

Hola:

Con respecto al último examen parcial quería comentaros que el hecho de no poner una clase de ejercicios no es solo por una cuestión de tiempo, sino también porque es importante que sepáis estudiar los temas con lo que recibís en clase sin necesidad que yo misma os de un resumen.

De todas formas, tampoco quiero que os agobiéis. Recordad que mis preguntas de examen nunca van a pillar y que no pregunto nada que no os haya dado.

De los exámenes anteriores sabéis que no suelo preguntar cosas donde tengáis que saber muchas cosas de memoria, sino que me interesa que sepáis aplicar los conceptos de modo que las preguntas de este examen serán de ese estilo.

En cuanto a fundamentos de los SD:

1. Descripción de una arquitectura y que tengáis que identificar cuál es y explicar sus características (C/S, SOA, MOM, Microservicios, Cluster, Grid, P2P, Cloud y Edge)
2. Quiero que tengáis claro las características de los Sistemas Distribuidos y que hay que tener en cuenta de ellas (primeras diapositivas de la presentación de fundamentos de SD)
3. Diferencias entre Sistemas Operativos en Red, Sistemas Operativos Distribuidos y Middlewares

Referente a Tecnologías:

1. Quiero que sepáis cómo van apareciendo tecnologías que añaden abstracción a los sockets, tanto para llamadas a procedimientos remotos (RPC), como para funciones de clases en programación Orientada a Objetos de Java (RMI) hasta llegar a los Middleware que tienen su origen en ORB y luego evolucionan como es JEE y .Net
2. Tecnología web:
 - a. Puedo preguntar en cuanto a tecnología web, las características tanto del HTTP Request como en Respond.
 - b. Las llamadas que hace el navegador cuando recibe una página que incluye más de un recurso (explicado en las diapositivas cuando como usuarios solo pedimos una URL y el navegador se encarga de solicitar todo lo que viene como recurso dentro del html)
 - c. Quiero que de las cabeceras controléis los principales elementos como por ejemplo el content-type que usando tipos MIME explica el tipo de contenido que viene en el body
 - d. Quiero que tengáis claro que es un cliente web (sería lo mismo que un cliente http) y un servidor web o http.
3. Servicios Web
 - a. Quiero que tengáis claro que van sobre http y que permiten la comunicación entre aplicaciones en diferentes plataformas
 - b. Que empezaron sobre todo con el auge de HTTP y con la arquitectura SOA que permitía una separación entre el servicio cliente y servidor ya que había un servicio de registro donde los servidores se registraban y luego los clientes, sin tener que saber previamente nada de los servidores,

- podían preguntar al servicio de registro sobre los datos y la forma en que podían acceder a los servicios que ofrecían los servidores (proveedores)
- c. Luego con REST esto se simplifica, porque ya REST estipula como van a ser las llamadas (endpoint) de los servicios y por tanto el servicio de registro deja de ser tan necesario en el contexto de aplicaciones más sencillas. En esta parte de REST tenéis más conocimientos porque es justo el servicio web (API-REST) que estamos haciendo en la práctica. Así que además quiero que tengáis claro como son los endpoints para un servicio REST.
 - d. Por otra parte, quiero que podáis identificar con una situación, que sería más conveniente usar dentro de las tecnologías de servicios web: SOA, REST, GraphQL, gRPC o Websocket

Referente al tema de seguridad:

1. Quiero que tengáis clara la triada de seguridad y lo que significa: integridad, disponibilidad y confidencialidad
2. Tenéis que saber qué es la tecnología de blockchain
3. Quiero que sepáis cómo Bitcoin resolvió el problema del doble gasto (teniendo en cuenta tanto el blockchain como el montar toda la arquitectura P2P y que todos tuvieran la cadena)
4. Quiero que sepáis qué son las funciones hash
5. ¿En qué consiste la firma digital mediante cifrado asimétrico?
6. En el ámbito de las criptomonedas ¿Qué es una transacción? ¿Qué es una UTXO? ¿Qué son las entradas y las salidas de una transacción, y qué relación existe entre ellas? ¿Qué es un Smart-Contract? ¿En qué consiste el bloqueo y el desbloqueo? ¿Qué credenciales se debe aportar en cada caso?
7. Qué es la prueba de esfuerzo en bitcoin y como se utiliza como mecanismo de sincronización y consenso.
8. Qué tiene que hacer un minero para cerrar un bloque

Bueno pues nada, no quería, pero casi que os he dejado un resumen de los principales elementos.

Saludos y nos vemos el viernes,

Iren

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

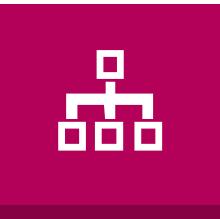
SOR, SOD, Middleware

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

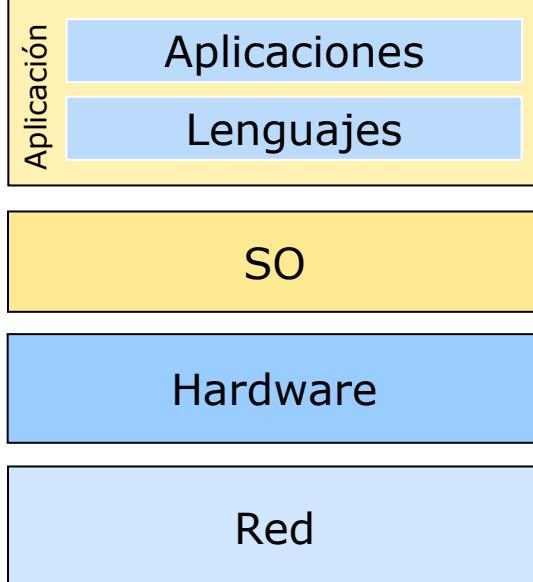
La que se desarrolla en un SD: servicios y aplicaciones de red

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

03

Escalabilidad

Extensibilidad

02

Seguridad

04



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

Firewalls, SSL, HTTPS, Radius, Kerberos

Introducción

Características básicas

01

Heterogeneidad

03

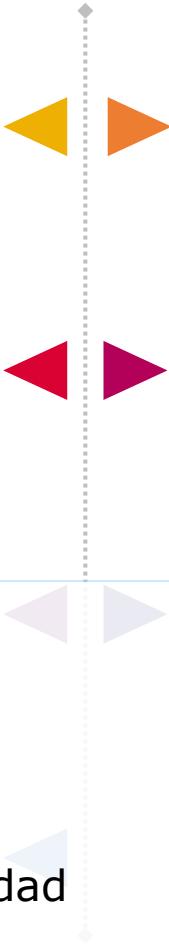
Escalabilidad

05

Concurrencia y sincronización

07

Transparencia



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

Extensibilidad

02

Seguridad

04

Tolerancia a fallos

06

Introducción

Características básicas

01

Heterogeneidad

03

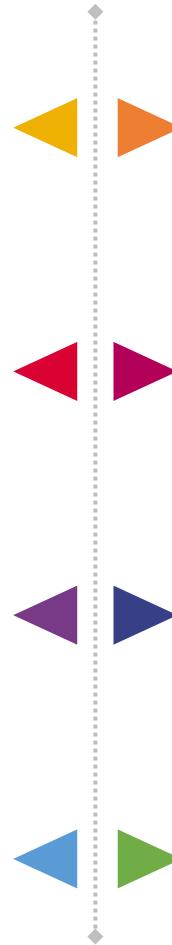
Escalabilidad

05

Concurrencia y sincronización

07

Transparencia



Extensibilidad

02

Seguridad

04

Tolerancia a fallos

06

Administración de recursos

08

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.

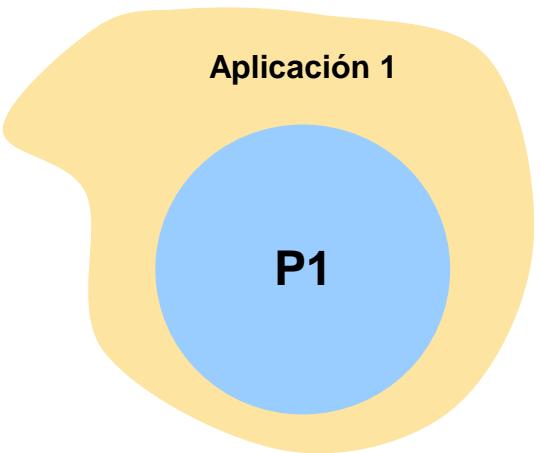
Consistencia de datos: Garantizar que los datos sean consistentes y estén actualizados en todos los nodos puede ser difícil, especialmente en sistemas con muchas réplicas.

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

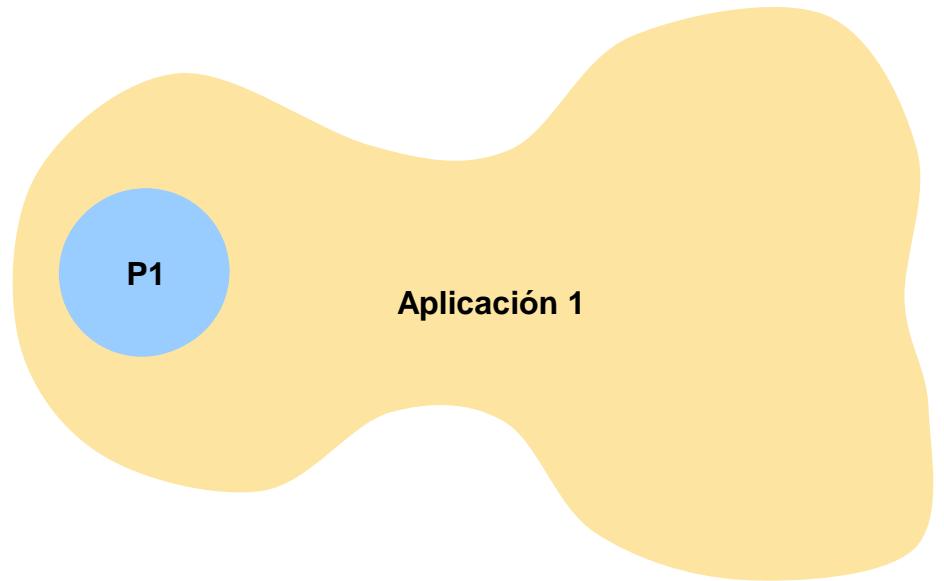
- | **Sistemas Monoprogramados:** Ejecutan un único programa a la vez, aprovechando todos los recursos del sistema. Ejemplo: los primeros sistemas operativos donde un proceso ocupaba toda la CPU.
- | **Sistemas Multiprogramados y Multitarea:** Introducen la capacidad de ejecutar múltiples programas concurrentemente, 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



