Sistemas Operativos y Distribuidos

Iren Lorenzo Fonseca
iren.fonseca@.ua.es

TEMA 3. Sistemas Distribuidos.

Fundamentos de la Computación Distribuida

Tema 1. Fundamentos de la Computación Distribuida

Contenidos

Introducción y conceptos clave

Evolución de los modelos de computación distribuida

Enfoques de Gestión

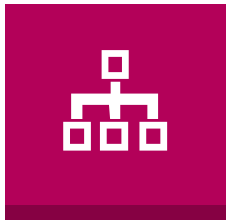
SOR, SOD, Middleware

Modelos arquitectónicos de sistemas distribuidos

C/S, N-Niveles, MOM, SOA, Cluster, Grid,P2P, Cloud, Edge

Introducción

Definiciones básicas

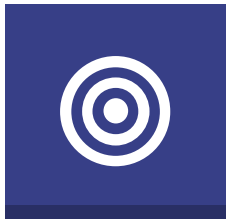


Sistema Distribuido

Elementos de computación **independientes**, **interconectados**, que **comunican** y **coordinan** sus **acciones**

Ejemplos de SD

Redes Sociales, Aplicaciones web, Sistemas de Mensajería, Sistema de Streaming, IoT, etc.



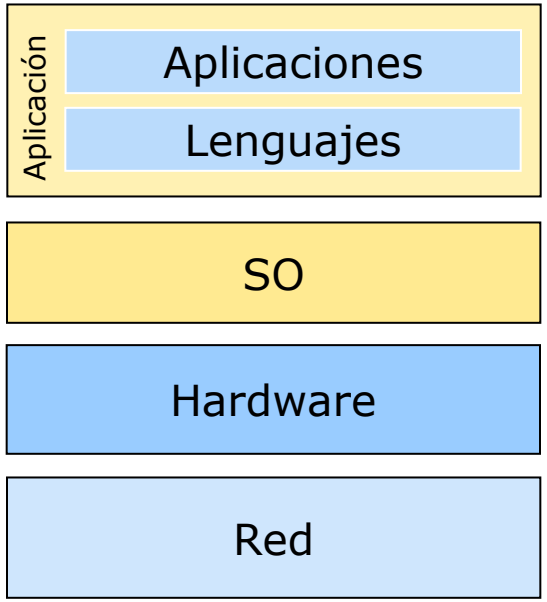
Computación Distribuida

La que se desarrolla en un SD: servicios y aplicaciones de red

Introducción

Características básicas

01 Heterogeneidad



- Integración
- Lenguajes intermedios
- UNIX, Windows, OS X
- Representación datos, precisión de las operaciones
- Ethernet, 802.11, ATM

- Estandarización
  - Representación de datos
  - Representación de código
  - Representación de objetos
- Protocolos

Introducción

Características básicas

01 **Heterogeneidad**

02 **Extensibilidad**

03 **Escalabilidad**

04 **Seguridad**

- Entornos proclives a ataques externos
- Confidencialidad
- Integridad
- Disponibilidad

Firewalls, SSL, HTTPS, Radius, Kerberos

Introducción

Características básicas

01 **Heterogeneidad**

02 **Extensibilidad**

03 **Escalabilidad**

04 **Seguridad**

05 **Concurrencia y sincronización**

06 **Tolerancia a fallos**

07 **Transparencia**

- De acceso
- De ubicación
- De movilidad
- De escalabilidad
- Frente a fallos

Introducción

Características básicas

01 **Heterogeneidad**

02 **Extensibilidad**

03 **Escalabilidad**

04 **Seguridad**

05 **Concurrencia y sincronización**

06 **Tolerancia a fallos**

07 **Transparencia**

08 **Administración de recursos**

Introducción

Características básicas. Resumen

Transparencia: Los usuarios no necesitan conocer la distribución física de los recursos. La transparencia de ubicación, acceso y replicación es clave para que los sistemas distribuidos se perciban como un único sistema.

Escalabilidad: Se pueden agregar o eliminar nodos (máquinas) sin que el sistema se vea significativamente afectado. Esto permite que los sistemas crezcan sin perder rendimiento.

Concurrencia: En un sistema distribuido, múltiples nodos pueden ejecutar procesos simultáneamente, lo que permite una mayor eficiencia y paralelismo.

Tolerancia a fallos: Los sistemas distribuidos pueden continuar operando incluso si uno o varios nodos fallan, gracias a mecanismos de redundancia y replicación.

Heterogeneidad: Los nodos en un sistema distribuido pueden ser de diferentes tipos, marcas o configuraciones, sin que esto le impida trabajar en conjunto.

Introducción

Ventajas

Mejor uso de recursos: Los sistemas distribuidos permiten aprovechar recursos de diferentes ubicaciones, aumentando la eficiencia y reduciendo el desperdicio de capacidad.

Disponibilidad: Gracias a la replicación y la distribución, los sistemas distribuidos suelen ser más resistentes a los fallos, garantizando la disponibilidad de los servicios.

Escalabilidad flexible: Se pueden escalar tanto horizontalmente (añadiendo más nodos) como verticalmente (mejorando los nodos existentes), lo que ofrece flexibilidad según las necesidades del sistema.

Introducción

Desafíos

Coordinación y comunicación: Mantener la sincronización y coordinación entre múltiples nodos puede ser complejo, sobre todo si están ubicados en diferentes regiones geográficas.

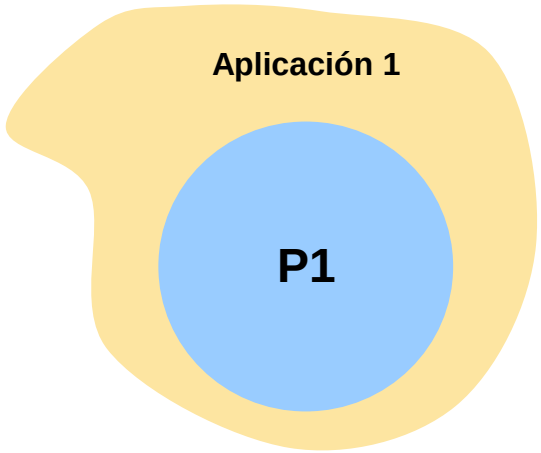
Consistencia de datos: Garantizar que los datos sean consistentes y estén actualizados en todos los nodos puede ser difícil, especialmente en sistemas con muchas réplicas.

Seguridad: Los sistemas distribuidos son más vulnerables a ataques debido a su mayor superficie de ataque, lo que requiere robustas políticas de seguridad y mecanismos de control de acceso.

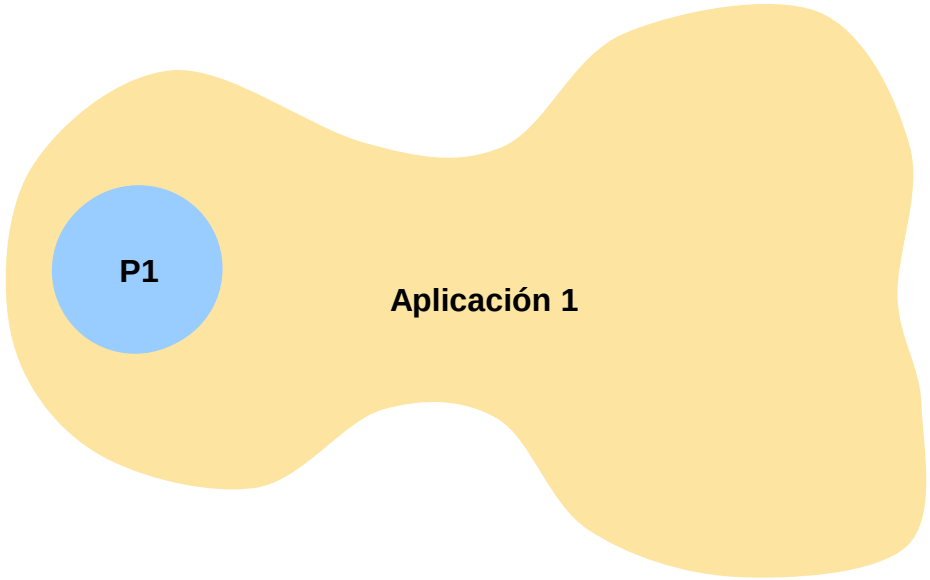
Evolución

- **Sistemas Monoprogramados**: Ejecutan un único programa a la vez, aprovechando todos los recursos del sistema. Ejemplo: los primeros sistemas operativos donde un proceso ocupaba toda la CPU.
- **Sistemas Multiprogramados y Multitarea**: Introducen la capacidad de ejecutar múltiples programas concurrentemente, optimizando tiempos de espera y uso de CPU.
- **Sistemas Multiprocesador**: Incorporan varias CPUs que operan de forma coordinada, con interconexión entre procesadores y mecanismos de comunicación.
- **Redes de Computadores**: Conectan varios equipos, expandiendo la distribución de tareas y procesos en múltiples máquinas conectadas en red.
- **Middleware**: Introduce una capa que facilita la comunicación y coordinación en sistemas complejos, abstrayendo la infraestructura. Ejemplo: CORBA, gRPC.
- **Microservicios y Orquestadores**: Dividen aplicaciones en módulos autónomos, gestionados por orquestadores (Kubernetes) y brokers de mensajes (Kafka), permitiendo escalabilidad y flexibilidad en entornos cloud.

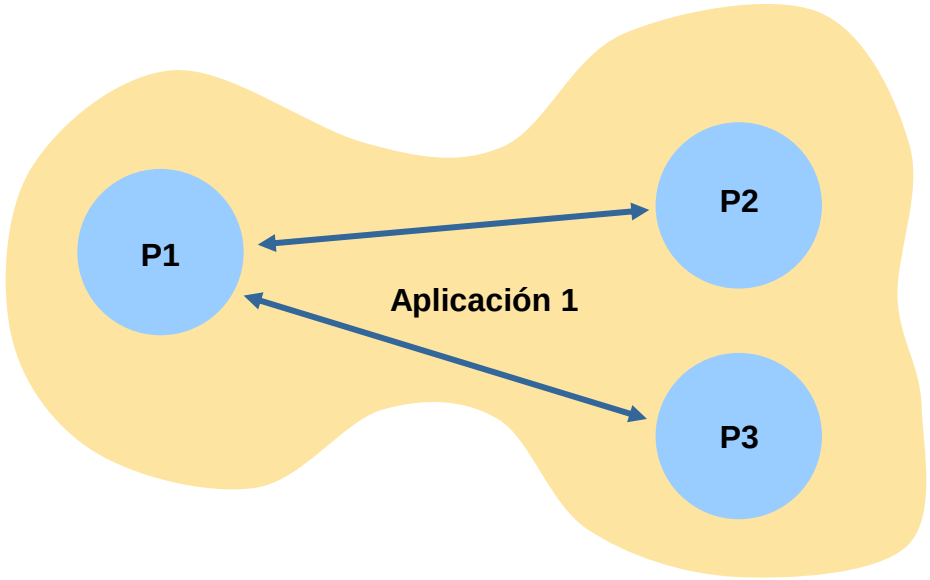
Evolución



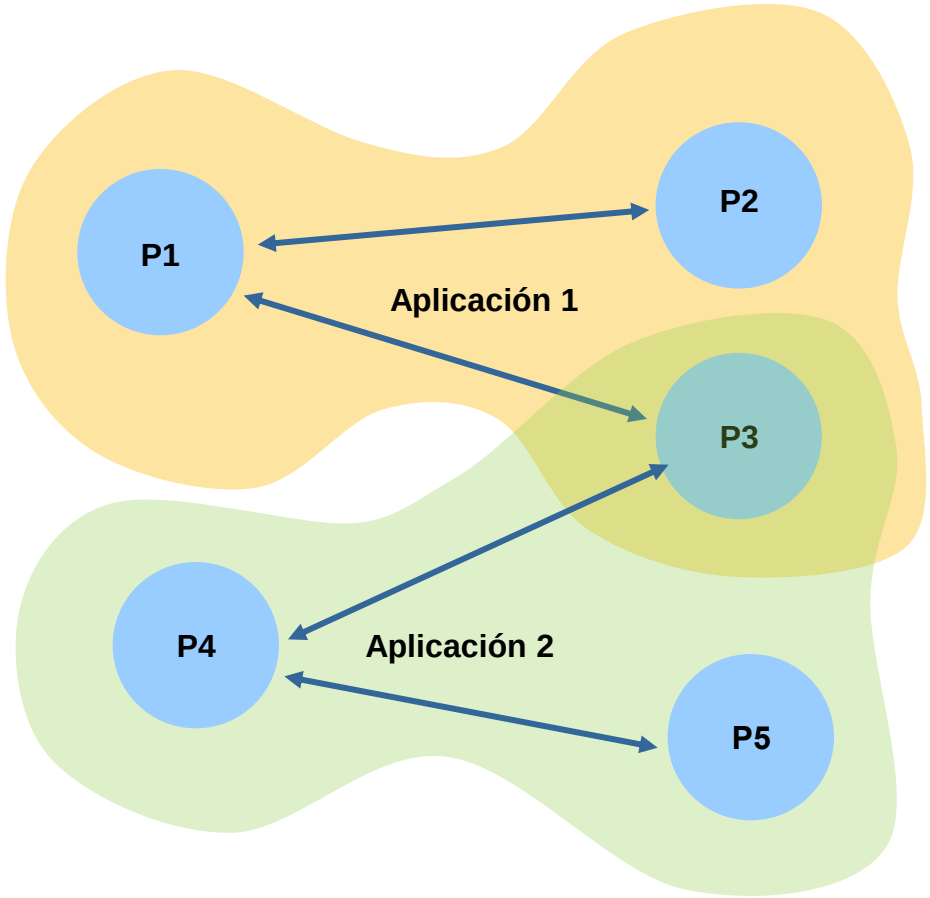
Evolución



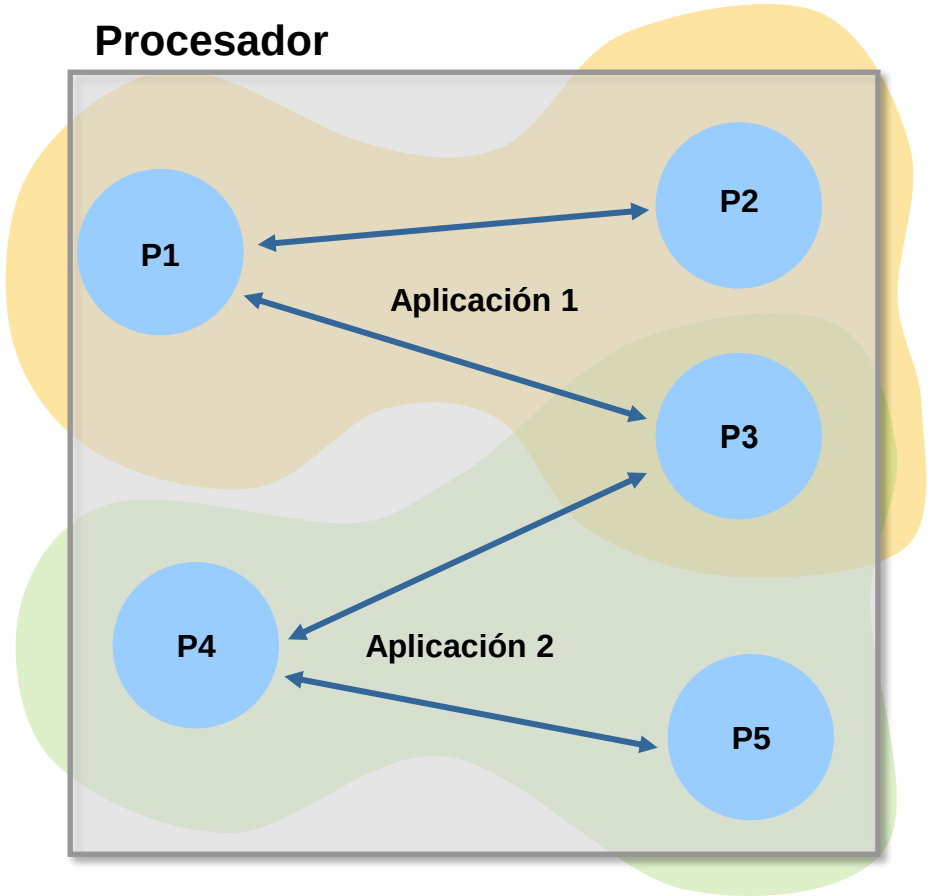
Evolución



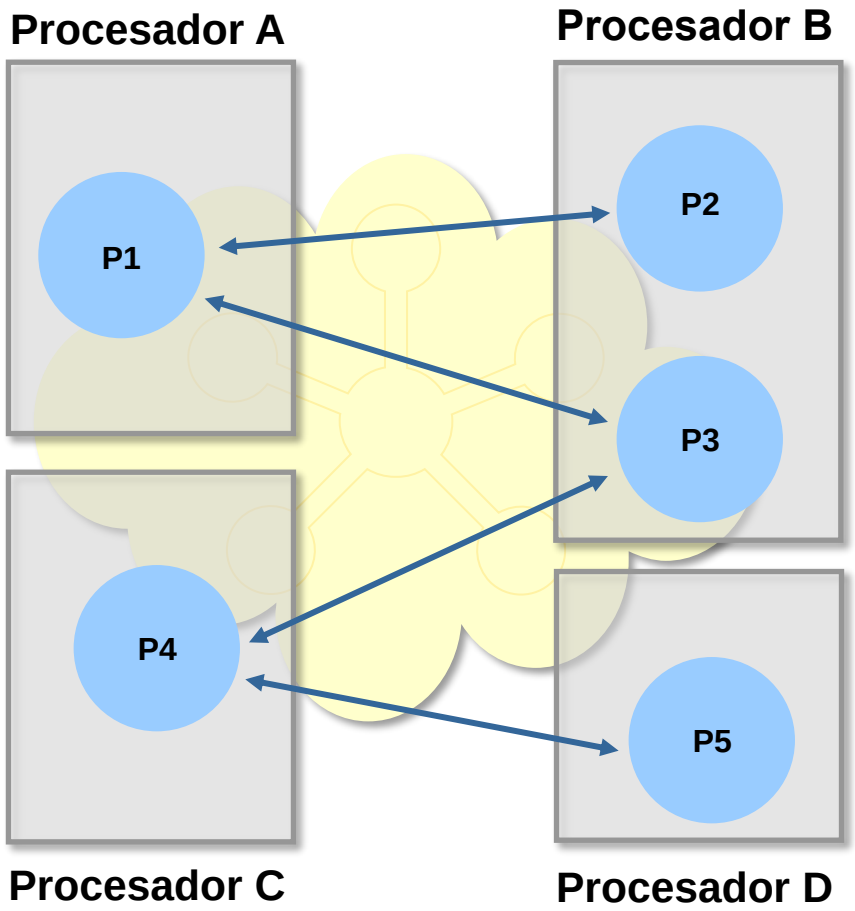
Evolución



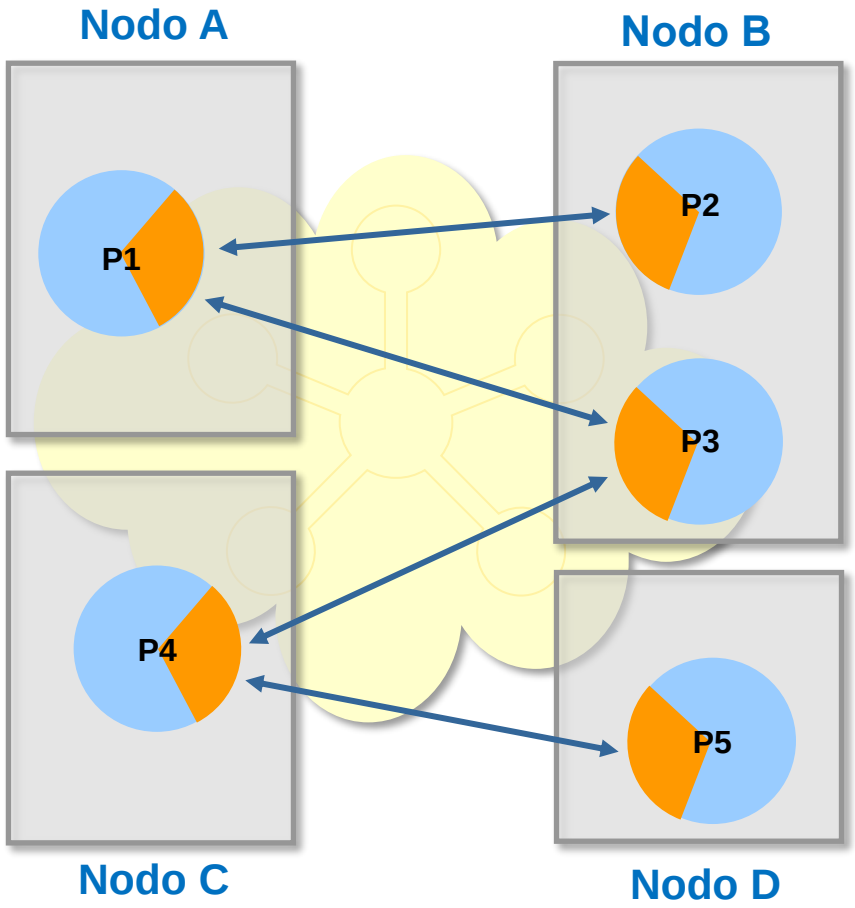
Evolución



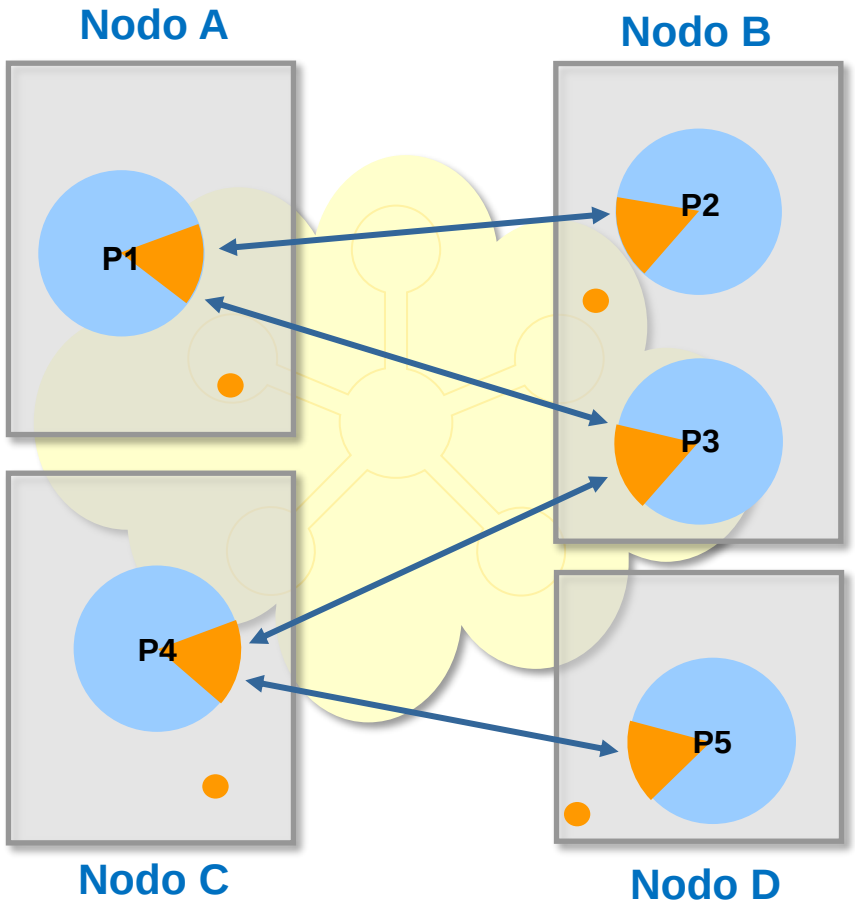
Evolución


Procesador A
Procesador B
P1
P2
P3
P4
P5
Procesador C
Procesador D

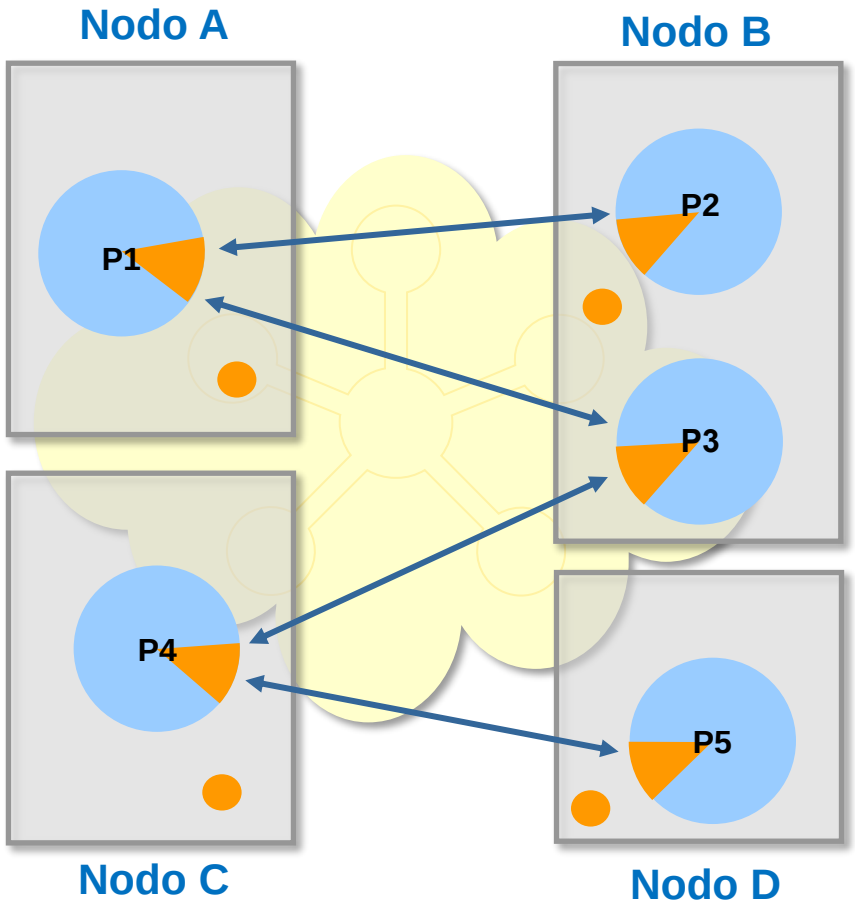
Evolución



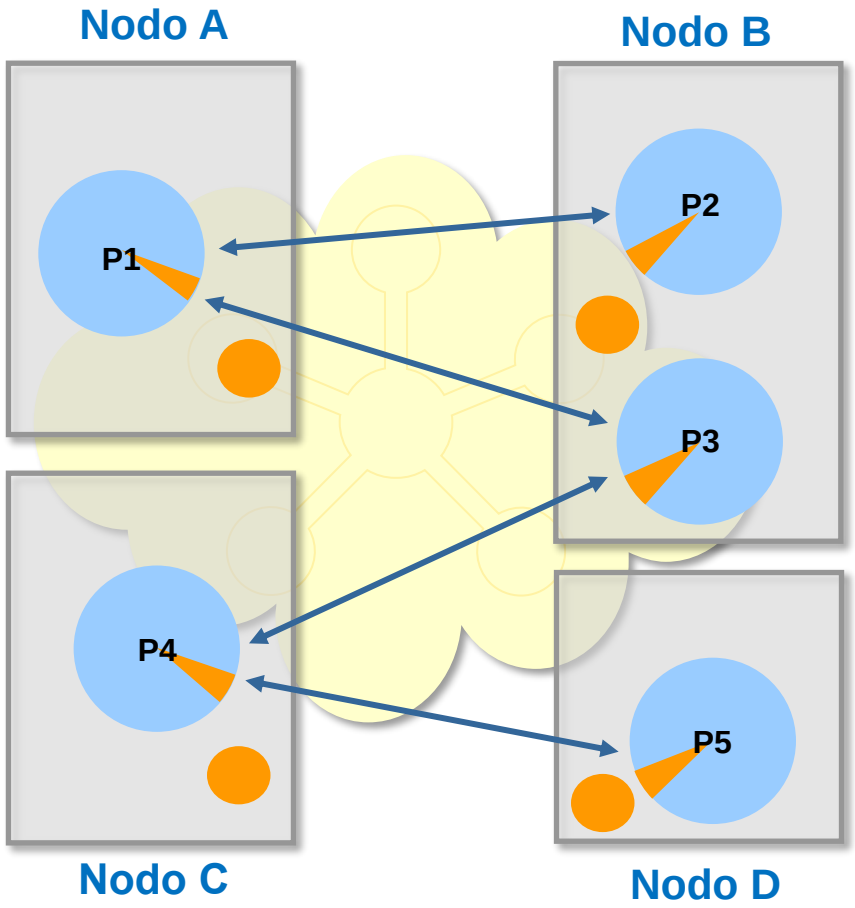
Evolución



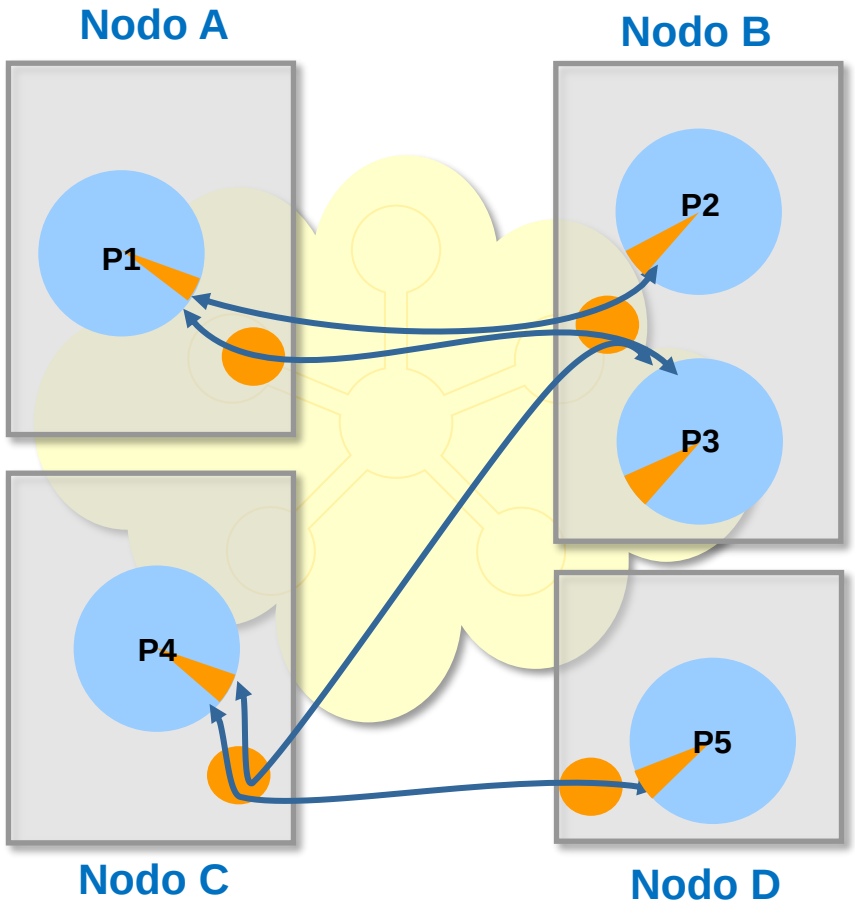
Evolución



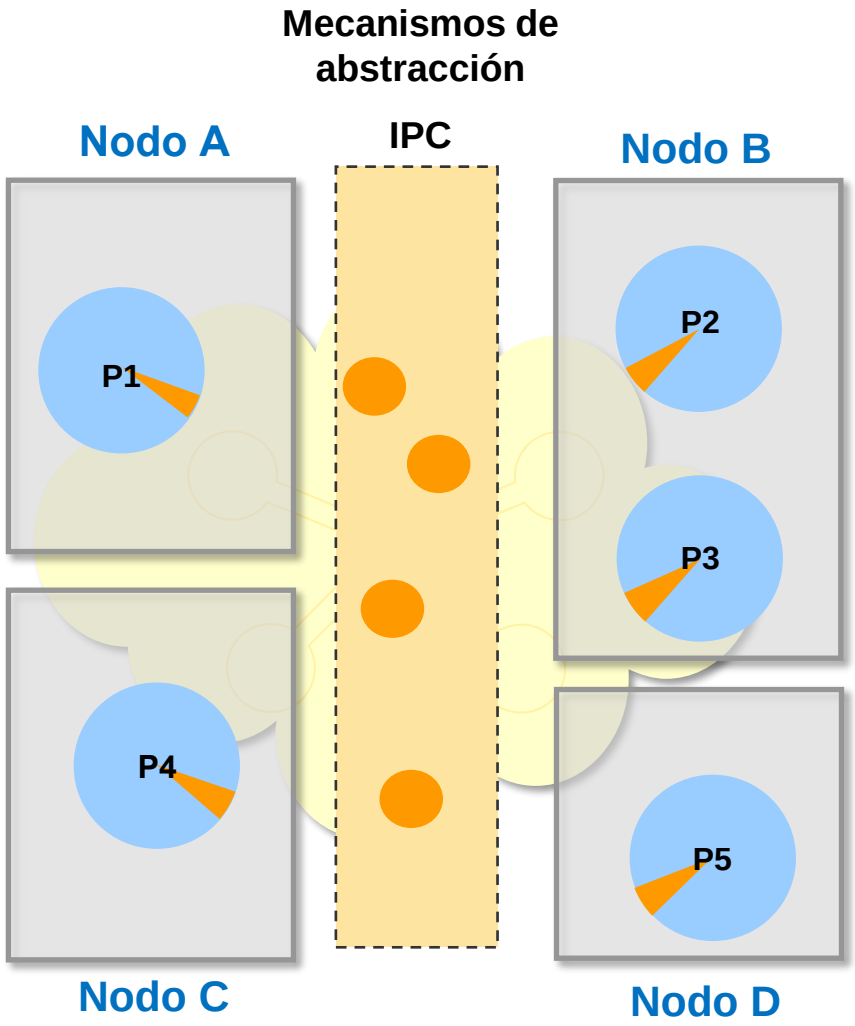
Evolución



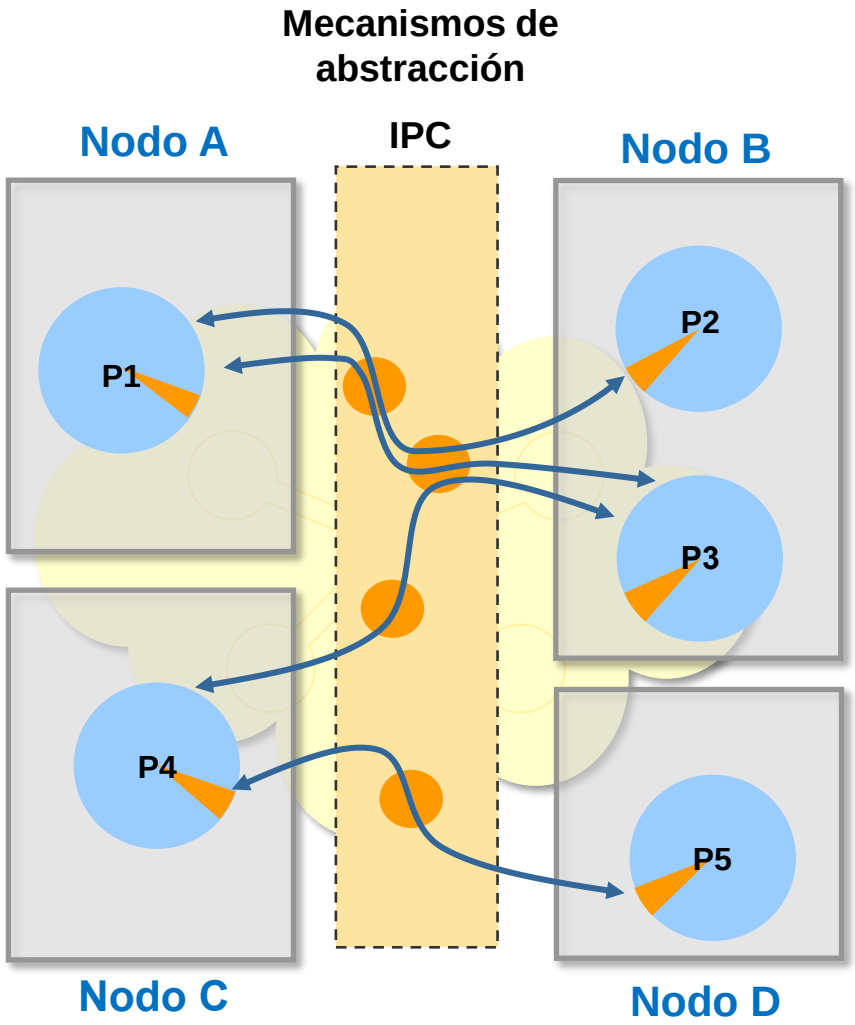
Evolución

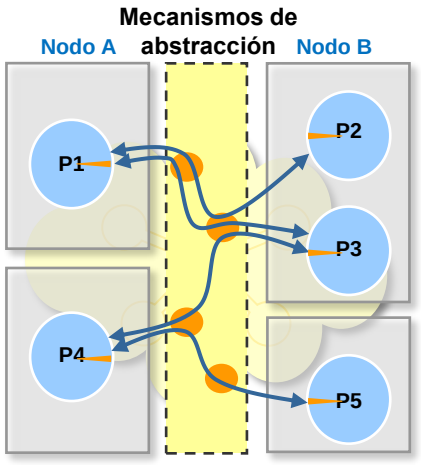


Evolución


Mecanismos de abstracción
Nodo A
IPC
Nodo B
P1
P2
P3
P4
P5
Nodo C
Nodo D

Evolución


Mecanismos de abstracción
Nodo A
IPC
Nodo B
P1
P2
P3
P4
P5
Nodo C
Nodo D

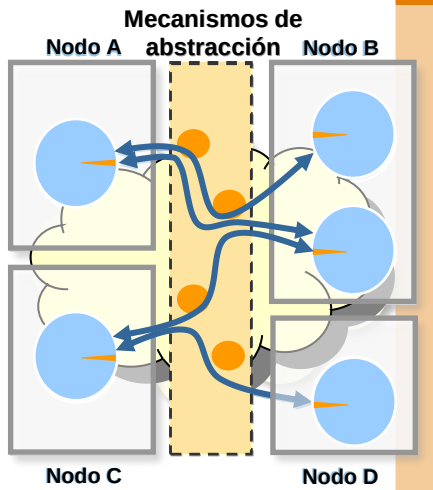


- [x] Sistemas Operativos en Red
- [x] Sistemas Operativos Distribuidos
- [x] Middleware

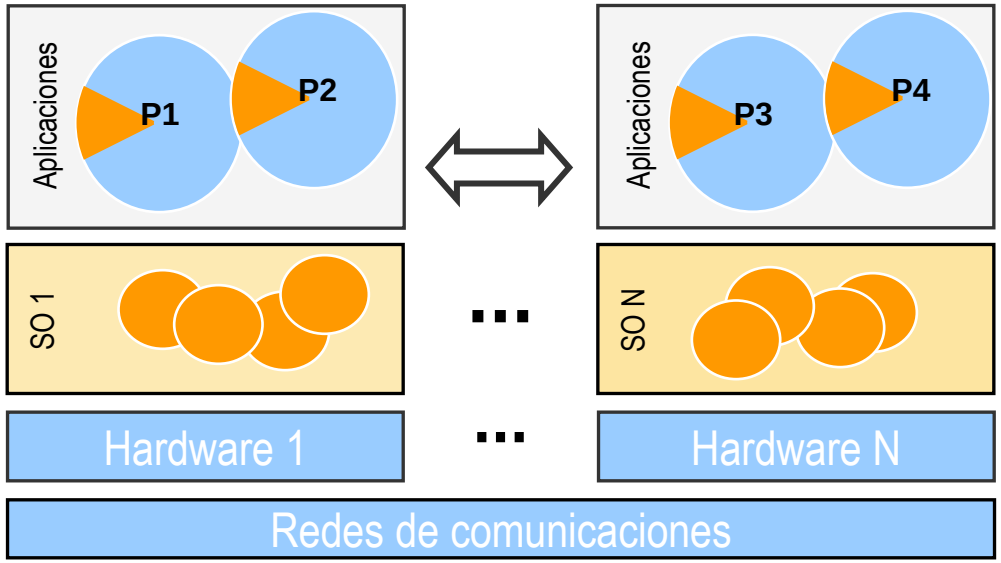
Enfoques de Gestión

Contenidos

SO en Red



- Ubicación IPC en el SO
- **Heterogéneo** → diferentes del SO
- Ejemplos:
  - Linux, Windows, OS X, Android, ….
- Ventajas
  - Flexibilidad
  - Técnicas maduras
- Desventajas
  - Falta de transparencia
  - Mayor esfuerzo de integración



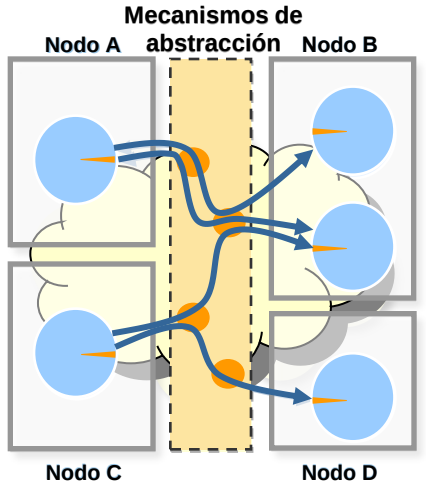
Enfoques de Gestión

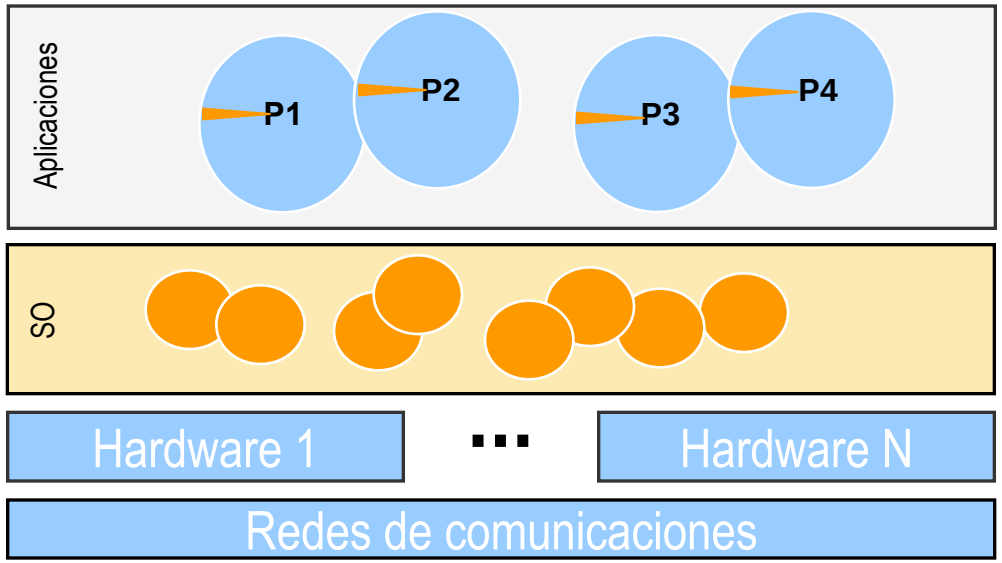
Ejemplo de código en SOR

```
void ClienteHTTP(char*dirIP, int puerto, char *recurso, char HTTP_response[])
{
    int    sfd;                              // descriptor del socket del cliente
    static struct sockaddr_in sa;            // dirección IPv4 del servidor
    int    bRecibidos;
    char   HTTP_request[8000];


    // SOCKET: Obtenemos el socket (INET)
    sfd = socket(AF_INET, SOCK_STREAM, 0);

    // CONNECT: Preparamos la conexión con el servidor...
    sa.sin_family = AF_INET;
    sa.sin_addr.s_addr = inet_addr( dirIP );
    sa.sin_port = htons( (uint16_t)puerto );

    // ...Conectamos con el servidor y puerto indicados
    connect(sfd, (struct sockaddr *)&sa, sizeof(sa));

    // HTTP_REQUEST: Enviamos la solicitud GET sobre el recurso solicitado (index.html por defecto)
    sprintf(HTTP_request, "GET %s HTTP/1.0\r\n\r\n", recurso );
    write(sfd, HTTP_request, strlen(HTTP_request));

    // HTTP_RESPONSE: leemos la respuesta en HTTP_response
    bRecibidos = (int)read(sfd, HTTP_response, 40000);

    // Tratamos la respuesta
    printf("%s", HTTP_response);

    // CLOSE: Cerramos el socket
    close(sfd);
}          // de ClienteHTTP
```

Sistemas operativos distribuidos

- Ubicación IPC en SO global
- **Homogéneo** → SO único
- Ventajas
  - Transparencia
  - Escalabilidad
  - Facilidad de integración
- Desventajas
  - Técnicas complejas
  - Rendimiento y Latencia
- Ejemplos:
  - Mach, Amoeba

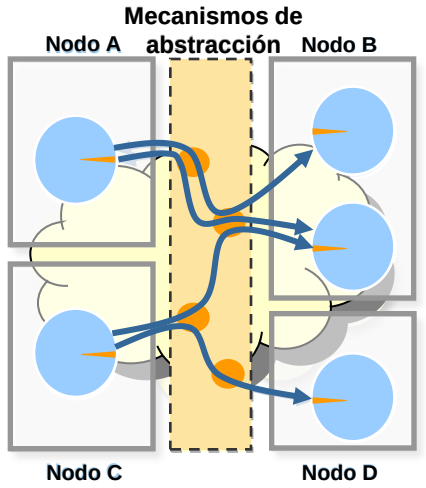


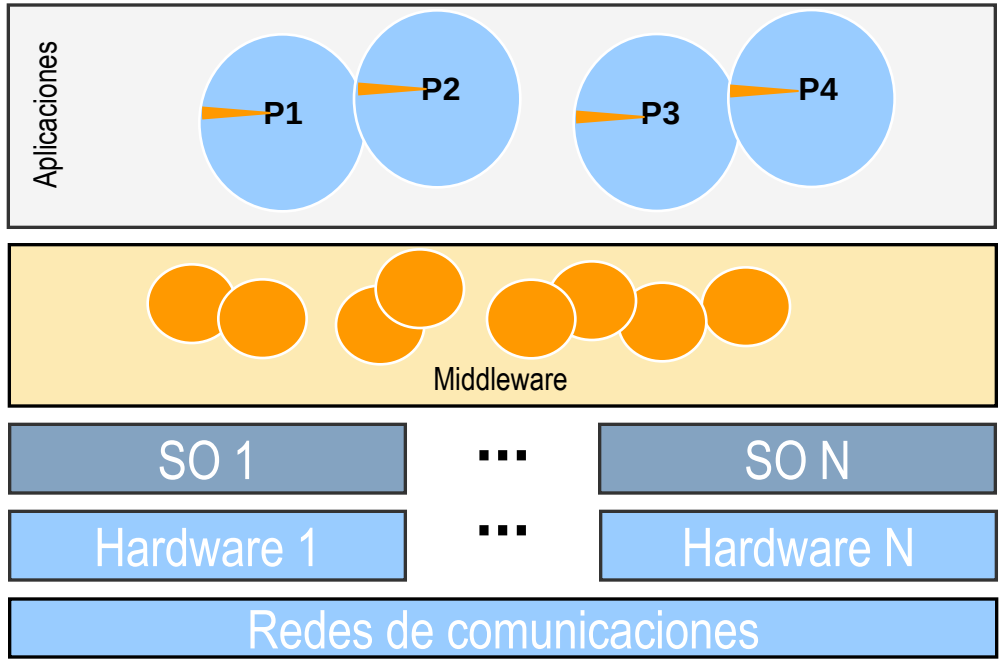


Enfoques de Gestión

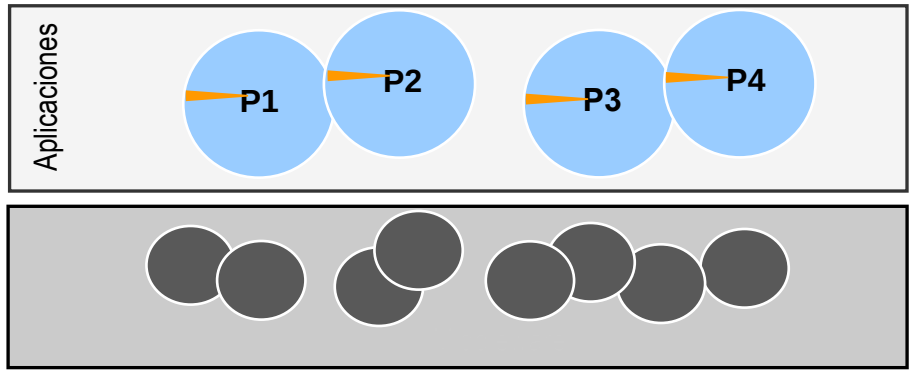
Middleware

- Enfoque **mixto**
  - Modelo **conceptual → SOD**
  - **Infraestructuras → SOR**
- Capa por encima del SO
- **Homogéneo**
- Ejemplos:
  - CORBA
  - JEE
  - .Net
- Ventajas
  - Flexibilidad
  - Transparencia
  - Integración
  - Madurez
  - Escalabilidad
- Desventajas
  - Plataformas heterogéneas
  - Necesidad de estandarización
  - Gran consumo de recursos





Enfoques de Gestión



- Modelo Cliente/Servidor
- Arquitectura de N-Niveles
- Middleware orientado a mensajes
- Arquitectura orientada a servicios
- Arquitectura de Microservicios
- Cluster y Grid
- Peer-to-Peer
- Arquitectura Cloud
- Edge computing

Modelos Arquitectónicos

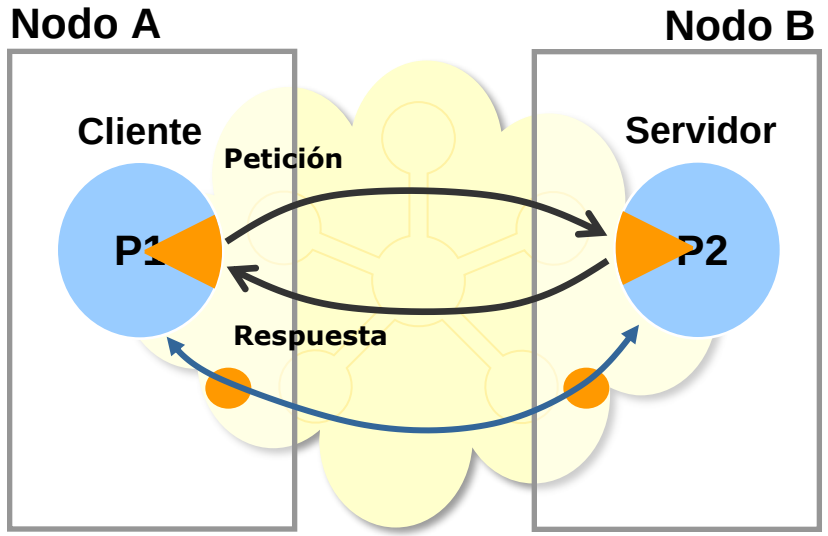
Contenidos

Modelo Cliente/Servidor

- El **modelo cliente-servidor** es una arquitectura ampliamente utilizada en sistemas distribuidos.
- Se basa en la interacción entre un **cliente** (que solicita servicios o recursos) y un **servidor** (que los provee).
- La comunicación sigue un esquema de **petición-respuesta**.