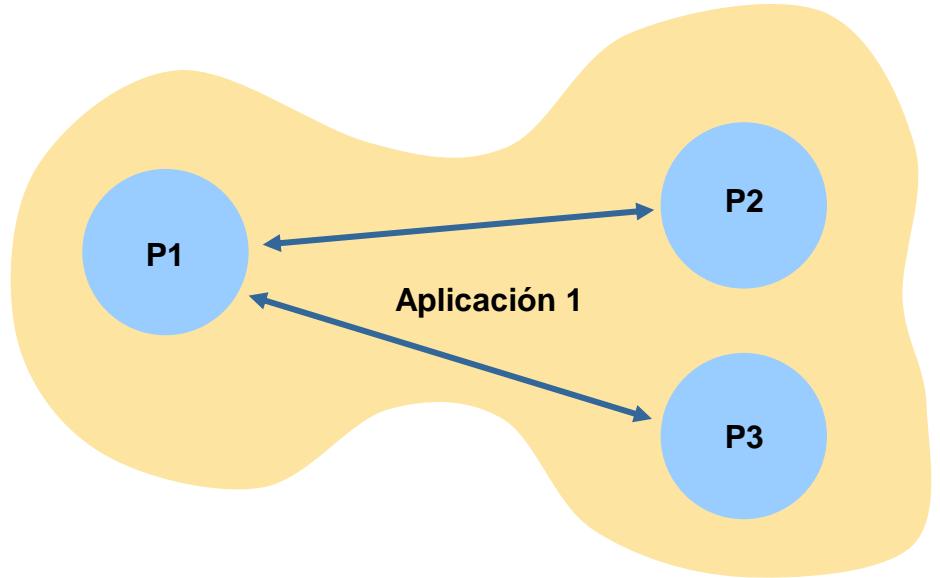
Introducción

Evolución



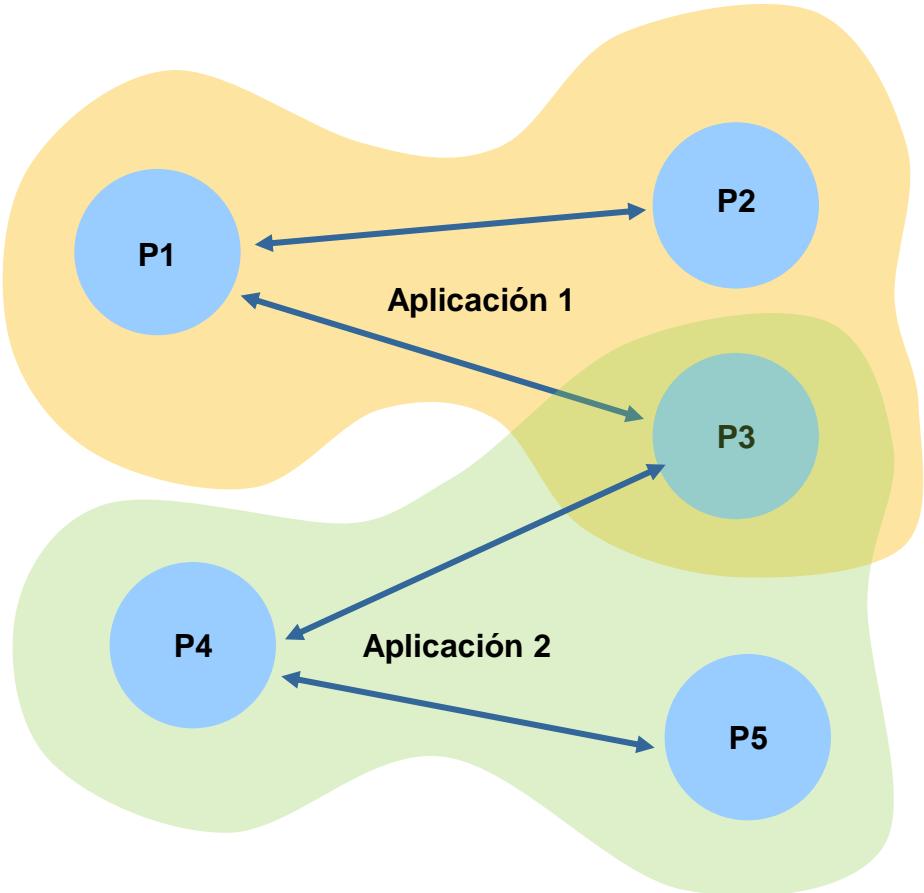
Introducción

Evolución

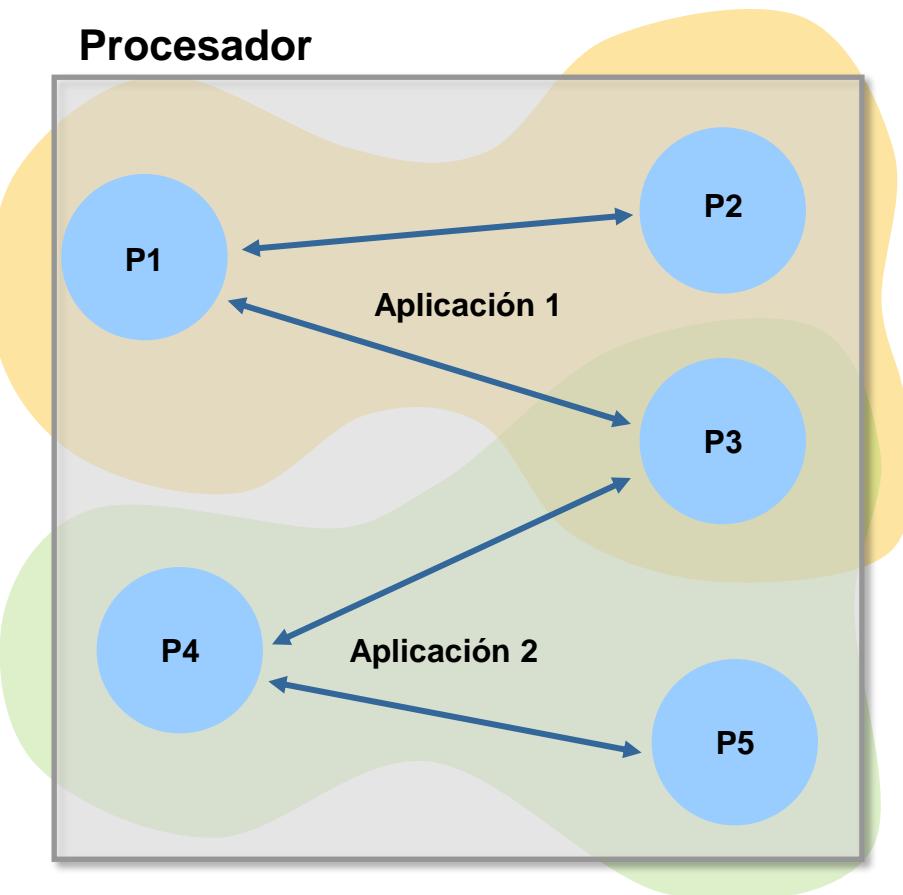


Introducción

Evolución

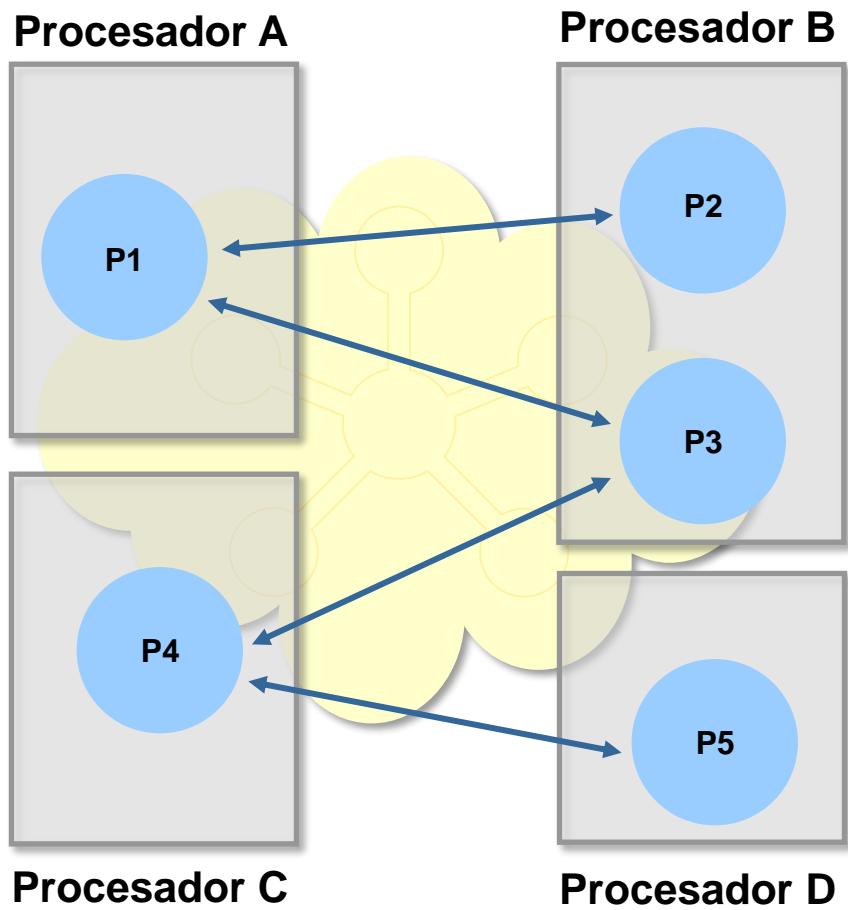


Evolución



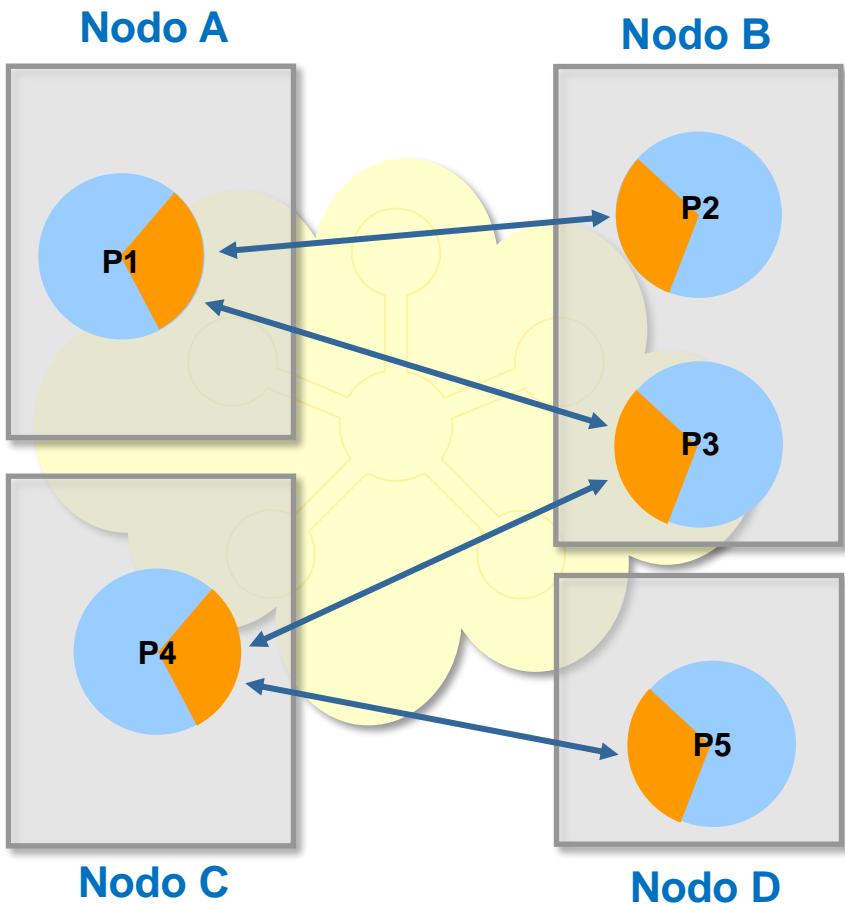
Introducción

Evolución



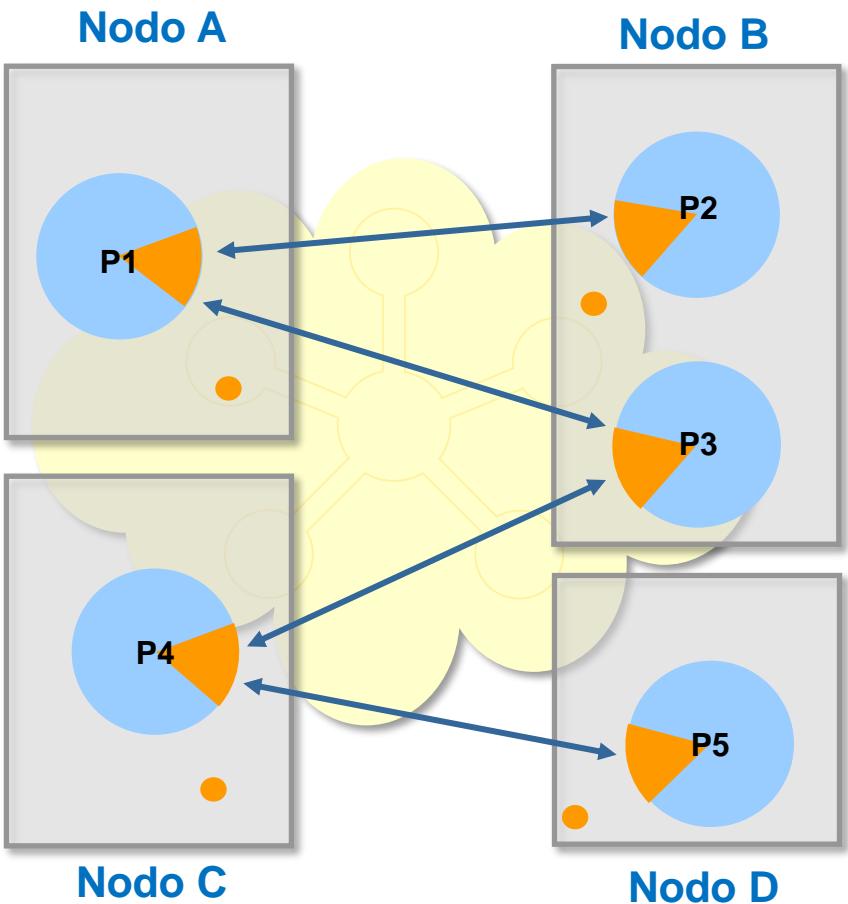
Introducción

Evolución



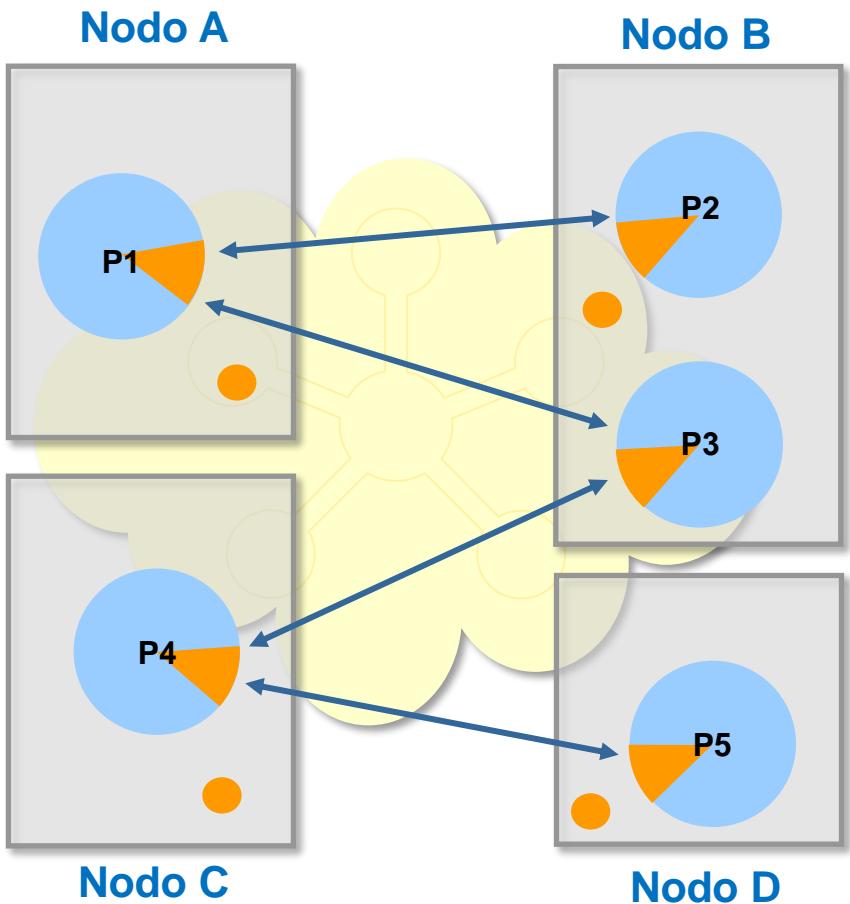
Evolución

Introducción



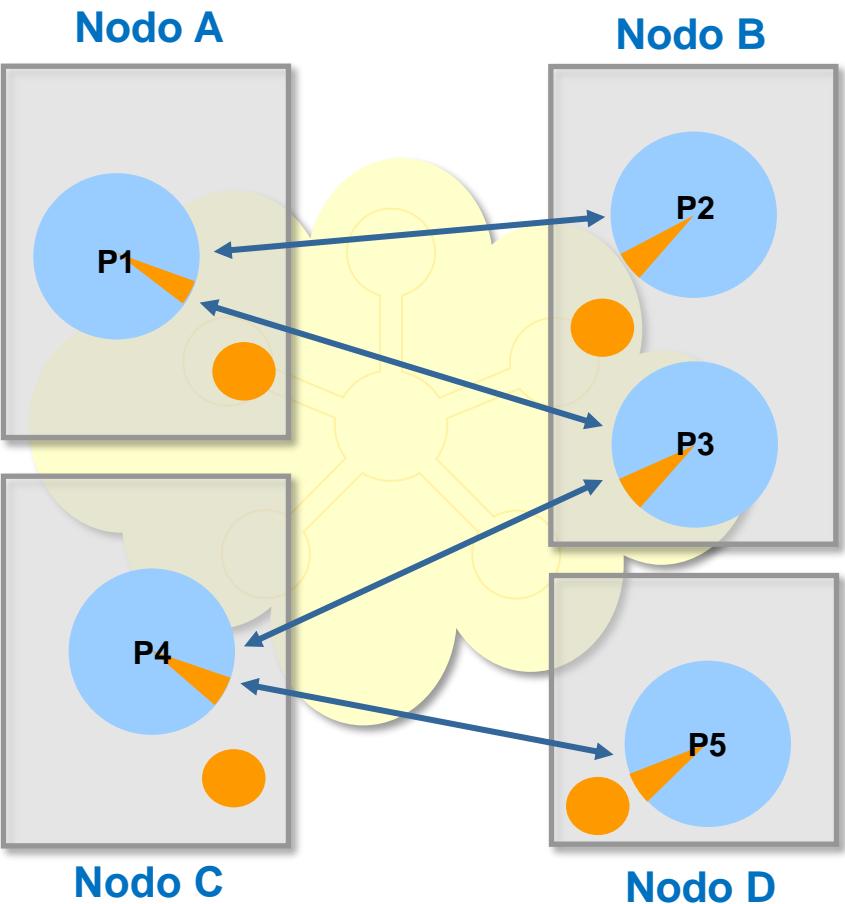
Evolución

Introducción



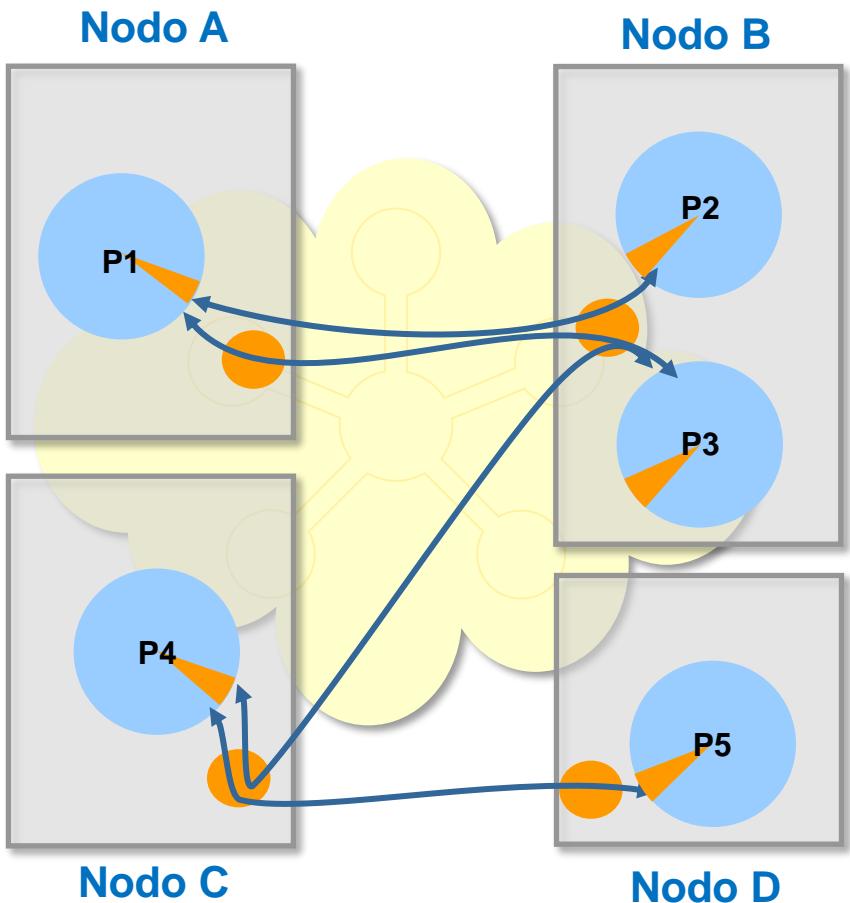
Evolución

Introducción



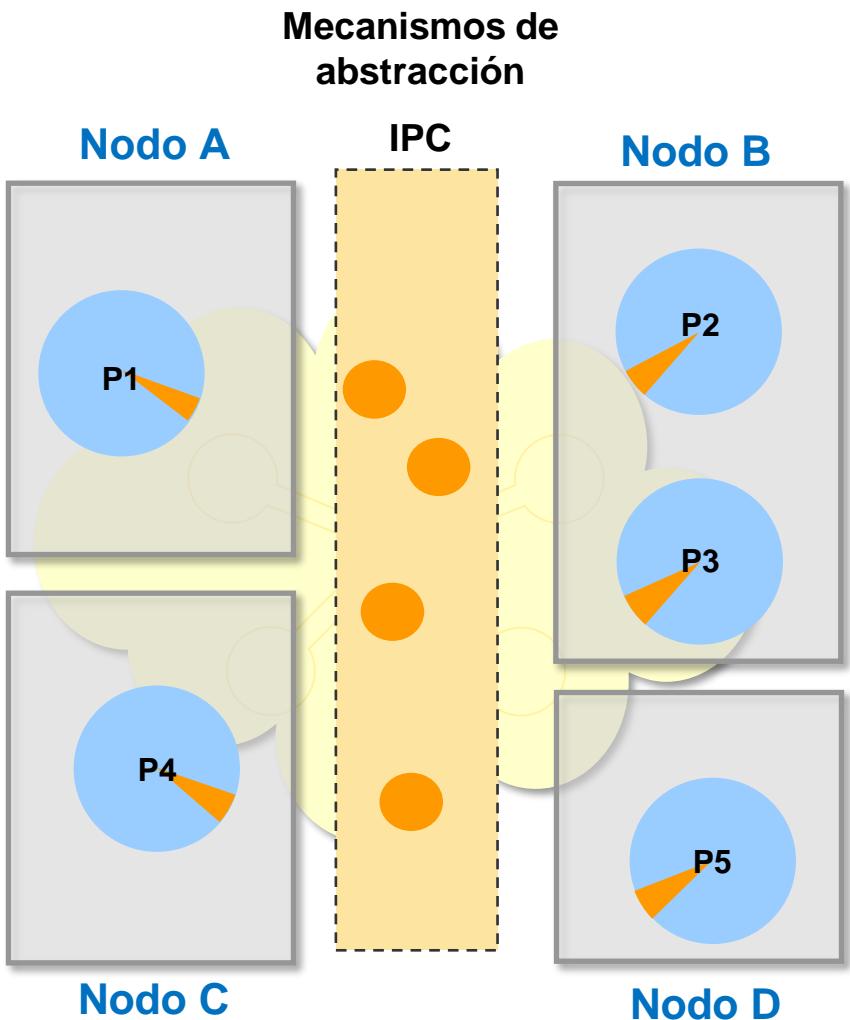
Evolución

Introducción



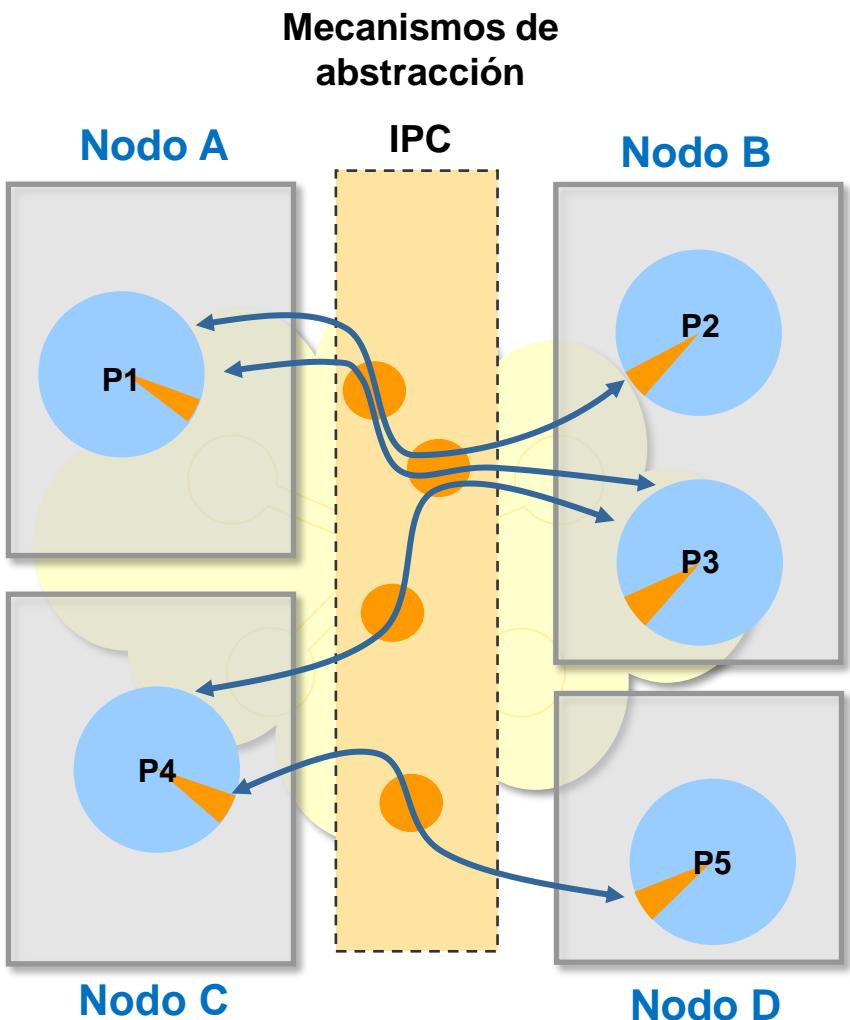
Evolución

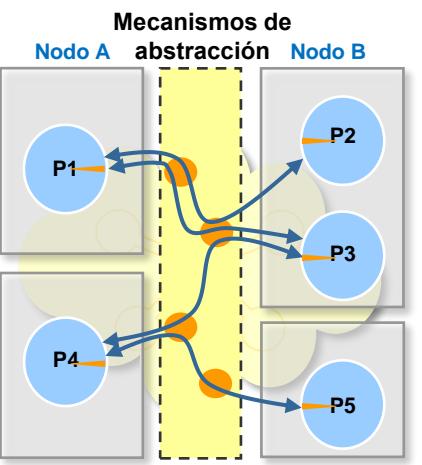
Introducción



Evolución

Introducción





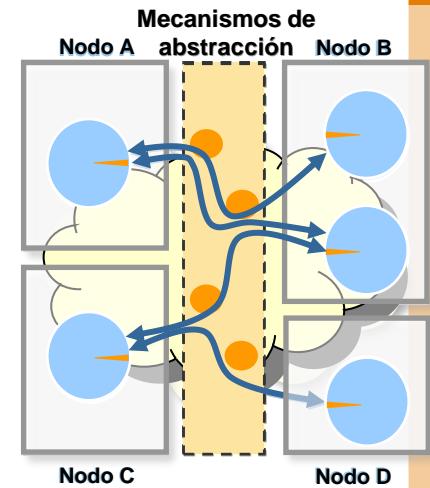
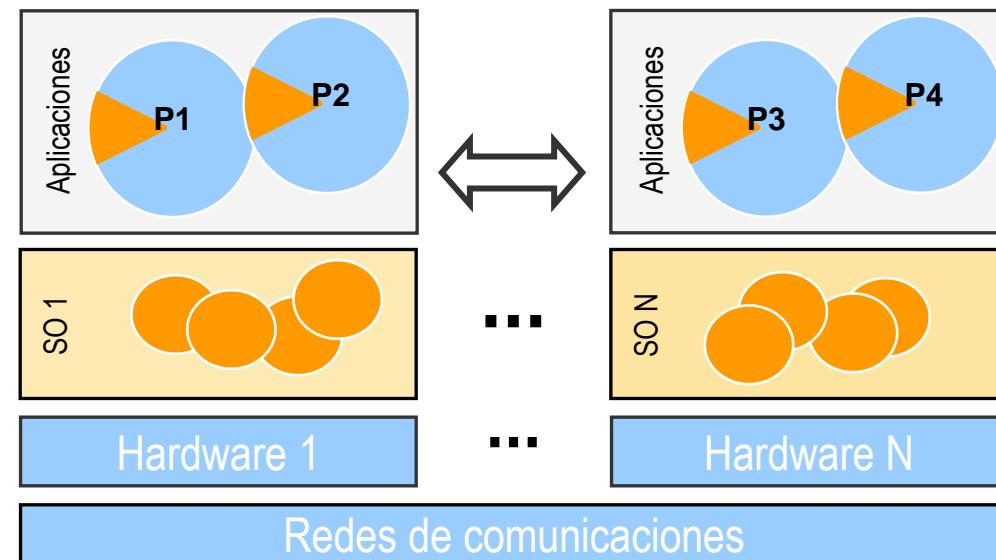
Enfoques de Gestión

- Sistemas Operativos en Red
- Sistemas Operativos Distribuidos
- Middleware

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



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);

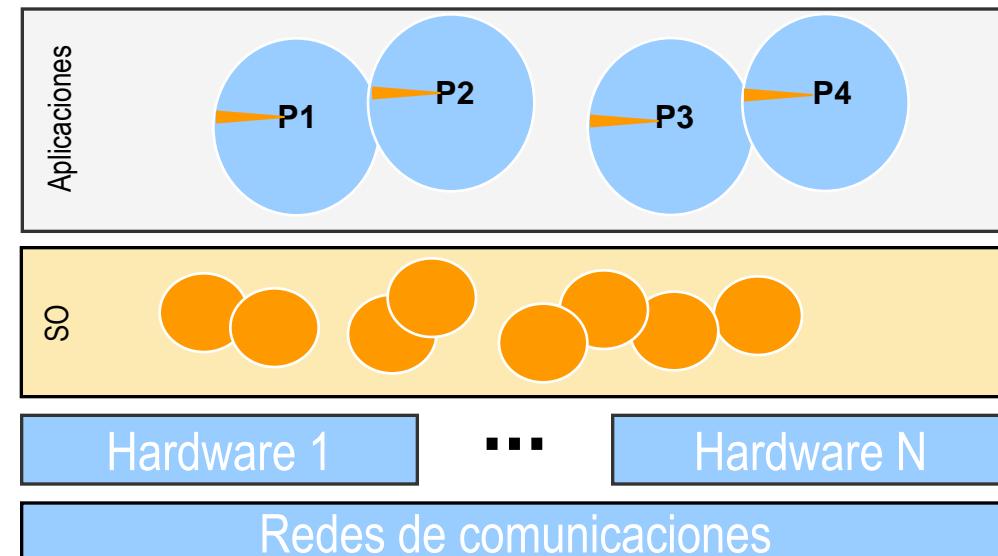
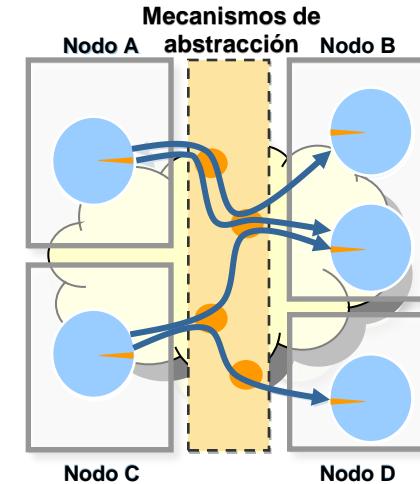
    // Tratamos la respuesta
    printf("%s", HTTP_response);

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



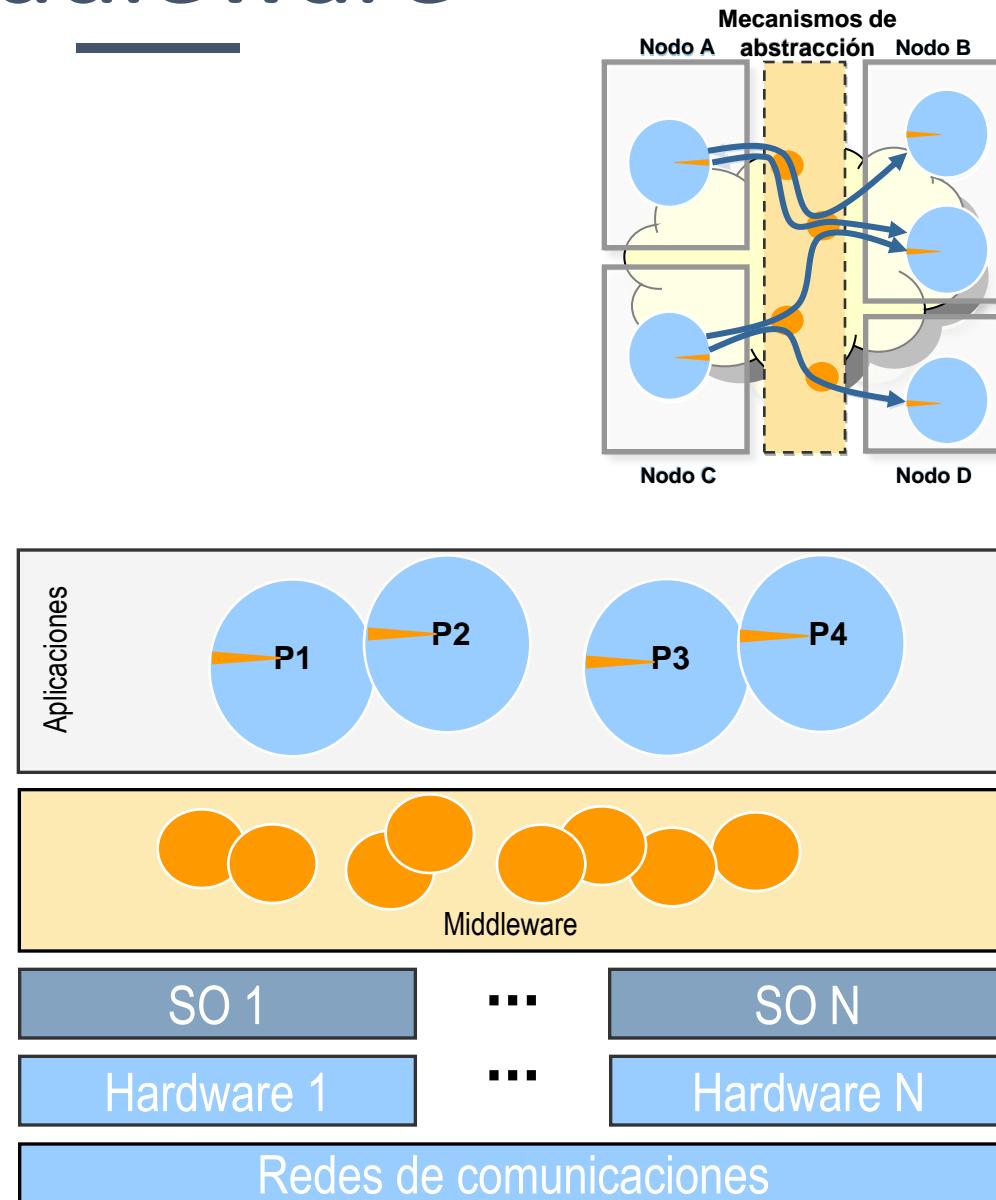
Sistemas operativos distribuidos

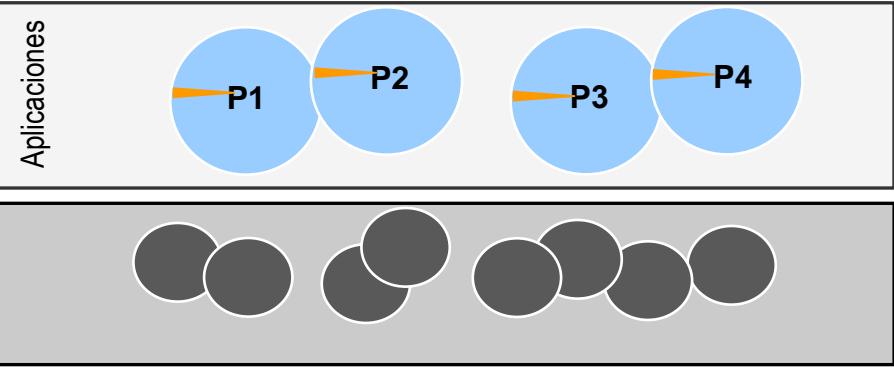
- Ubicación IPC en SO global
- **Homogéneo → SO único**
- Ventajas
 - Transparencia
 - Escalabilidad
 - Facilidad de integración
- Desventajas
 - Técnicas complejas
 - Rendimiento y Latencia
- Ejemplos:
 - Mach, Amoeba