Modelo Cliente/Servidor

- **Procesos**
  - **Servidor** (Proveedor de servicio en espera pasiva)
  - **Cliente** (Solicita servicio)
- **Características principales:**
  - **Simplicidad**: fácil de implementar y gestionar.
  - **Escalabilidad**: los servidores pueden escalar horizontal o verticalmente.
  - **Centralización del control**: facilita la seguridad y gestión.
- **Ejemplo de aplicación:**
  - Aplicaciones web (cliente: navegador, servidor: servidor web).
  - Servicio ftp



Ejemplo de código en SOR

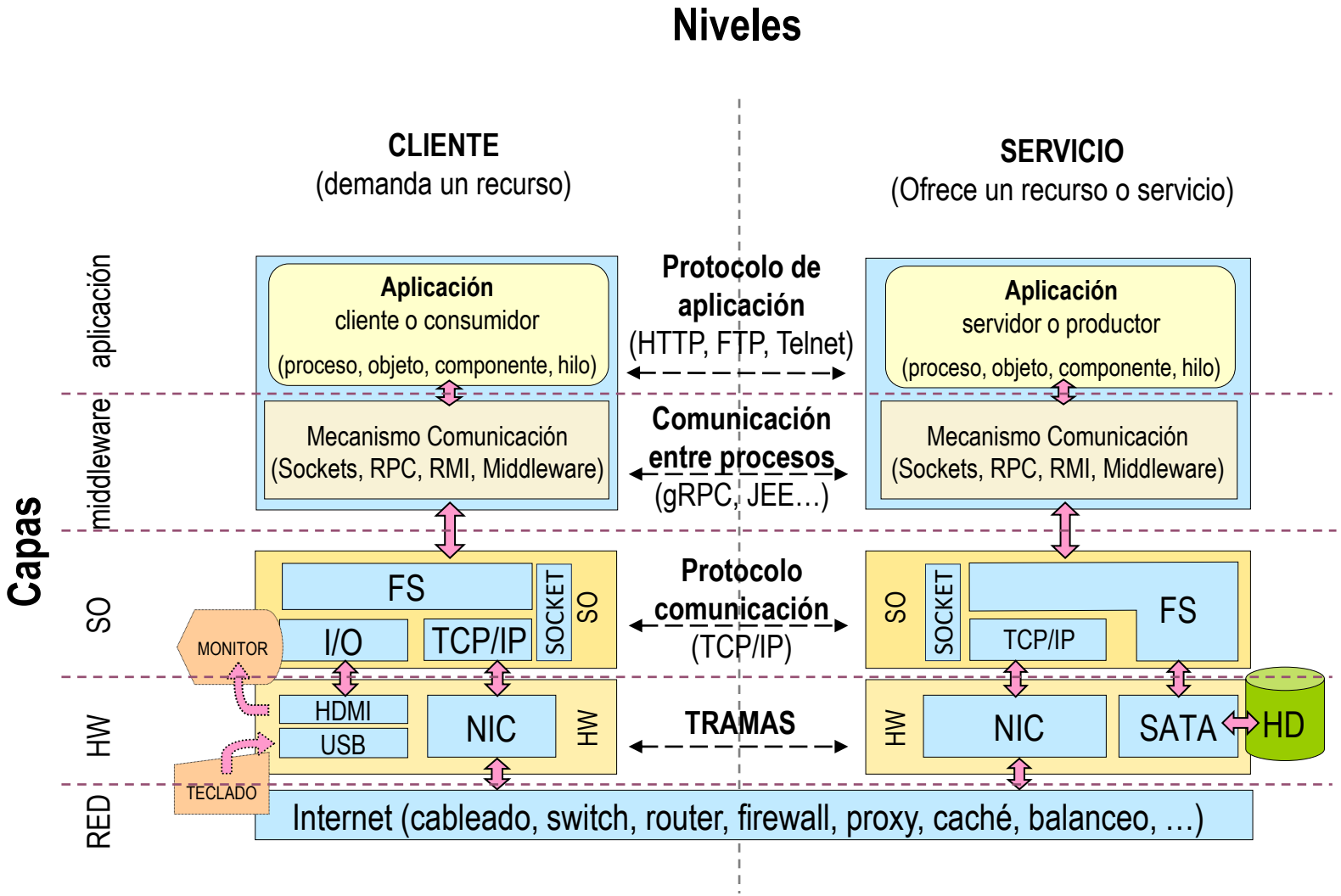
```
void ClienteHTTP(char*dirIP, int puerto, char *recurso, char HTTP_response[])
{
    int    sfd;                          // descriptor del socket del cliente
    static struct sockaddr_in sa;        // dirección IPv4 del servidor
    int    bRecibidos;
    char   HTTP_request[8000];


    // SOCKET: Obtenemos el socket (INET)
    sfd = socket(AF_INET, SOCK_STREAM, 0);

    // CONNECT: Preparamos la conexión con el servidor...
    sa.sin_family = AF_INET;
    sa.sin_addr.s_addr = inet_addr( dirIP );
    sa.sin_port = htons( (uint16_t)puerto );

    // ...Conectamos con el servidor y puerto indicados
    connect(sfd, (struct sockaddr *)&sa, sizeof(sa));

    // HTTP_REQUEST: Enviamos la solicitud GET sobre el recurso solicitado (index.html por defecto)
    sprintf(HTTP_request, "GET %s HTTP/1.0\r\n\r\n", recurso );
    write(sfd, HTTP_request, strlen(HTTP_request));

    // HTTP_RESPONSE: leemos la respuesta en HTTP_response
    bRecibidos = (int)read(sfd, HTTP_response, 40000);

    // Tratamos la respuesta
    printf("%s", HTTP_response);

    // CLOSE: Cerramos el socket
    close(sfd);
}       // de ClienteHTTP
```

Enfoques de SO

Modelos Arquitectónicos

Modelo Cliente/Servidor

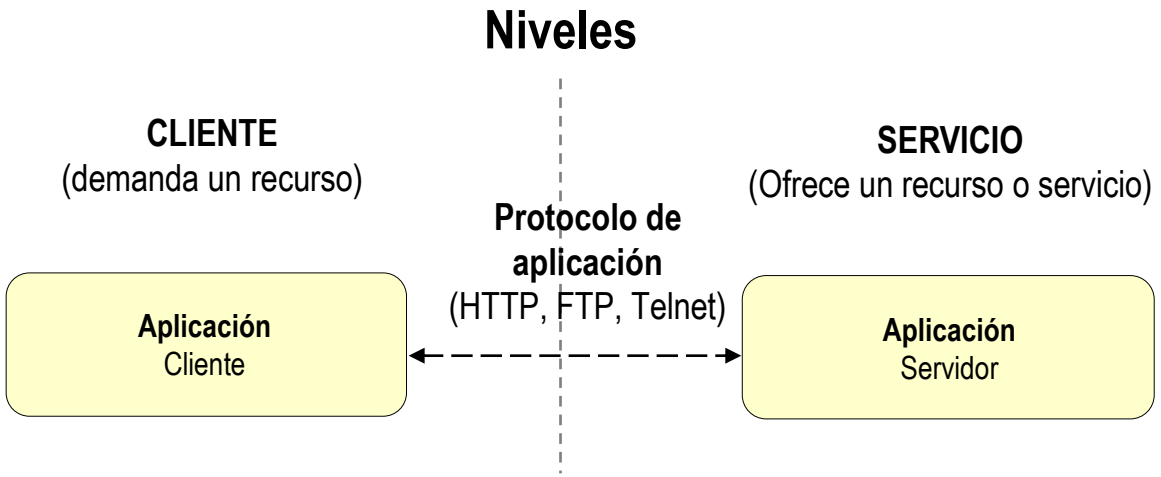
Arquitectura C/S de 2-niveles



Modelos Arquitectónicos

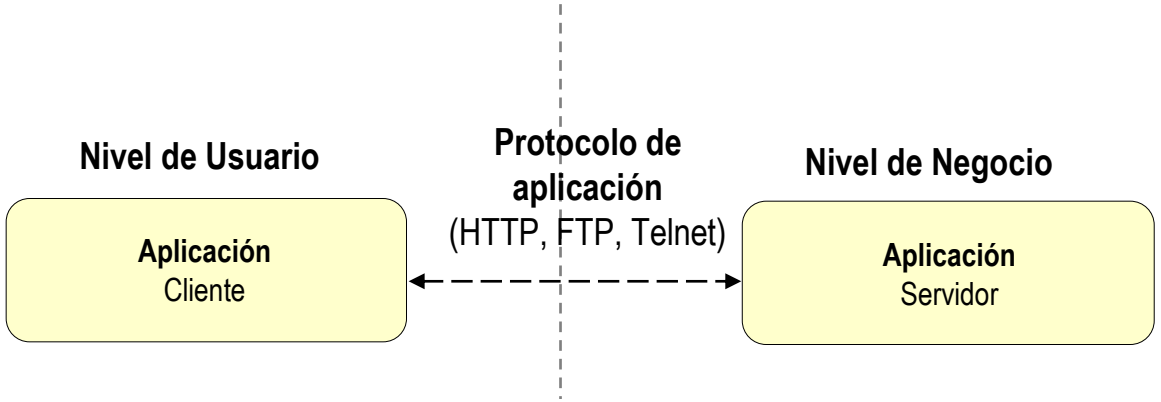
Modelo Cliente/Servidor

Arquitectura C/S



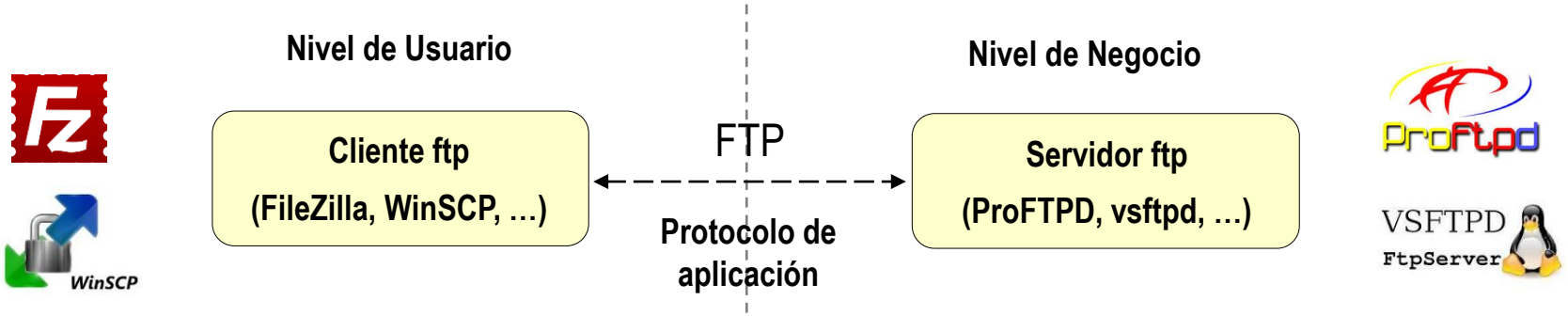
Modelo Cliente/Servidor

Arquitectura C/S

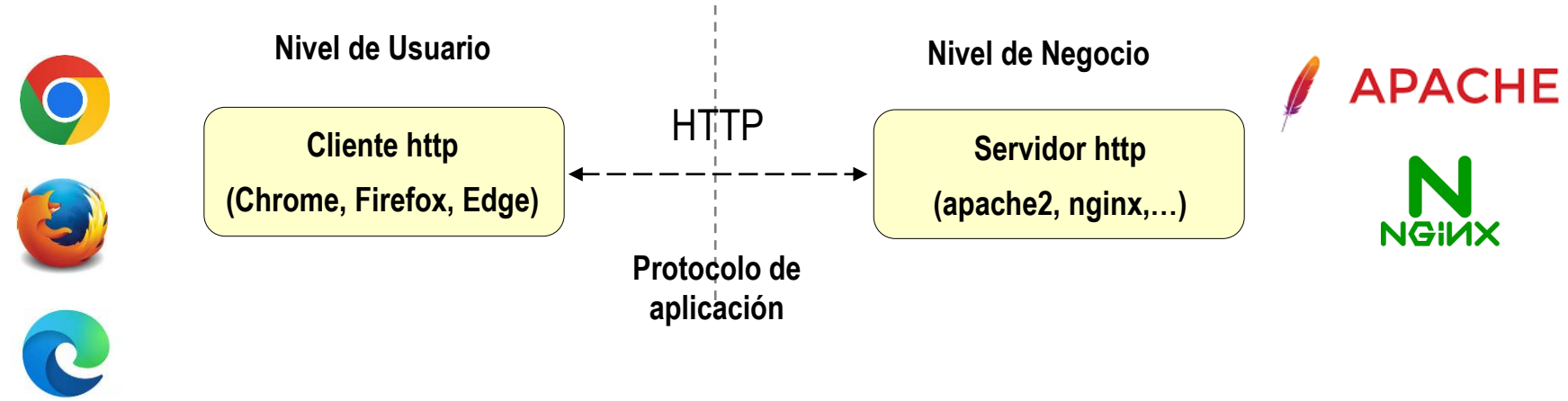


Modelo Cliente/Servidor

Arquitectura C/S de 2-niveles del servicio FTP

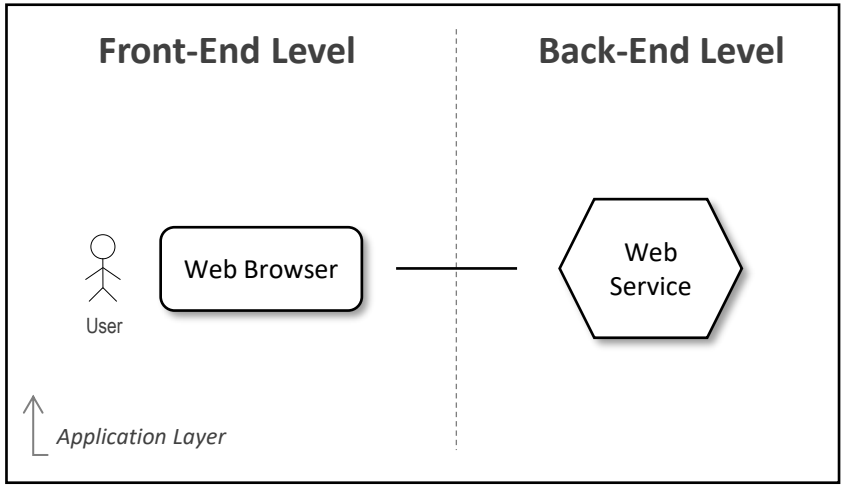


Arquitectura C/S de 2-niveles del servicio HTTP



Modelo Cliente/Servidor

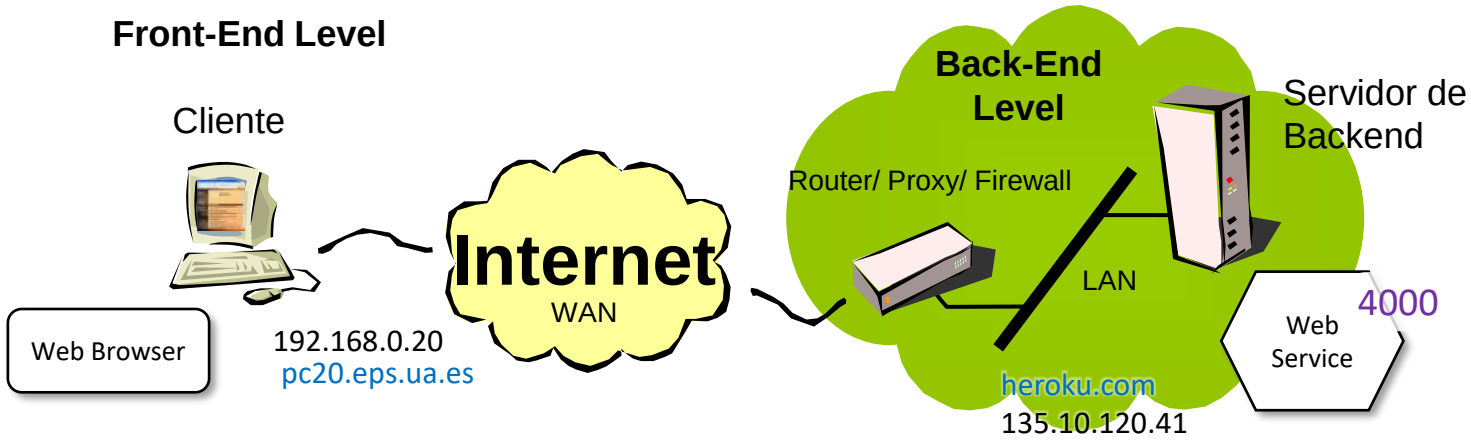
Diagrama de la Arquitectura Distribuida de una Aplicación Web