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





- 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

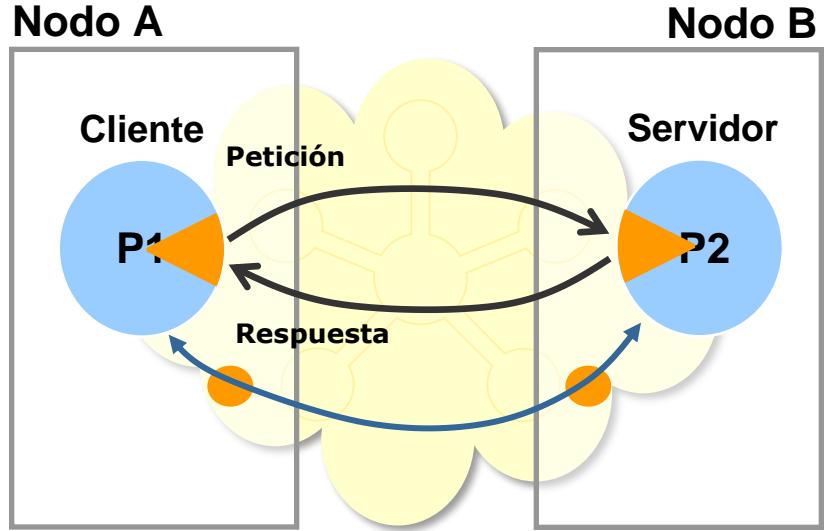
Contenidos

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);

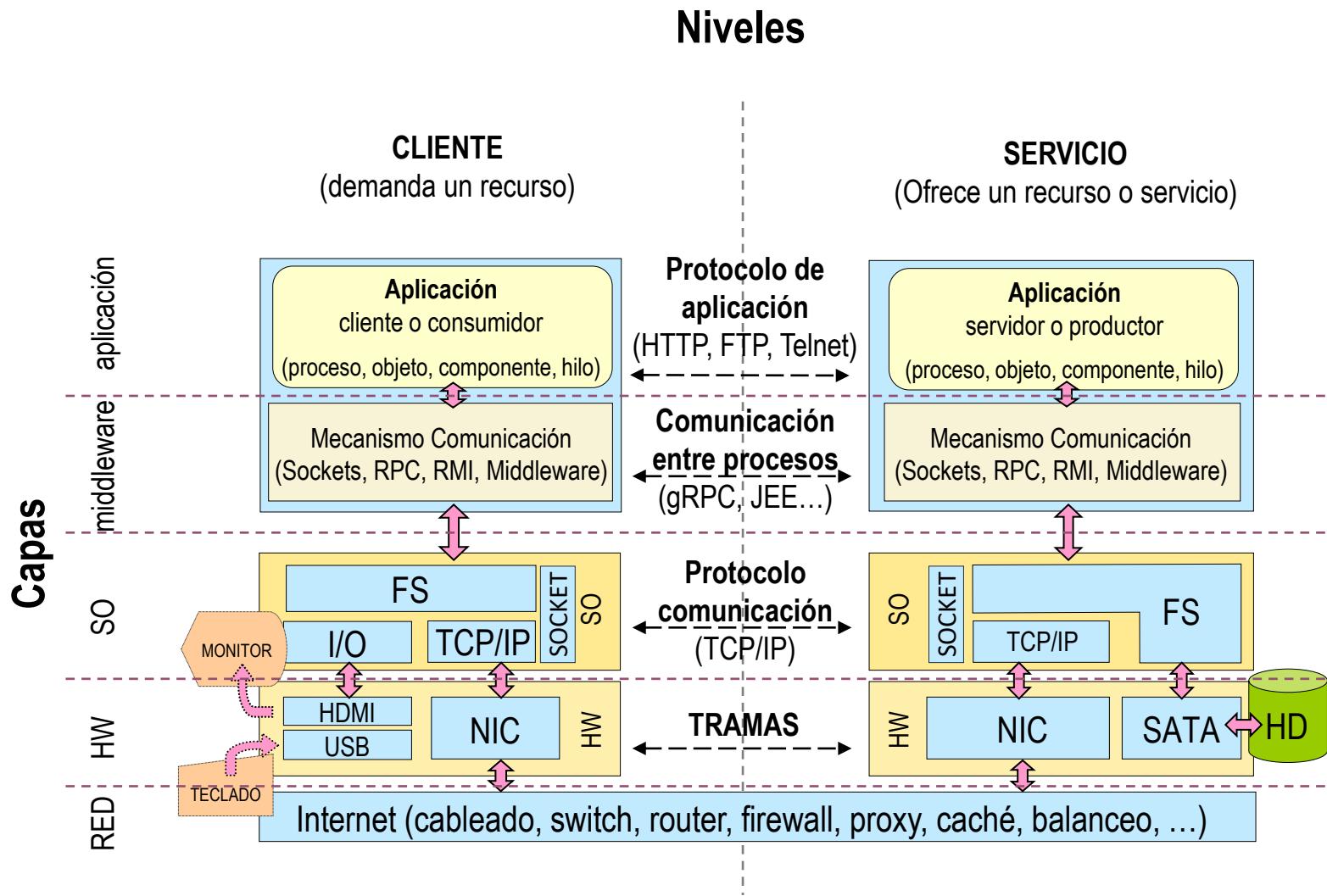
    // Tratamos la respuesta
    printf("%s", HTTP_response);

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

Enfoques de SO

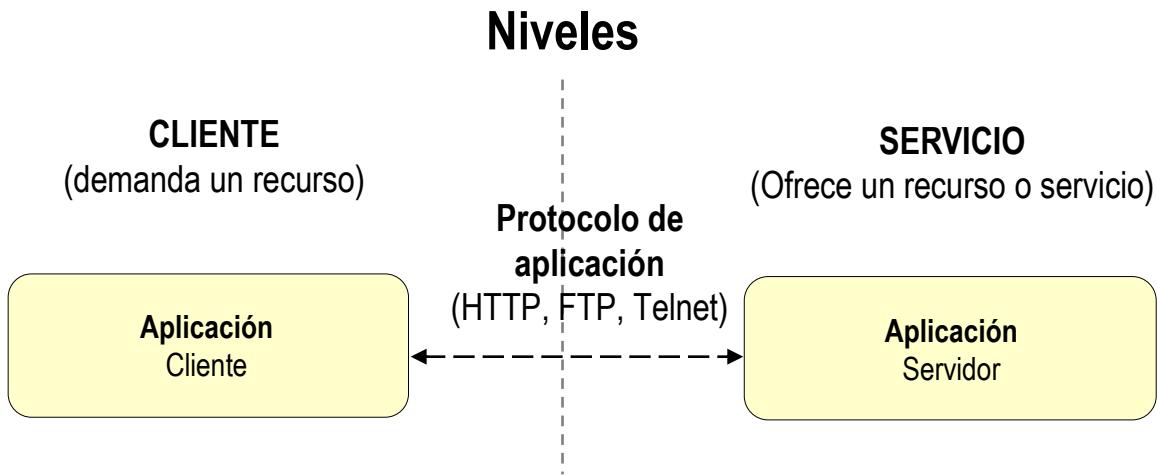
Modelo Cliente/Servidor

Arquitectura C/S de 2-niveles



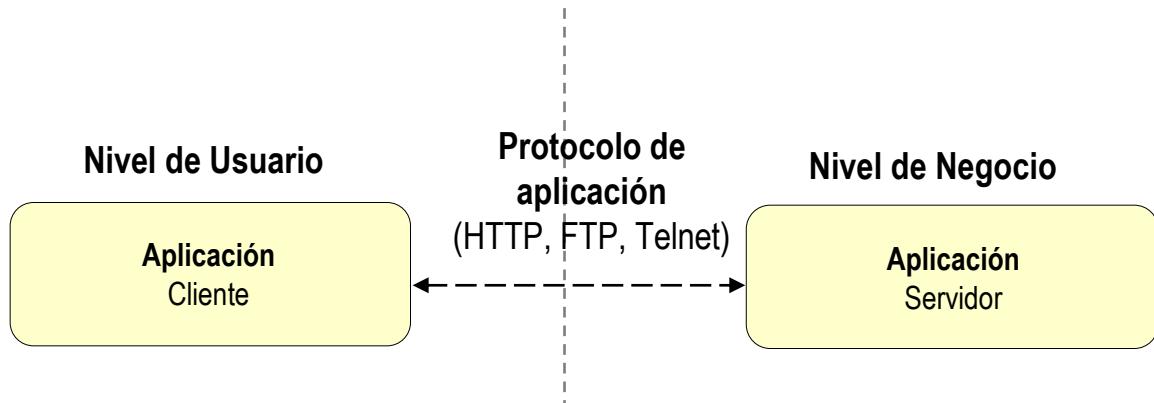
Modelo Cliente/Servidor

Arquitectura C/S



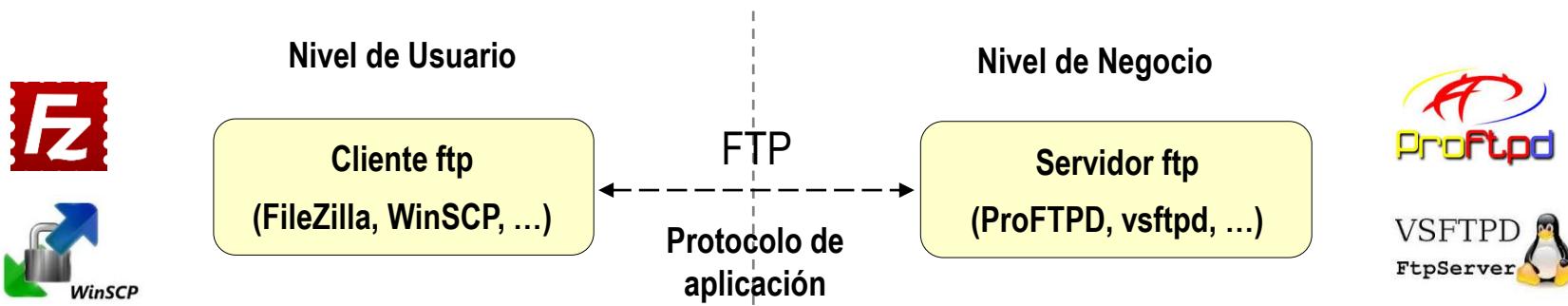
Modelo Cliente/Servidor

Arquitectura C/S

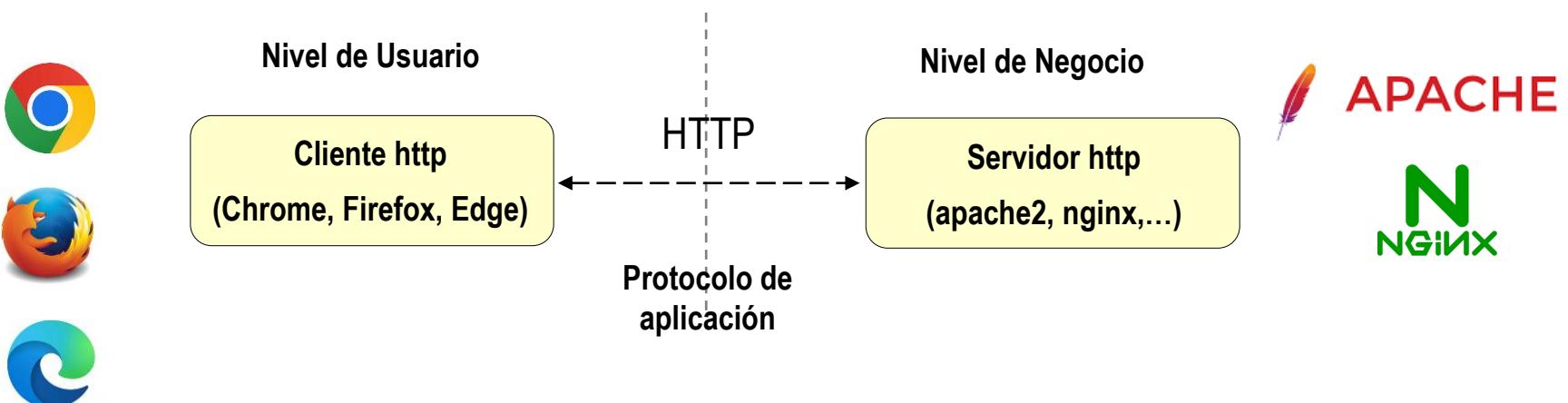


Modelo Cliente/Servidor

Arquitectura C/S de 2-niveles del servicio FTP

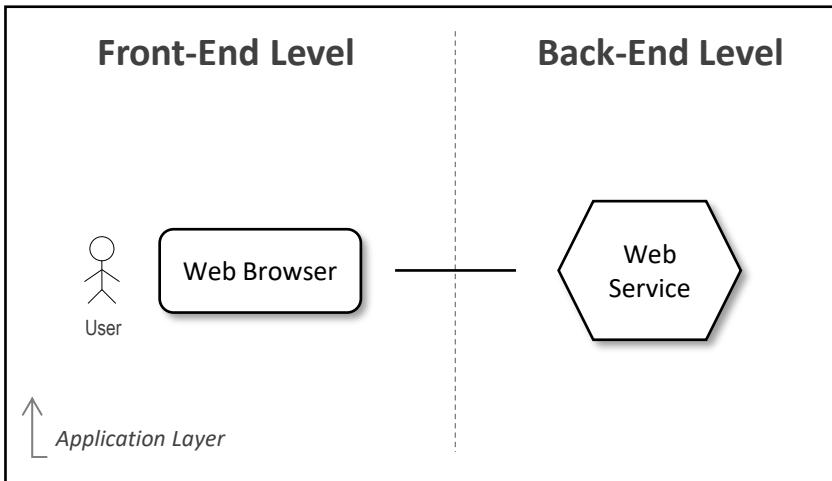


Arquitectura C/S de 2-niveles del servicio HTTP



Modelo Cliente/Servidor

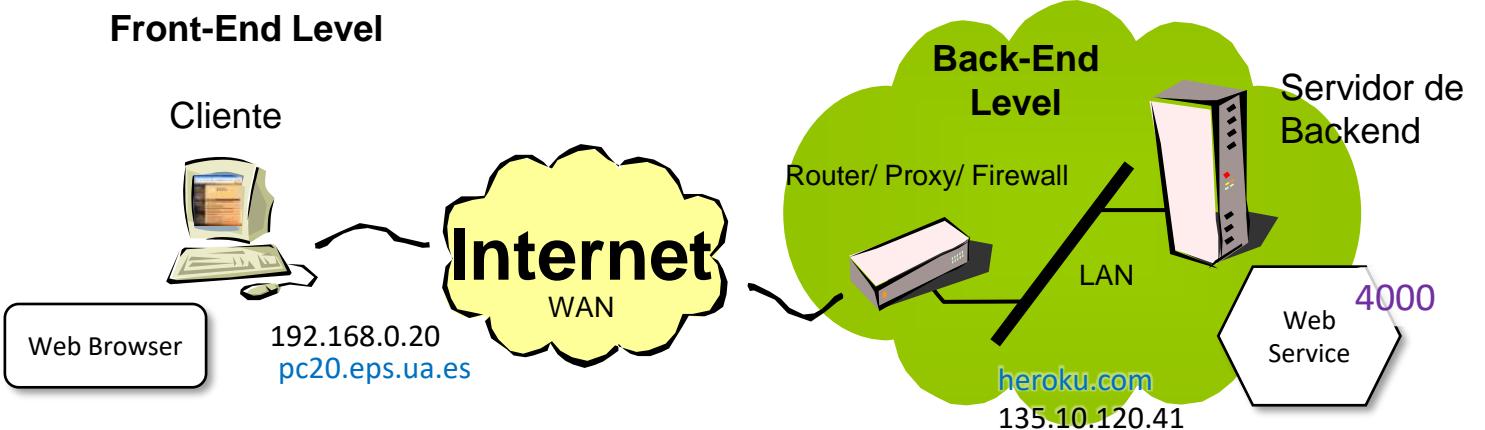
Diagrama de la Arquitectura Distribuida de una Aplicación Web