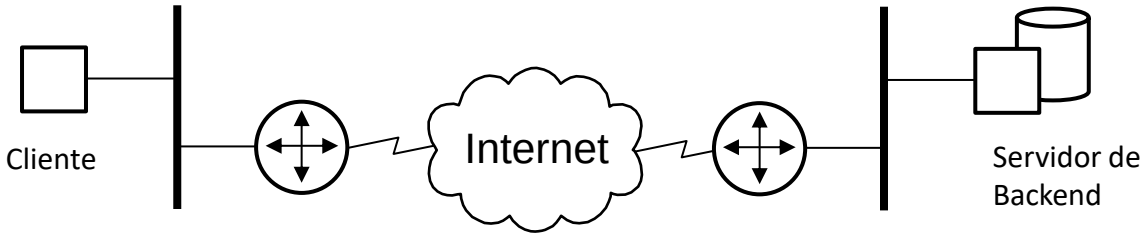
Modelo Cliente/Servidor

Posible Escenario de Despliegue para la Aplicación Web

escenario de despliegue



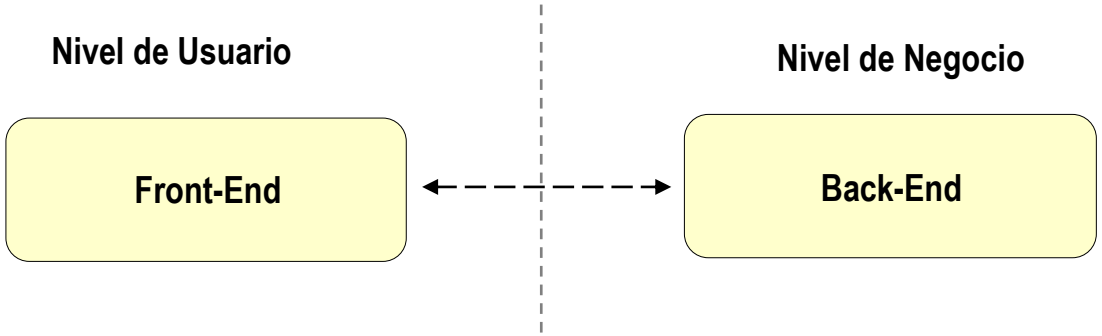
arquitectura física de despliegue



Arquitectura de N-Niveles

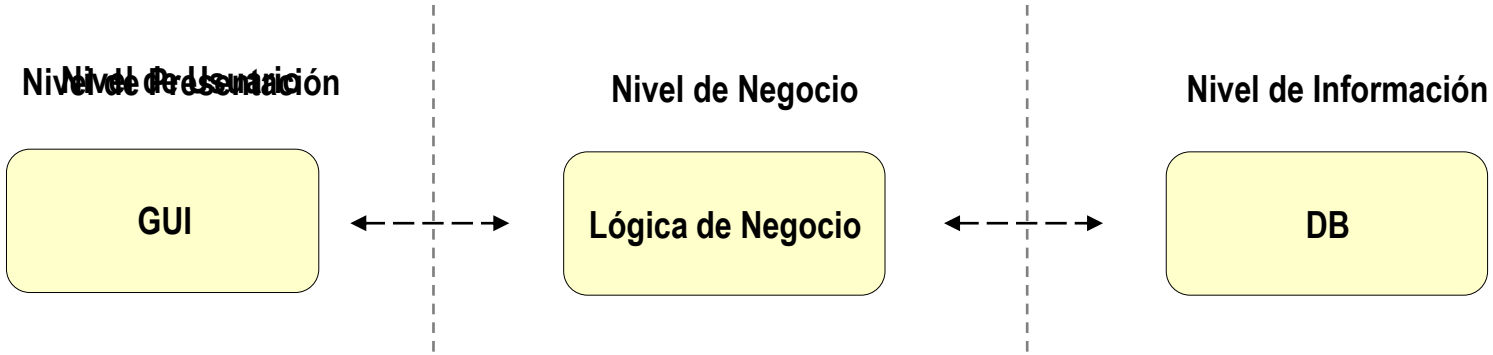
Arquitectura de N-Niveles

Arquitectura C/S de 2-niveles



Arquitectura de N-Niveles

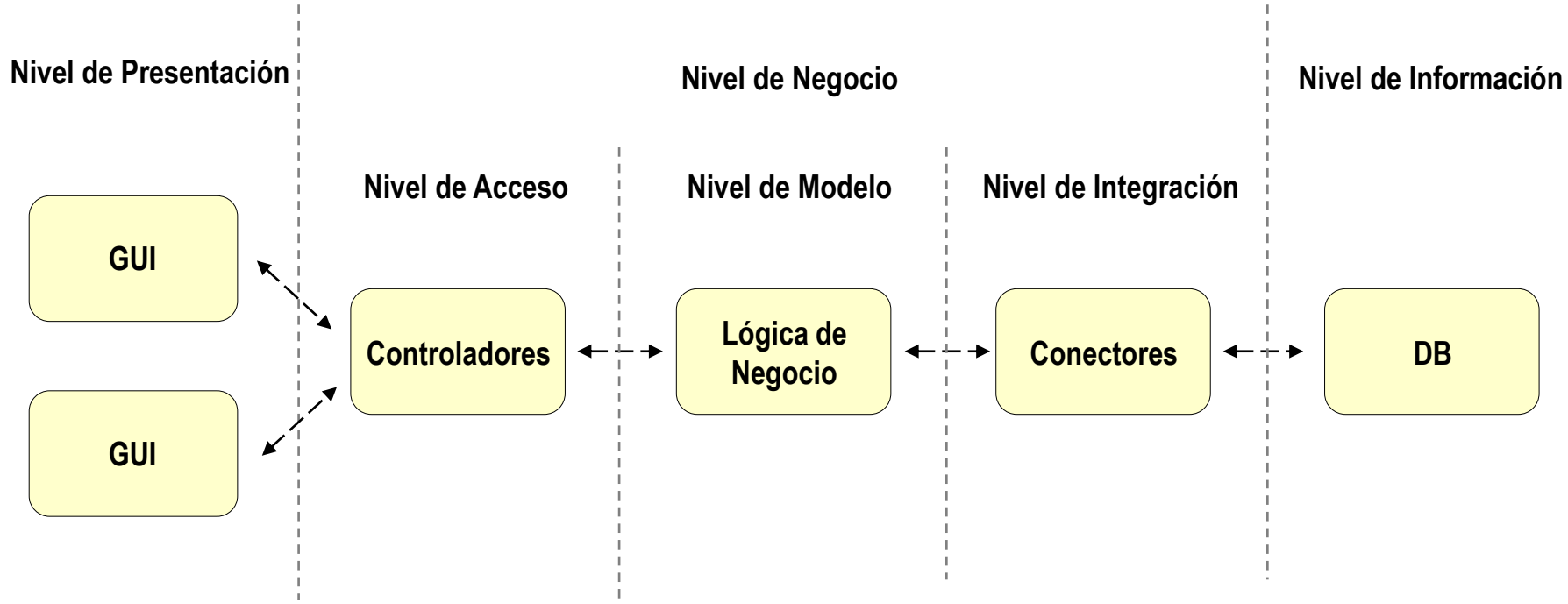
Arquitectura C/S de 3-niveles



Arquitectura C/S de 3-niveles empresarial

Arquitectura de N-Niveles

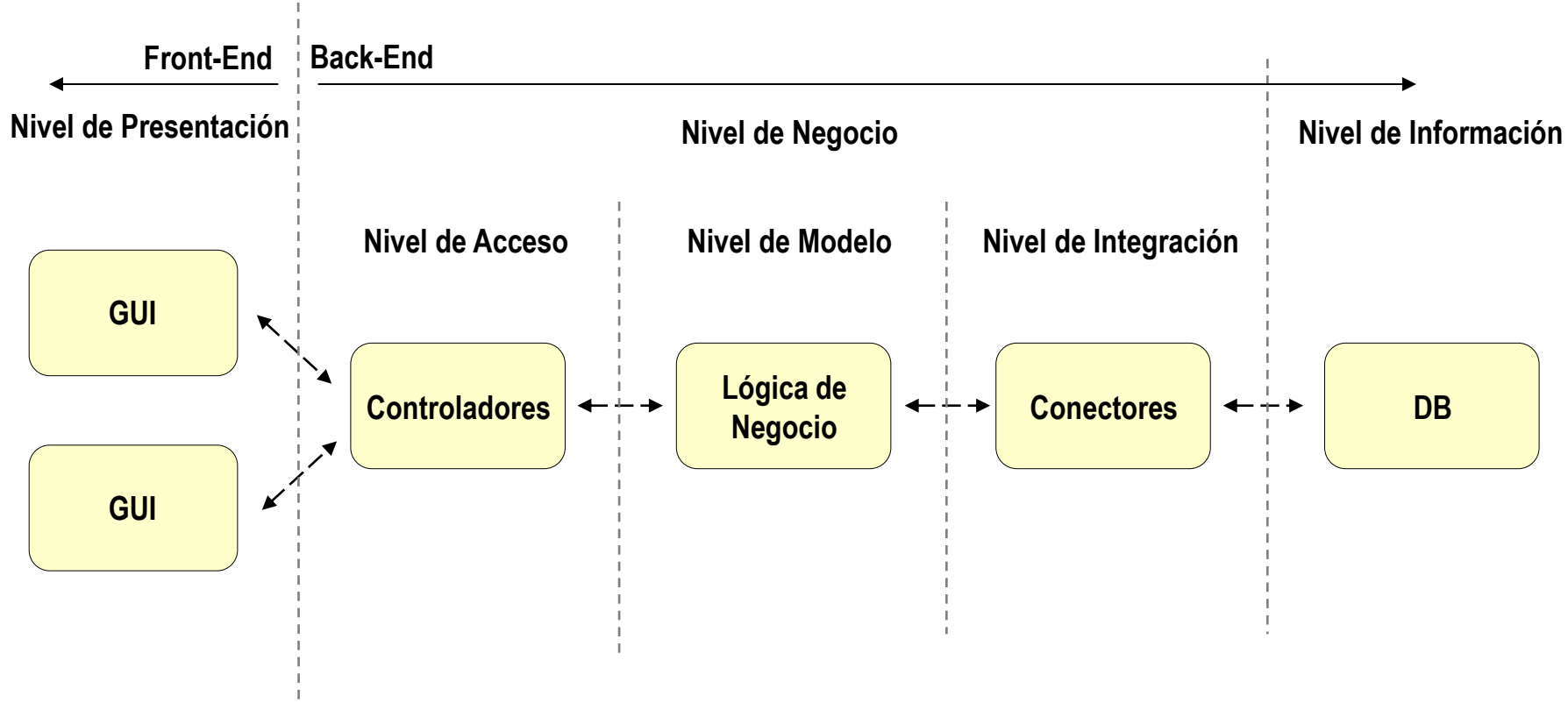
Arquitectura C/S de n-niveles



Arquitectura C/S de n-niveles empresarial

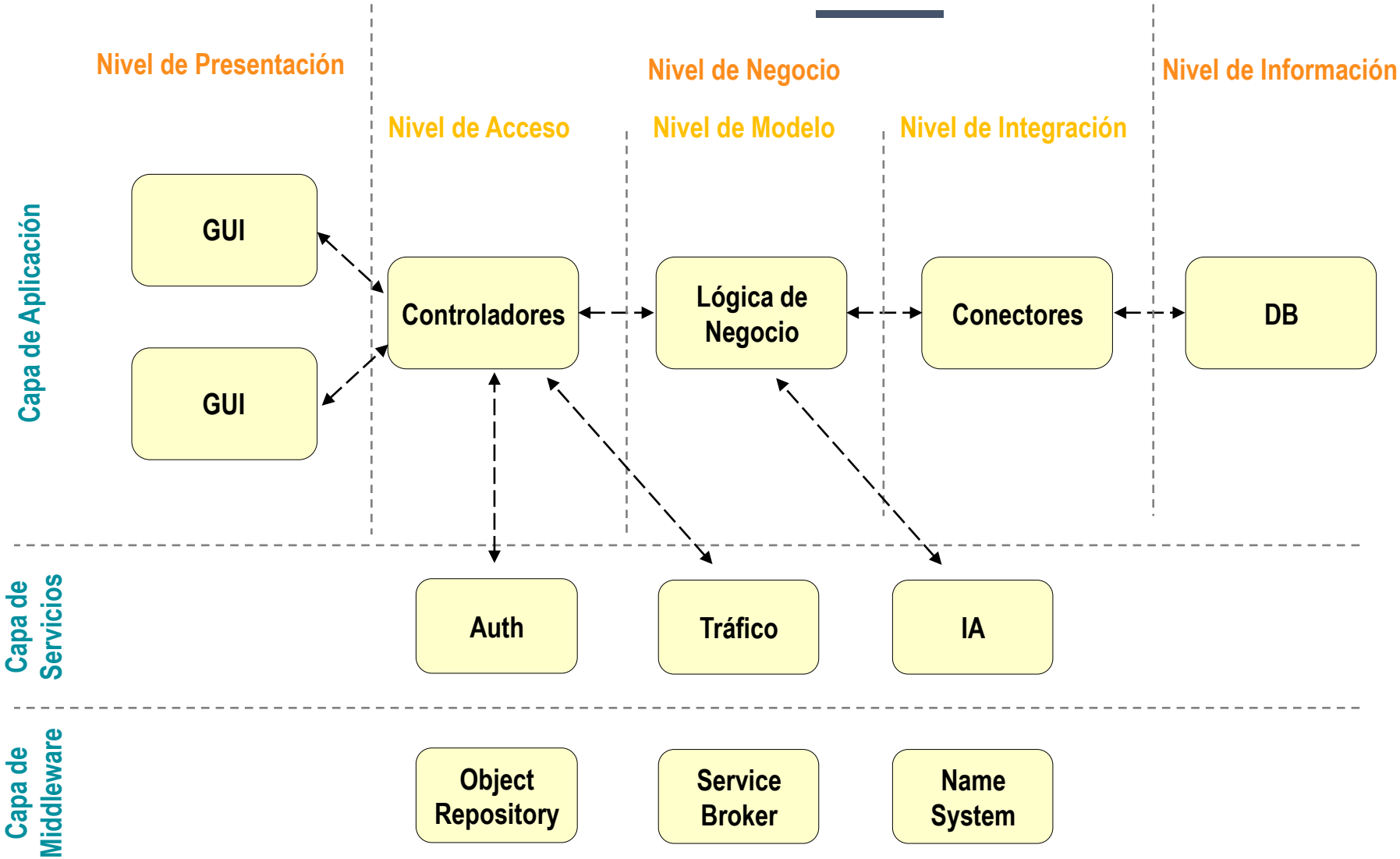
Arquitectura de N-Niveles

Arquitectura C/S de n-niveles



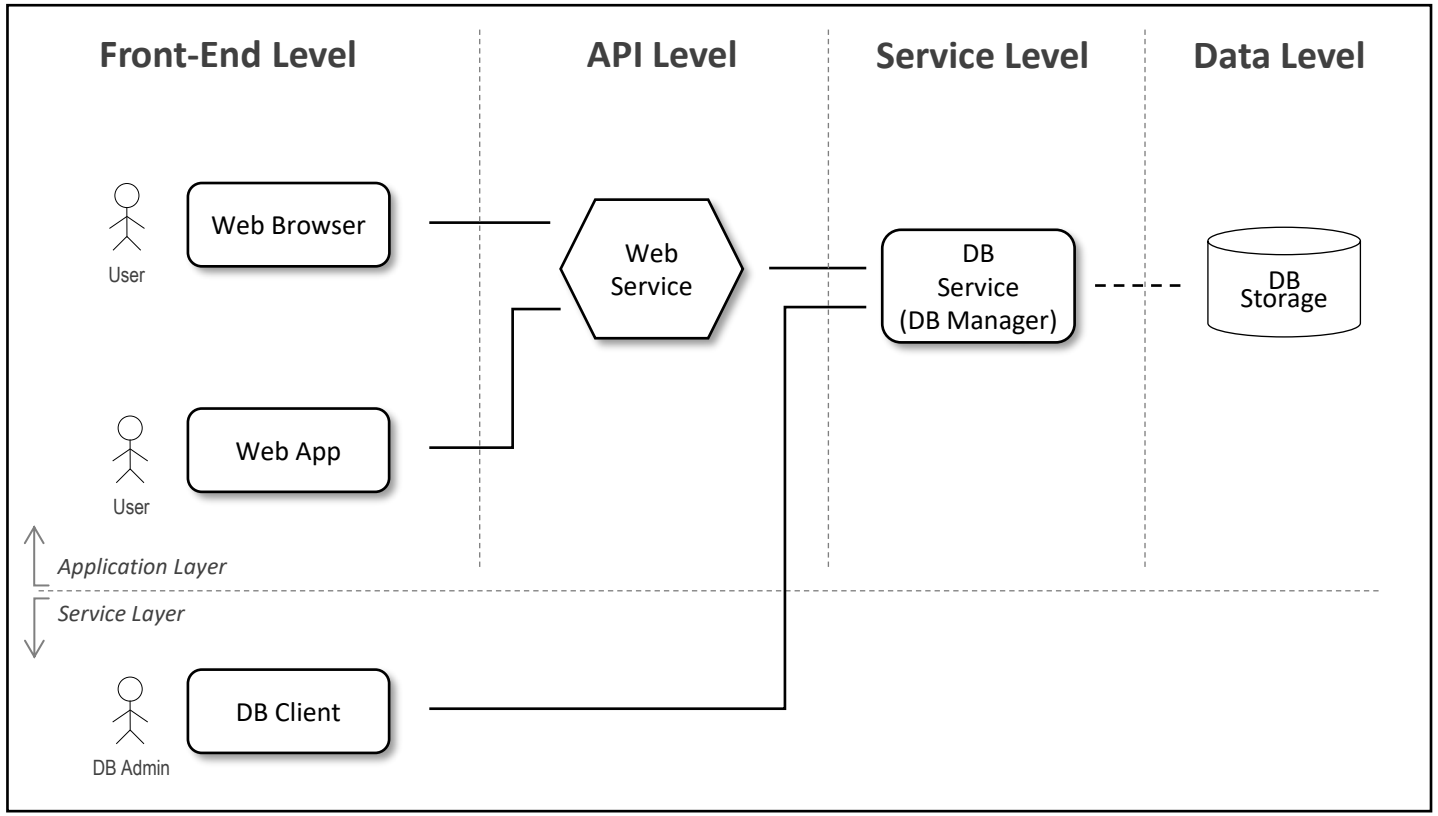
Arquitectura de N-Niveles

Arquitectura C/S de n-niveles



Arquitectura de N-Niveles

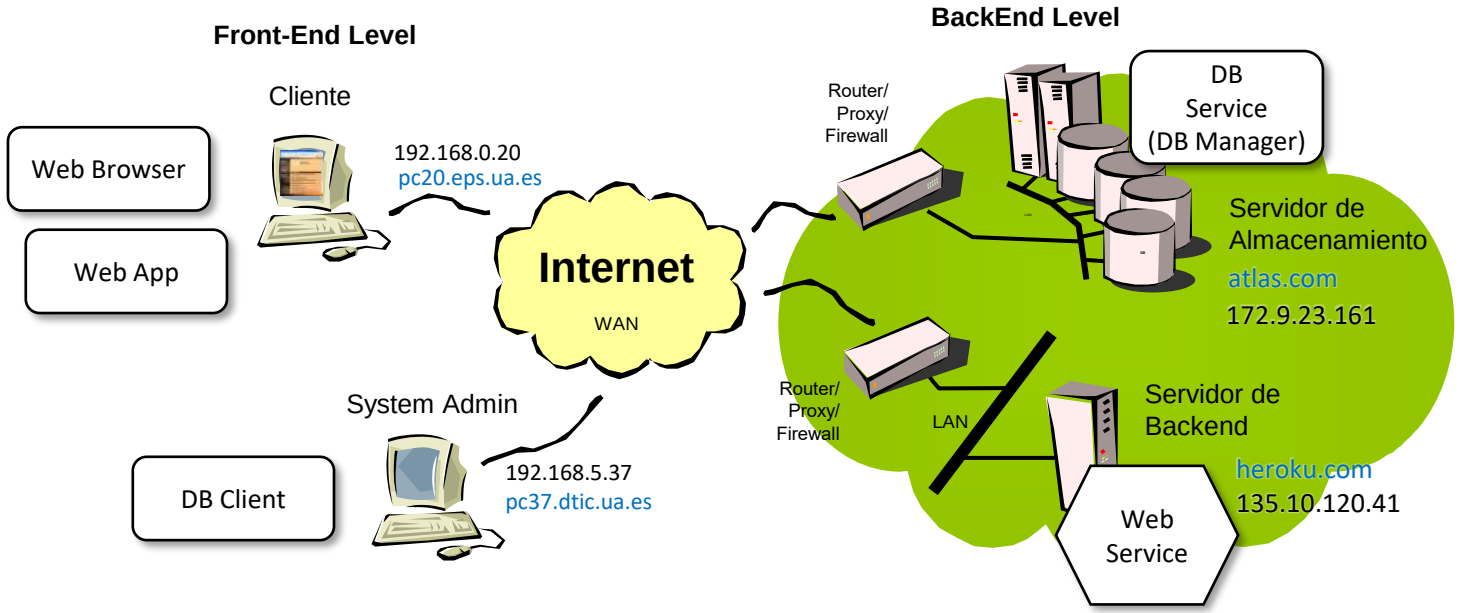
Diagrama de la Arquitectura Distribuida de una Aplicación Web



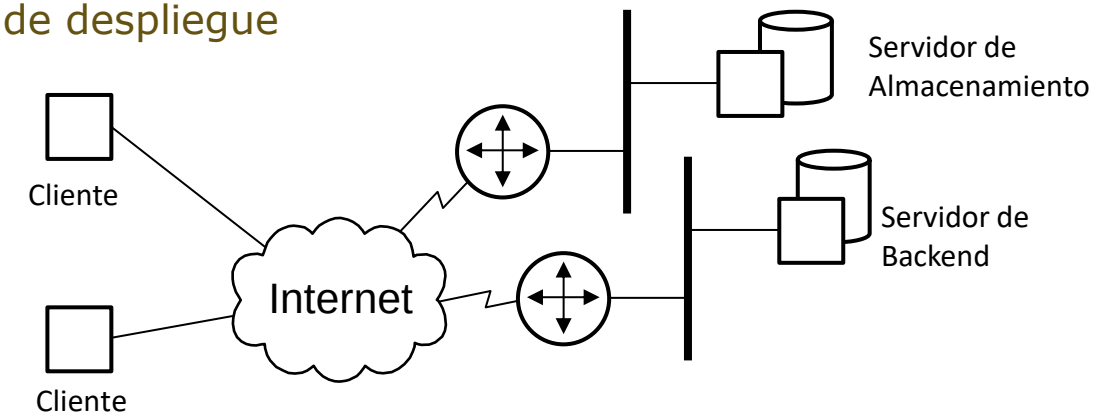
Arquitectura de N-Niveles

Posible Escenario de Despliegue para la Aplicación Web

escenario de despliegue



arquitectura física de despliegue

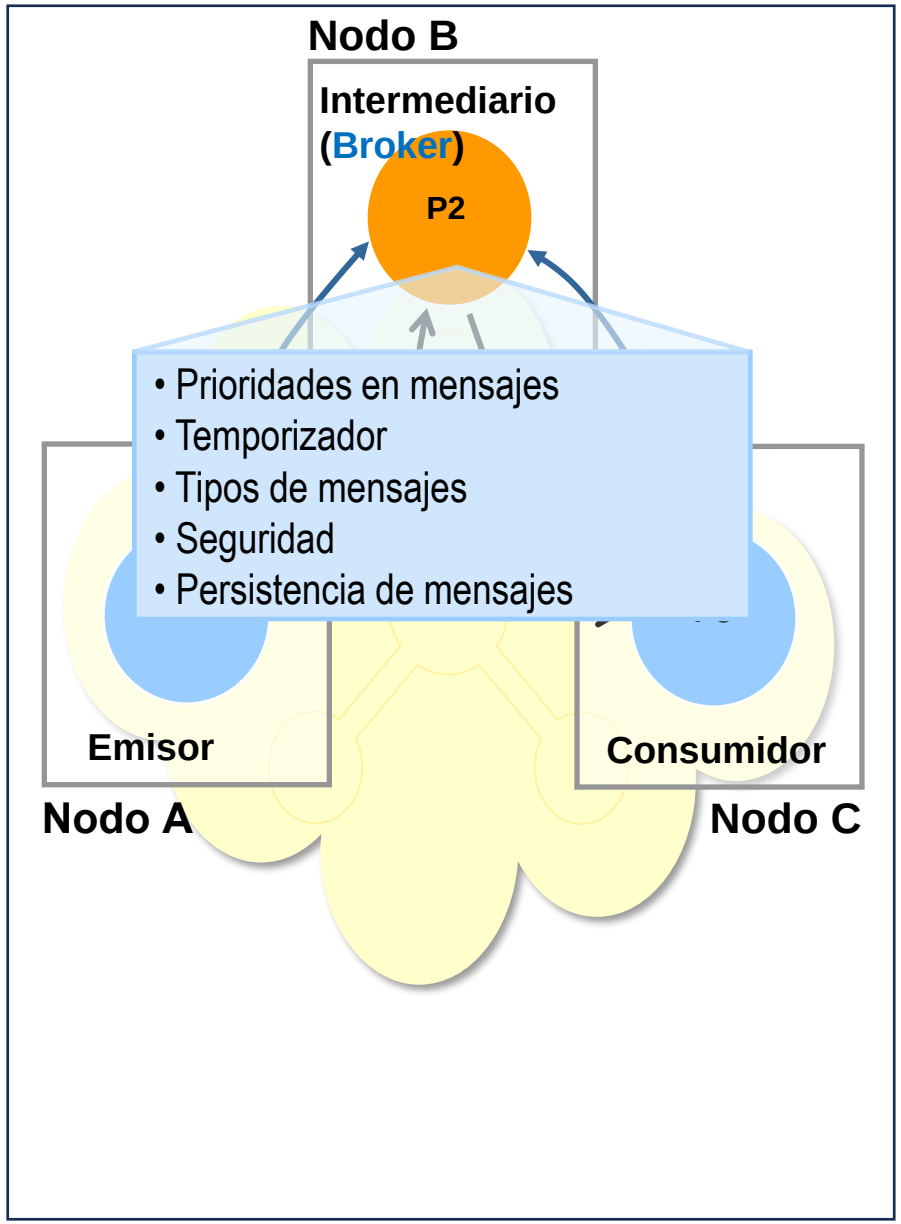


Modelos Arquitectónicos

Middleware orientado a mensajes

Middleware orientado a mensajes

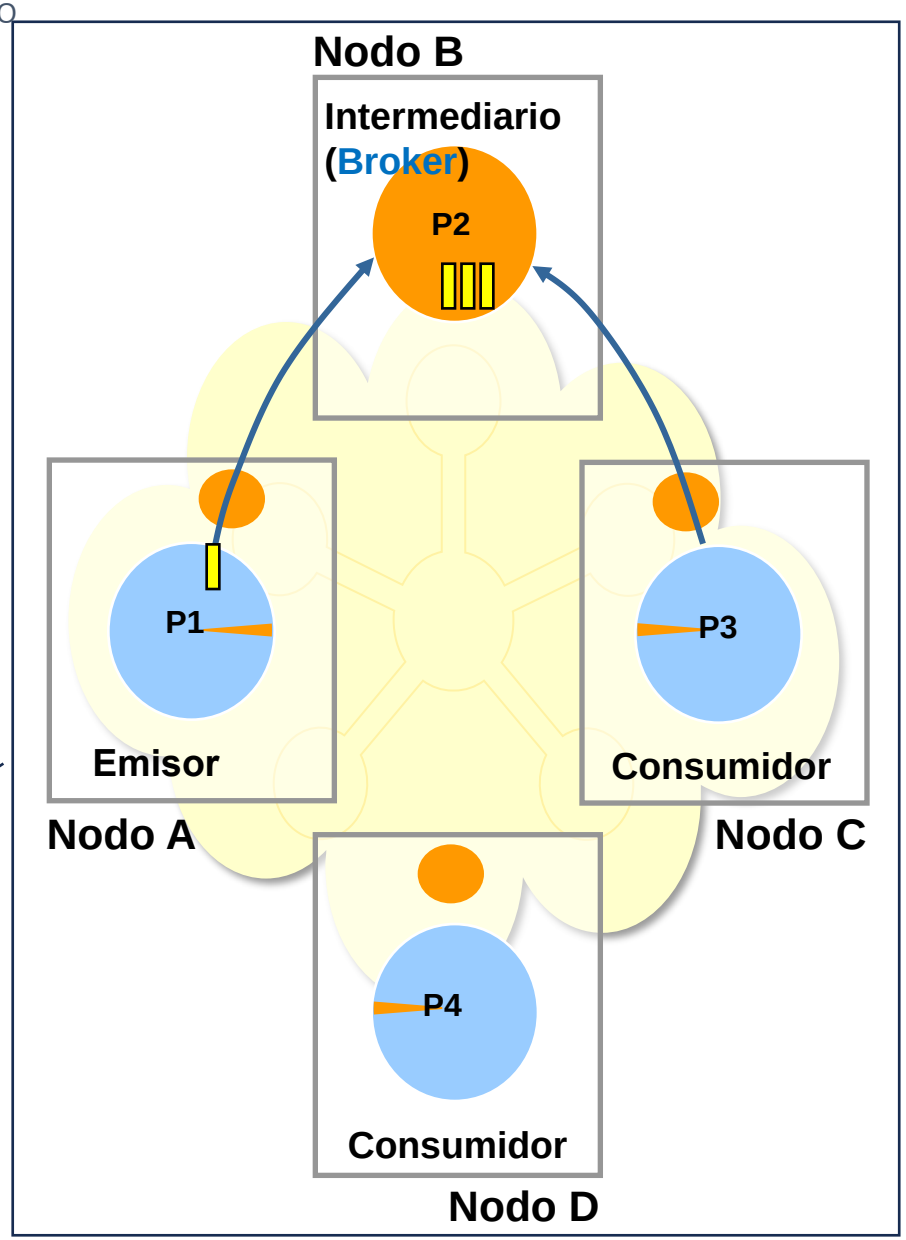
- MOM (Middleware Orientado a Mensajes)
  - Evolución del paso de mensajes
- Desacoplamiento
- Sistemas asíncronos
- Intermediario en el proceso de comunicación
- Herramientas:
  - MQTT, JMS, MSMQ
- Dos tipos:
  - Punto a punto → 1:1
  - Publicación/suscripción → 1:M



Middleware orientado a mensajes

Punto a punto

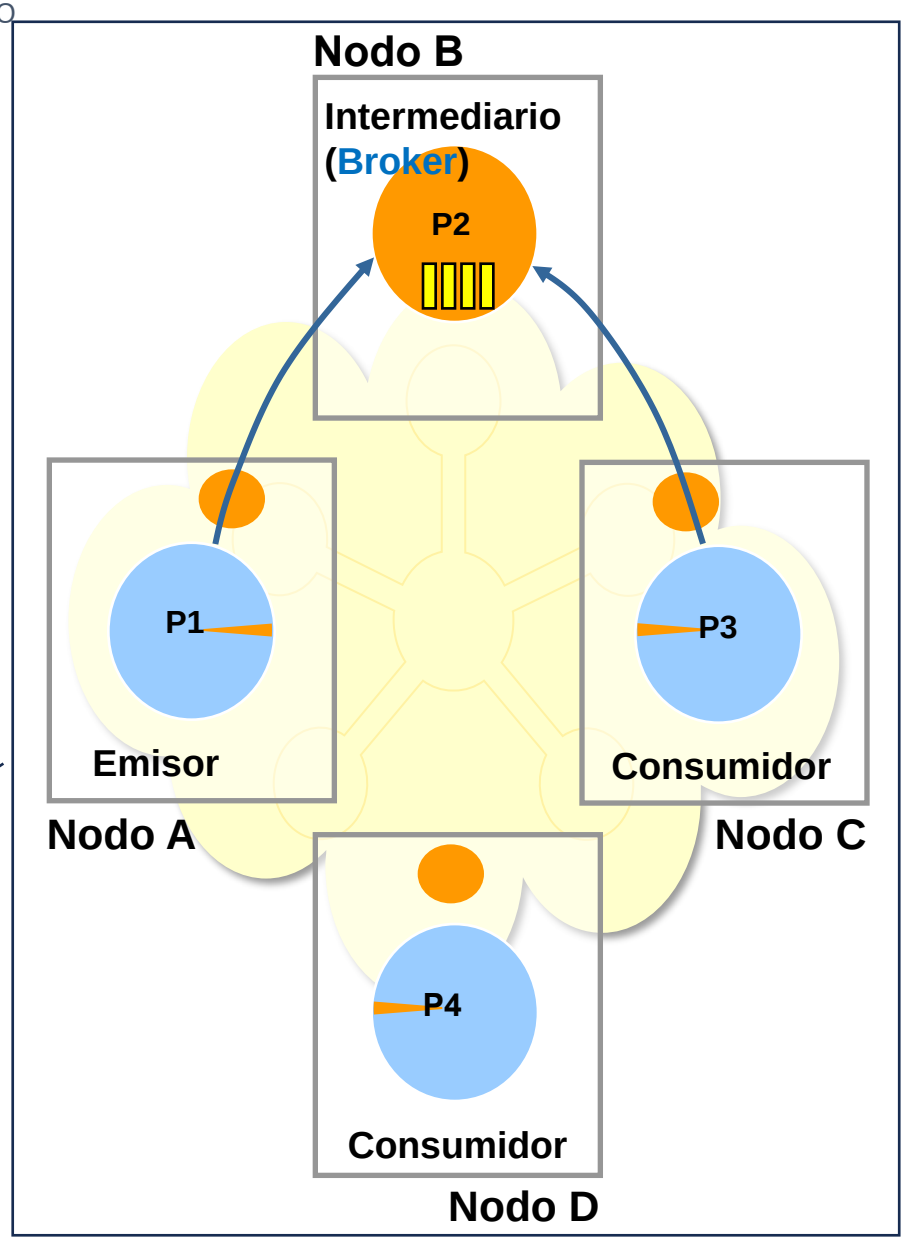
- MOM (Middleware Orientado a Mensajes)
  - Evolución del paso de mensajes
- Desacoplamiento
- Sistemas asíncronos
- Intermediario en el proceso de comunicación
- Herramientas:
  - MQTT, JMS, MSMQ
- Dos tipos:
  - Punto a punto → 1:1
  - Publicación/suscripción → 1:M



Middleware orientado a mensajes

Punto a punto

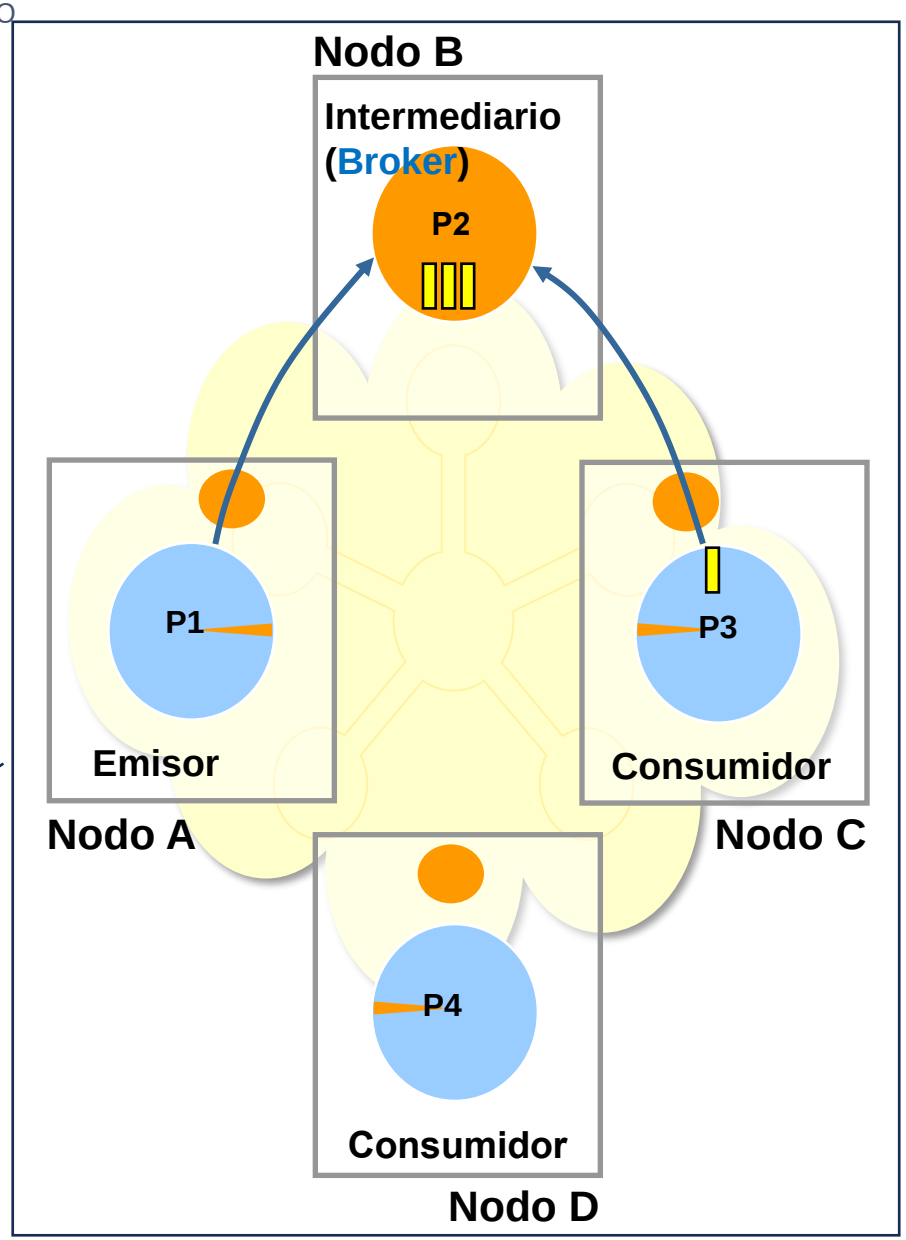
- MOM (Middleware Orientado a Mensajes)
  - Evolución del paso de mensajes
- Desacoplamiento
- Sistemas asíncronos
- Intermediario en el proceso de comunicación
- Herramientas:
  - MQTT, JMS, MSMQ
- Dos tipos:
  - Punto a punto → 1:1
  - Publicación/suscripción → 1:M



Middleware orientado a mensajes

Punto a punto

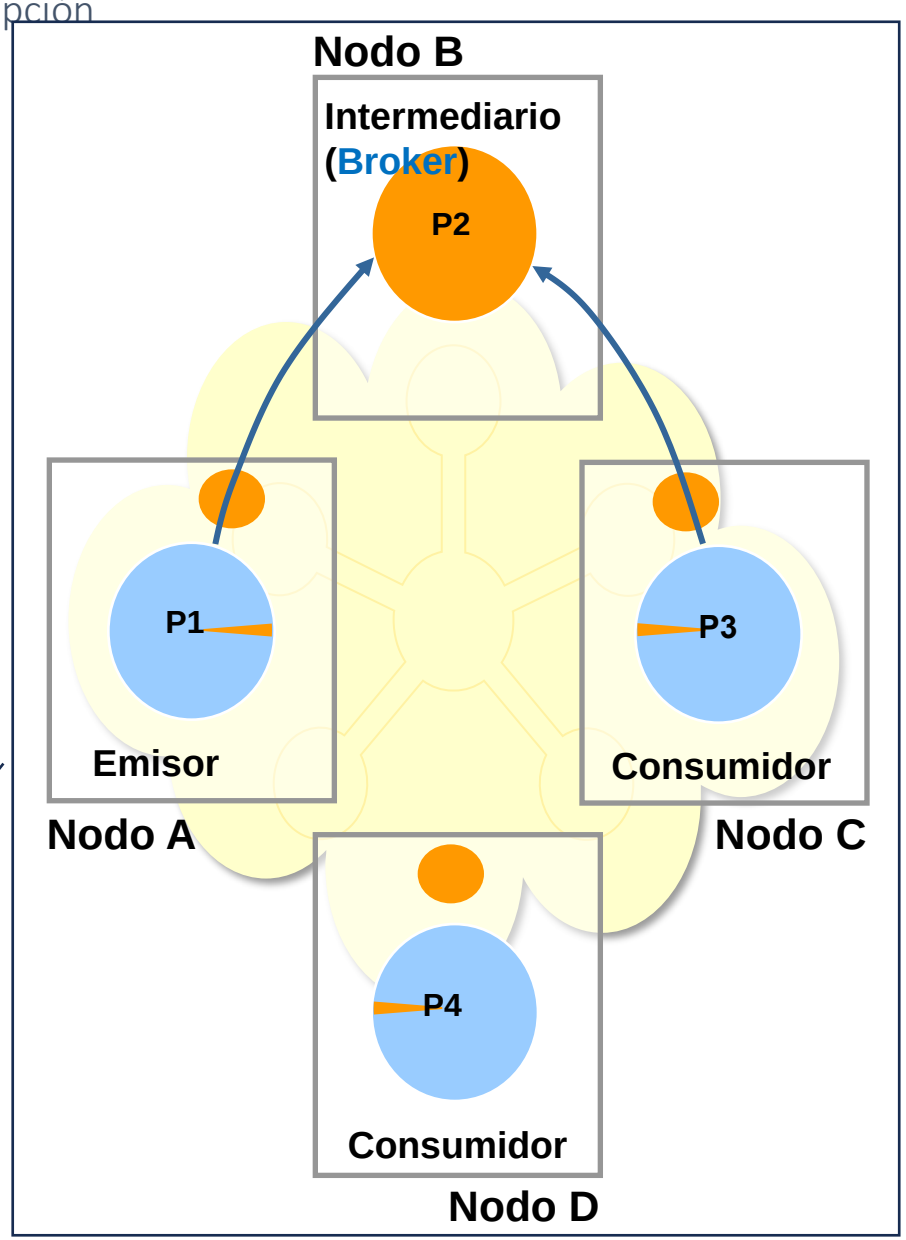
- MOM (Middleware Orientado a Mensajes)
  - Evolución del paso de mensajes
- Desacoplamiento
- Sistemas asíncronos
- Intermediario en el proceso de comunicación
- Herramientas:
  - MQTT, JMS, MSMQ
- Dos tipos:
  - Punto a punto → 1:1
  - Publicación/suscripción → 1:M



Middleware orientado a mensajes

Publicación/suscripción

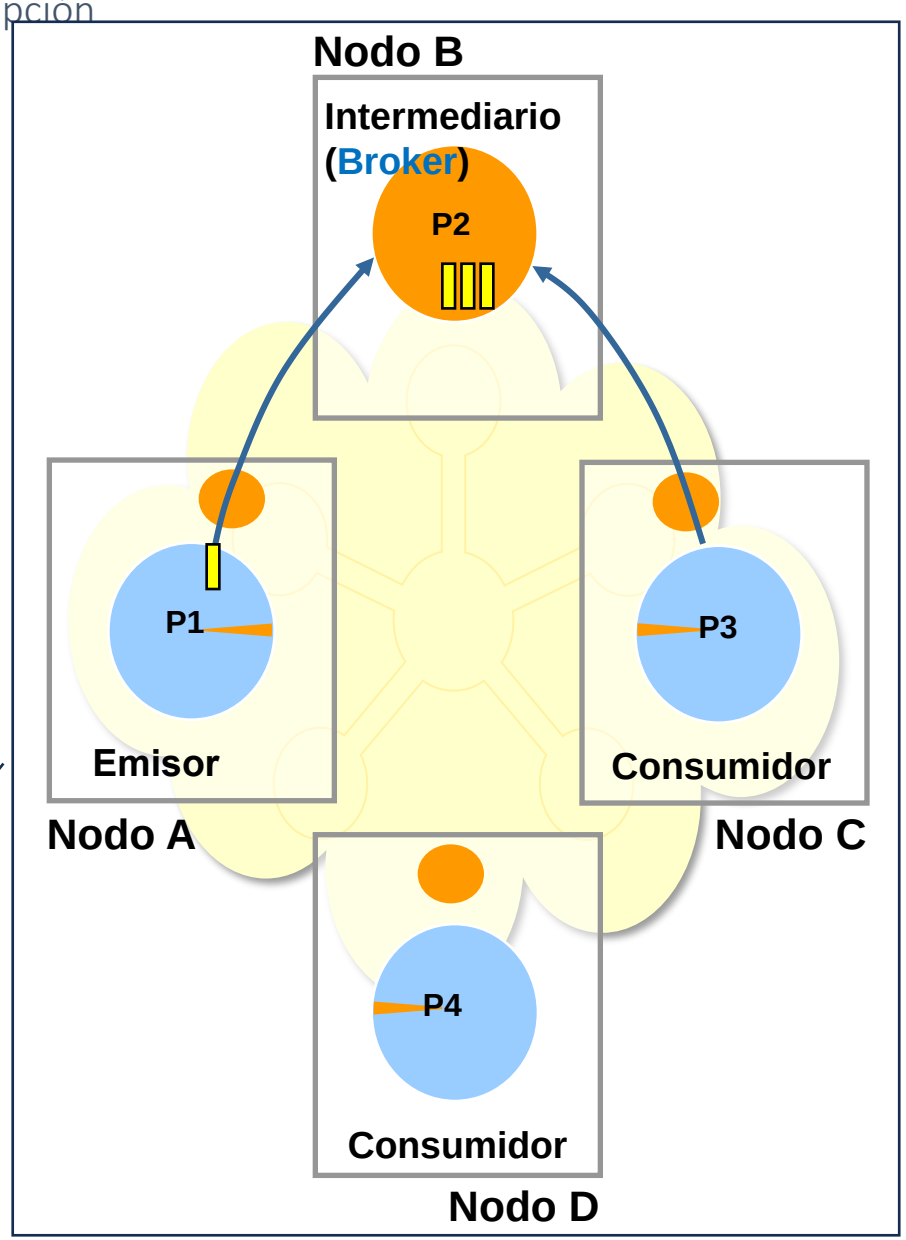
- MOM (Middleware Orientado a Mensajes)
  - Evolución del paso de mensajes
- Desacoplamiento
- Sistemas asíncronos
- Intermediario en el proceso de comunicación
- Herramientas:
  - MQTT, JMS, MSMQ
- Dos tipos:
  - Punto a punto → 1:1
  - Publicación/suscripción → 1:M



Middleware orientado a mensajes

Publicación/suscripción

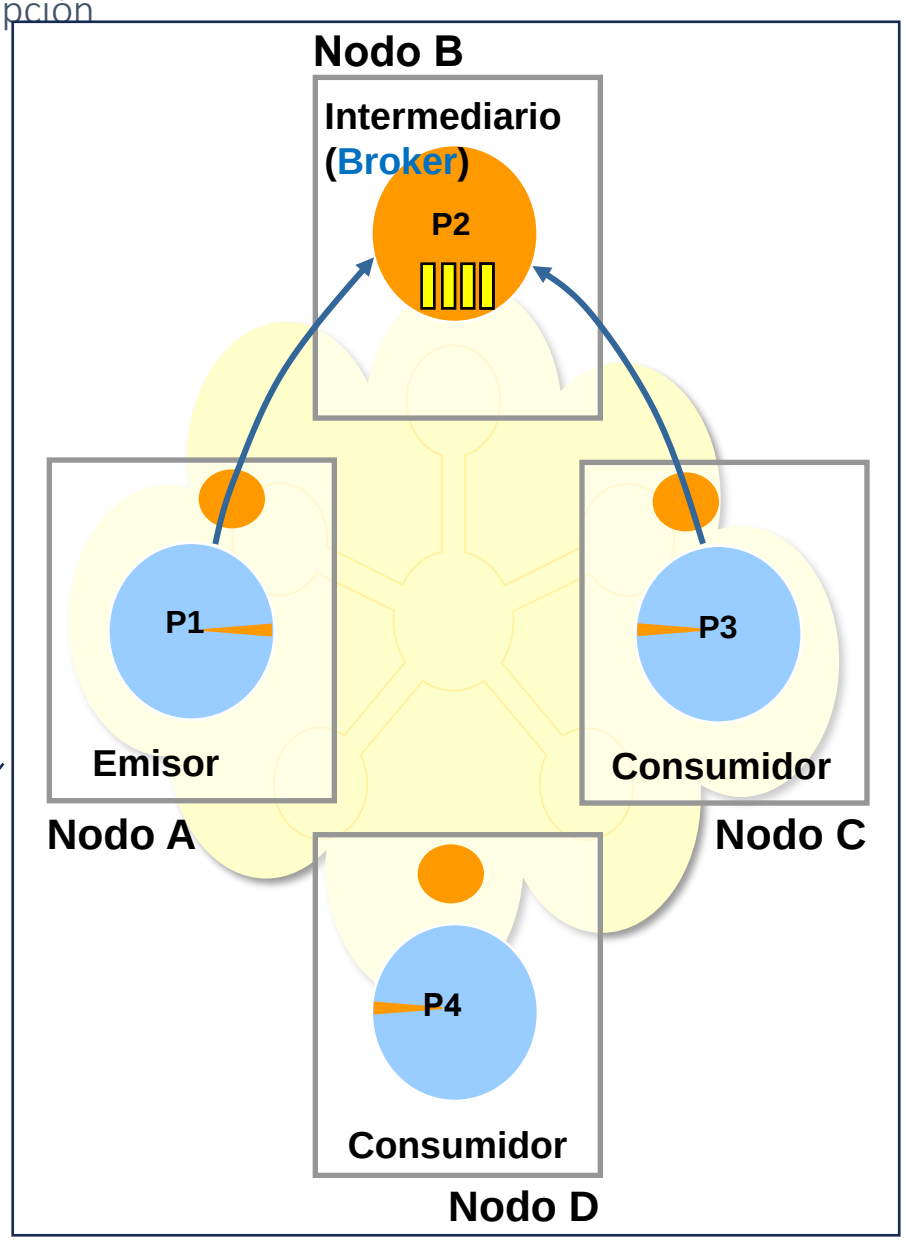
- MOM (Middleware Orientado a Mensajes)
  - Evolución del paso de mensajes
- Desacoplamiento
- Sistemas asíncronos
- Intermediario en el proceso de comunicación
- Herramientas:
  - MQTT, JMS, MSMQ
- Dos tipos:
  - Punto a punto → 1:1
  - Publicación/suscripción → 1:M



Middleware orientado a mensajes

Publicación/suscripción

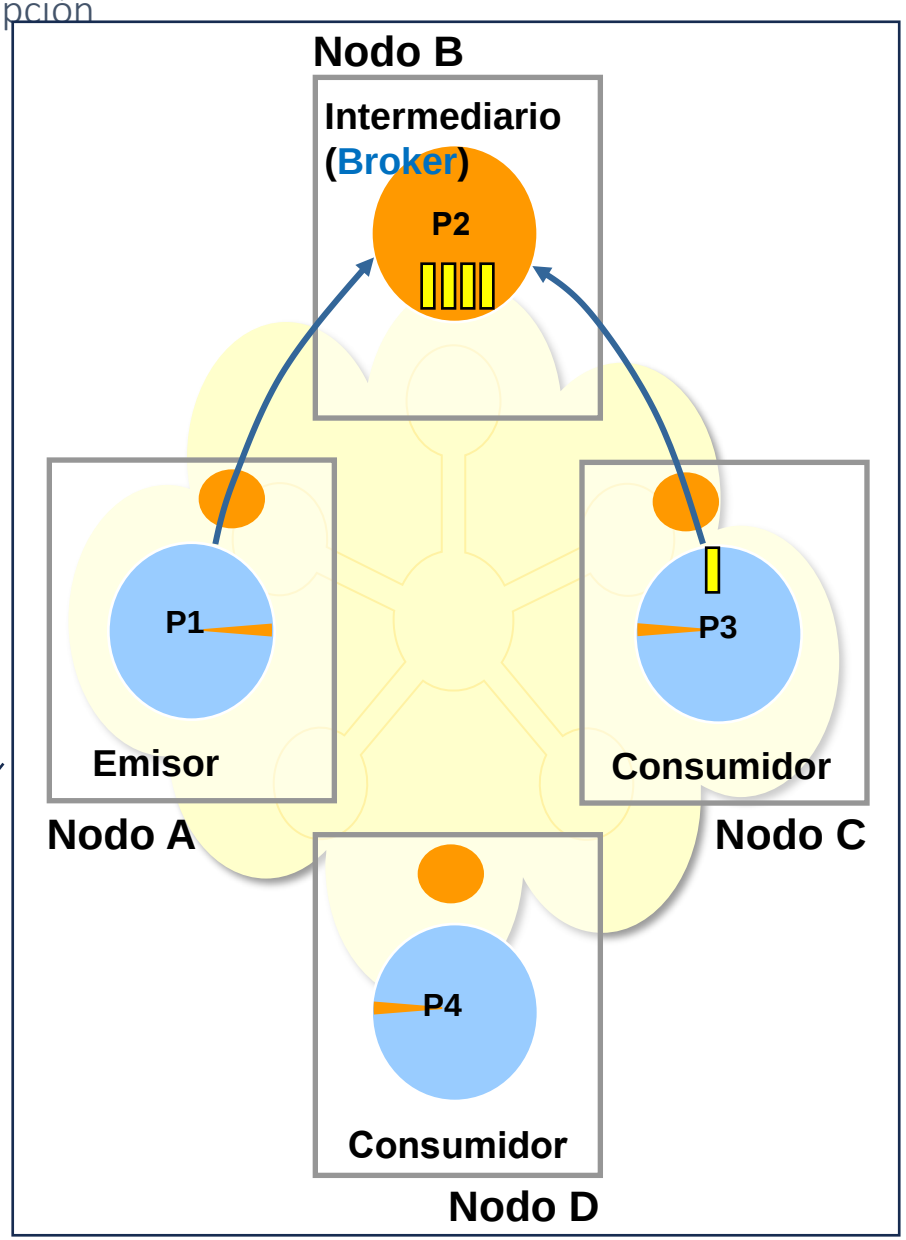
- MOM (Middleware Orientado a Mensajes)
  - Evolución del paso de mensajes
- Desacoplamiento
- Sistemas asíncronos
- Intermediario en el proceso de comunicación
- Herramientas:
  - MQTT, JMS, MSMQ
- Dos tipos:
  - Punto a punto → 1:1
  - Publicación/suscripción → 1:M



Middleware orientado a mensajes

Publicación/suscripción

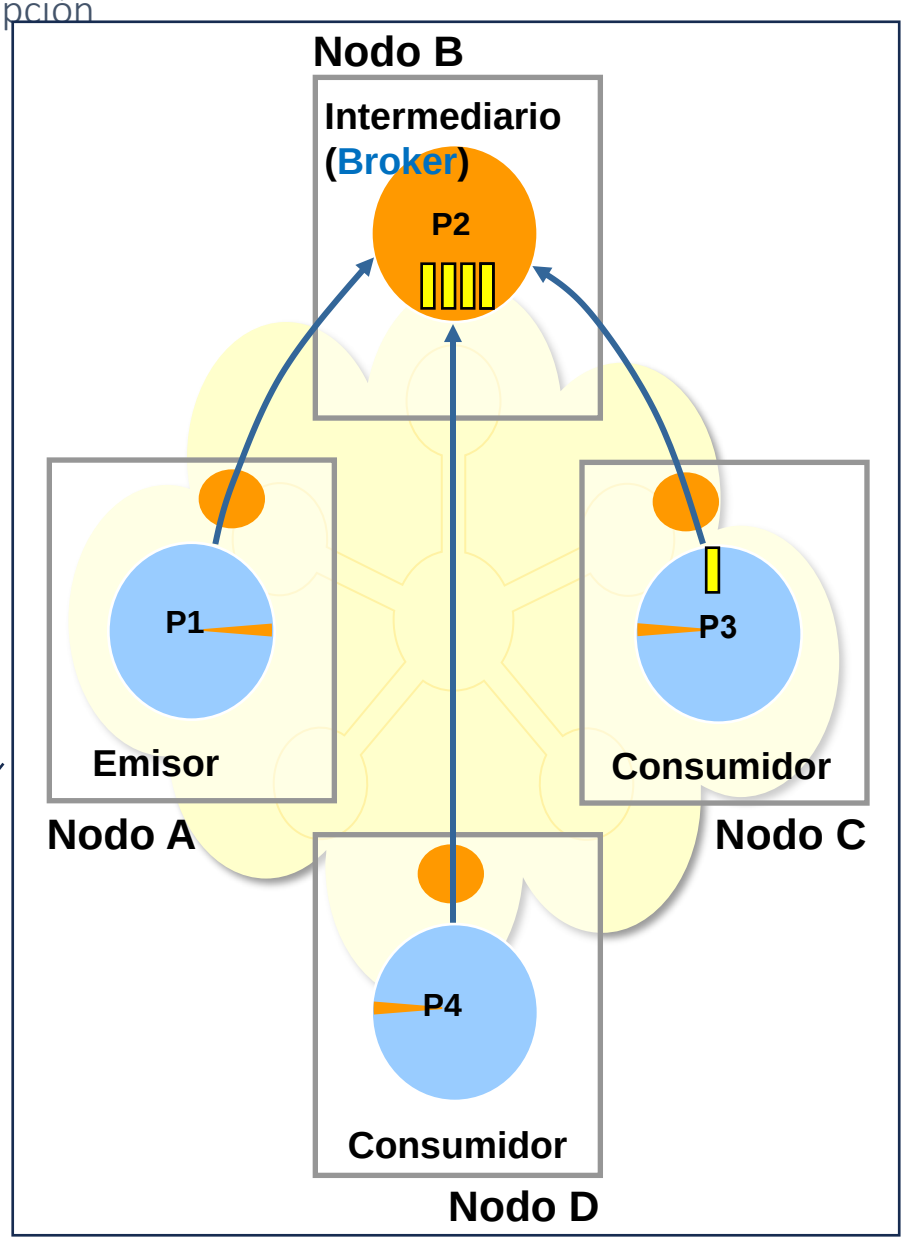
- MOM (Middleware Orientado a Mensajes)
  - Evolución del paso de mensajes
- Desacoplamiento
- Sistemas asíncronos
- Intermediario en el proceso de comunicación
- Herramientas:
  - MQTT, JMS, MSMQ
- Dos tipos:
  - Punto a punto → 1:1
  - Publicación/suscripción → 1:M



Middleware orientado a mensajes

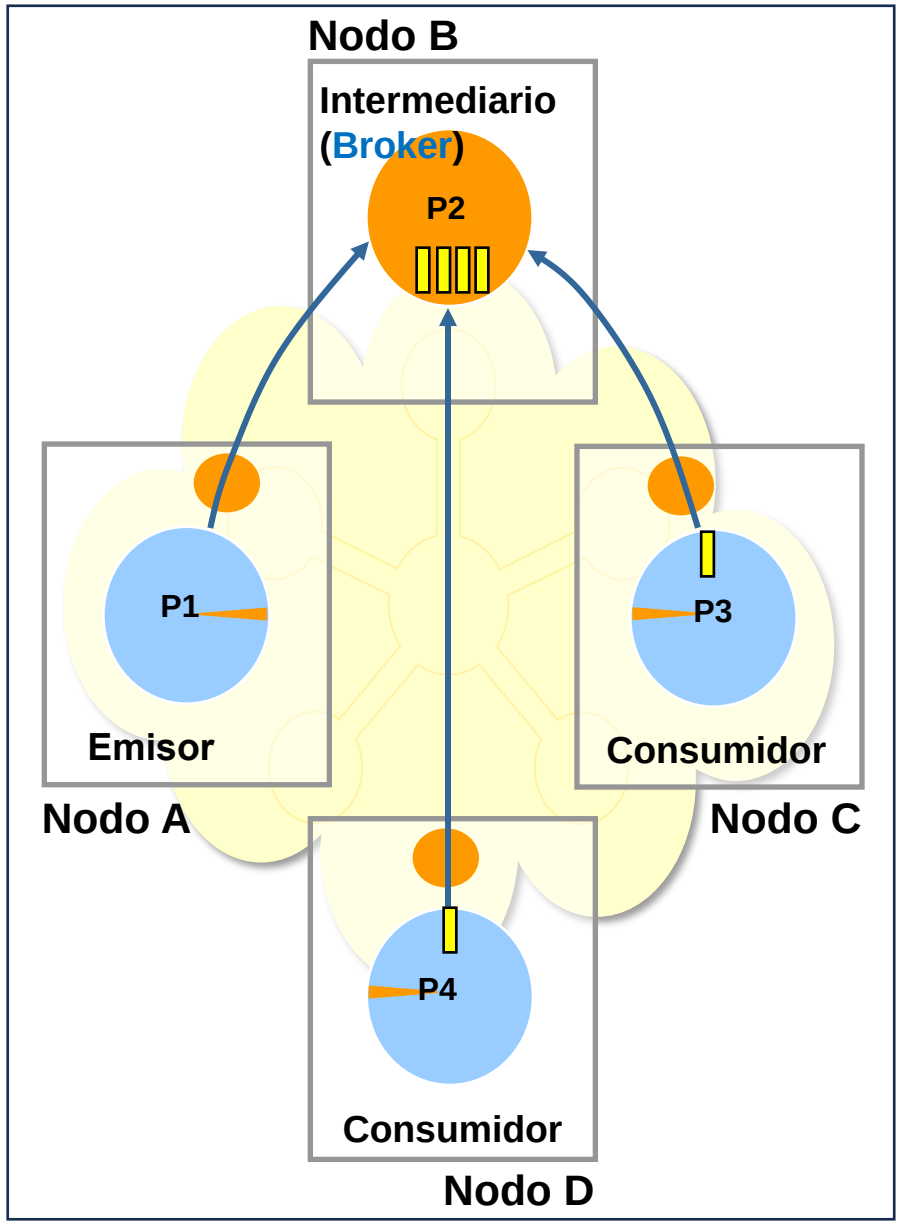
Publicación/suscripción

- MOM (Middleware Orientado a Mensajes)
  - Evolución del paso de mensajes
- Desacoplamiento
- Sistemas asíncronos
- Intermediario en el proceso de comunicación
- Herramientas:
  - MQTT, JMS, MSMQ
- Dos tipos:
  - Punto a punto → 1:1
  - Publicación/suscripción → 1:M