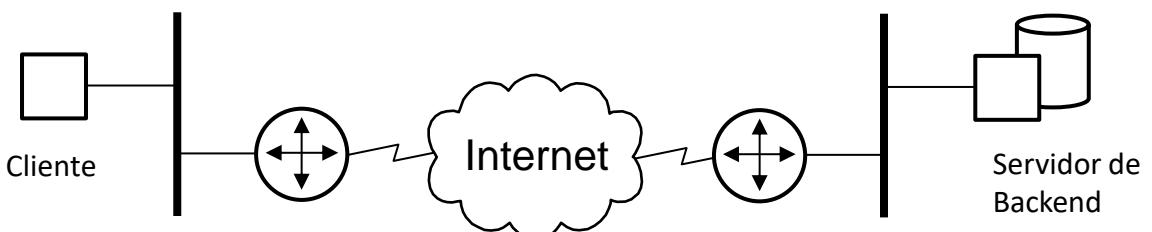
Modelo Cliente/Servidor

Possible Escenario de Despliegue para la Aplicación Web

escenario de despliegue



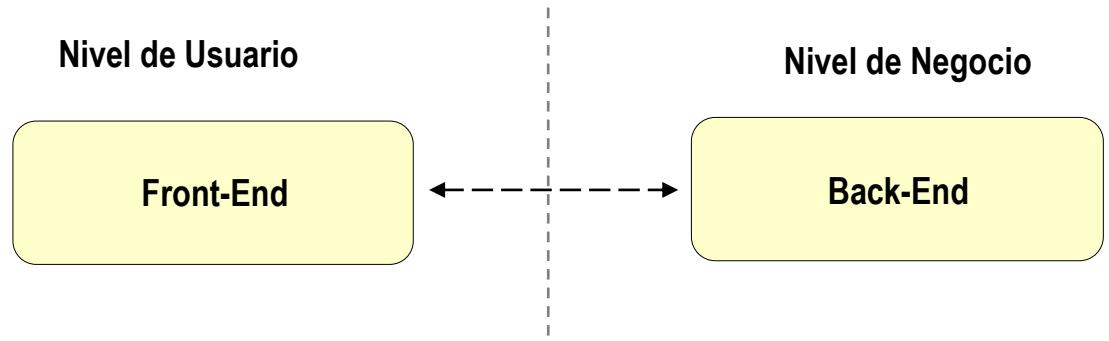
arquitectura física de despliegue



Arquitectura de N- Niveles

Arquitectura de N-Niveles

Arquitectura C/S de 2-niveles



Arquitectura de N-Niveles

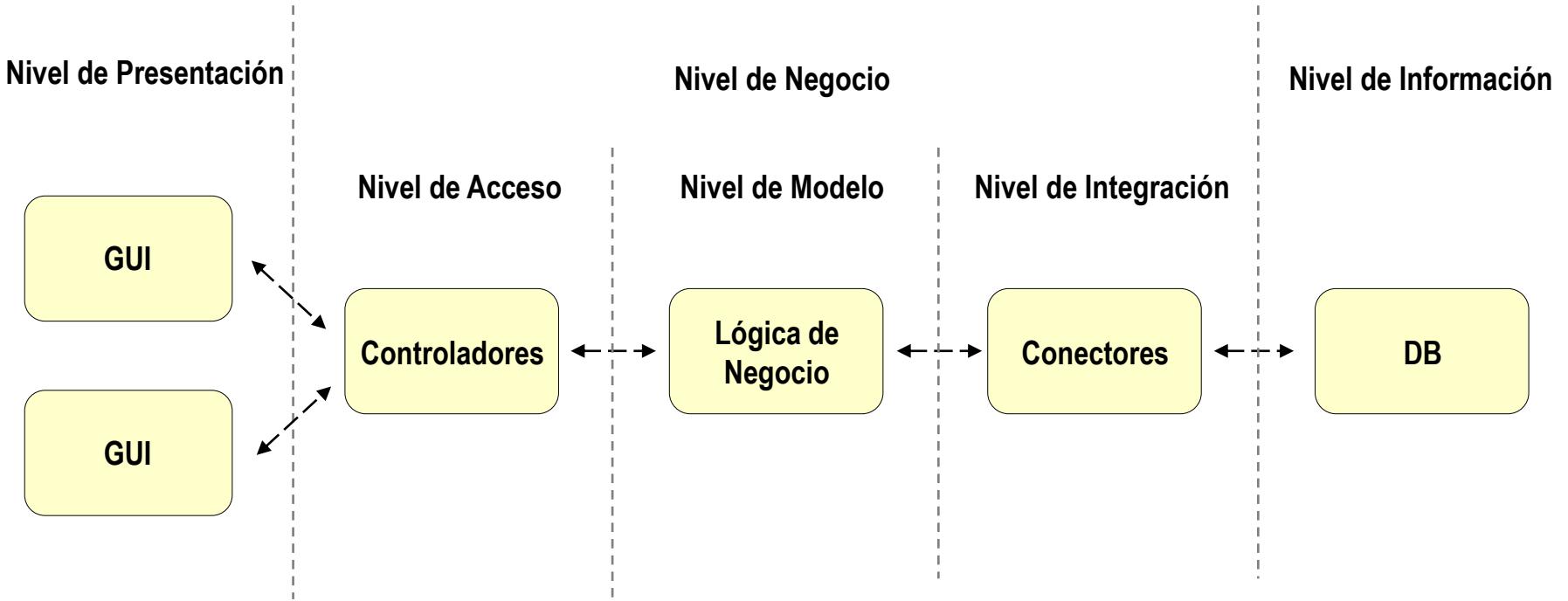
Arquitectura C/S de 3-niveles



Arquitectura C/S de 3-niveles empresarial

Arquitectura de N-Niveles

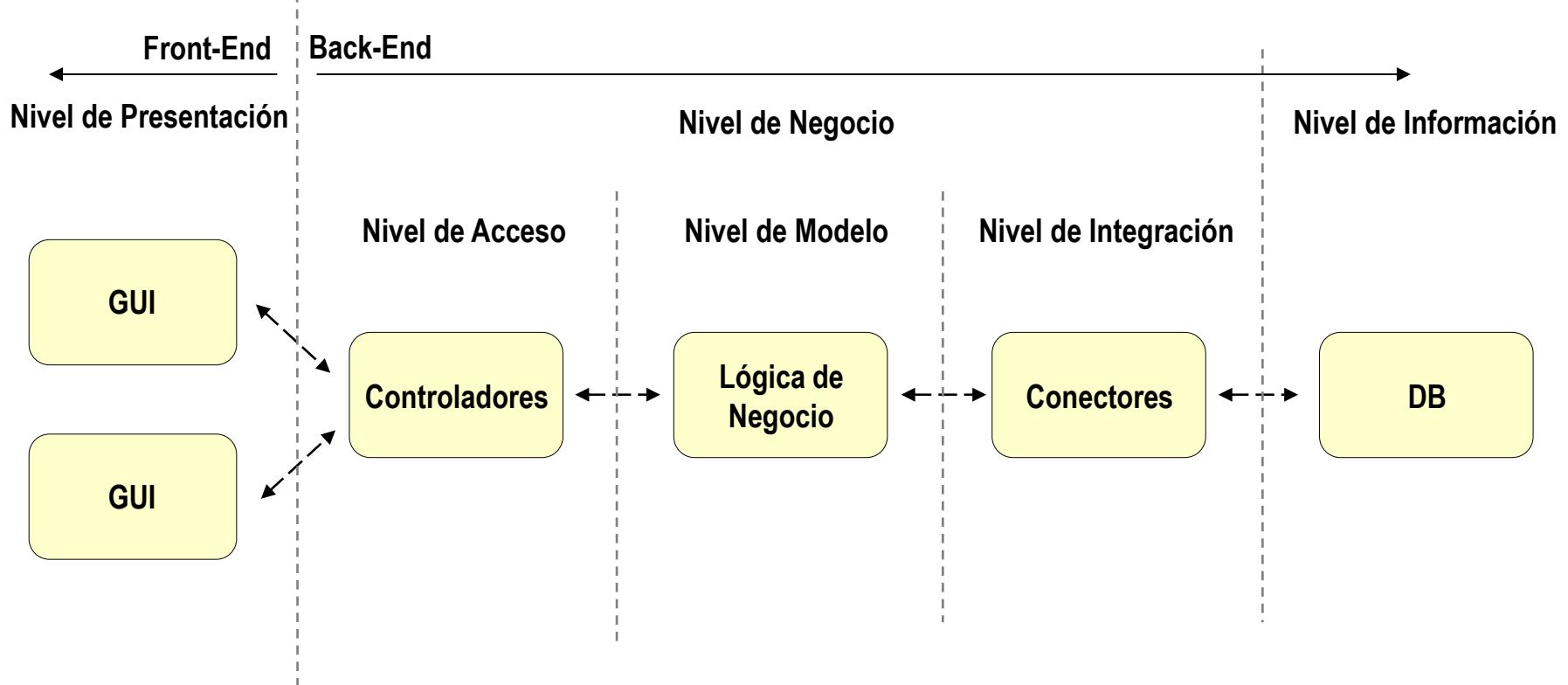
Arquitectura C/S de n-niveles



Arquitectura C/S de n-niveles empresarial

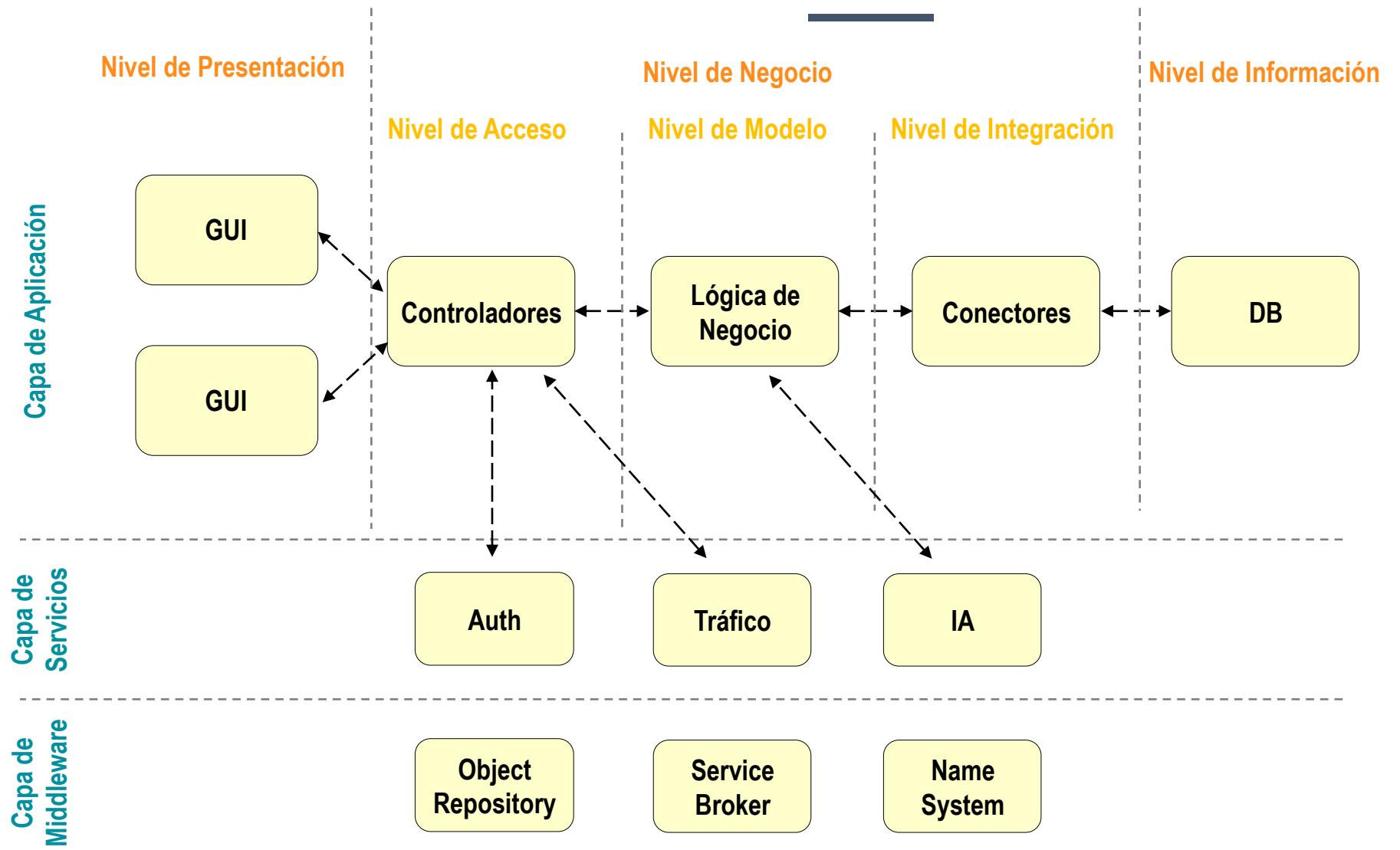
Arquitectura de N-Niveles

Arquitectura C/S de n-niveles



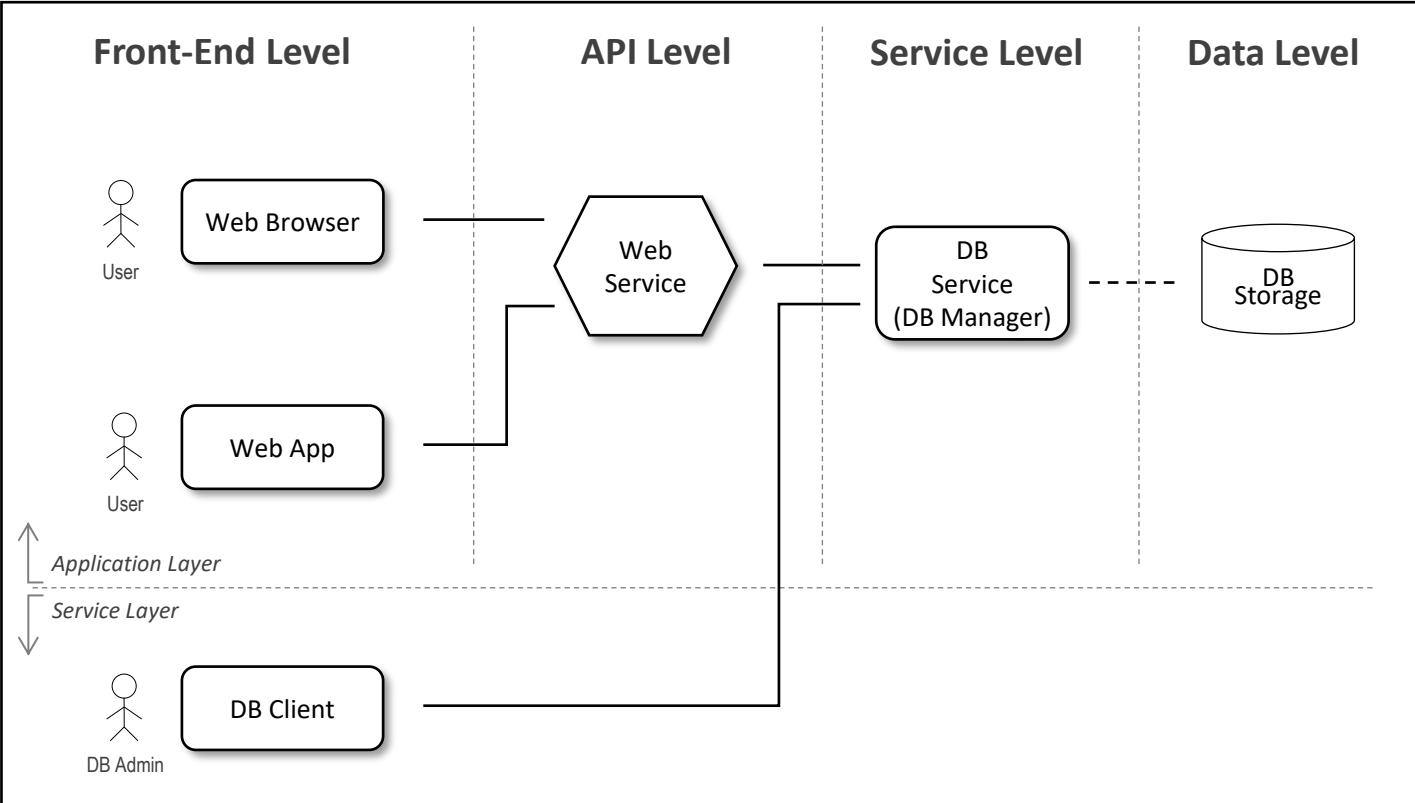
Arquitectura de N-Niveles

Arquitectura C/S de n-niveles



Arquitectura de N-Niveles

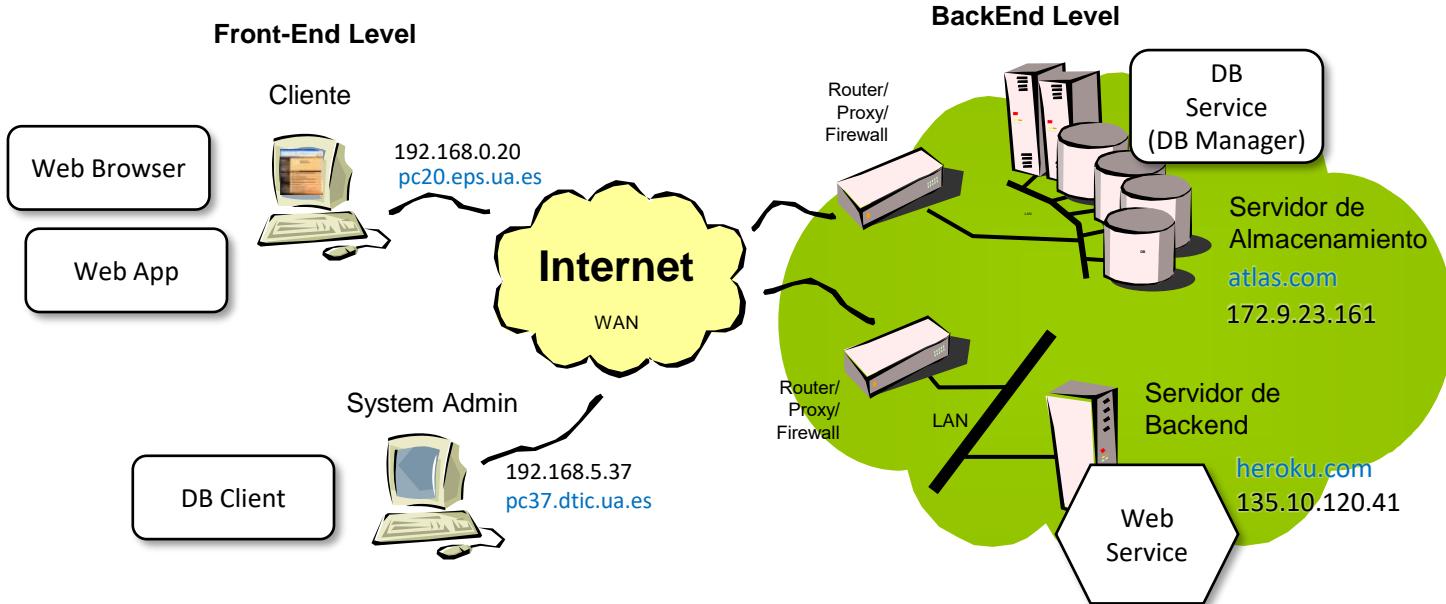
Diagrama de la Arquitectura Distribuida de una Aplicación Web



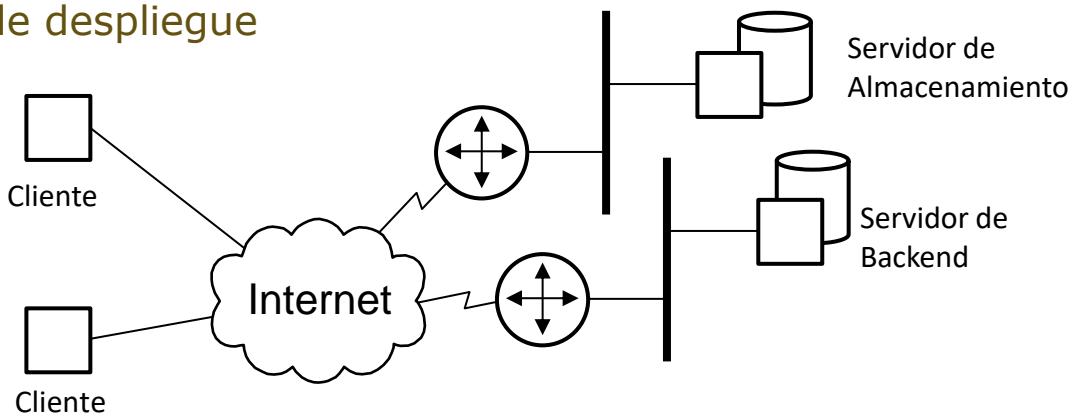
Arquitectura de N-Niveles

Possible Escenario de Despliegue para la Aplicación Web

escenario de despliegue



arquitectura física de despliegue

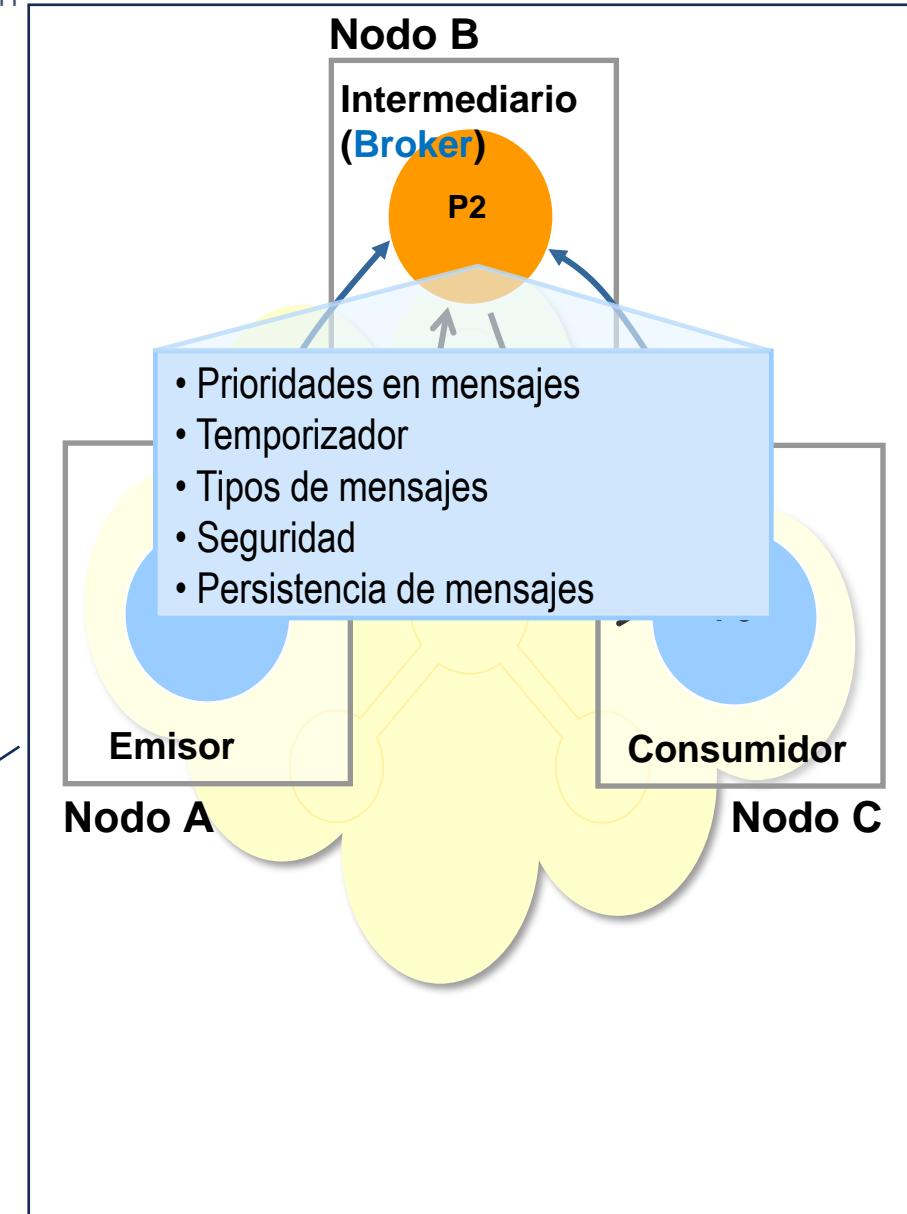


Middleware orientado a mensajes

Middleware orientado a mensajes

Introducción

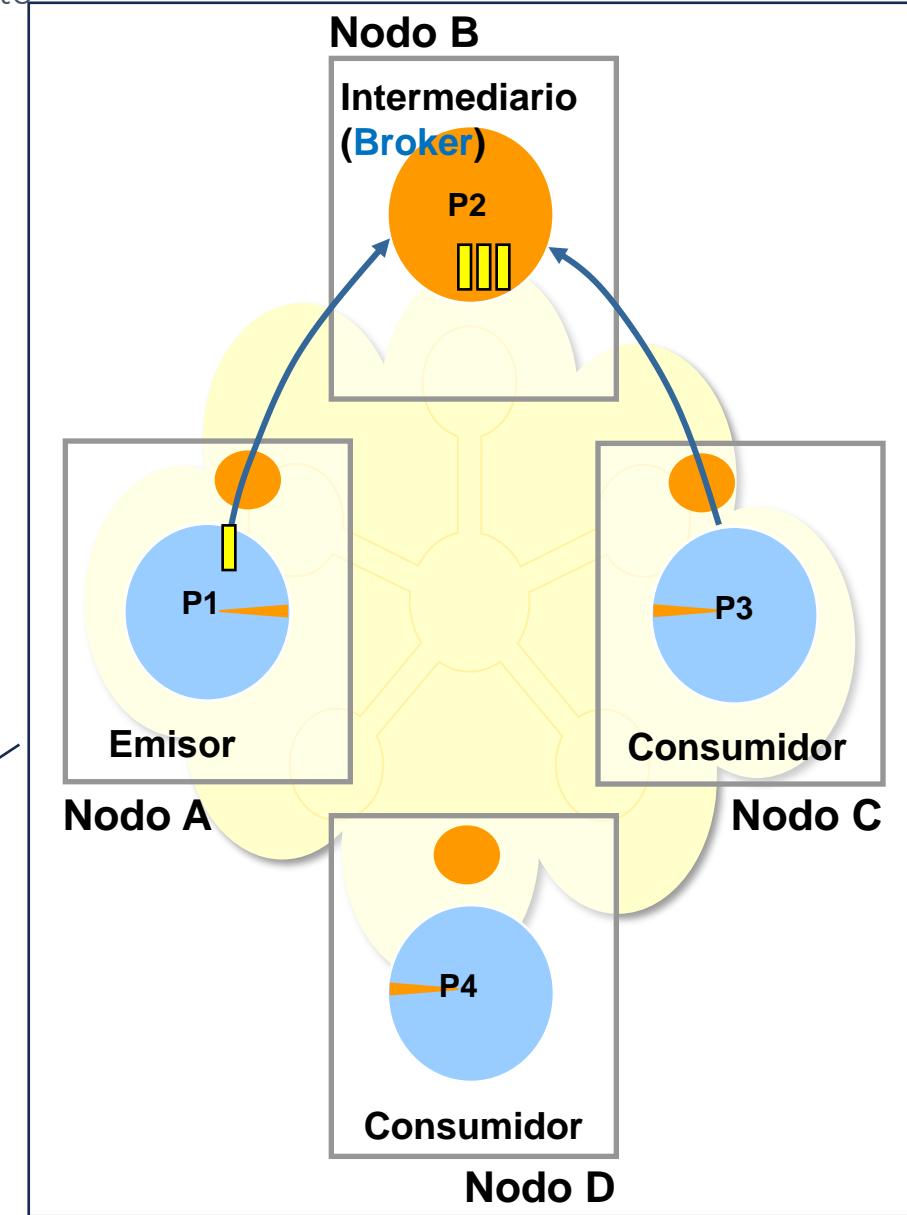
- MOM (Middleware Orientado a Mensajes)
 - Evolución del paso de mensajes
- Desacoplamiento
- Sistemas asíncronos
- Intermediario en el proceso de comunicación
- Herramientas:
 - MQTT, JMS, MSMQ
- Dos tipos:
 - Punto a punto → 1:1
 - Publicación/suscripción → 1:M



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

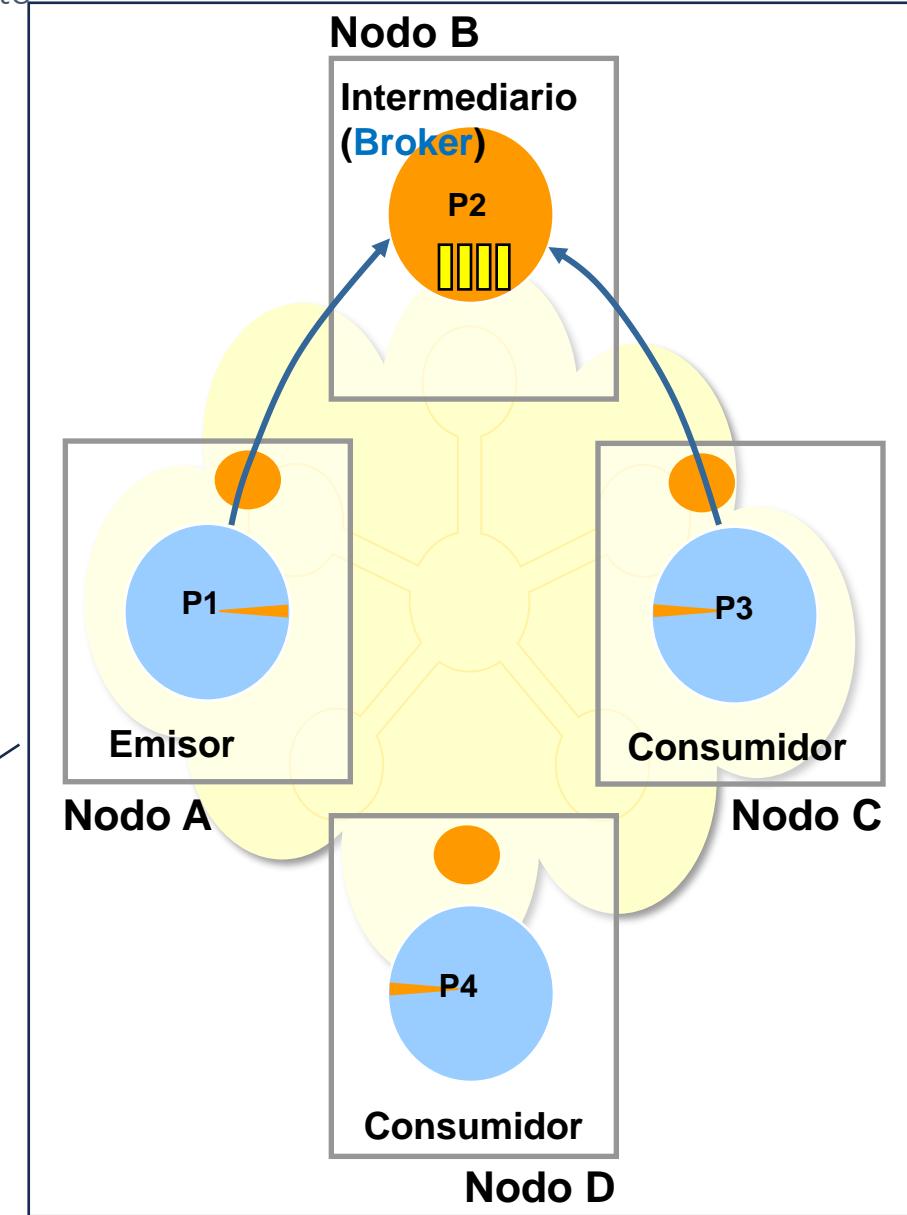
Punto a punto



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

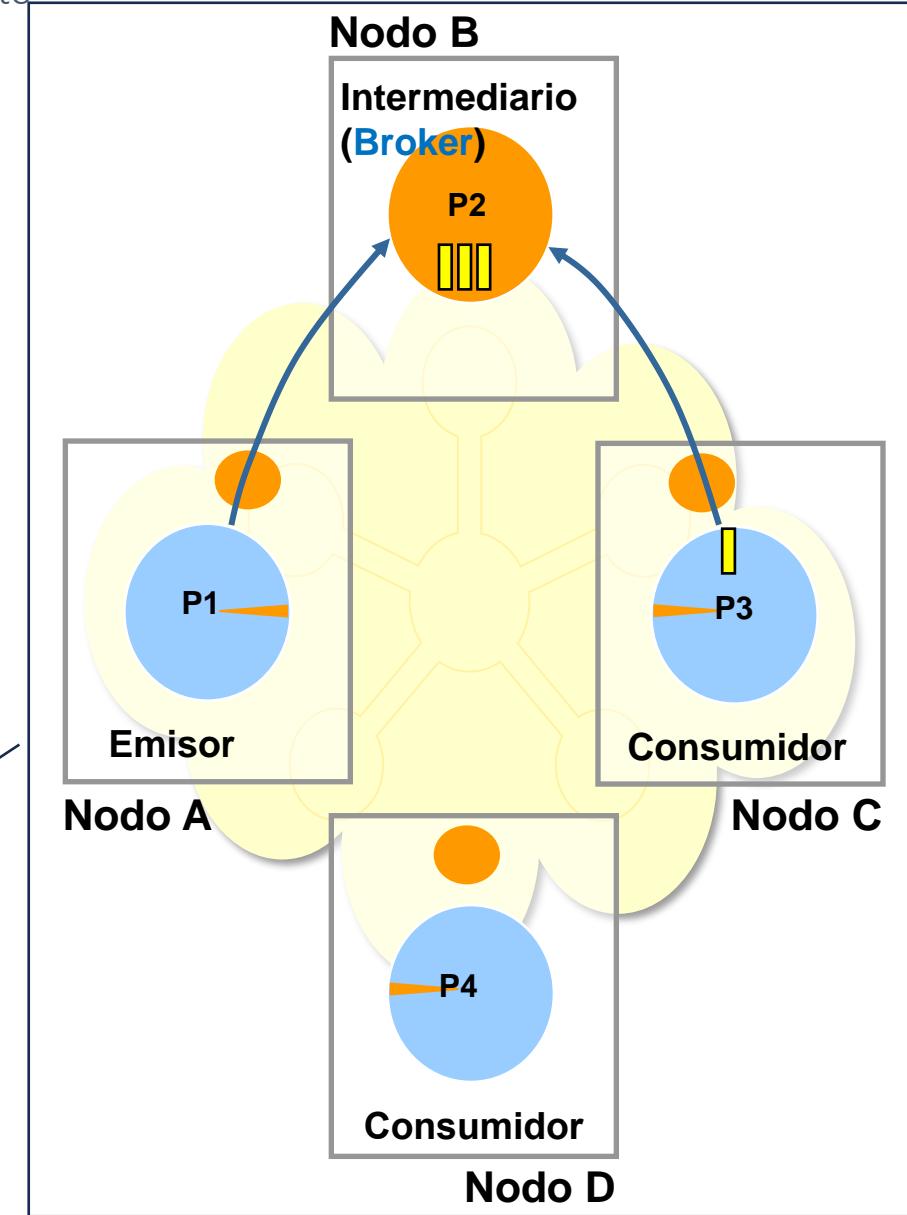
Punto a punto



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

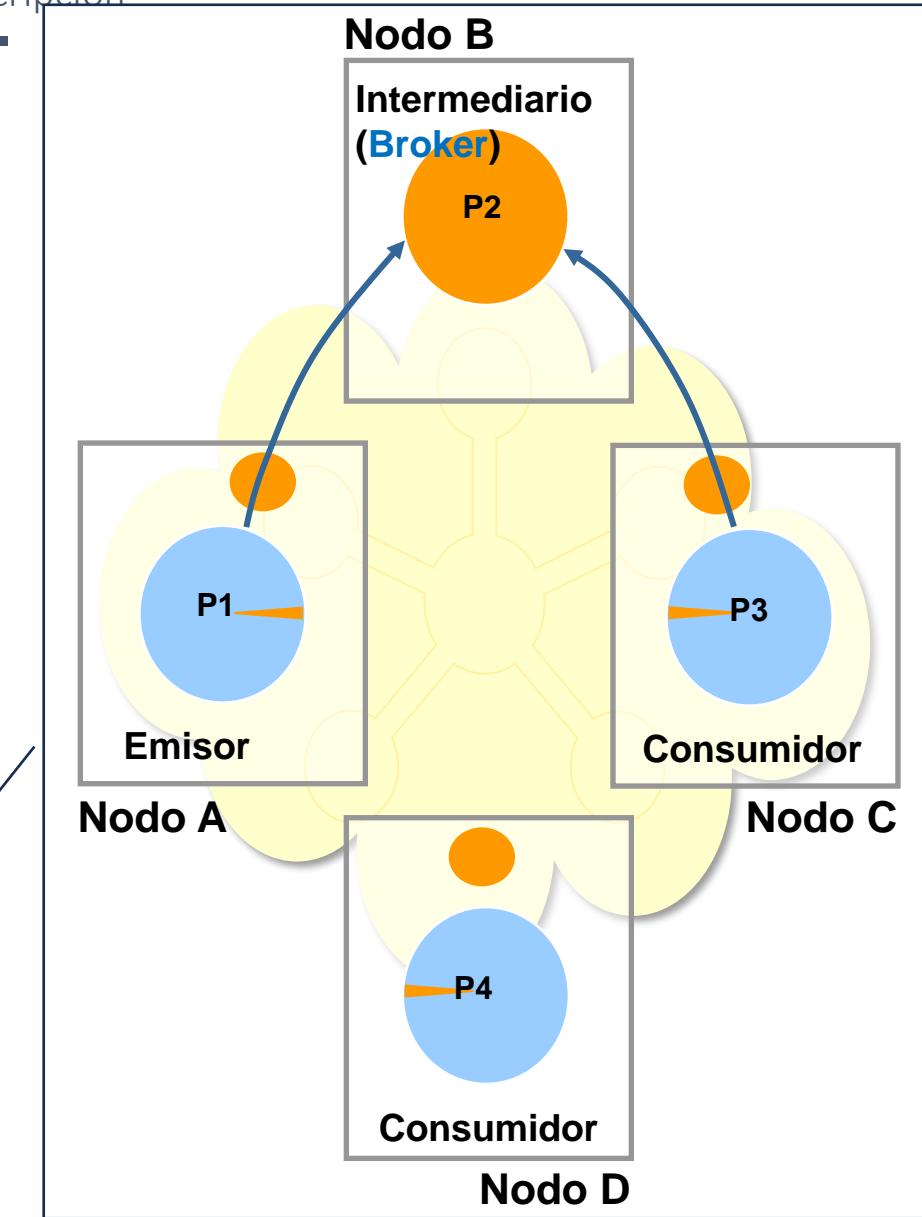
Punto a punto



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

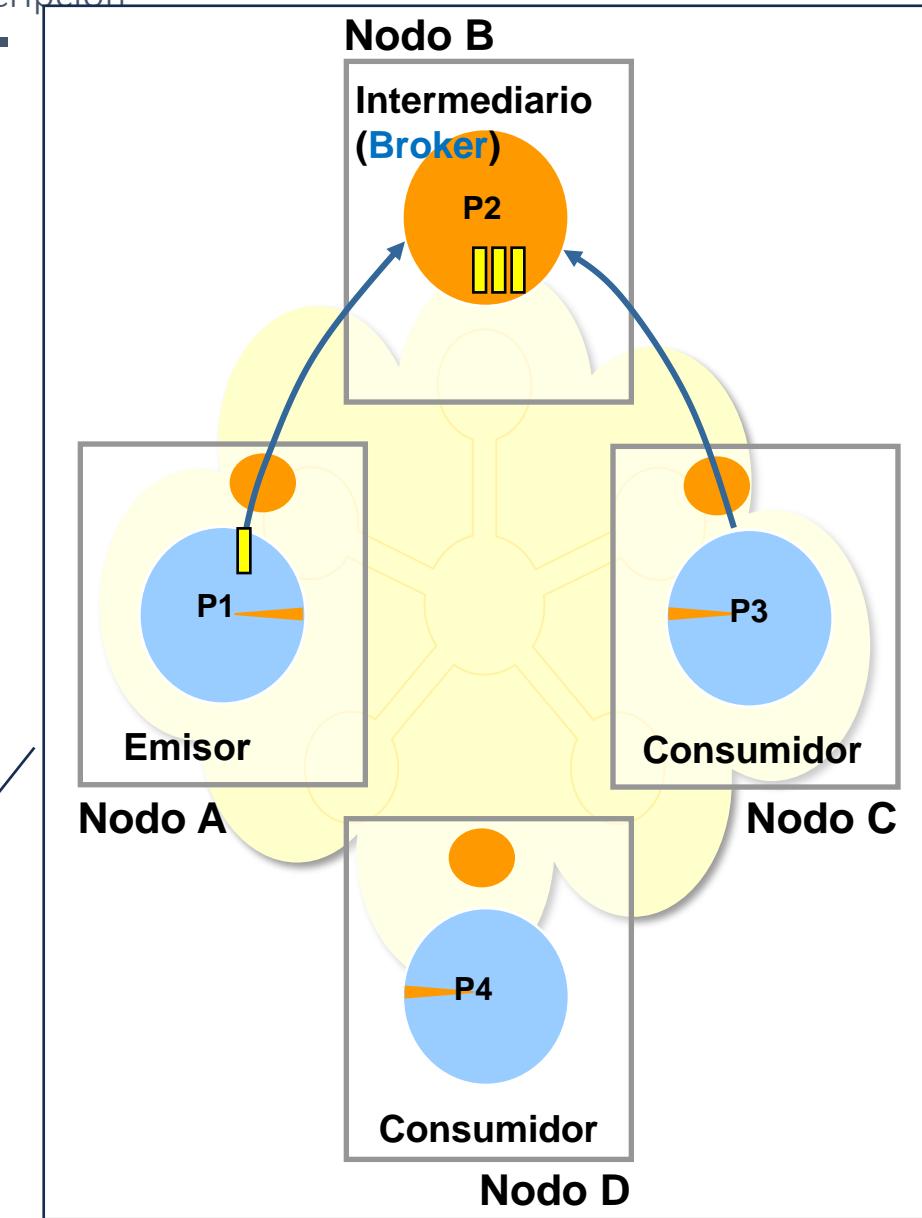
Publicación/suscripción



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

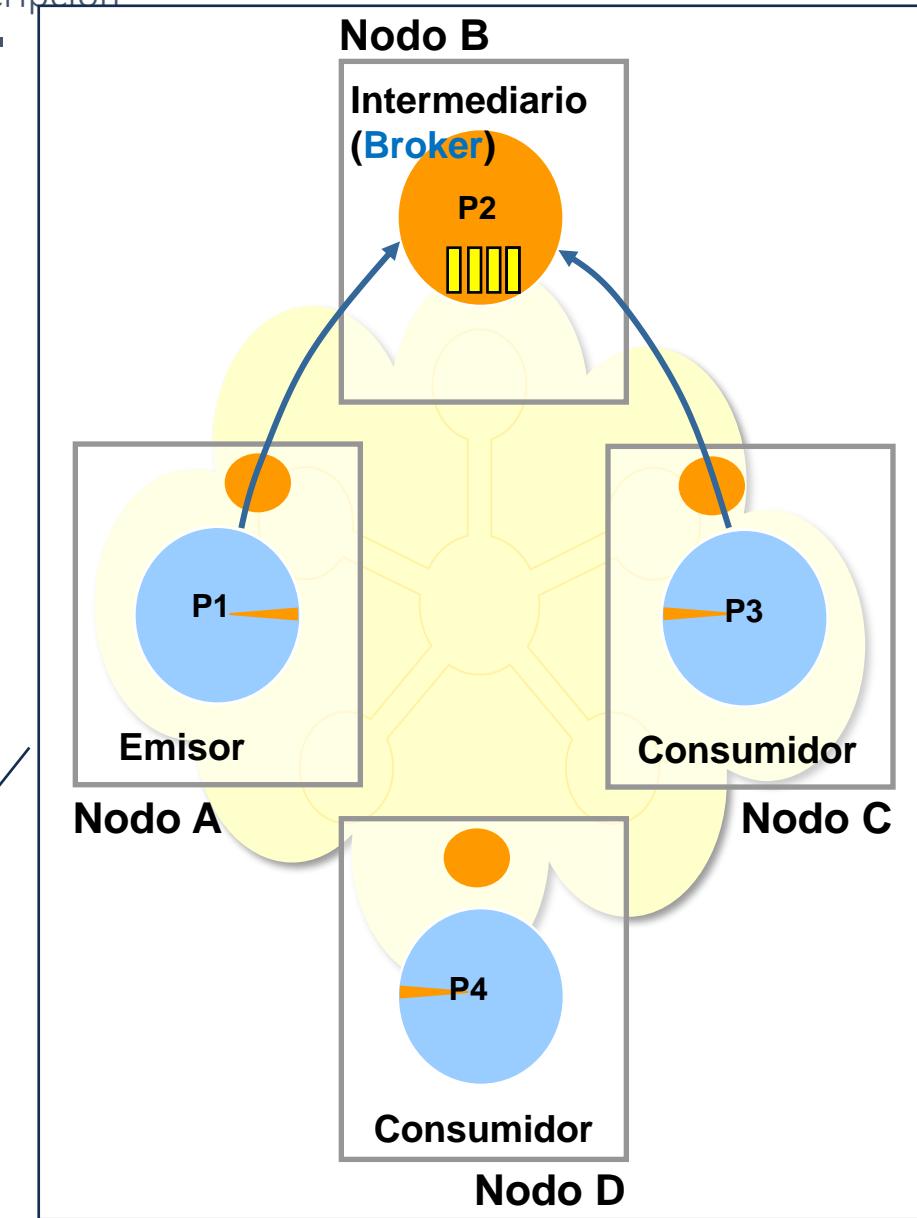
Publicación/suscripción



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

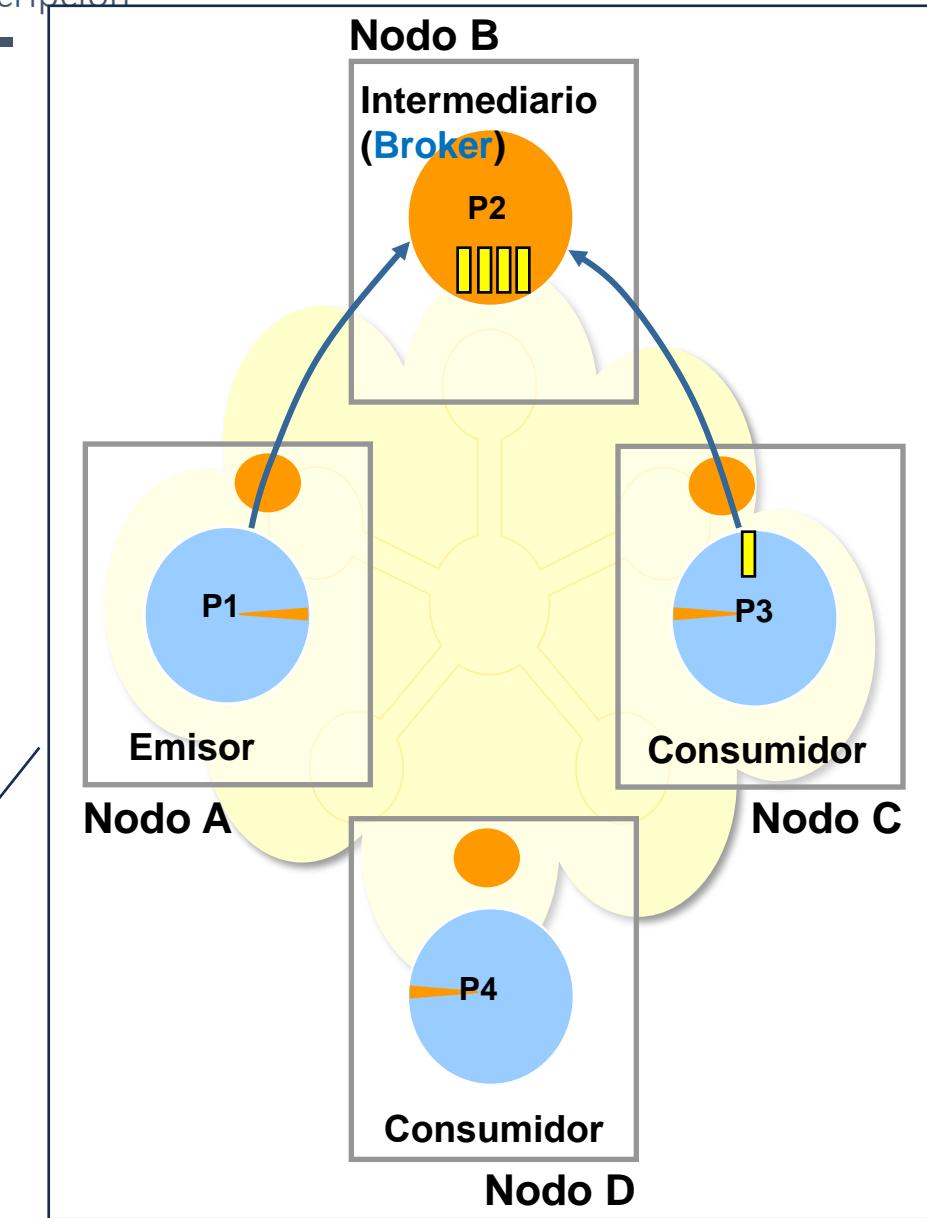
Publicación/suscripción



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

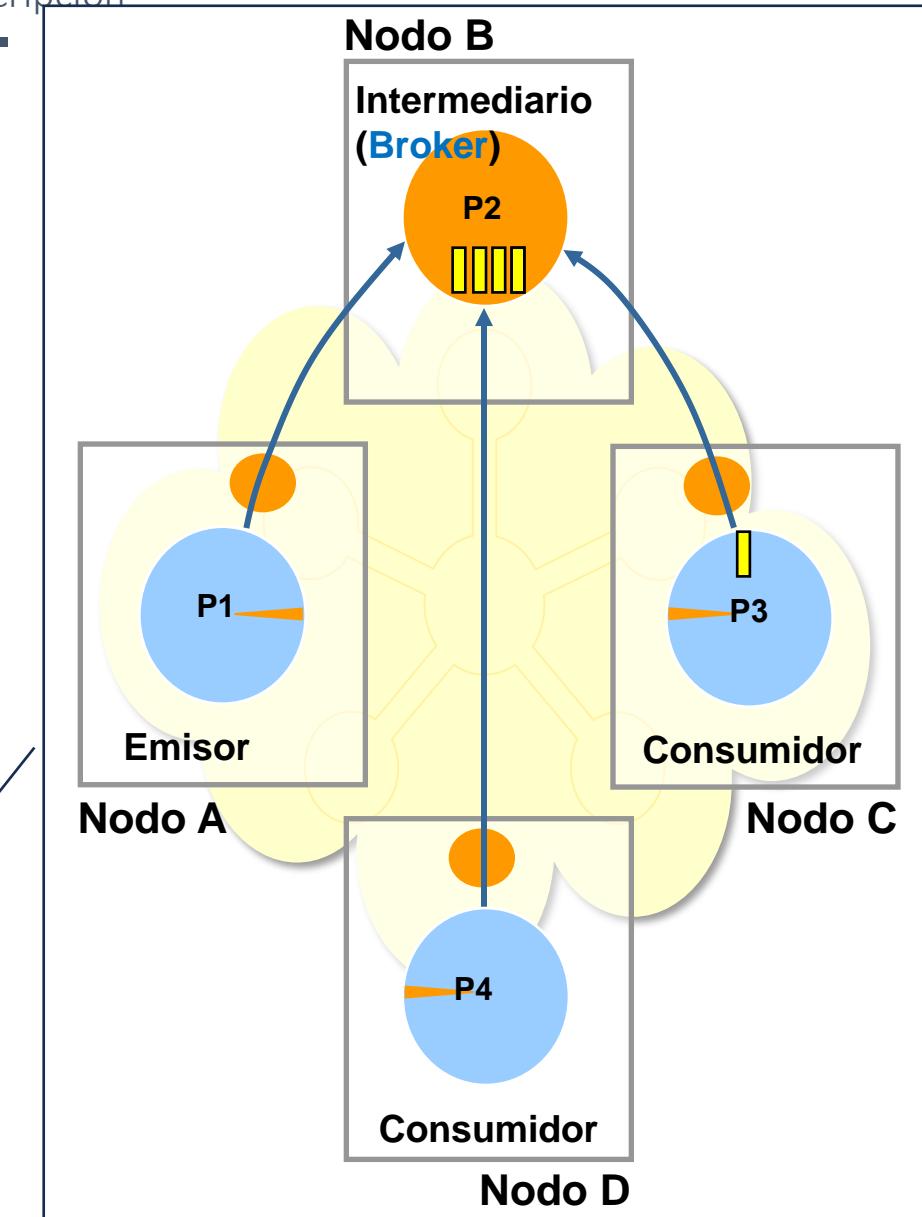
Publicación/suscripción



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

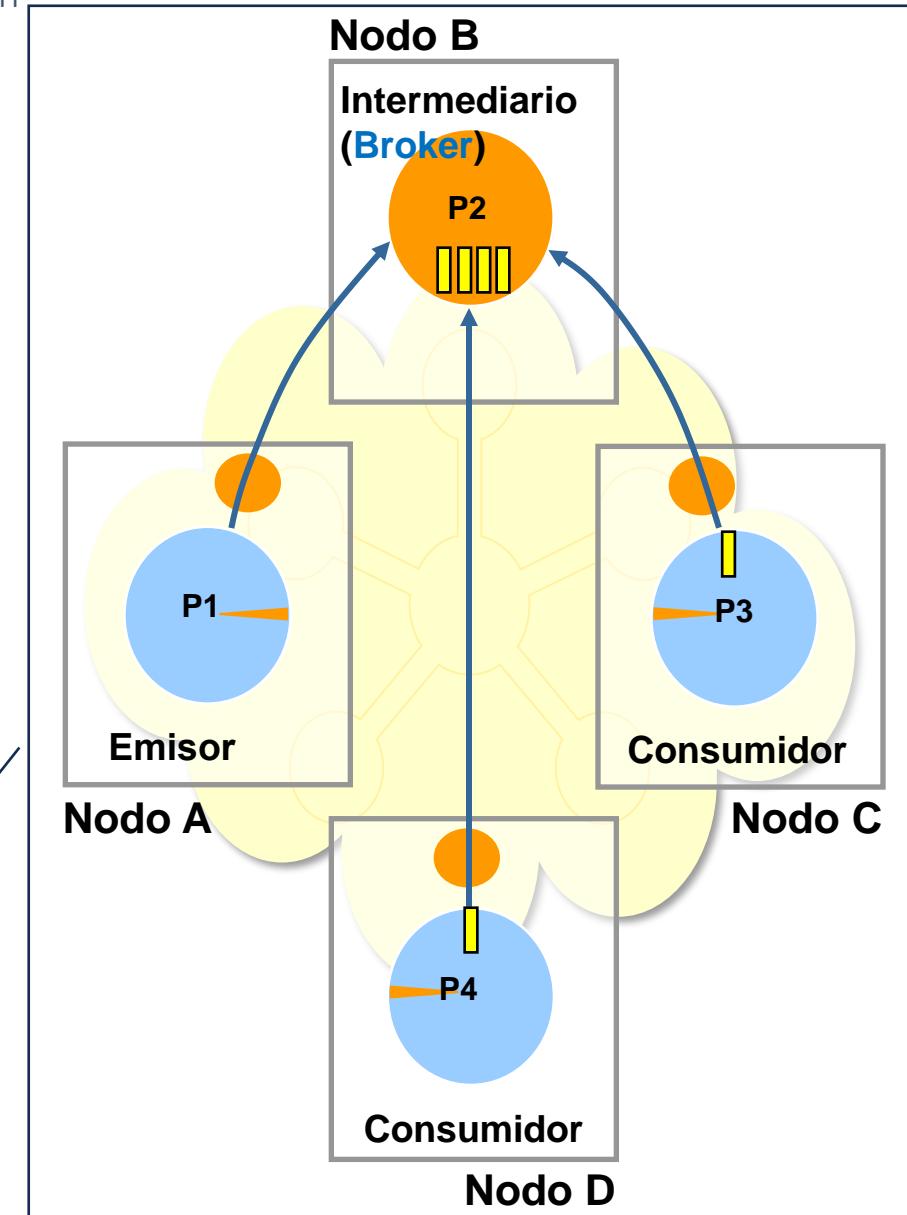
Publicación/suscripción



Middleware orientado a mensajes

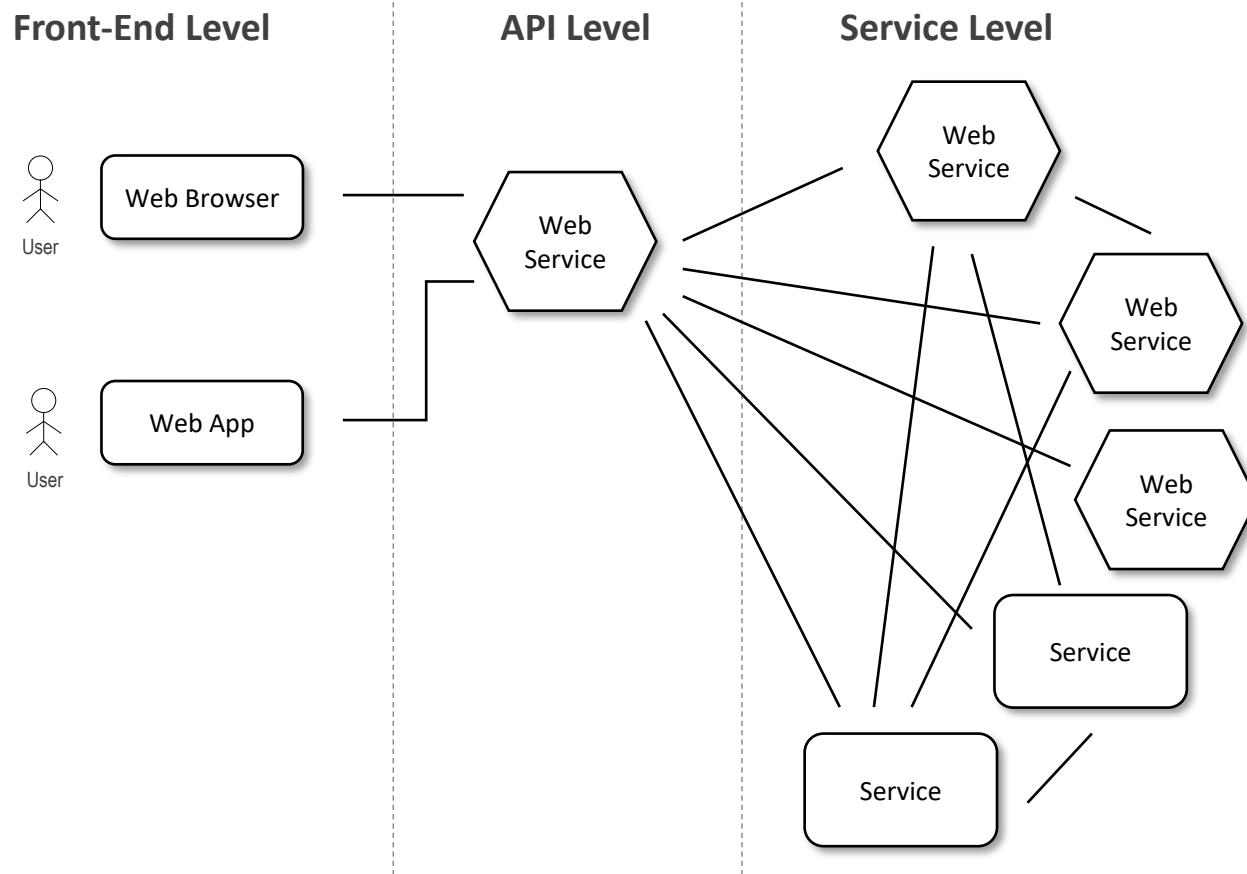
Introducción