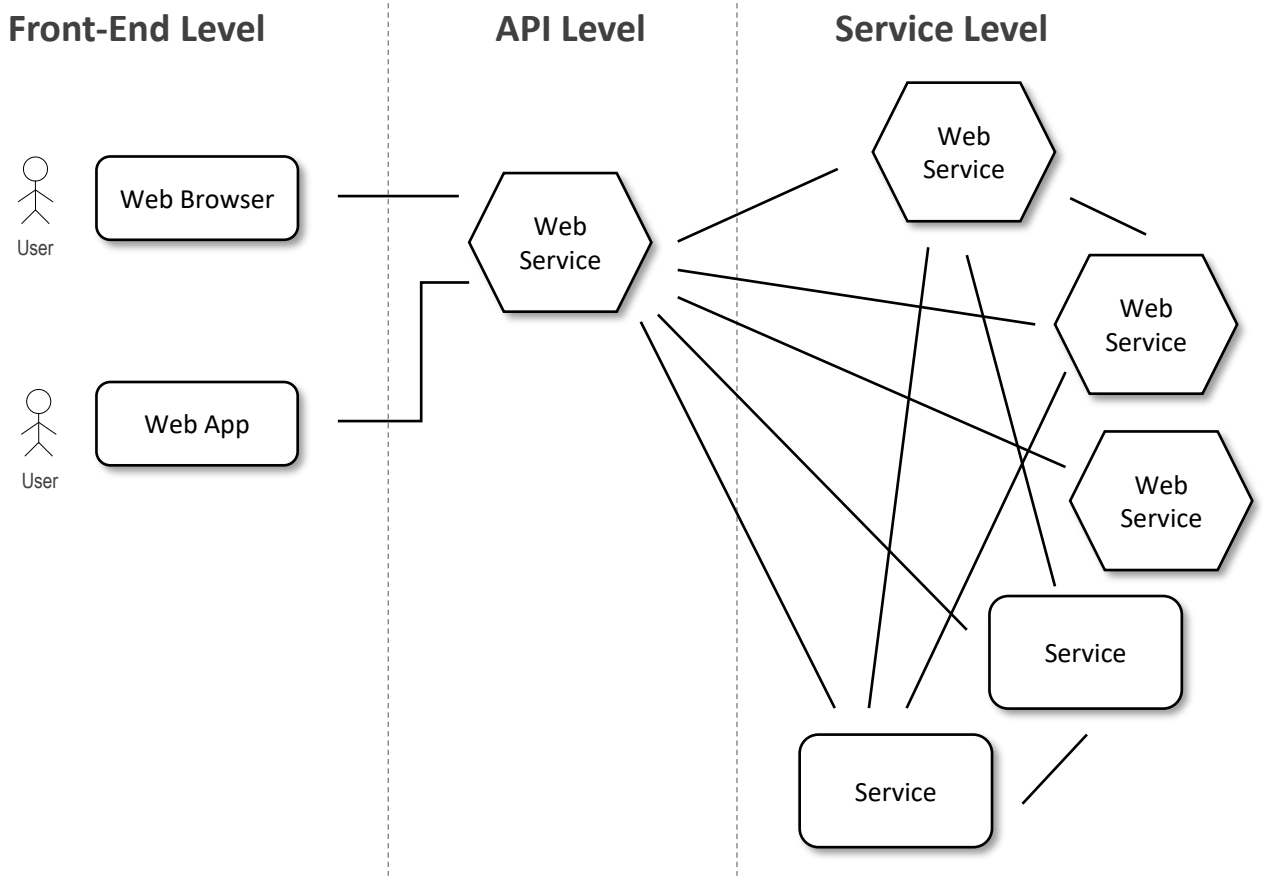
Middleware orientado a mensajes

- MOM (Middleware Orientado a Mensajes)
  - Evolución del paso de mensajes
- Desacoplamiento
- Sistemas asíncronos
- Intermediario en el proceso de comunicación
- Herramientas:
  - MQTT, JMS, MSMQ
- Dos tipos:
  - Punto a punto → 1:1
  - Publicación/suscripción → 1:M



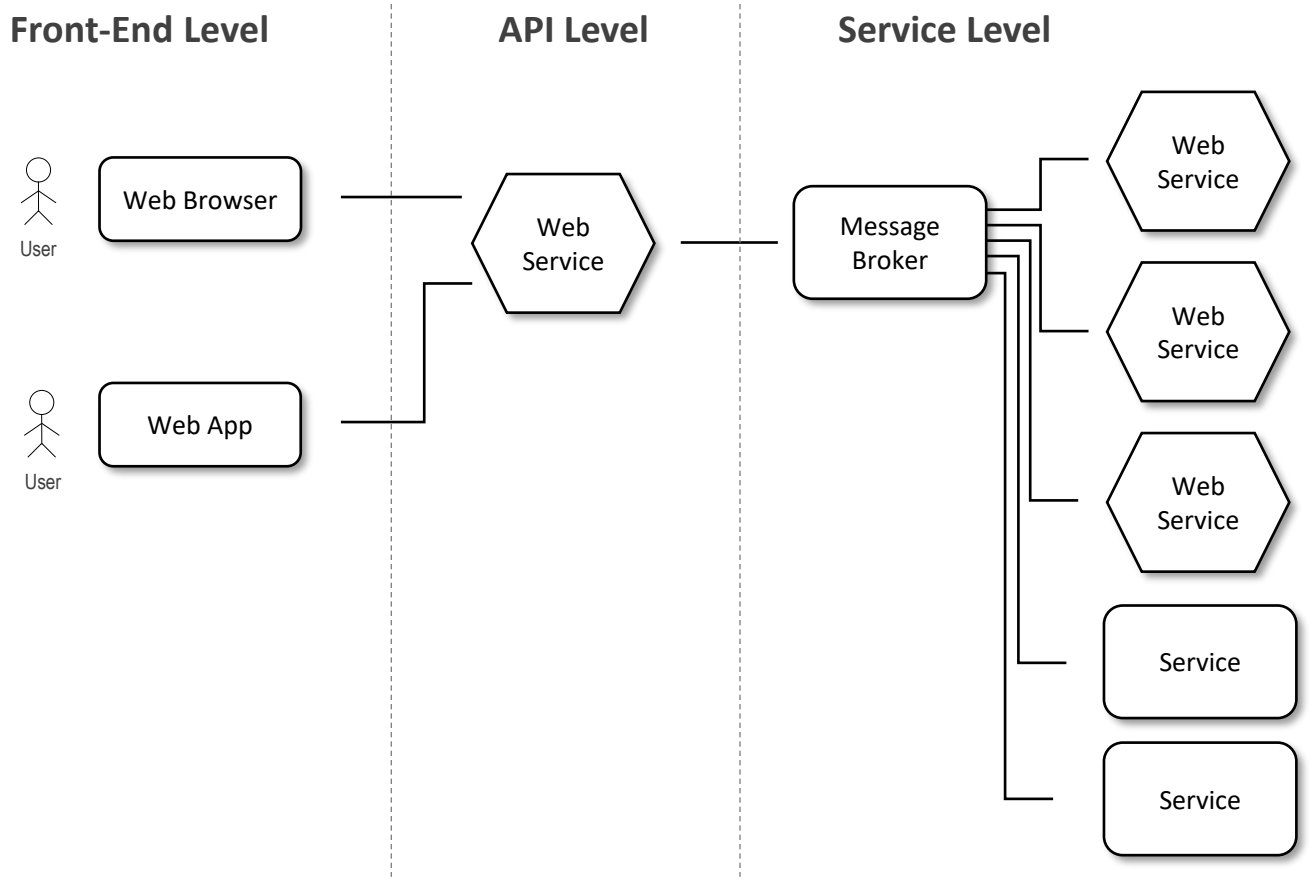
Middleware orientado a mensajes

Arquitectura distribuida



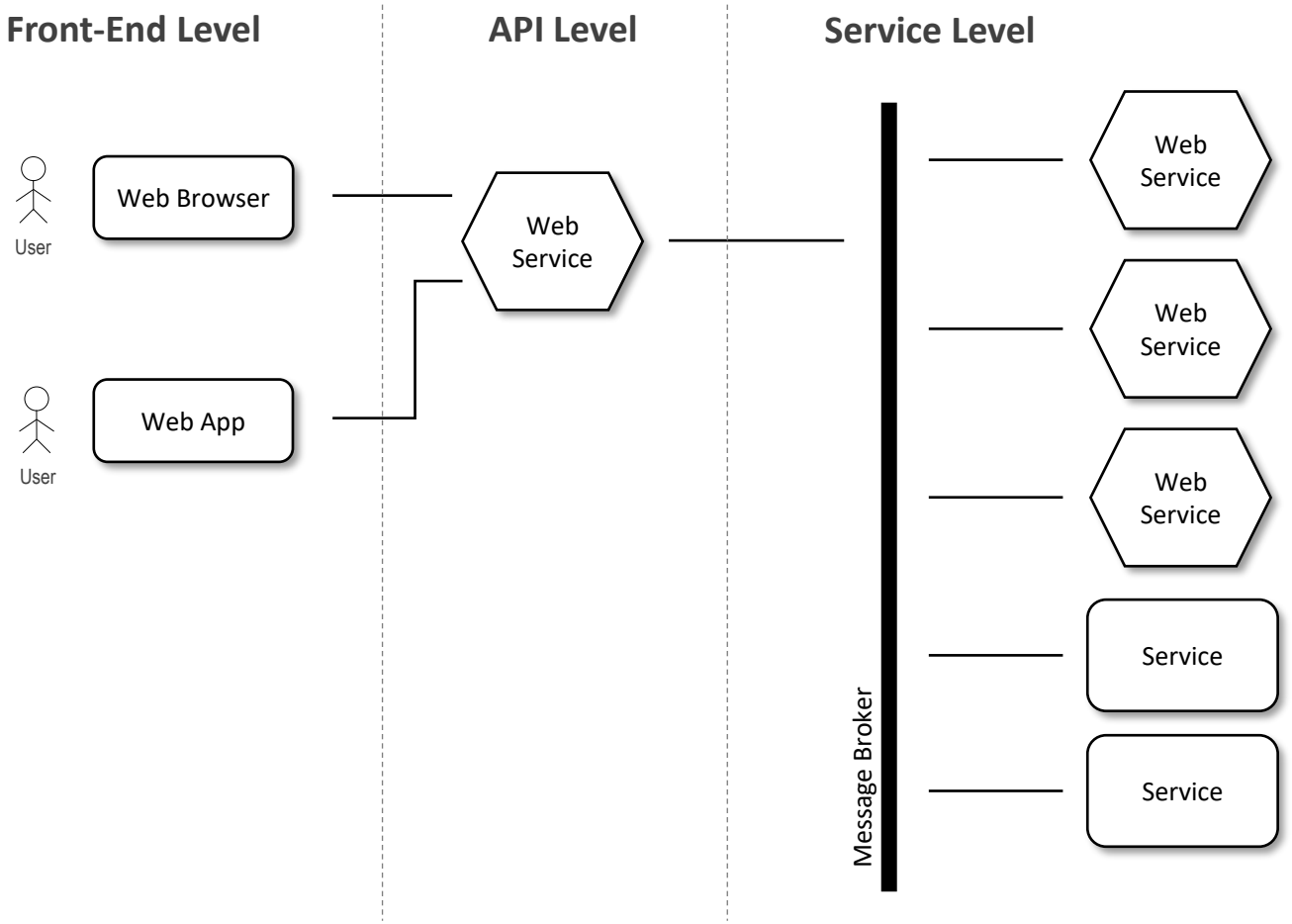
Middleware orientado a mensajes

Arquitectura distribuida



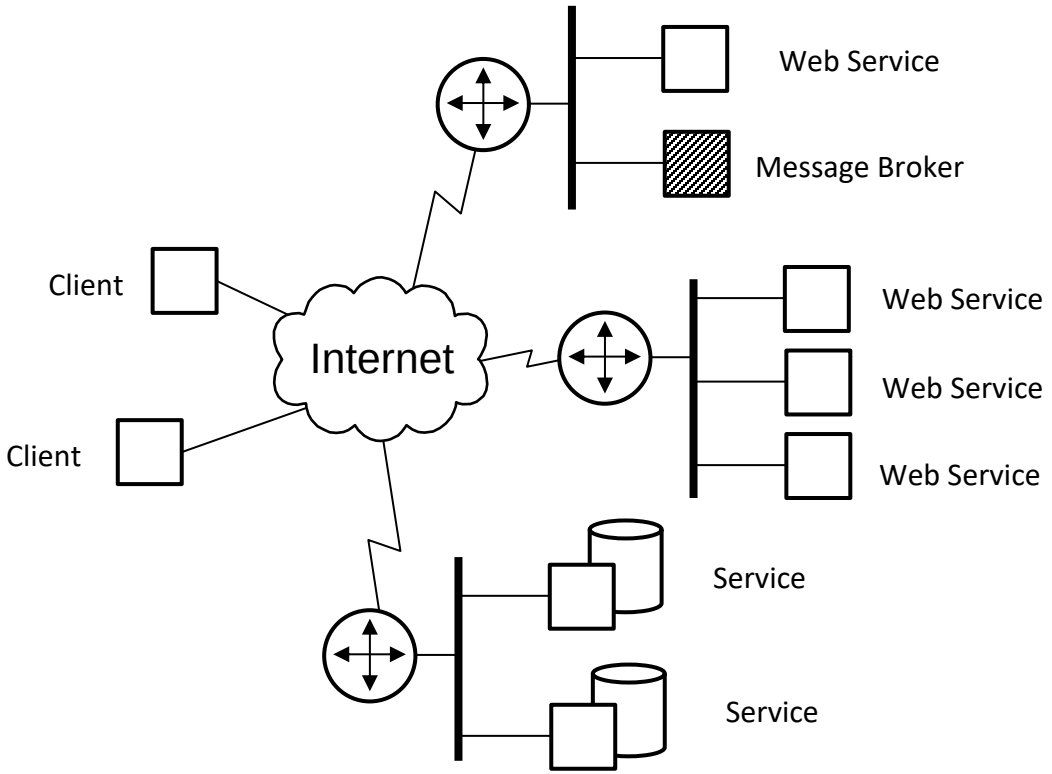
Middleware orientado a mensajes

Arquitectura distribuida



Middleware orientado a mensajes

Arquitectura física



Arquitectura orientada a servicios

Arquitectura orientada a servicios

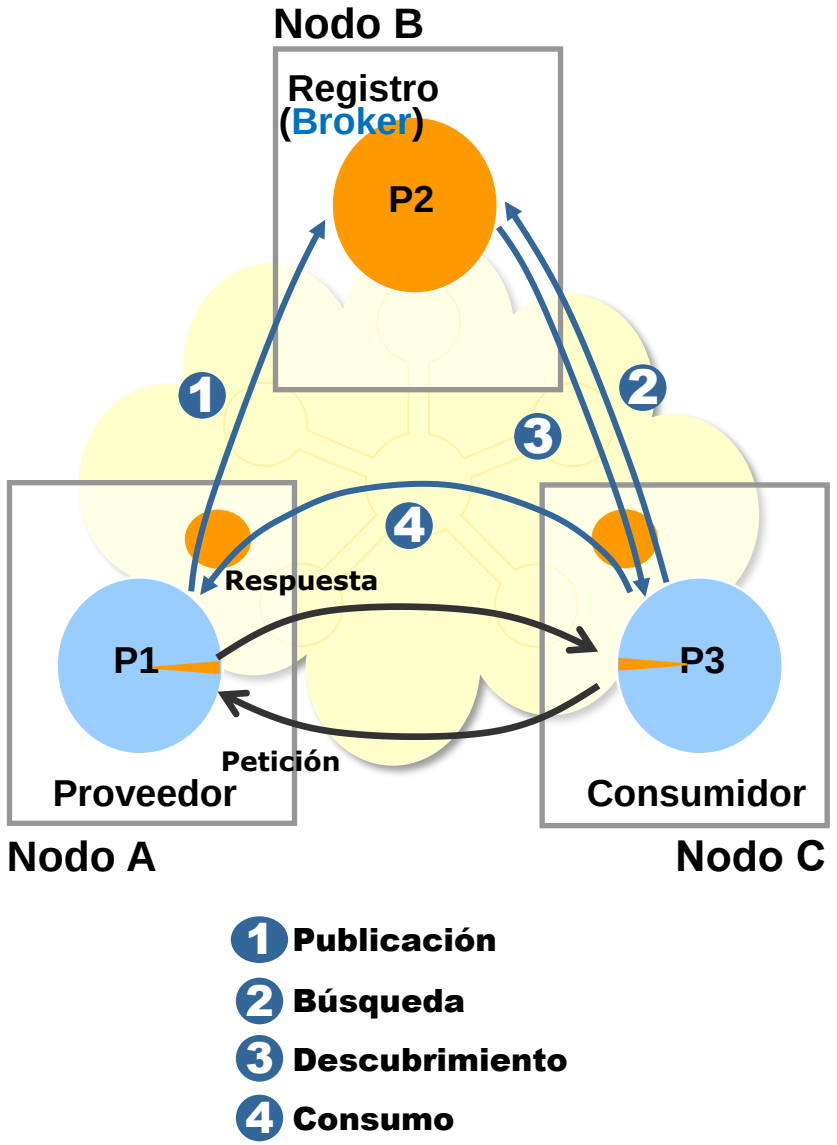
Introducción

- **SOA** organiza sistemas distribuidos en torno a servicios independientes que se comunican a través de protocolos estándar (HTTP, SOAP, REST).
- Componentes Clave:
  - Servicios Web: Realizan funciones de negocio mediante interfaces.
  - **Descubrimiento** de Servicios: a través de brokers de registro.
  - Contratos de Servicios: Definen cómo los servicios interactúan.

Arquitectura orientada a servicios

Introducción

- **Servicios** que **exponen** el **acceso** a **recursos** a través de la **red** (aplicaciones, archivos, bases de datos, servidores, sistemas de información, dispositivos)
- **Componentes**
  - **Proveedor** de servicios
  - **Consumidor** de servicios
  - **Servicio de registro**
- Transparencia de localización
- Características
  - Mayor grado de desacoplamiento
  - Hacia una gestión semántica
- Herramientas:
  - SOAP, WSDL, UDDI



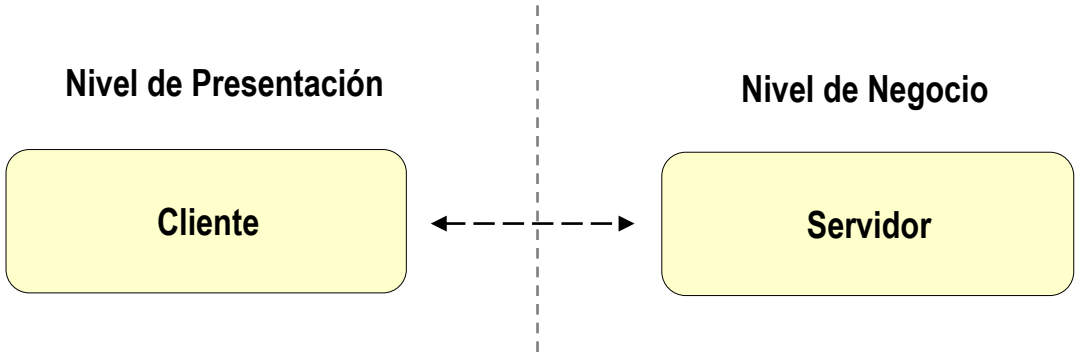
Arquitectura orientada a servicios

Ventajas y Desventaja

- Ventajas:
  - **Escalabilidad Horizontal**: Servicios distribuidos en varios nodos.
  - **Tolerancia a Fallos**: Resiliencia mediante replicación.
  - **Transparencia** e **Interoperabilidad**: Facilita la comunicación entre plataformas heterogéneas.
- Desventajas:
  - **Sobrecarga** Operacional: Gestión compleja de comunicación y contratos.
  - Rendimiento: **Latencia** en la comunicación entre servicios.
- Ejemplo:
  - Comercio electrónico: servicios independientes para gestión de pedidos, inventario, y facturación.

Arquitectura orientada a servicios

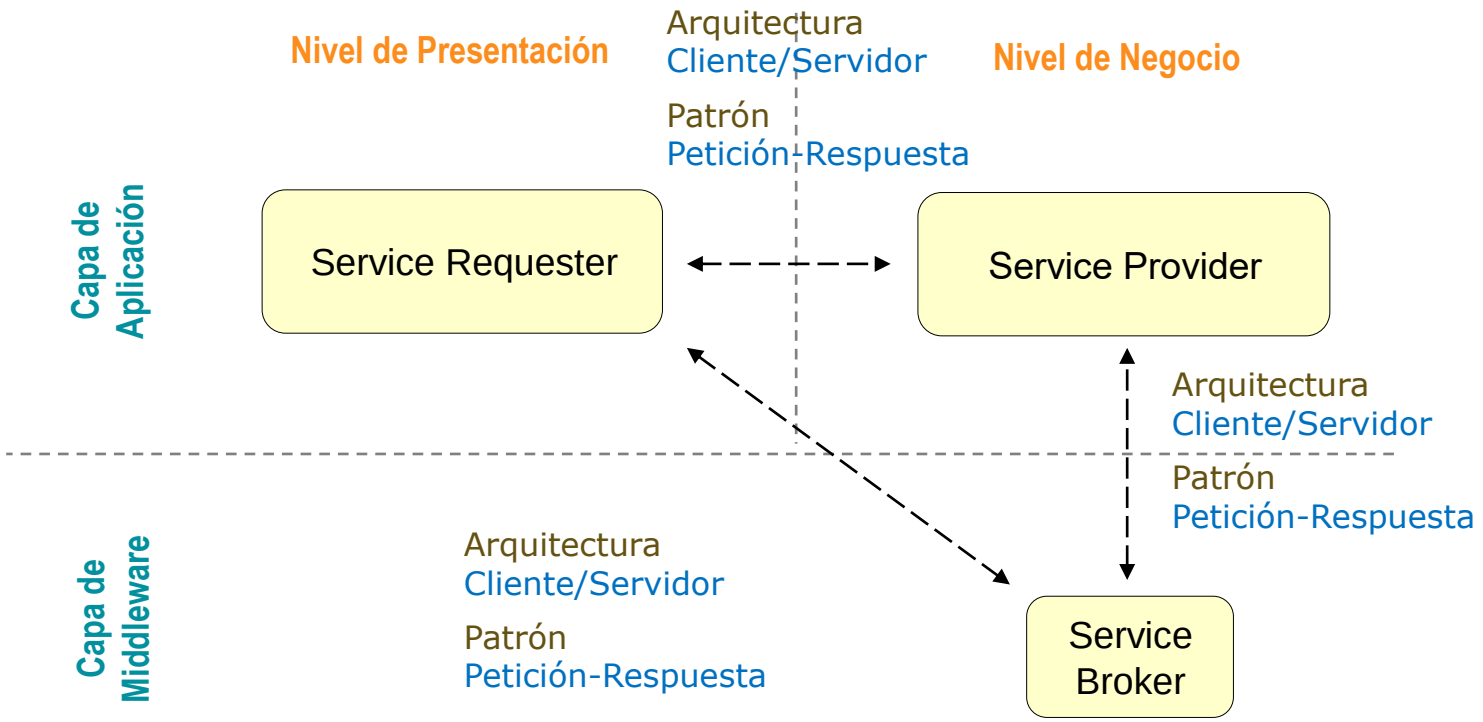
Arquitectura SOA por niveles



Arquitectura Cliente/Servidor

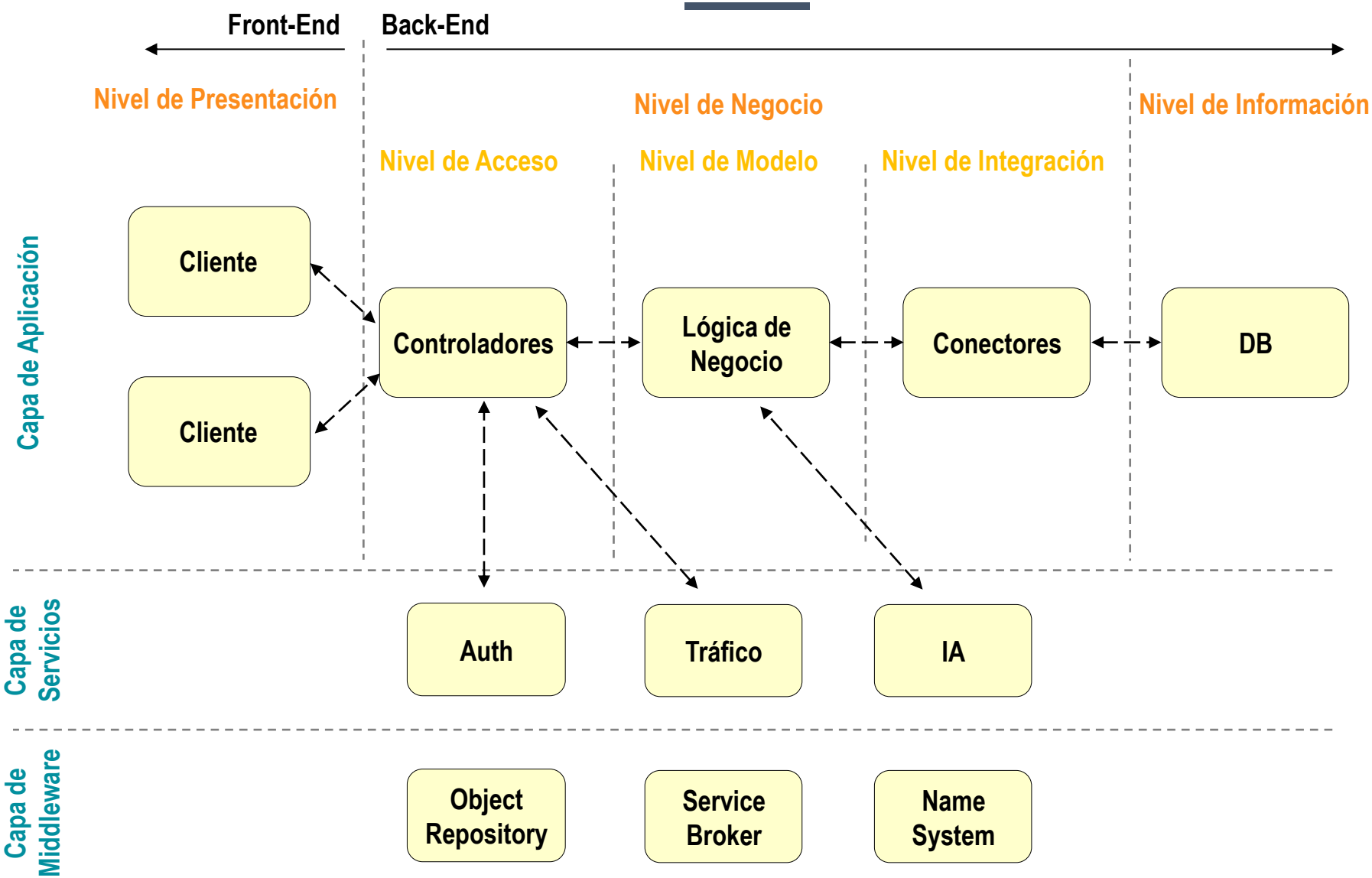
Arquitectura orientada a servicios

Arquitectura SOA por niveles



Arquitectura orientada a servicios

Arquitectura SOA por niveles



Arquitectura de Microservicios

Arquitectura de Microservicios

Introducción

- La arquitectura de microservicios es un enfoque de diseño de software en el que una aplicación se compone de pequeños servicios independientes que se comunican entre sí.
- Cada microservicio es una unidad funcional completa y autónoma, que realiza una tarea específica dentro del sistema.
- Estos servicios son desplegados y gestionados de forma independiente, lo que facilita la escalabilidad, la flexibilidad y la velocidad en el desarrollo y despliegue de aplicaciones.

Arquitectura de Microservicios

Ventajas

- Agilidad y Rapidez en el Desarrollo:
  - Los equipos pueden trabajar en paralelo en diferentes microservicios, acelerando el desarrollo y despliegue de nuevas funcionalidades.
- Escalabilidad:
  - Permite escalar solo los microservicios que lo requieren, optimizando el uso de recursos y reduciendo costos.
- Facilidad de Mantenimiento:
  - La modularidad del sistema facilita el mantenimiento y la actualización de los microservicios de manera independiente.
- Flexibilidad Tecnológica:
  - Los equipos pueden elegir las tecnologías más adecuadas para cada microservicio, sin estar atados a una única tecnología para toda la aplicación.

Arquitectura de Microservicios

Desafíos

- Complejidad Operacional:
  - La gestión de múltiples microservicios puede incrementar la complejidad operativa, especialmente en términos de despliegue, monitoreo y mantenimiento.
- Gestión de Datos:
  - La descentralización de datos puede presentar desafíos en cuanto a la consistencia y coordinación entre diferentes microservicios.
- Comunicación y Latencia:
  - La comunicación entre microservicios puede añadir latencia y requerir una gestión robusta de la comunicación y posibles fallos.
- Seguridad:
  - Asegurar la comunicación y la autenticación entre microservicios añade un nivel adicional de complejidad.

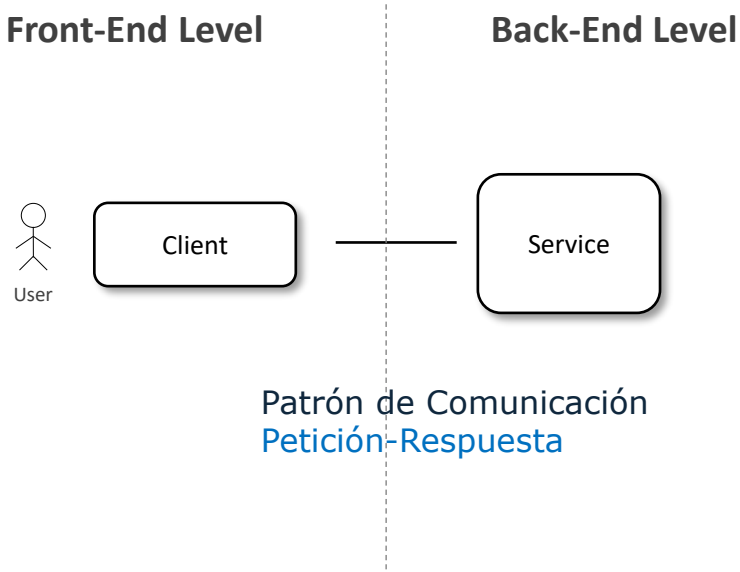
Arquitectura de Microservicios

Ejemplos

- Netflix:
  - Netflix utiliza una arquitectura de microservicios para gestionar sus servicios de streaming, recomendaciones, gestión de usuarios y más, permitiendo una escalabilidad masiva y alta disponibilidad.
- Amazon:
  - Amazon adopta microservicios para manejar diferentes funcionalidades de su plataforma de comercio electrónico, como inventario, pagos, envíos, y servicios de recomendaciones.
- Uber:
  - Uber utiliza microservicios para manejar la funcionalidad de su plataforma de transporte, incluyendo gestión de conductores, mapas, tarifas y comunicación en tiempo real.

Arquitectura de Microservicios

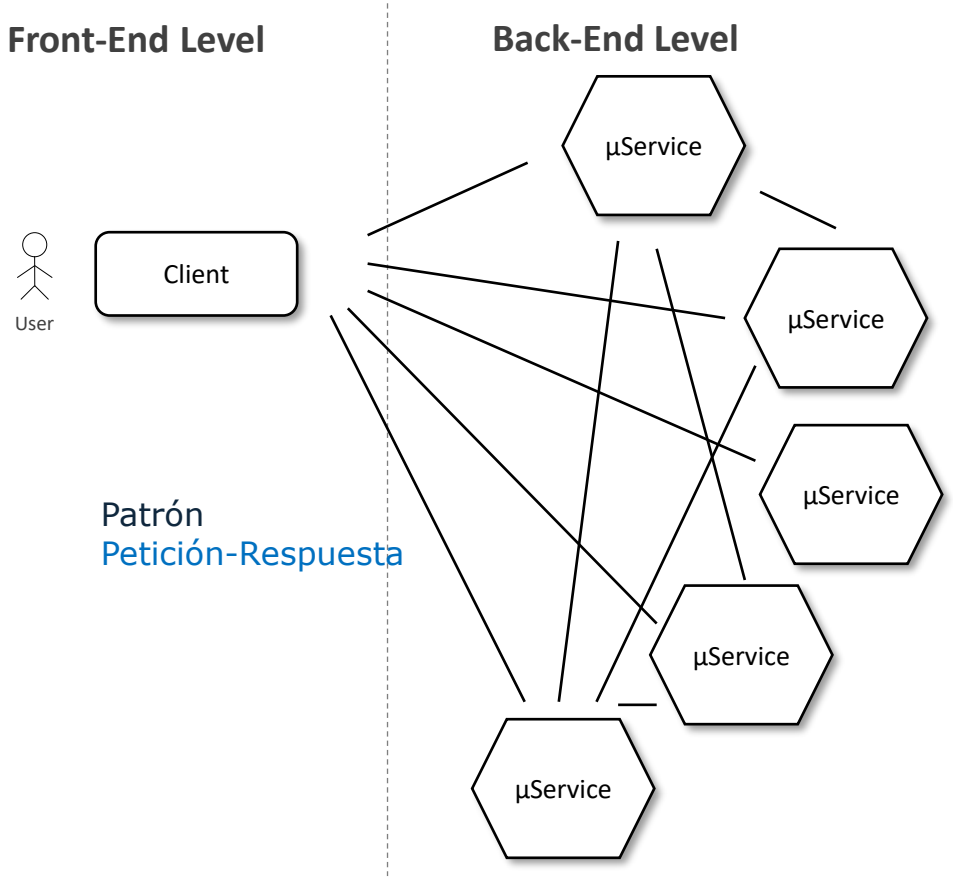
Arquitectura de Microservicios



Patrón de Comunicación
Petición-Respuesta

Arquitectura de Microservicios

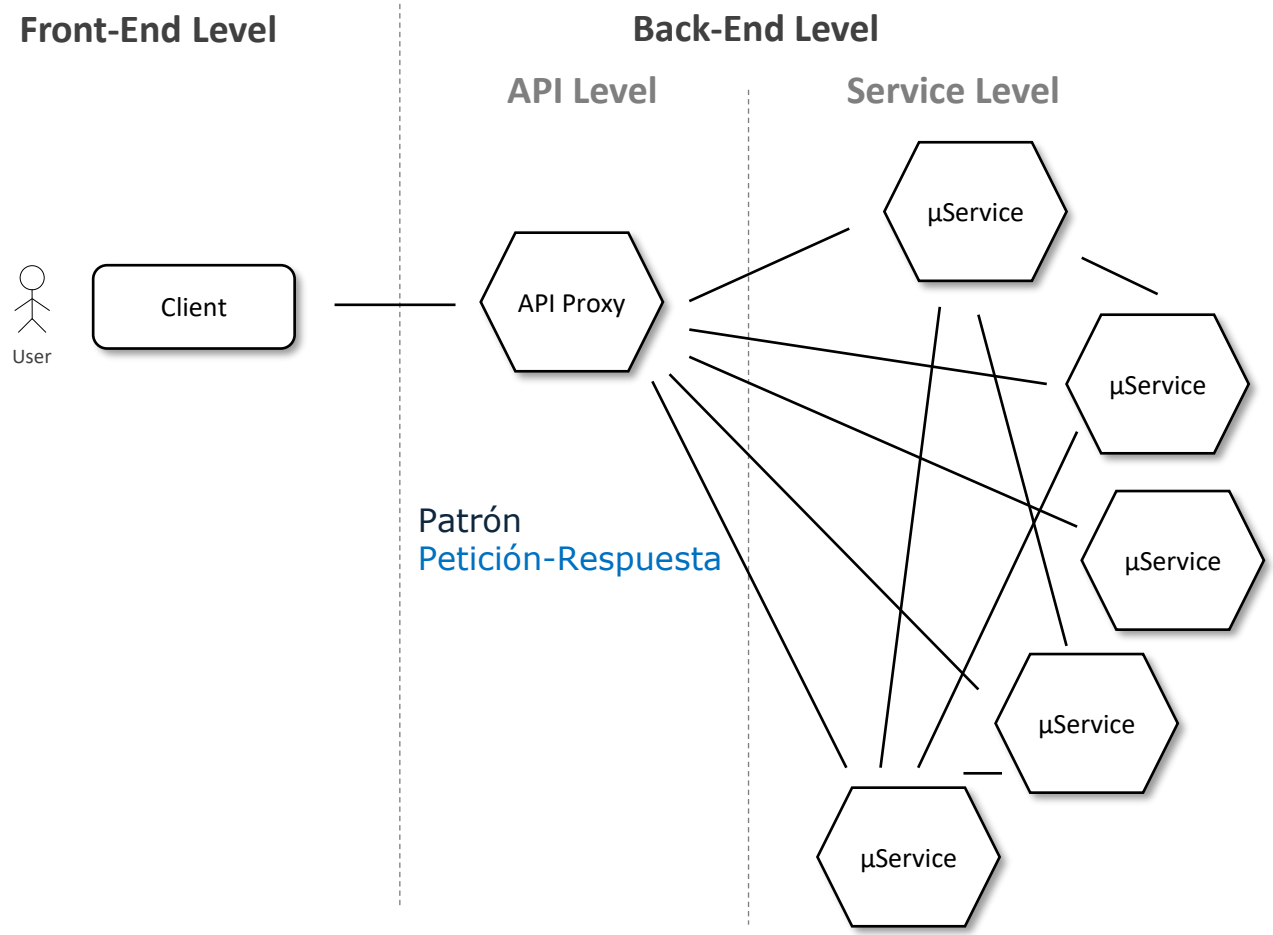
Arquitectura Distribuida basada en Patrón Petición-Respuestas



Arquitectura de Microservicios

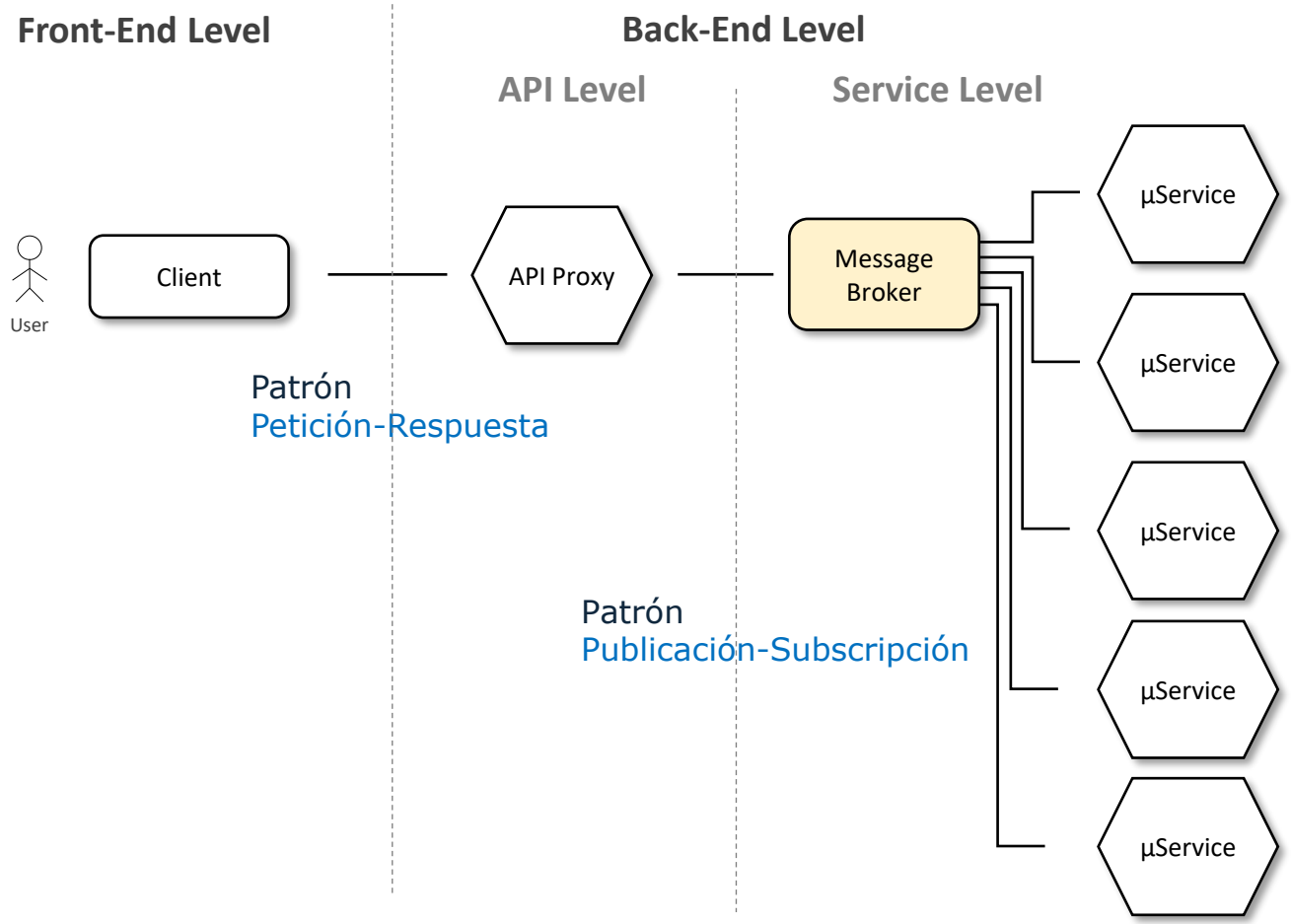
Arquitectura Distribuida basada en µServicios + Patrón API Proxy

Arquitectura distribuida basada en C/S



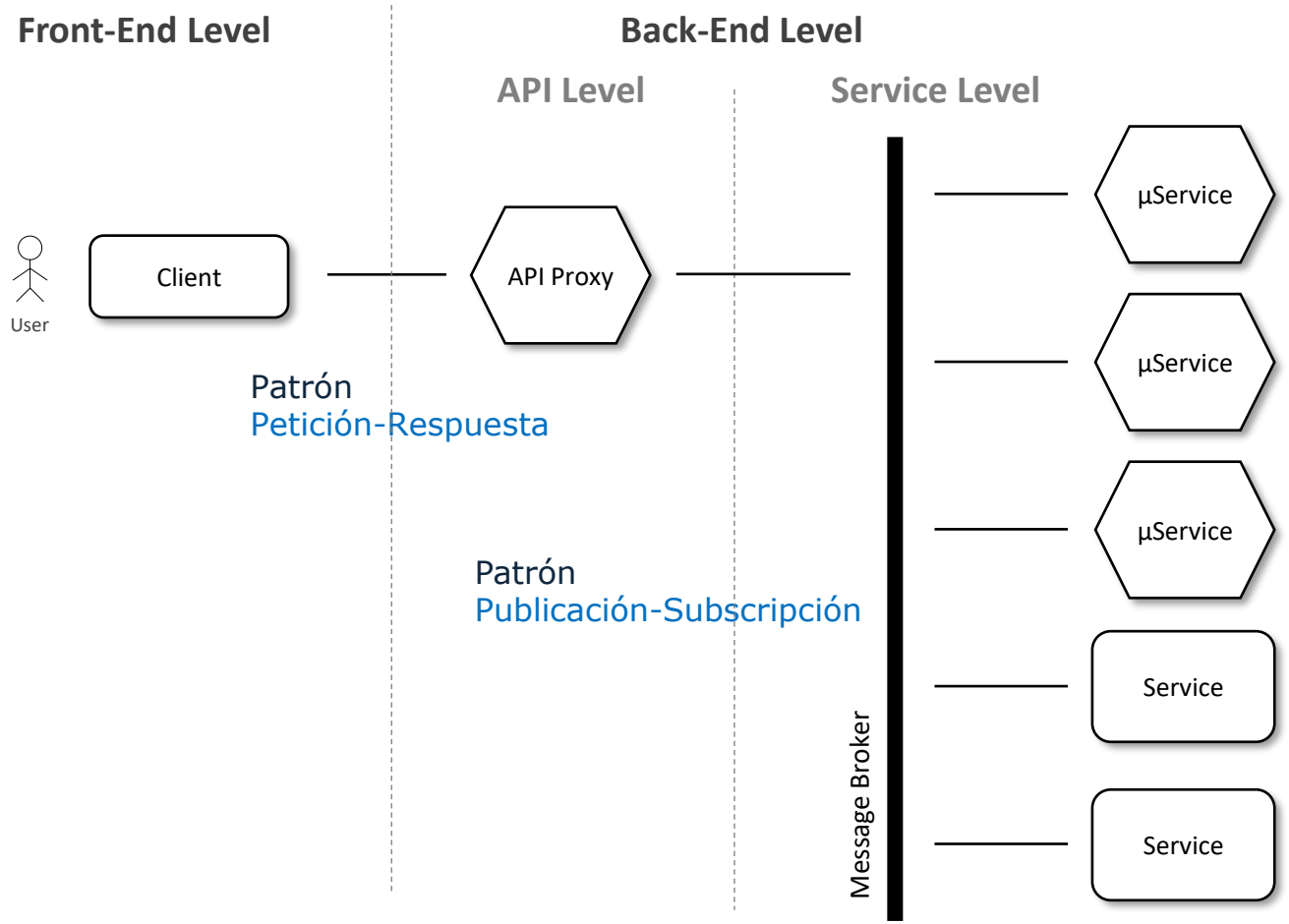
Arquitectura de Microservicos

Arquitectura Distribuida basada µServicios + Patrón API Proxy con Publicación-Subscripción



Arquitectura de Microservicios

Arquitectura Distribuida basada µServicios + Patrón API Proxy con Publicación-Subscripción



Cluster y Grid

Cluster y Grid

Aspectos comunes

Infraestructura

Infraestructuras hardware y software para ofrecer mayor capacidad de procesamiento y almacenamiento

01

Arquitectura

Arquitecturas de computación distribuida conformado por un conjunto de ordenadores

02

Objetivo

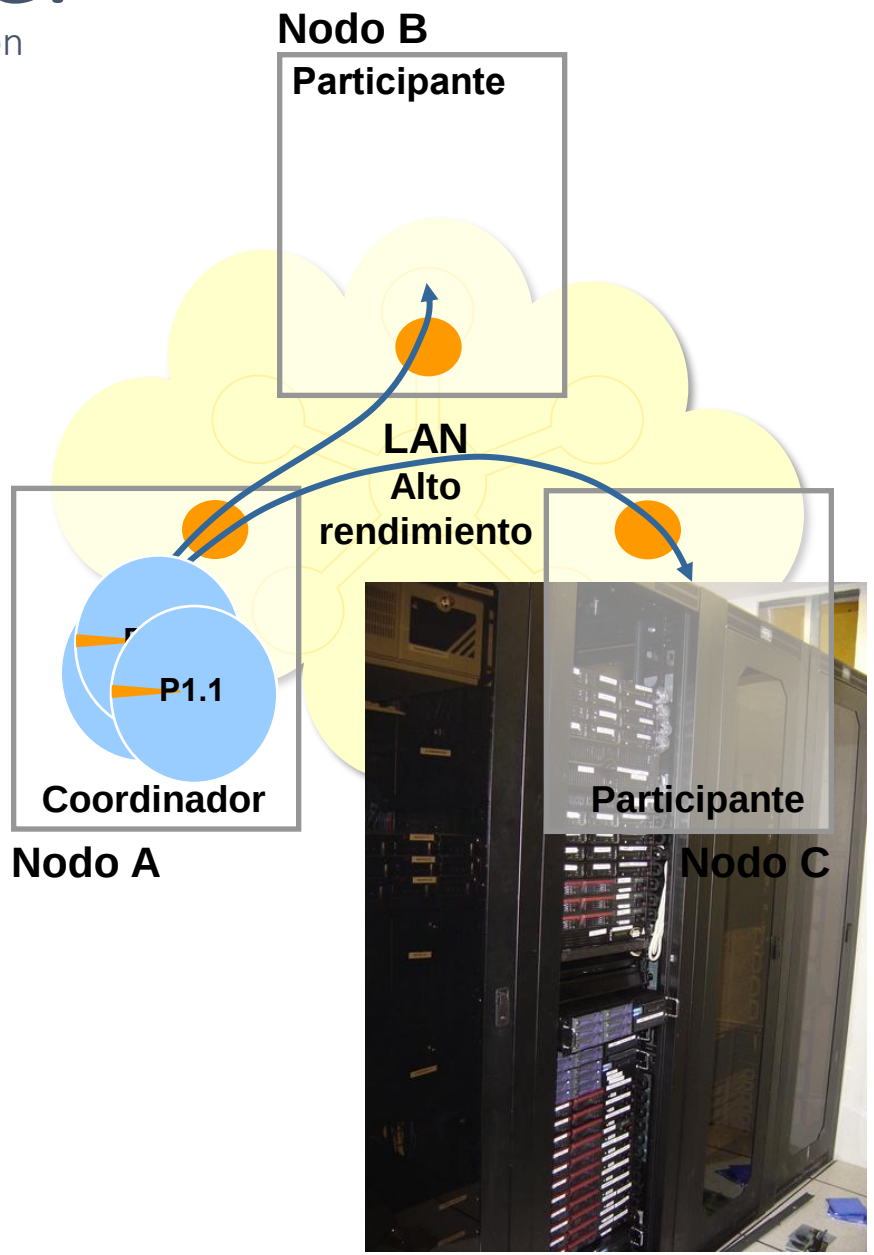
Mejorar la escalabilidad y el rendimiento de los sistemas informáticos

03

Cluster

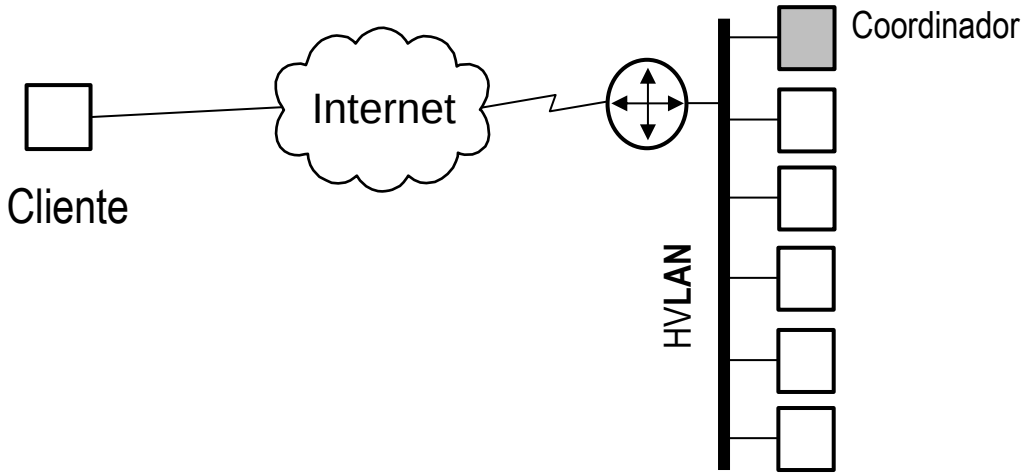
Introducción

- **Homogeneidad**
- Red local de alta velocidad
- Entorno **dedicado**
- Gestor de recursos centralizado
- Tipos
  - Alta disponibilidad
  - Balanceo de carga
  - Escalabilidad
  - Alto rendimiento
- Herramientas
  - MOSIX, OpenMosix, Heartbeat
- Aplicaciones
  - Servidores Web y de aplicaciones
  - Sistemas de información



Cluster

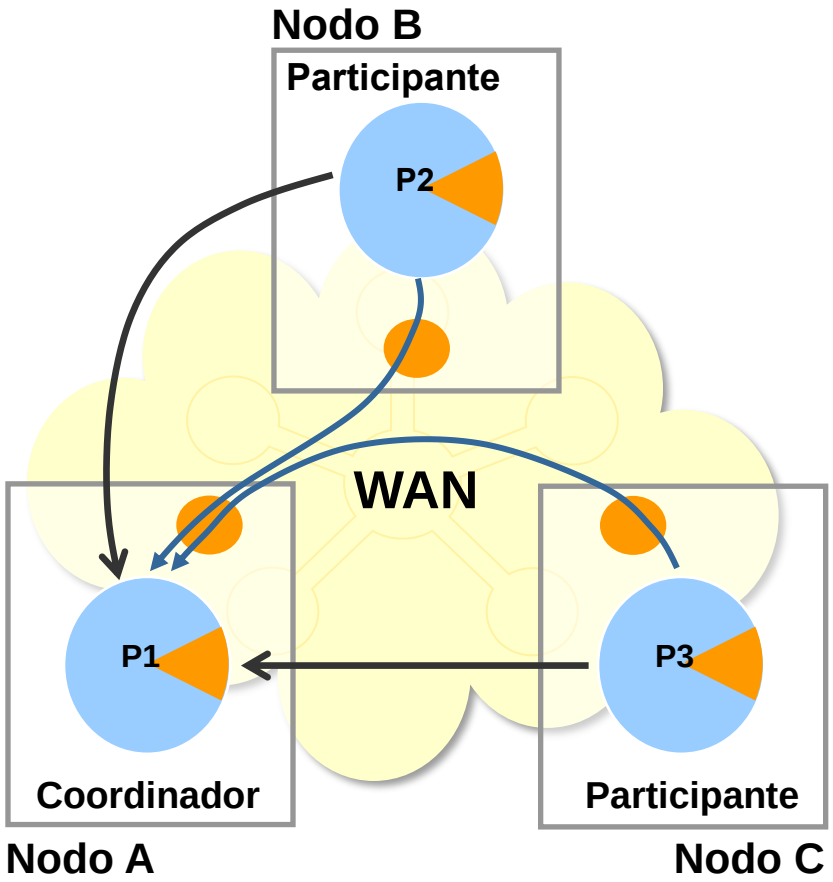
arquitectura física de despliegue



Grid

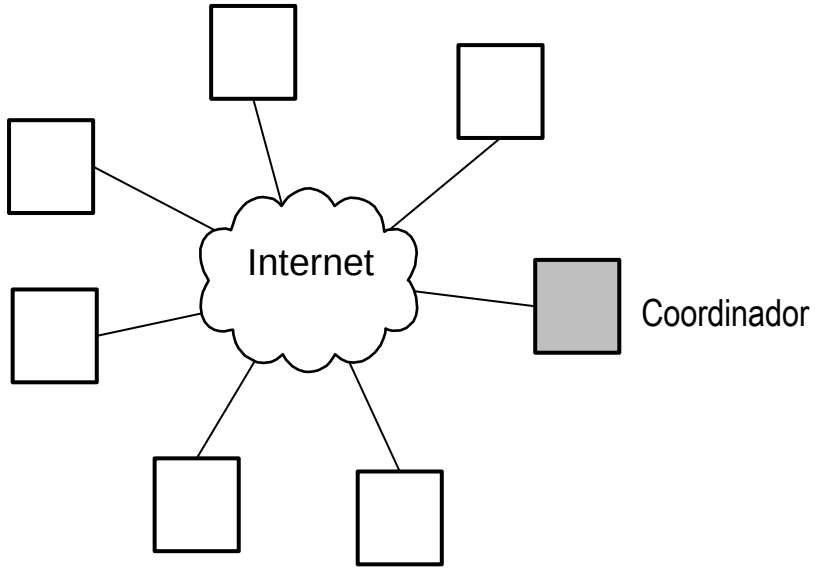
Introducción

- **Heterogeneidad**
- Internet
- **Desacoplamiento**
- Procesamiento y almacenamiento
- No pierde la independencia
  - Tiempos muertos
- OGSA (Open Grid Service Architecture)
  - Grid sobre tecnología Web
- Herramientas:
  - Globus Toolkit 4.0 (código abierto)
- Aplicaciones
  - Proyecto SETI@home, Folding@home, ESGF, etc.