- 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



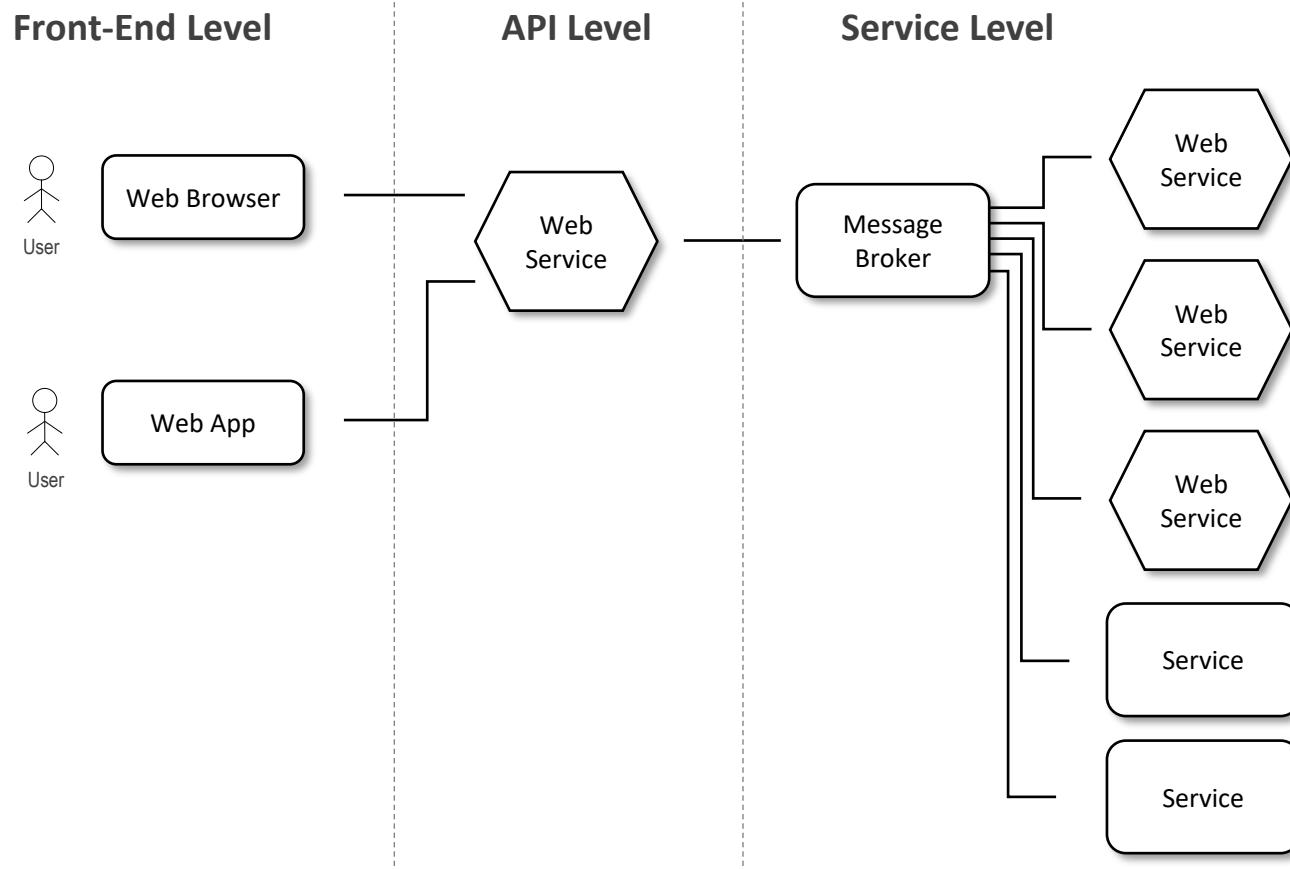
Middleware orientado a mensajes

Arquitectura distribuida



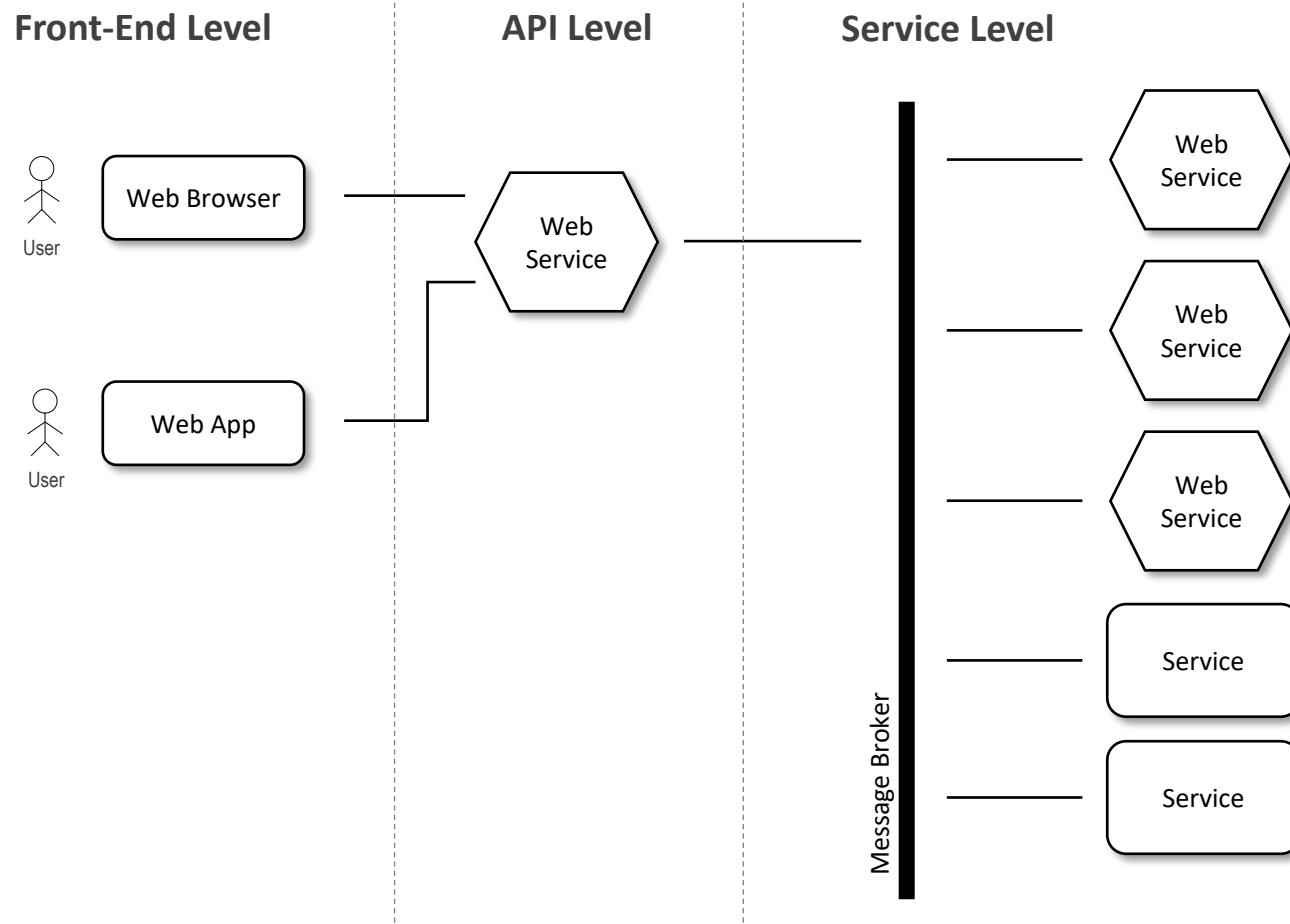
Middleware orientado a mensajes

Arquitectura distribuida



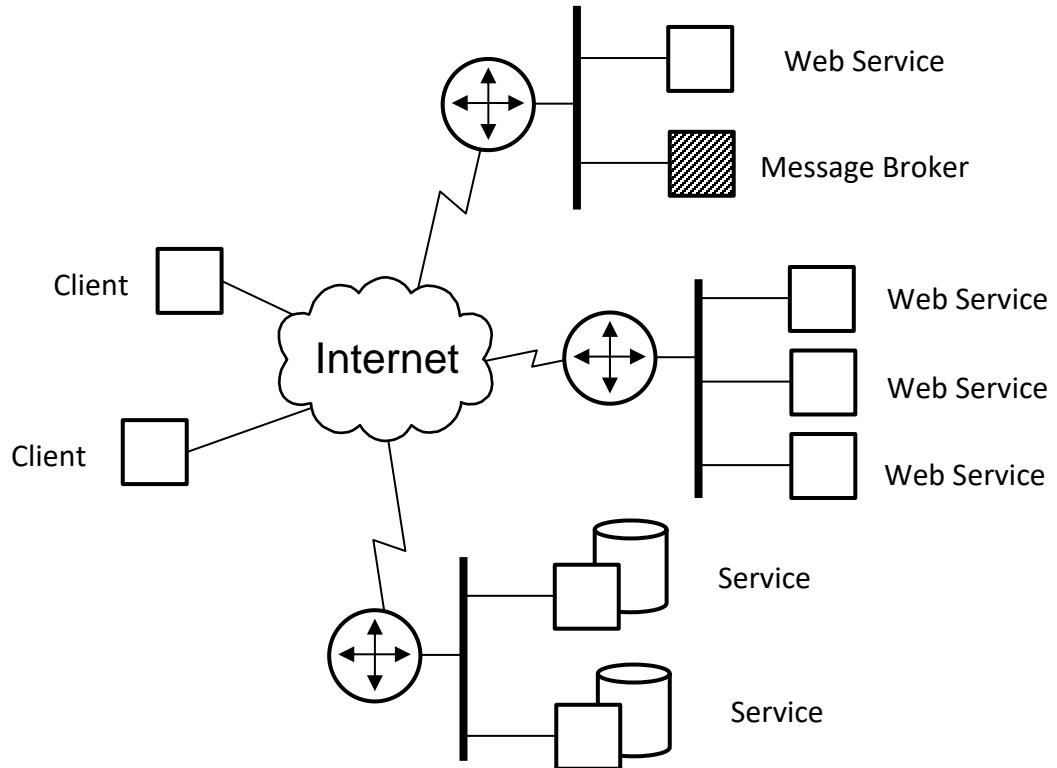
Middleware orientado a mensajes

Arquitectura distribuida



Middleware orientado a mensajes

Arquitectura física



Arquitectura orientada a servicios

Arquitectura orientada a servicios

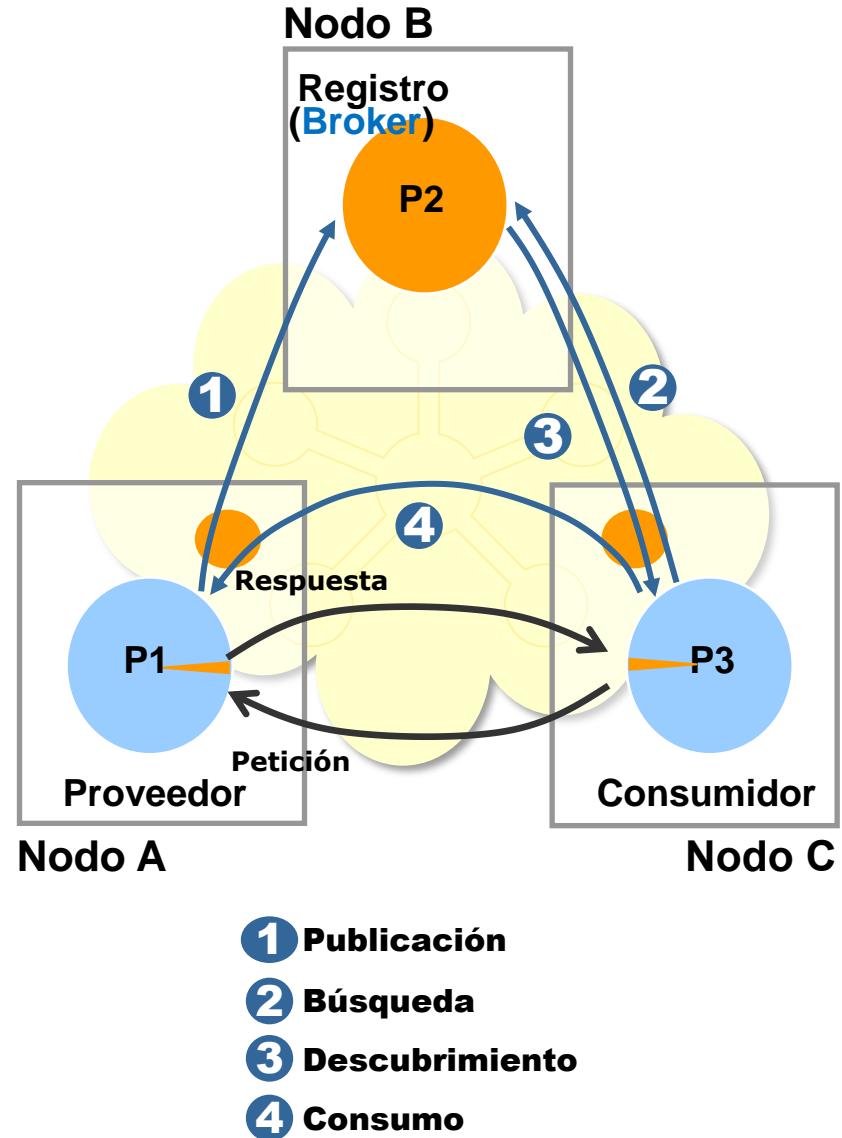
Introducción

- **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



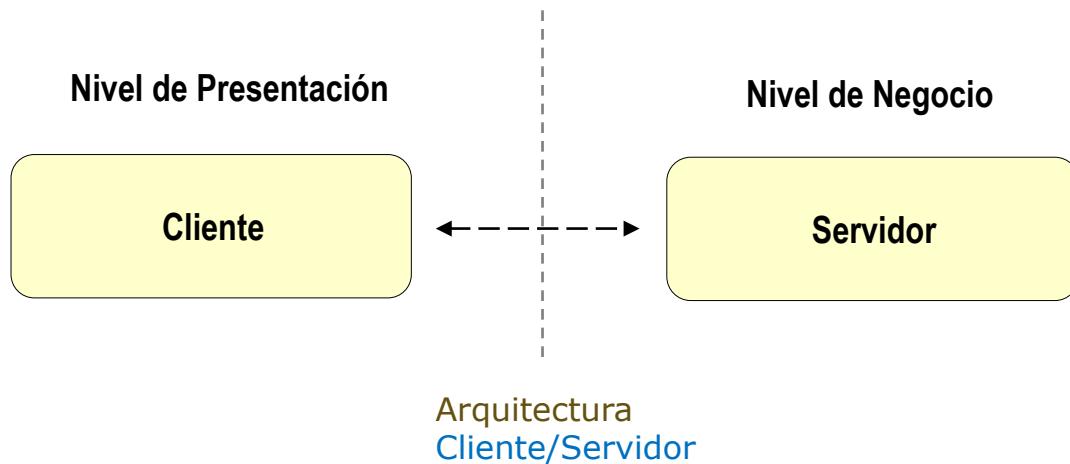
Arquitectura orientada a servicios

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.

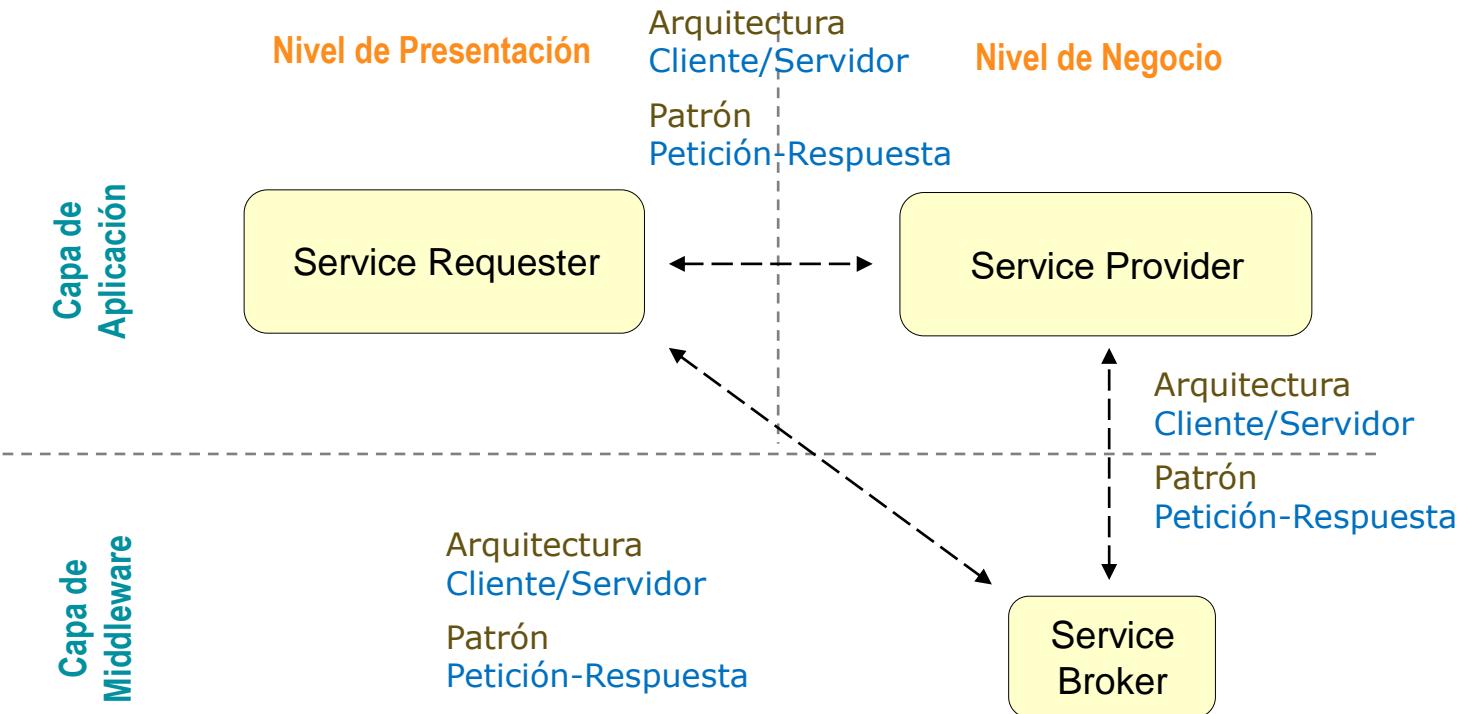
Arquitectura orientada a servicios

Arquitectura SOA por niveles

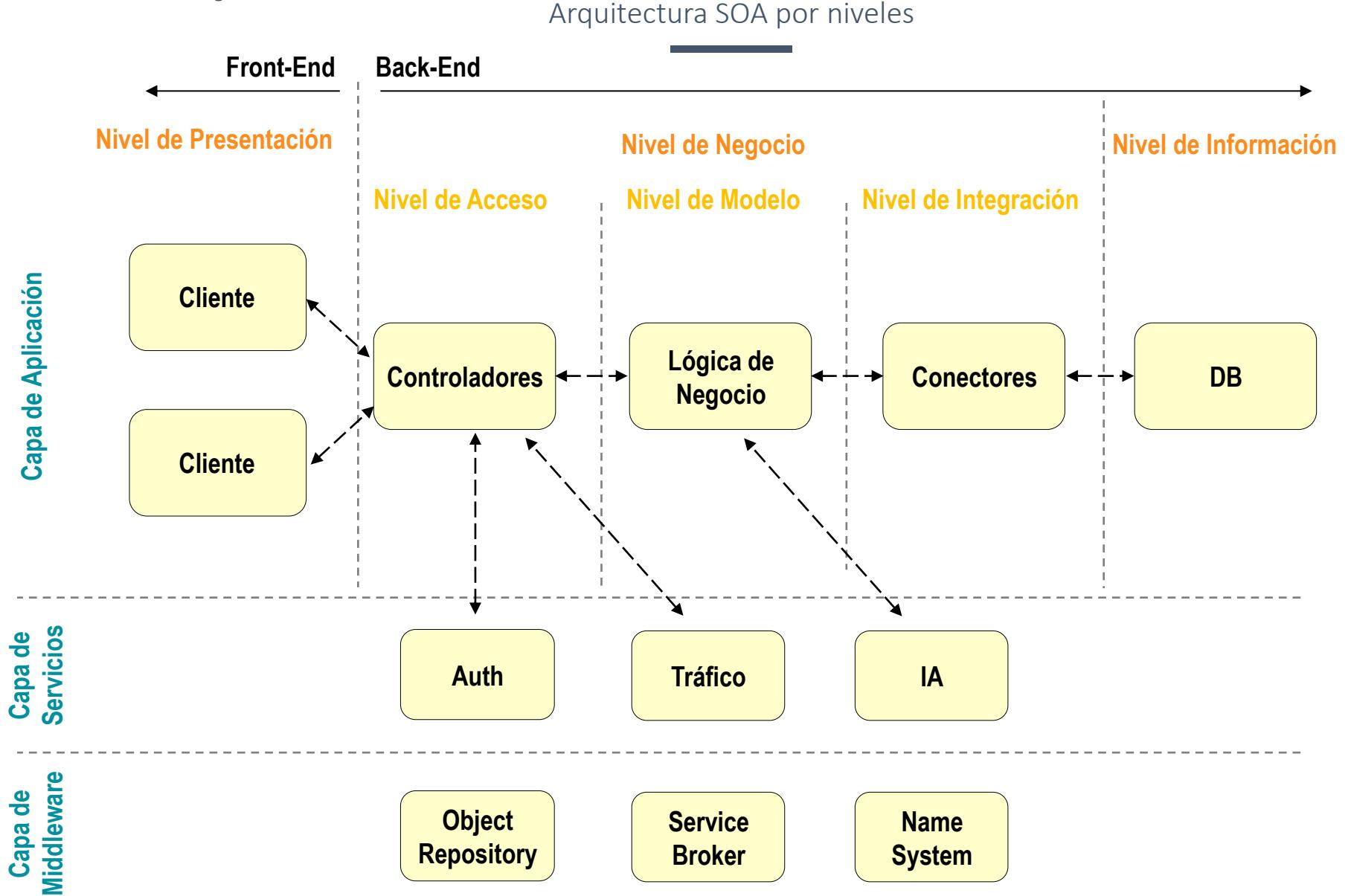


Arquitectura orientada a servicios

Arquitectura SOA por niveles



Arquitectura orientada a servicios



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.