Grid

arquitectura física de despliegue



Peer-to-peer (P2P)

Peer-to-peer (P2P)

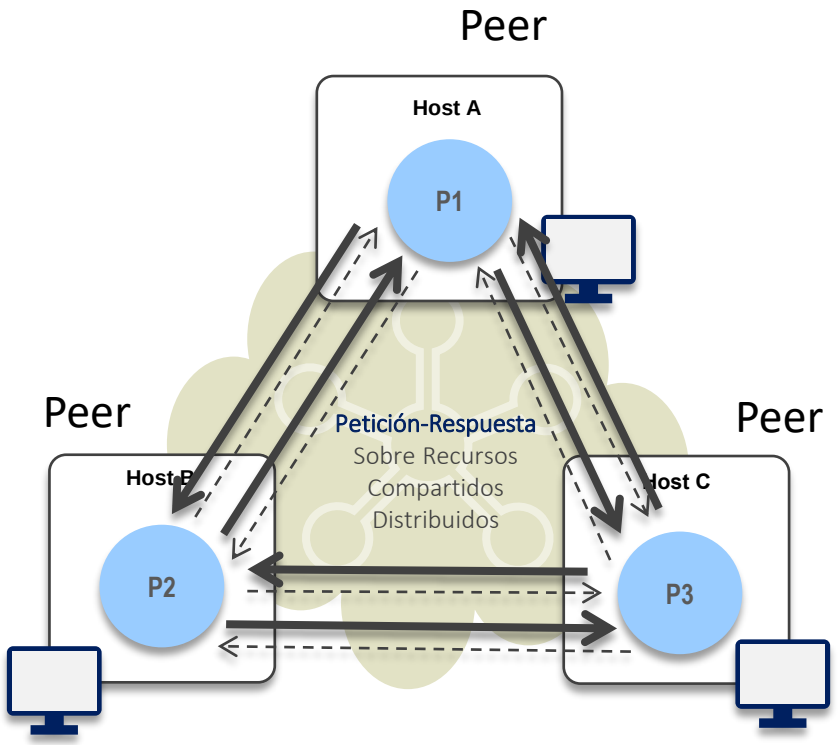
El **modelo peer-to-peer** (**P2P**) no tiene roles fijos: todos los nodos (peers) actúan como de cliente o servidor, solicitando y proporcionando servicios → es descentralizado: mejora la escalabilidad y tolerancia a fallos.

Características principales:

- **Escalabilidad dinámica:** a mayor número de nodos, mayor capacidad.
- **Alta tolerancia a fallos:** la caída de nodos no afecta el sistema global.
- **Descentralización:** no depende de un servidor central.

Apropiado para aplicaciones tipo: mensajería instantánea, transferencia de archivos, video conferencia y trabajo colaborativo. Por ejemplo :

- BitTorrent y µTorrent, redes blockchain, ZeroNet

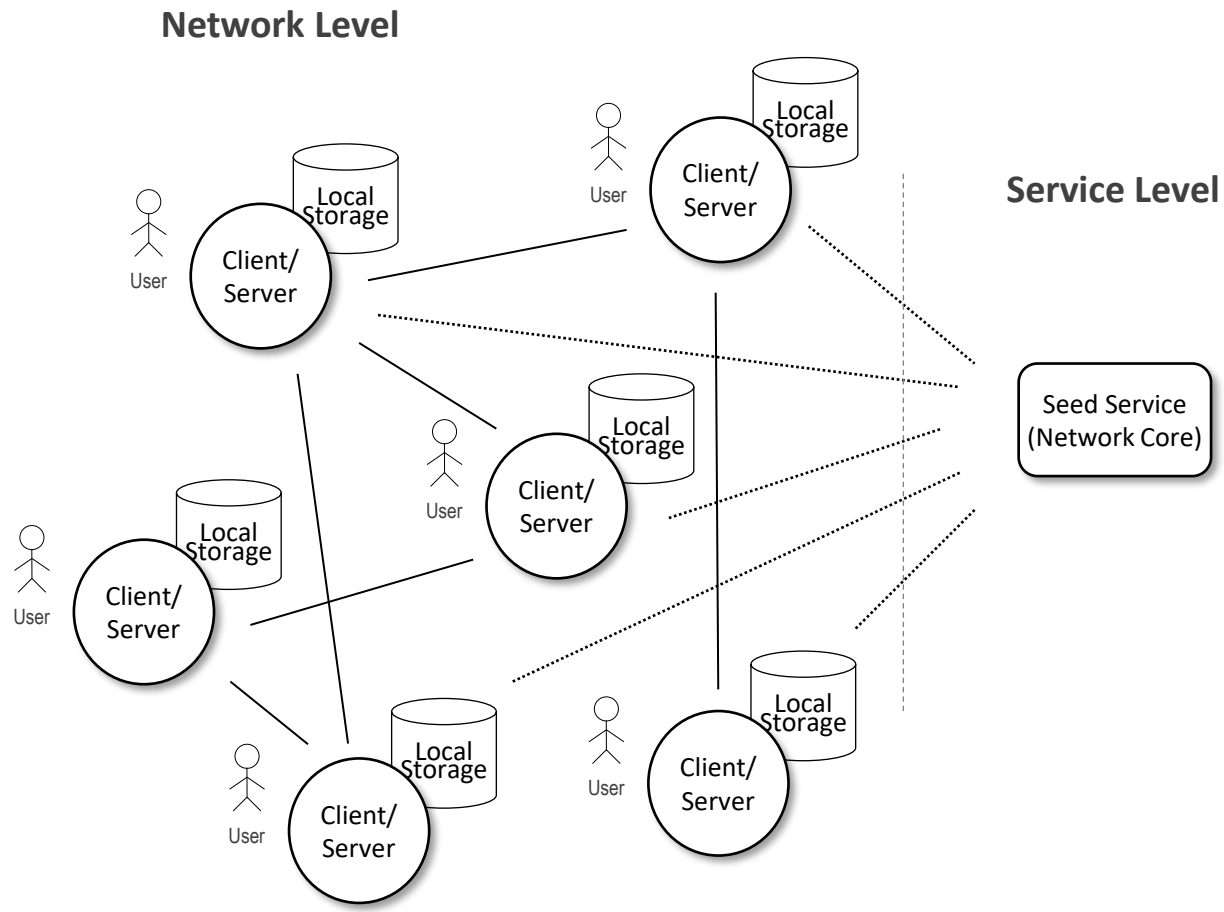


Actuando como Cliente: envío de peticiones, recepción de respuesta.

Actuando como Servidor: recepción de solicitudes, procesamiento de solicitudes, envío de respuesta, propagación de solicitudes.

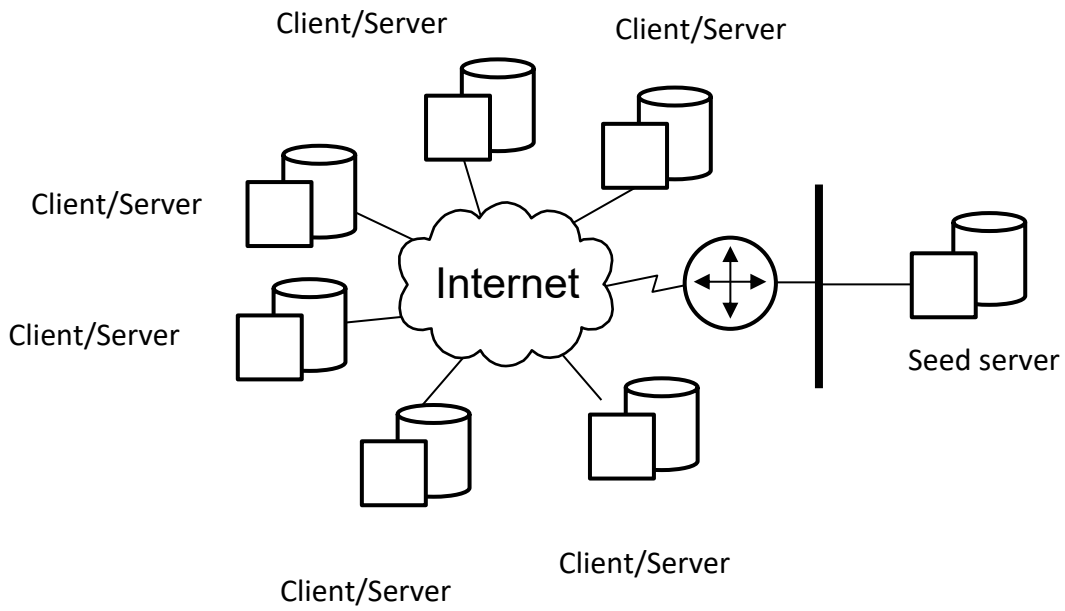
Peer-to-peer (P2P)

Arquitectura de n-niveles



Peer-to-peer (P2P)

arquitectura física de despliegue



Arquitectura Cloud

Arquitectura Cloud

Características

Utiliza **infraestructura distribuida** accesible a través de **Internet** para ofrecer **servicios escalables** y **bajo demanda**.

Recursos gestionados por proveedores de servicios en la nube,

→ Las organizaciones pueden escalar aplicaciones de manera flexible sin gestionar infraestructura física.

Características principales:

Escalabilidad automática (horizontal y vertical).

Pago por uso.

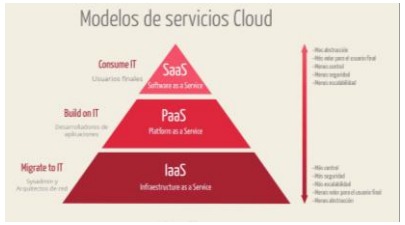
Alta disponibilidad y redundancia.

Tipos de modelos de servicios:

SaaS (Software como Servicio)

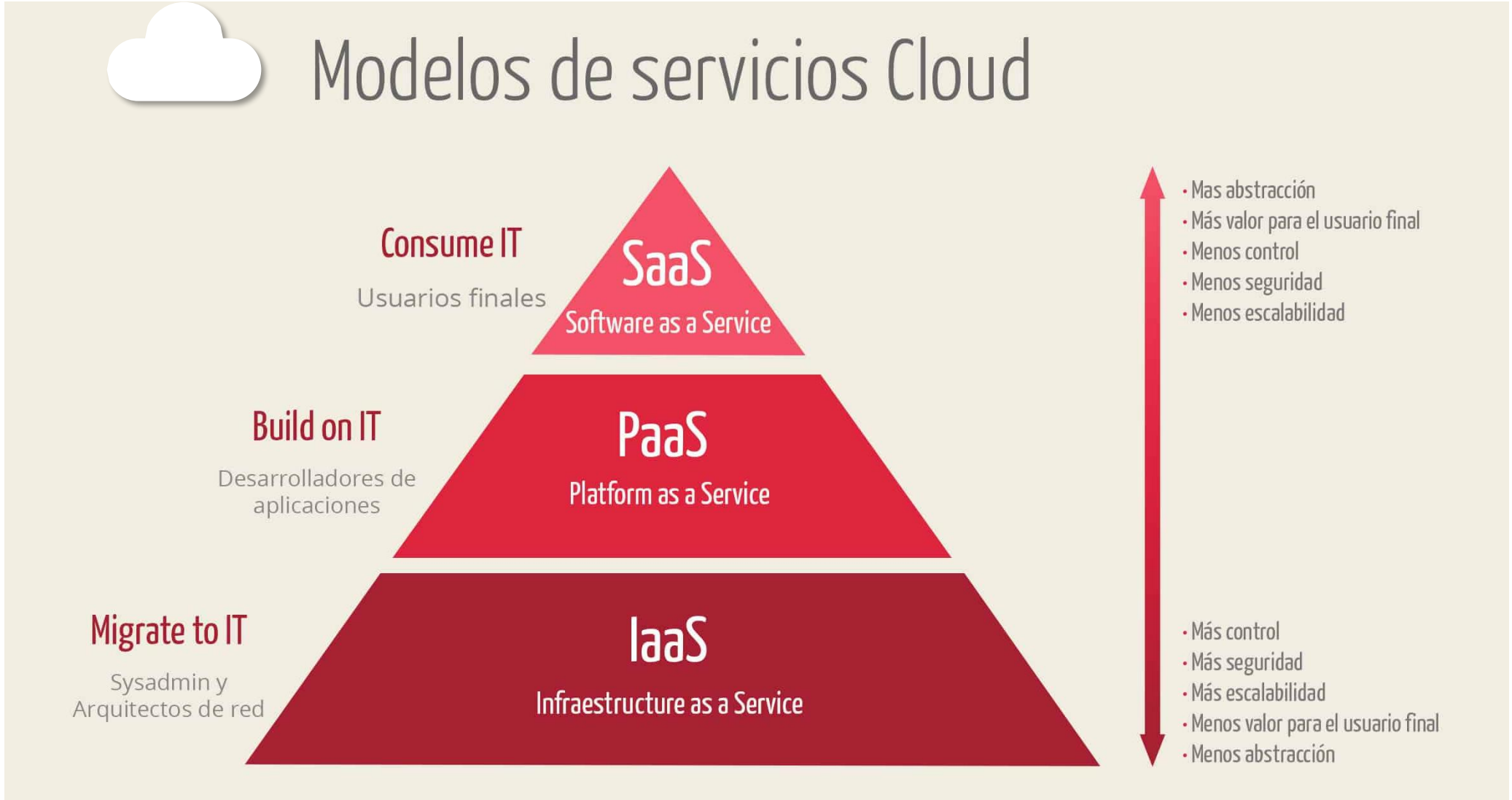
PaaS (Plataforma como Servicio)

IaaS (Infraestructura como Servicio)



Arquitectura Cloud

Características



Arquitectura Cloud

SaaS

- SaaS es un modelo de distribución de software en el que las aplicaciones están alojadas por un proveedor de servicios y están disponibles para los usuarios a través de Internet.
- Los usuarios no tienen que preocuparse por la instalación, el mantenimiento o la gestión de la infraestructura subyacente, ya que todo esto es manejado por el proveedor.
- Ejemplos comunes de SaaS incluyen Google Workspace (anteriormente G Suite) y Microsoft Office 365.

Arquitectura Cloud

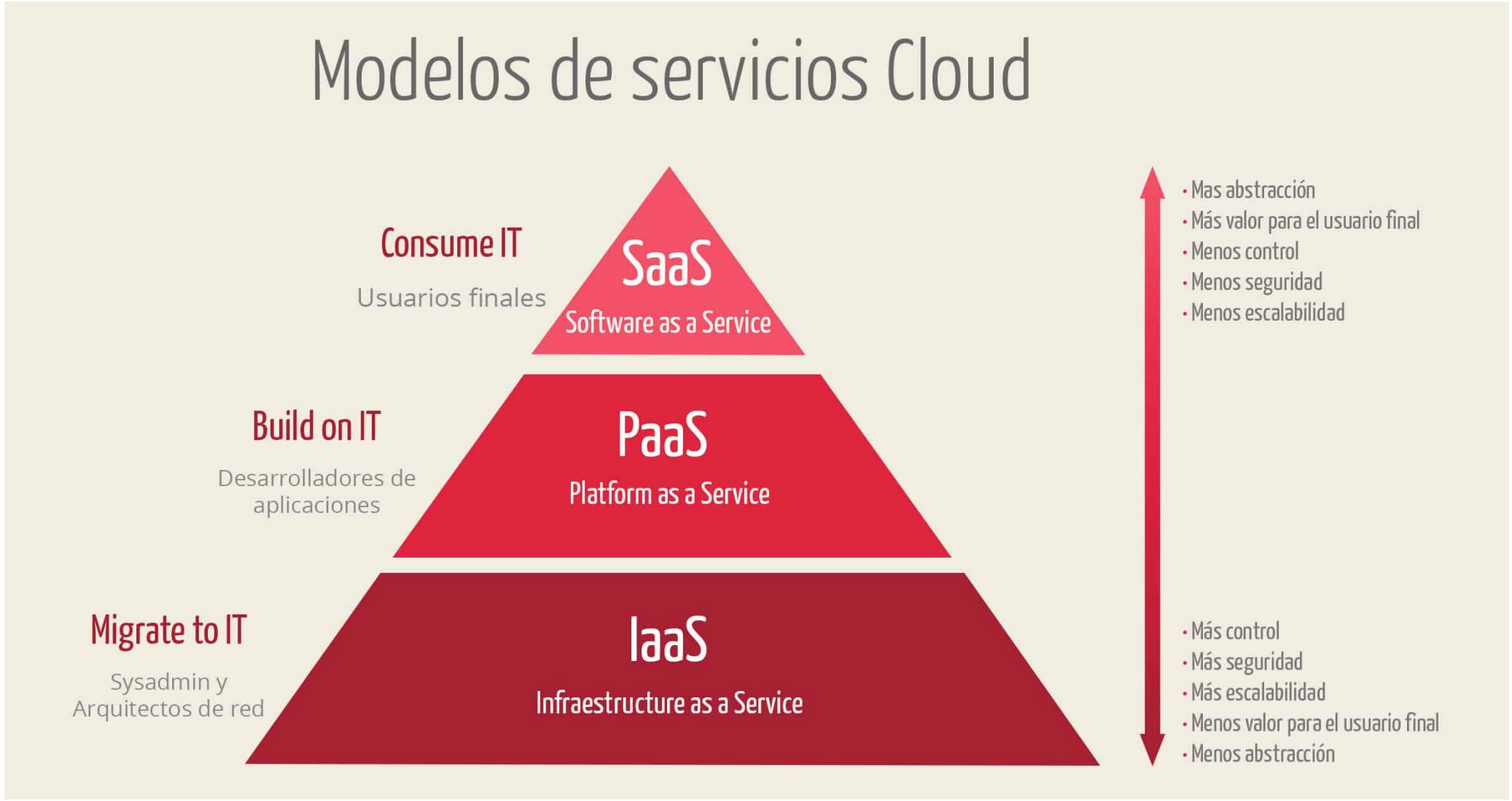
SaaS. Ventajas y desventajas

- Ventajas:
  - No requiere instalación o mantenimiento del software por parte del usuario.
  - Acceso desde cualquier lugar con conexión a Internet.
  - Actualizaciones y mantenimiento gestionados por el proveedor.
  - Escalabilidad fácil según las necesidades del negocio.

- Desventajas:
  - Menor control sobre la personalización y la funcionalidad del software.
  - Dependencia del proveedor para la disponibilidad y el rendimiento del servicio.
  - Consideraciones de seguridad y privacidad de los datos almacenados en la nube.

Arquitectura Cloud

Características



Arquitectura Cloud

PaaS

- PaaS proporciona una plataforma que permite a los desarrolladores construir, desplegar y gestionar aplicaciones sin tener que gestionar la infraestructura subyacente (hardware y sistemas operativos).
- PaaS ofrece un entorno de desarrollo completo, incluyendo sistemas operativos, bases de datos, servidores web y herramientas de desarrollo. Ejemplos de PaaS: Google App Engine.

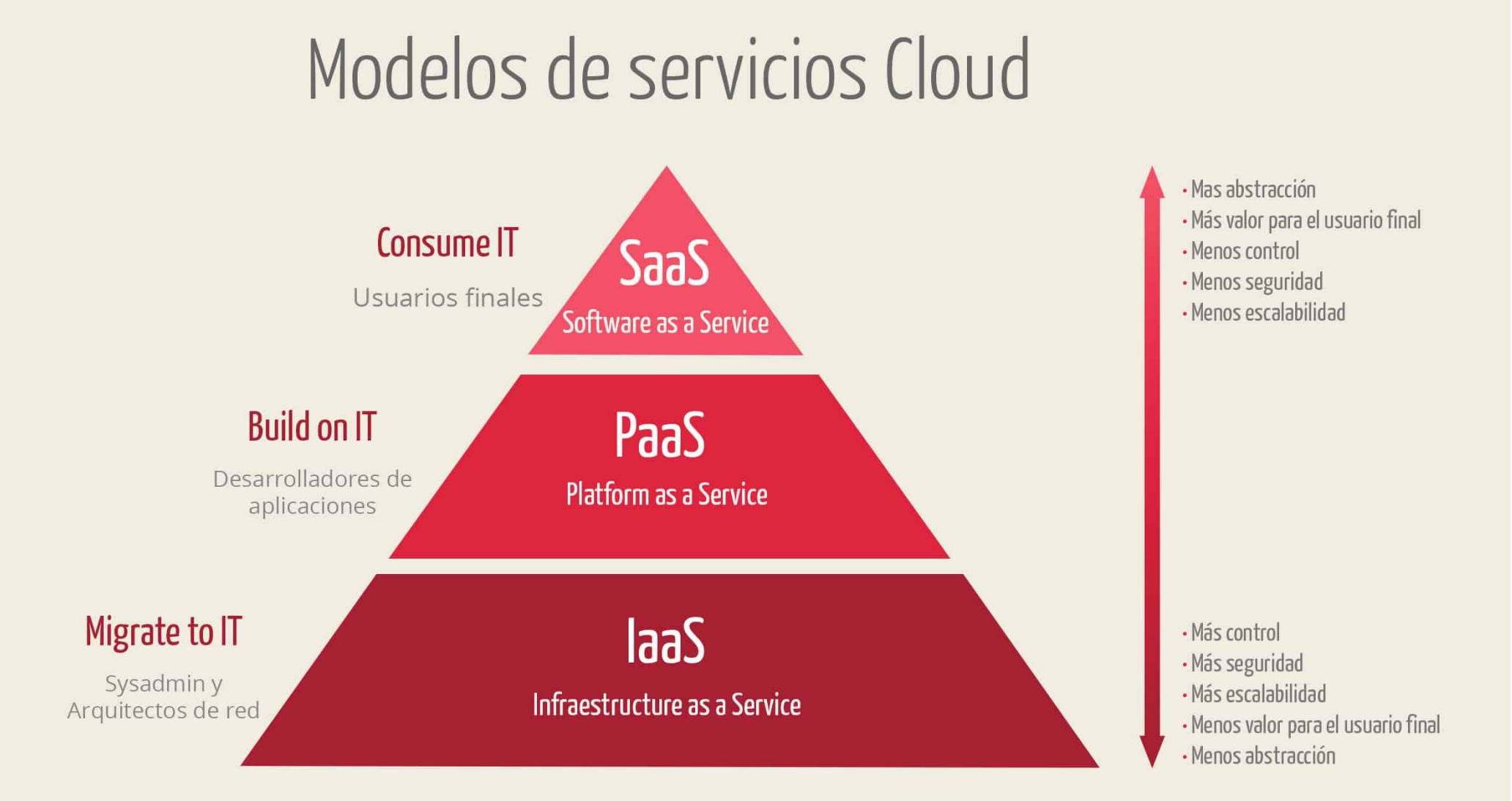
Arquitectura Cloud

PaaS. Ventajas y desventajas

- Ventajas:
  - Simplifica el proceso de desarrollo y despliegue de aplicaciones.
  - Proporciona herramientas y servicios integrados para el desarrollo, pruebas y despliegue.
  - Escalabilidad y gestión automática de la infraestructura.
  - Permite a los desarrolladores centrarse en el código y la funcionalidad de la aplicación.
- Desventajas:
  - Dependencia del proveedor para la disponibilidad y el rendimiento de la plataforma.
  - Posibles limitaciones en la personalización y control de la infraestructura subyacente.
  - Consideraciones de interoperabilidad y portabilidad de las aplicaciones.

Arquitectura Cloud

Características



Arquitectura Cloud

IaaS

- IaaS proporciona recursos informáticos virtualizados a través de Internet.
- Los usuarios tienen acceso a servidores, almacenamiento y redes virtuales, y pueden instalar y gestionar cualquier software, incluyendo sistemas operativos y aplicaciones.
- IaaS ofrece una infraestructura escalable y flexible, y los usuarios pagan solo por lo que utilizan.
- Ejemplos de IaaS incluyen Amazon Web Services (AWS), Microsoft Azure y Google Cloud Platform (GCP).

Arquitectura Cloud

IaaS. Ventajas y desventajas

- Ventajas:
  - Control total sobre la infraestructura virtual.
  - Escalabilidad según las necesidades de recursos.
  - Pago por uso, lo que puede reducir costos.
  - Flexibilidad para instalar y configurar cualquier software necesario.

- Desventajas:
  - Mayor responsabilidad en la gestión y mantenimiento de la infraestructura.
  - Requiere conocimientos técnicos para configurar y administrar la infraestructura.
  - Consideraciones de seguridad y cumplimiento normativo.

Edge Computing

Edge Computing

Definición

- Procesa los datos cerca del origen, en dispositivos locales o intermedios, en lugar de centralizarlos en la nube.
- Esto reduce la latencia y el uso de ancho de banda
- Crucial en aplicaciones como IoT y análisis en tiempo real.
- Características:
  - Procesamiento en el borde de la red.
  - Menor latencia y mayor velocidad de respuesta.
  - Uso eficiente de recursos locales y reducción de tráfico a la nube.
- Ejemplo:
  - Cámaras de vigilancia con modelos de IA integrados para detectar objetos o contar personas en tiempo real, enviando solo los resultados, no las imágenes completas.

Tema 1. Fundamentos de la Computación Distribuida

Contenidos

Paradigmas de computación distribuida

Evolución de los modelos de computación distribuida

Enfoques de Sistemas Operativos

SOR, SOD, Middleware

Modelos arquitectónicos de sistemas distribuidos

C/S, P2P, MOM, SOA, Cluster y Grids

Sistemas Operativos y Distribuidos

Iren Lorenzo Fonseca
iren.fonseca@.ua.es

TEMA 3. Sistemas Distribuidos.

Fundamentos de la Computación Distribuida