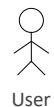
- Uber:

- 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

Front-End Level Back-End Level



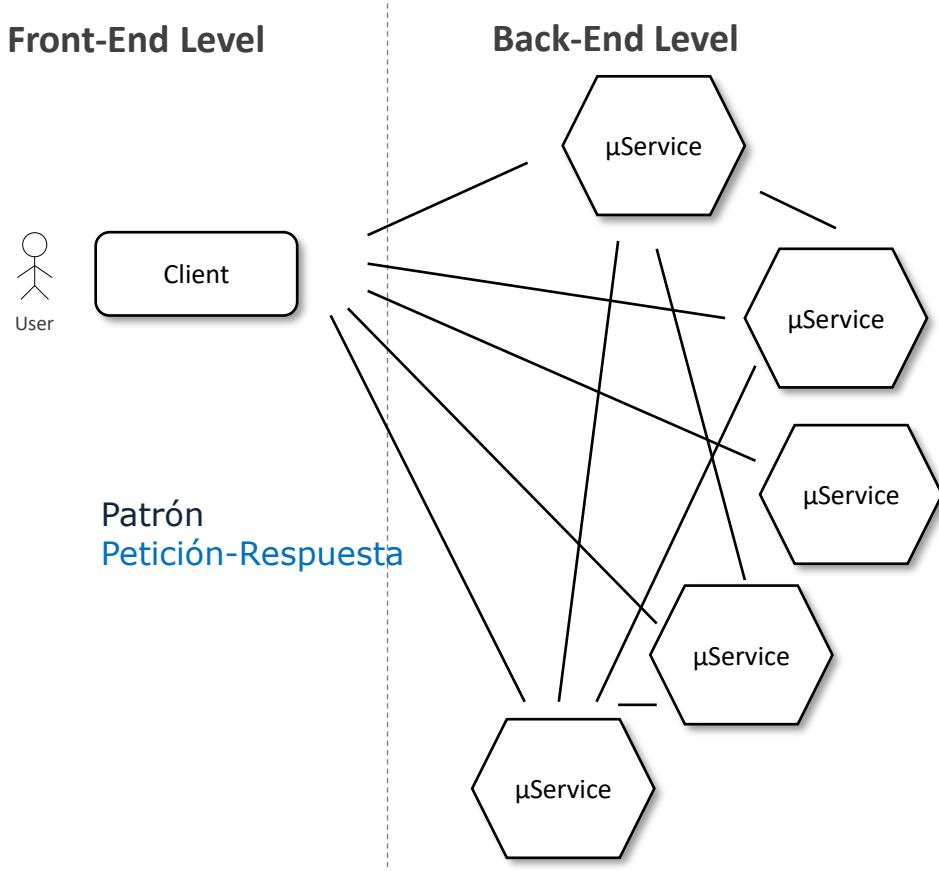
User



Patrón de Comunicación
Petición-Respuesta

Arquitectura de Microservicios

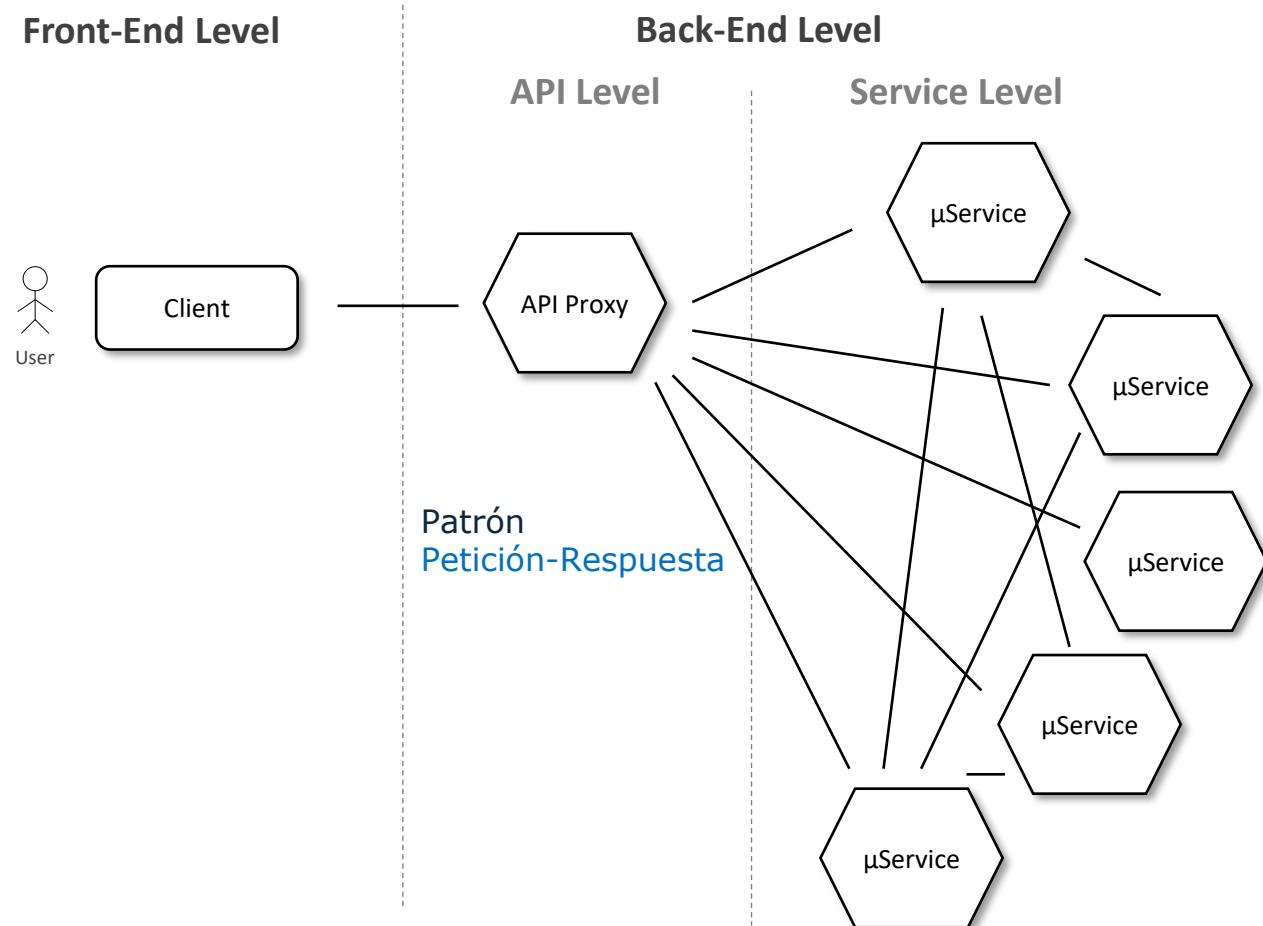
Arquitectura Distribuida basada en Patrón Petición-Respuestas



Arquitectura de Microservicios

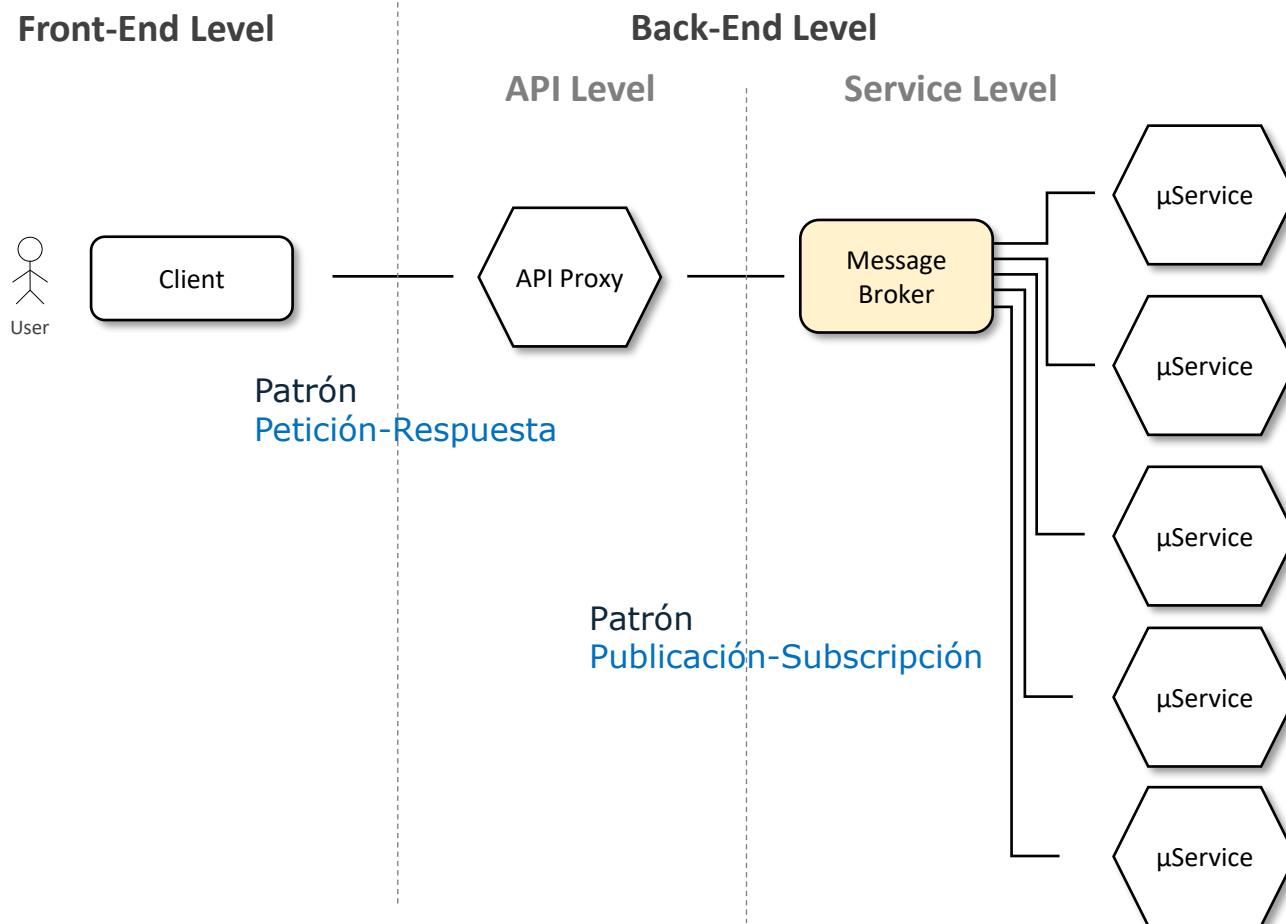
Arquitectura Distribuida basada en µServicios + Patrón API Proxy

Arquitectura distribuida basada en C/S



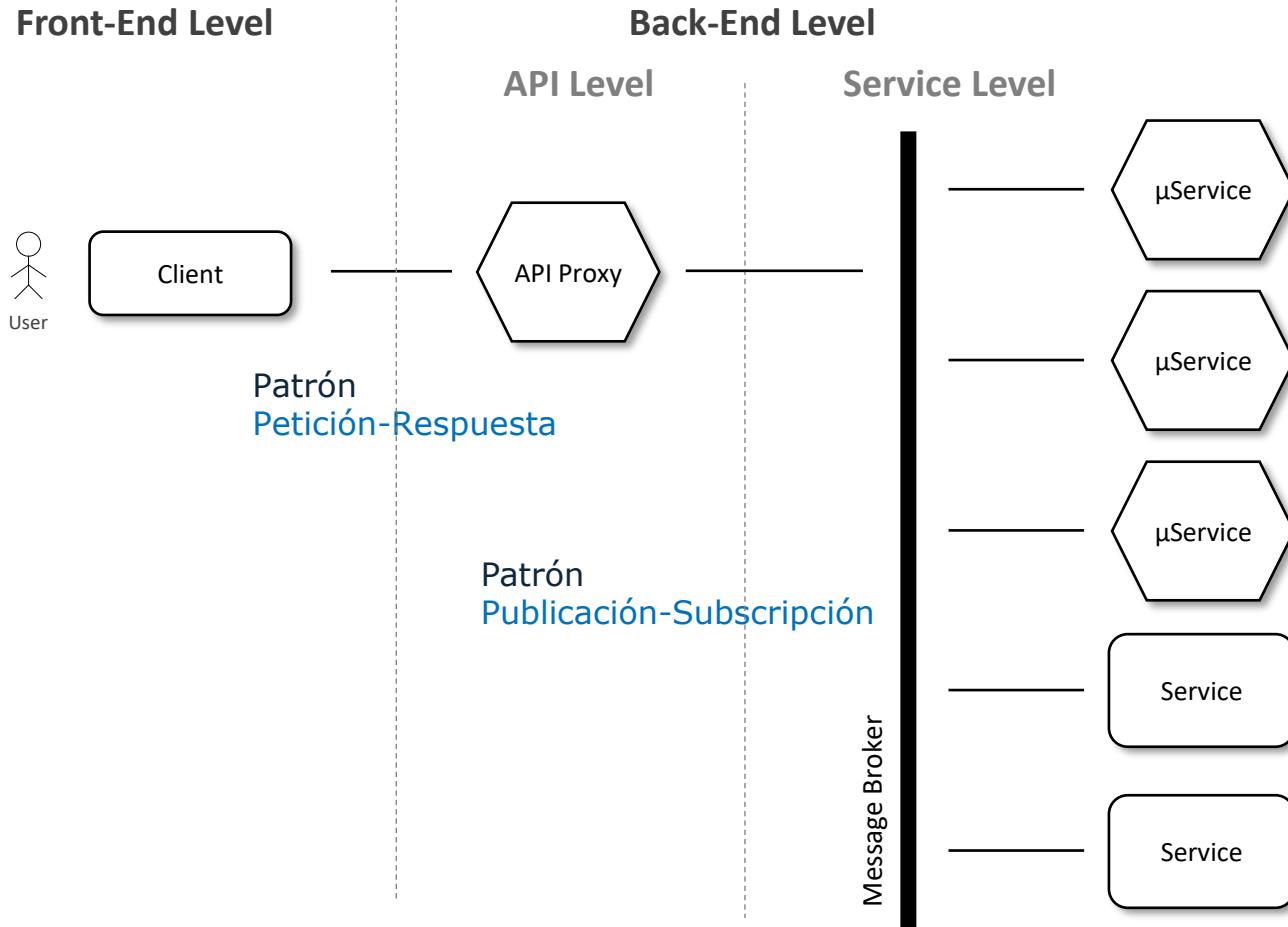
Arquitectura de Microservicios

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

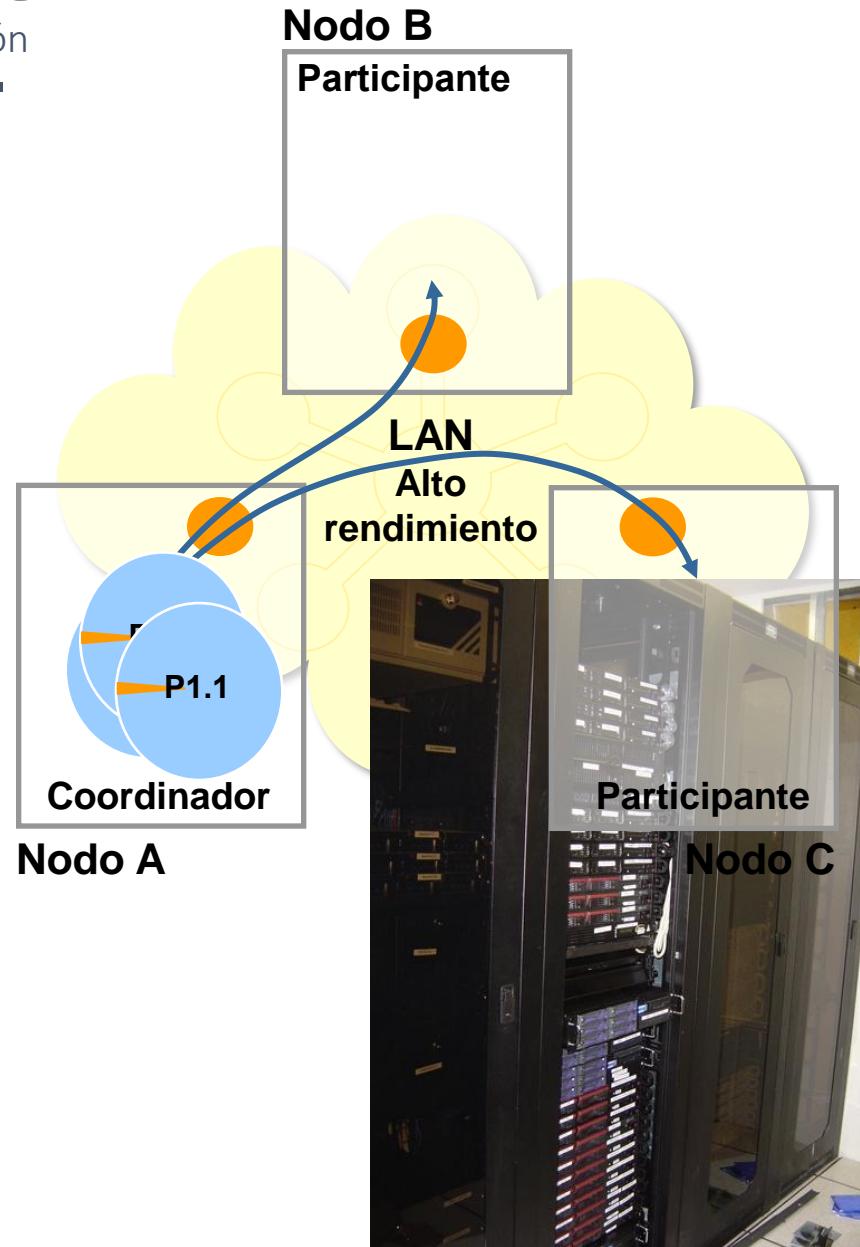
Mejorar la escalabilidad y el rendimiento de los sistemas informáticos

03

Cluster

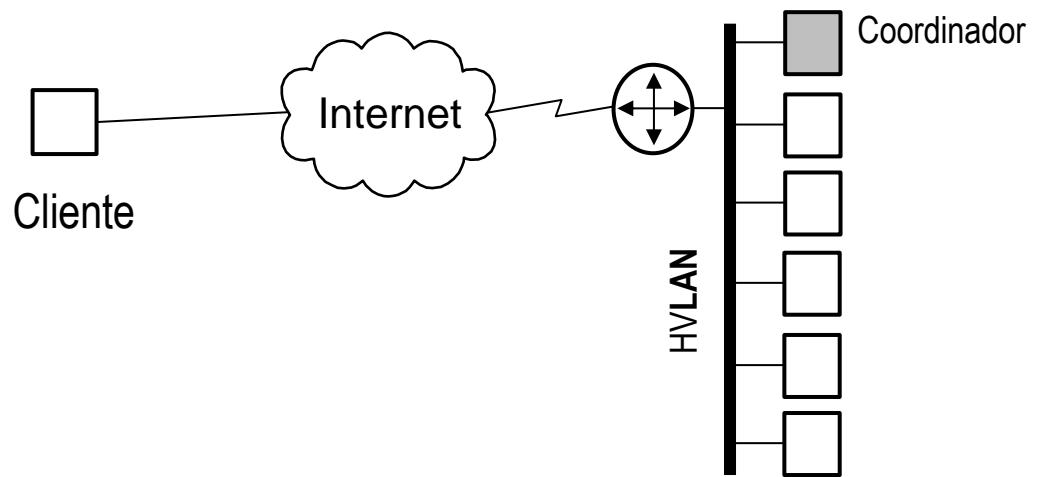
Introducción

- **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

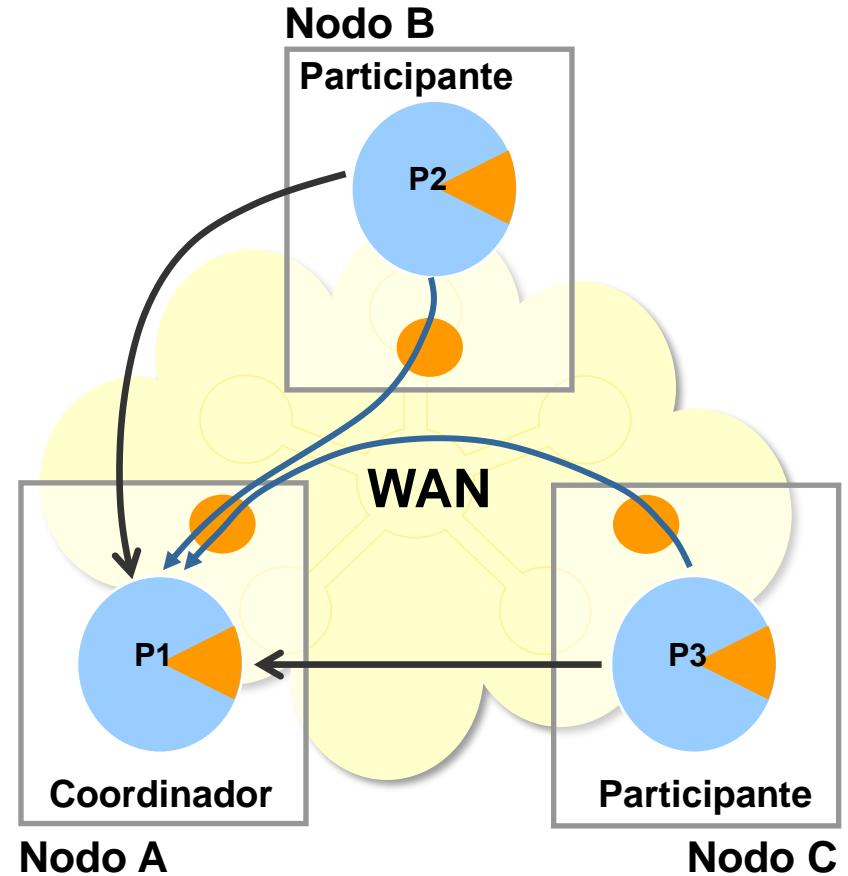
arquitectura física de despliegue



Grid

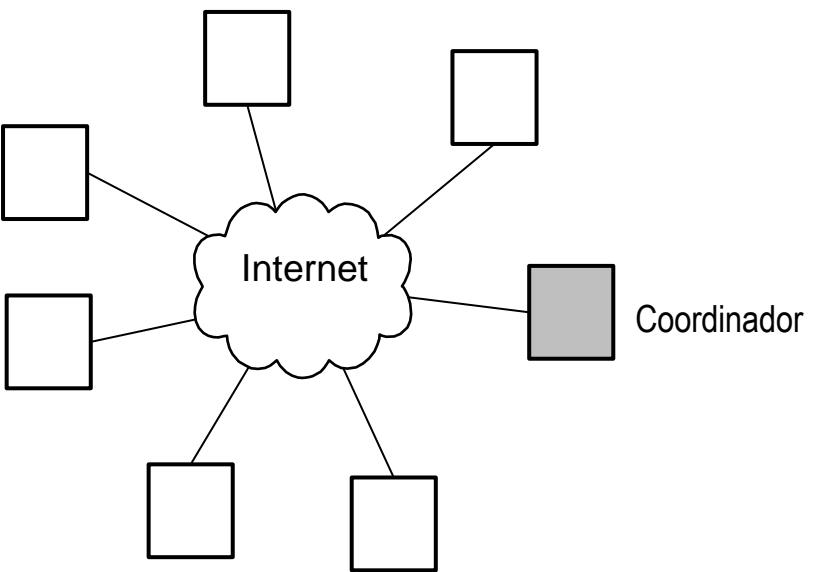
Introducción

- **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)

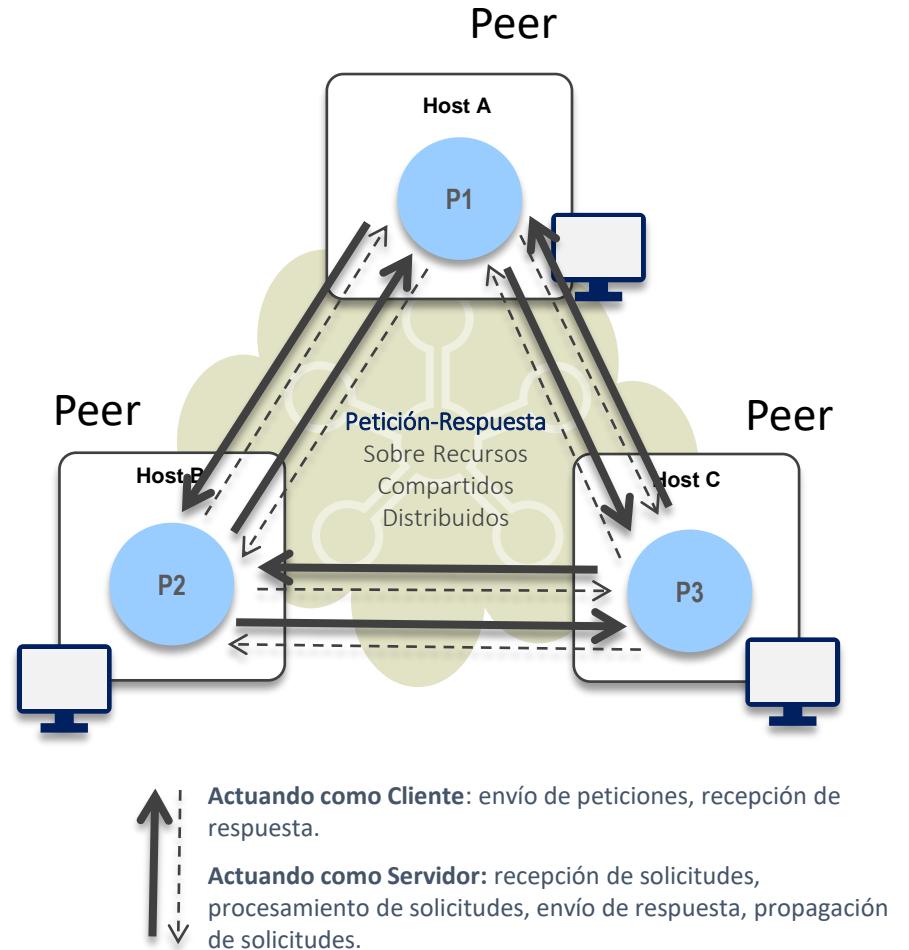
El modelo peer-to-peer (P2P) no tiene roles fijos: todos los nodos (**peers**) actúan como de cliente o servidor, solicitando y proporcionando servicios → es descentralizado: mejora la escalabilidad y tolerancia a fallos.

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.

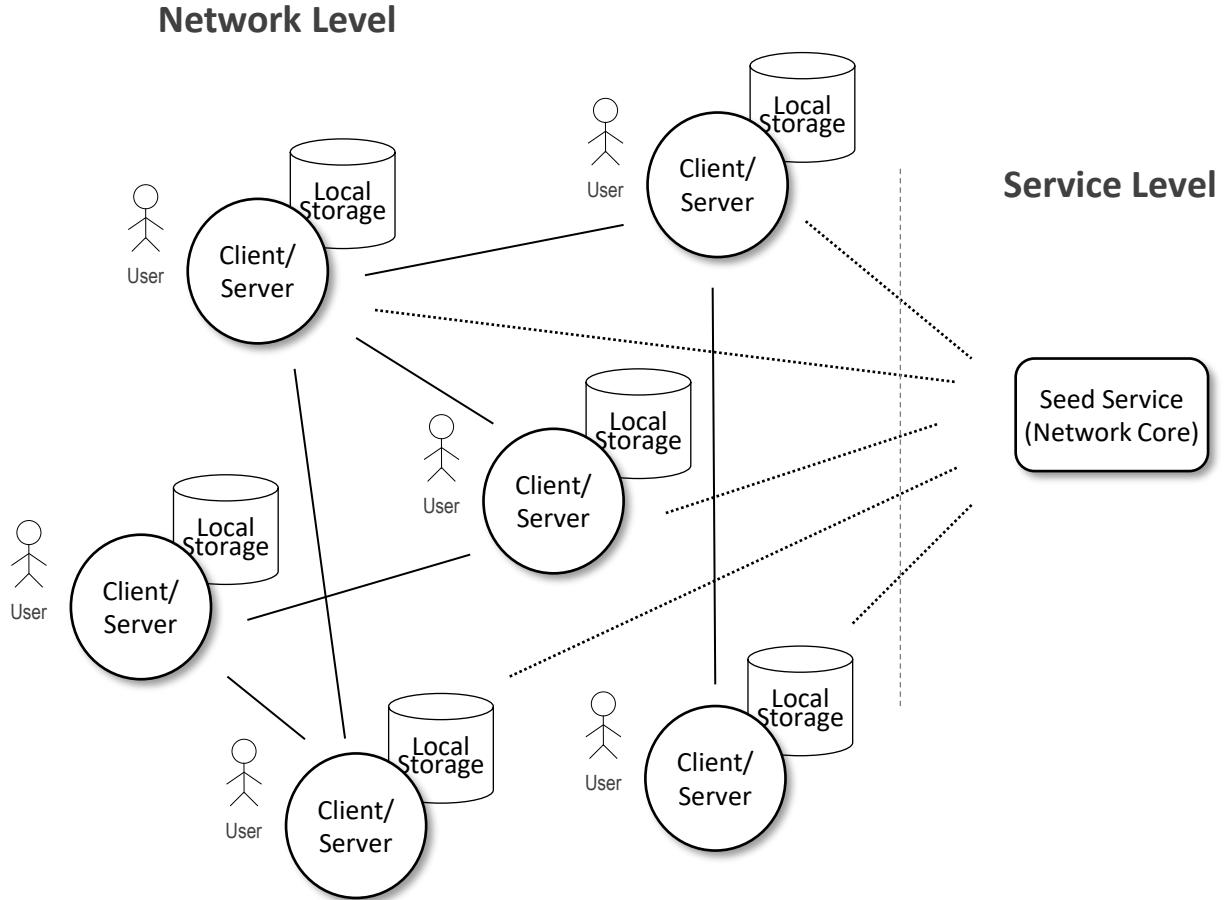
Apropiado para aplicaciones tipo: mensajería instantánea, transferencia de archivos, video conferencia y trabajo colaborativo. Por ejemplo :

- BitTorrent y µTorrent, redes blockchain, ZeroNet



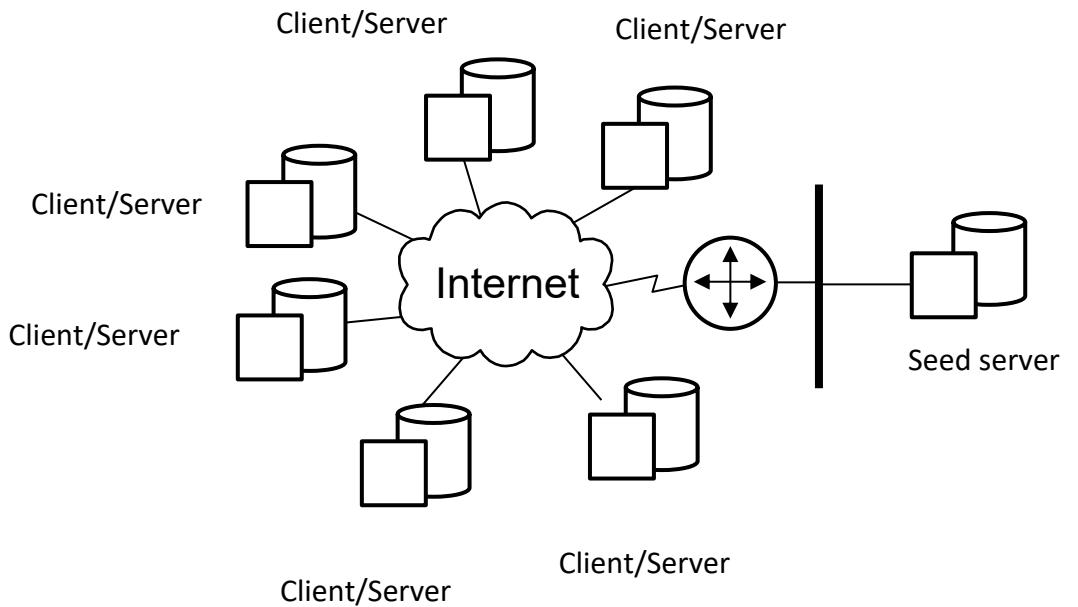
Peer-to-peer (P2P)

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)

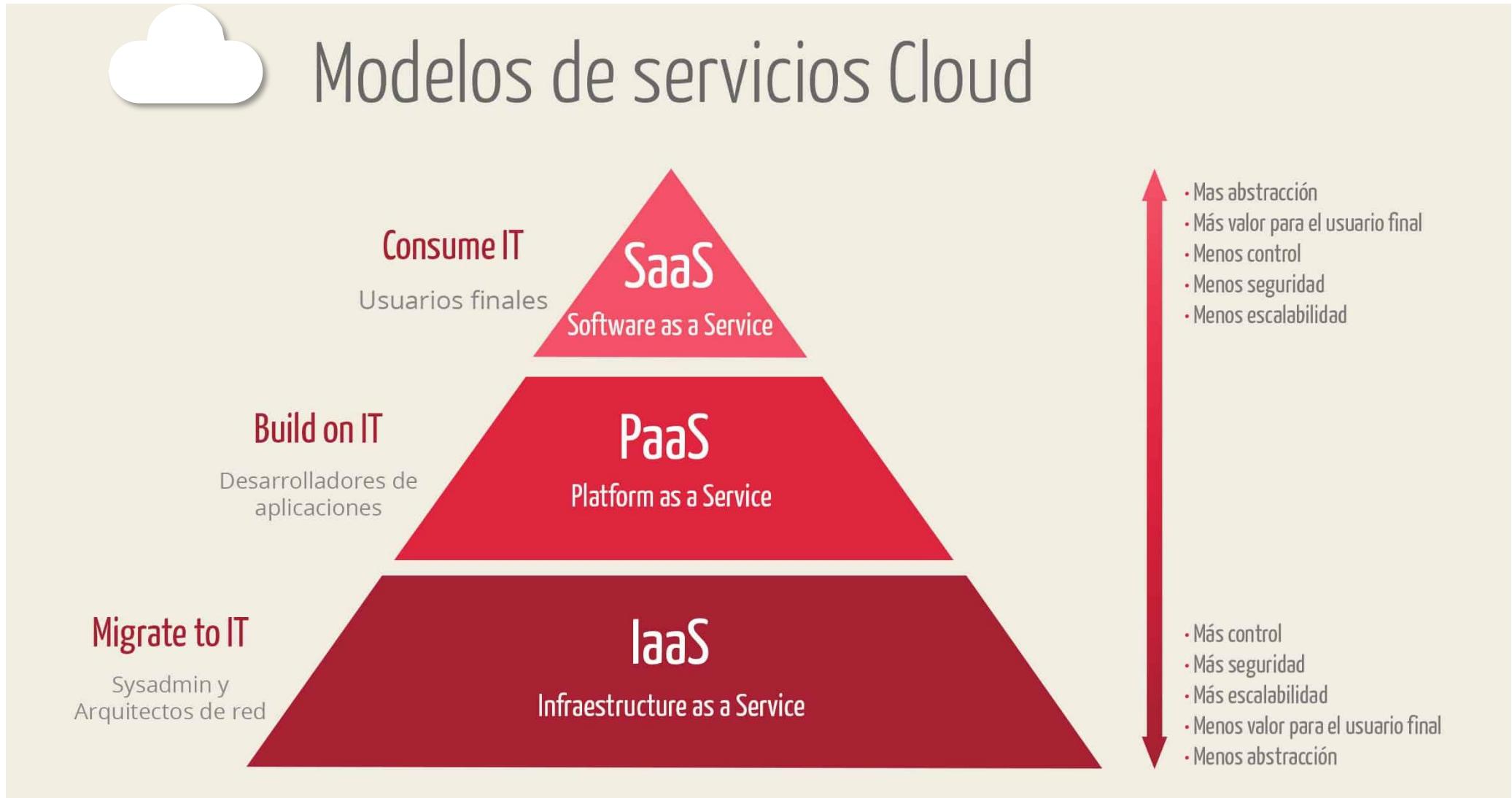
PaaS (Plataforma como Servicio)

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

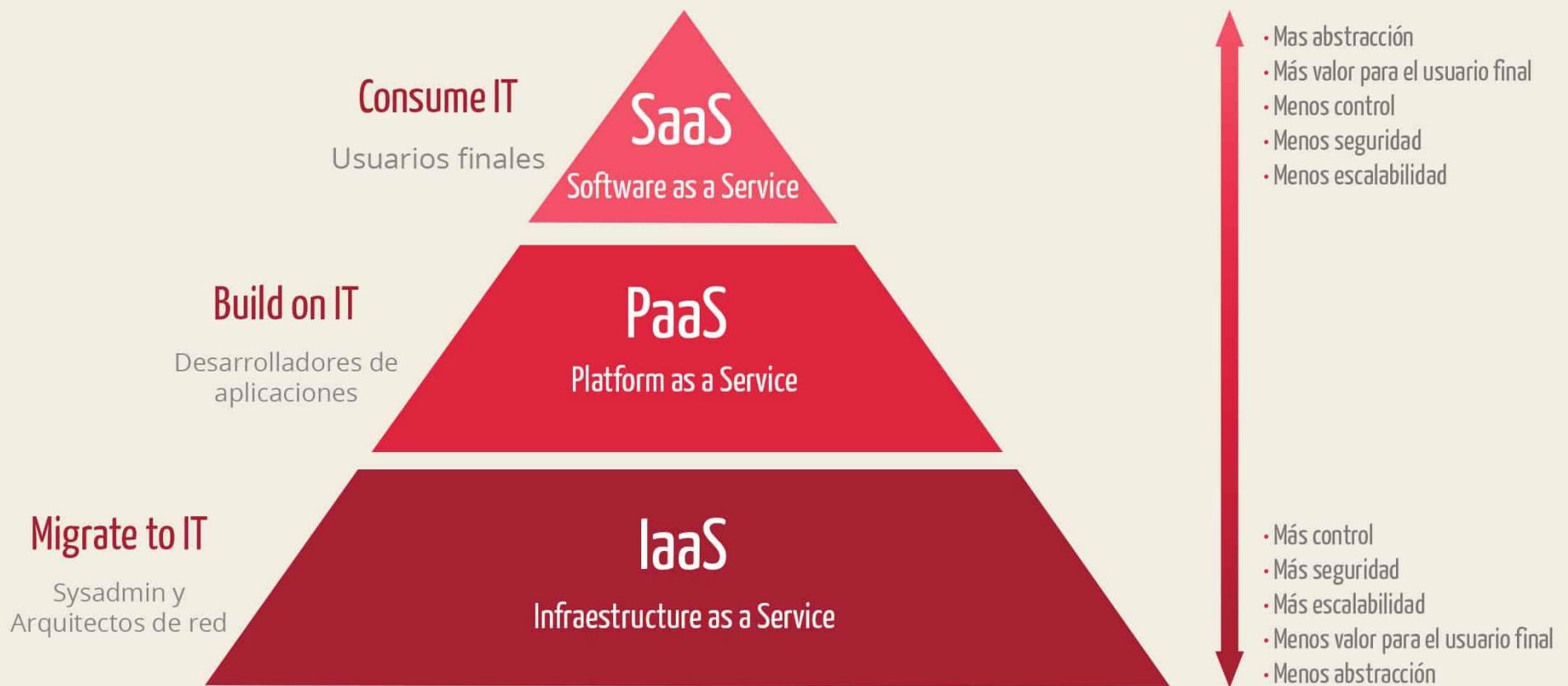


- **Ventajas:**
 - No requiere instalación o mantenimiento del software por parte del usuario.
 - Acceso desde cualquier lugar con conexión a Internet.
 - Actualizaciones y mantenimiento gestionados por el proveedor.
 - Escalabilidad fácil según las necesidades del negocio.
- **Desventajas:**
 - Menor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

Modelos de servicios Cloud



Arquitectura Cloud



- 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

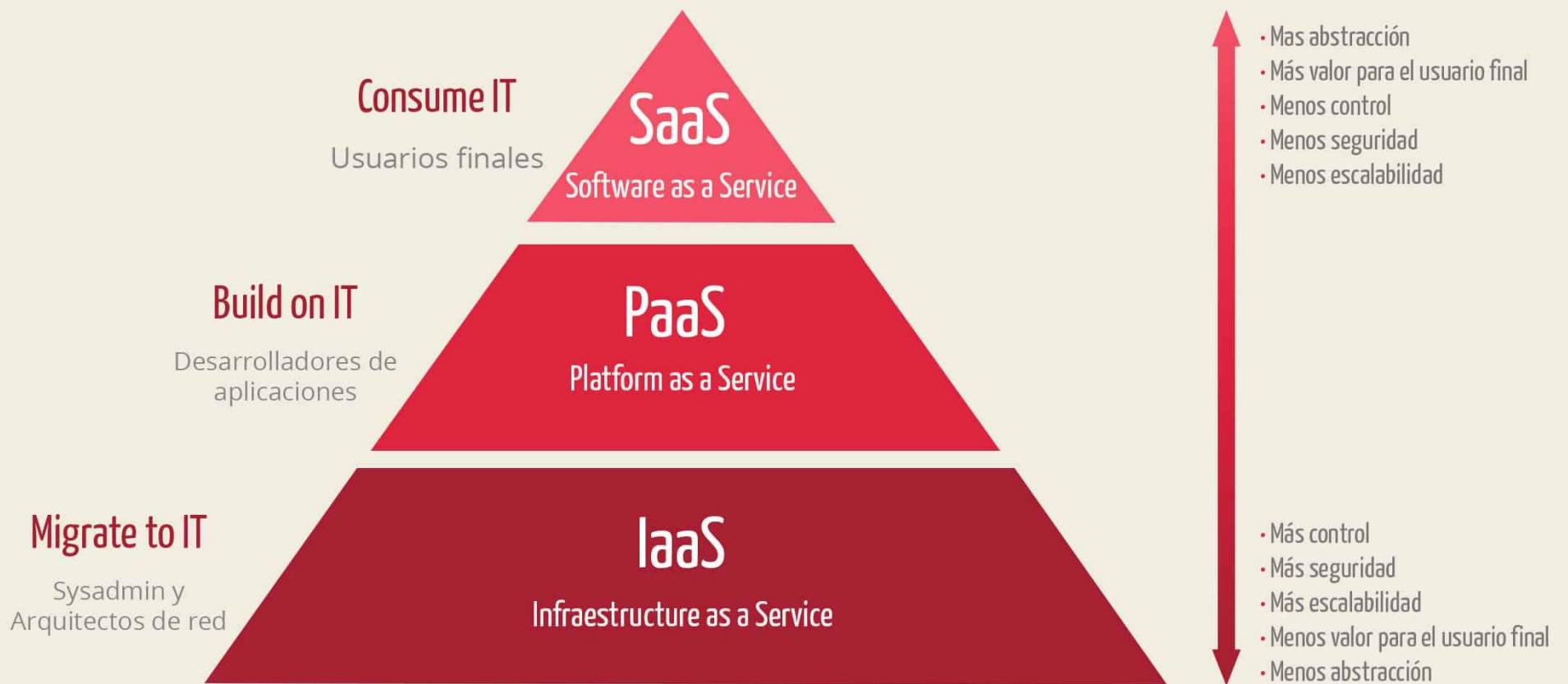


- **Ventajas:**
 - Simplifica el proceso de desarrollo y despliegue de aplicaciones.
 - Proporciona herramientas y servicios integrados para el desarrollo, pruebas y despliegue.
 - Escalabilidad y gestión automática de la infraestructura.
 - Permite a los desarrolladores centrarse en el código y la funcionalidad 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

Modelos de servicios Cloud



Arquitectura Cloud



- 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

Sistemas Operativos y Distribuidos

Iren Lorenzo Fonseca
iren.fonseca@.ua.es



TEMA 3. Sistemas Distribuidos.

Tecnologías para los Sistemas Distribuidos

Tecnologías para los Sistemas Distribuidos

Contenidos

Mecanismos de comunicación distribuida

IPC, Sockets, RPC, RMI, ORB

Revisión de tecnologías Web

Modelo HTTP Básico

Servicios Web

SOA, REST, gRPC, GraphQL, WebSocket

- Teoría General
- Transmisión de información
- Protocolos
- Sockets, RPC, RMI, ORB

Mecanismos de comunicación

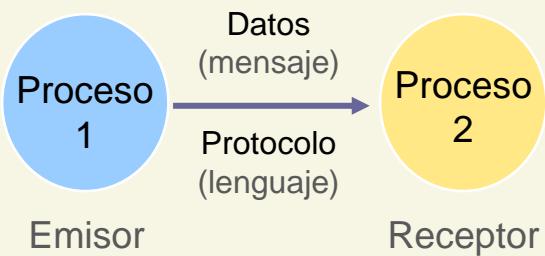
Contenidos

Teoría General

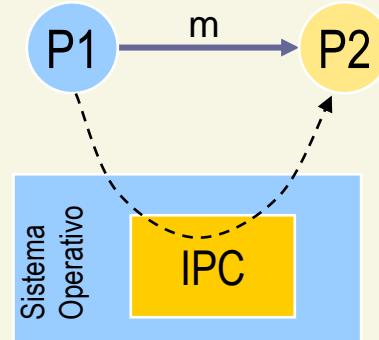
Teoría General

Introducción

| **IPC**: conjunto de herramientas básicas del SO
→ comunicación entre procesos distribuidos



Elementos que intervienen en la comunicación entre procesos. Teoría de la comunicación



Arquitectura para la comunicación entre procesos mediante IPC

Teoría General

Introducción

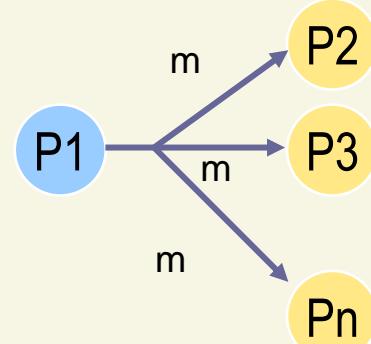
| **IPC**: conjunto de herramientas básicas del OS
→ comunicación entre procesos distribuidos

| Modelos básicos:

- | Unidifusión (unicast)
- | Multidifusión (multicast)



Enfoque Unicast



Enfoque Multicast

Teoría General

Introducción

IPC: conjunto de herramientas básicas del OS
→ comunicación entre procesos distribuidos

Modelos básicos:

- ─ Unidifusión (unicast)
- ─ Multidifusión (multicast)

Definición de interfaz:

- ─ Conectar (solicitar-conexión/aceptar-conexión) y Desconectar
- ─ Enviar y Recibir

Sincronización básica, temporizadores e interbloqueos

- ─ Operaciones bloqueantes o síncronas
- ─ Operaciones no bloqueantes o asíncronas

Teoría General

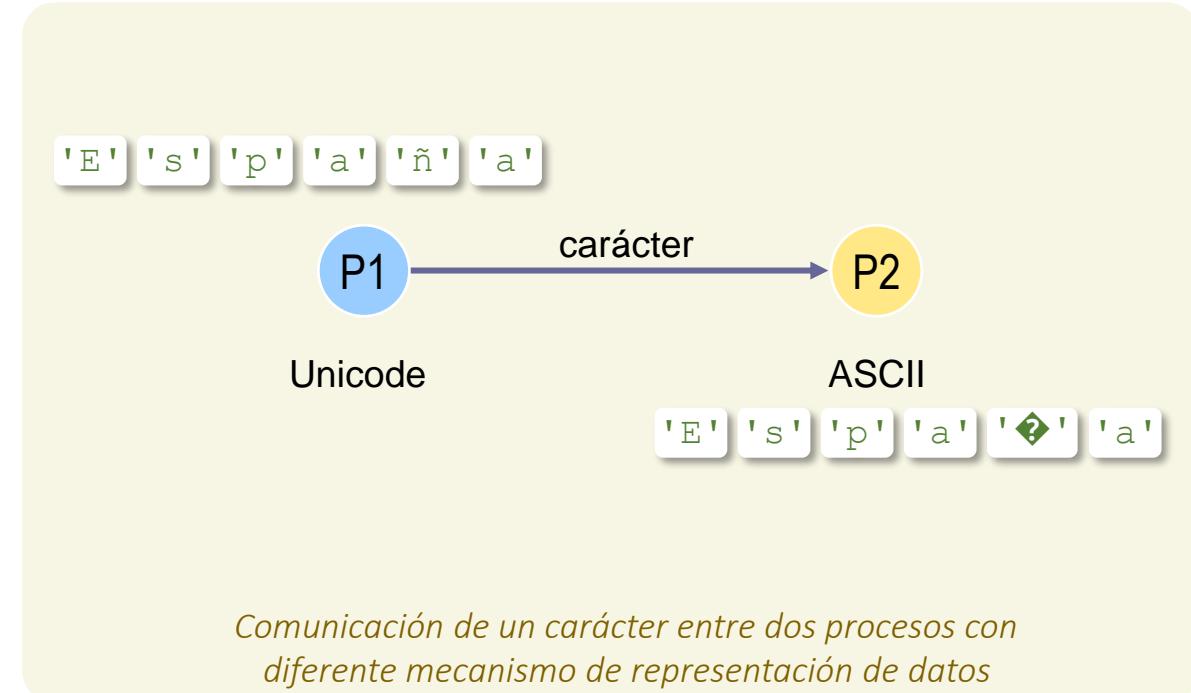
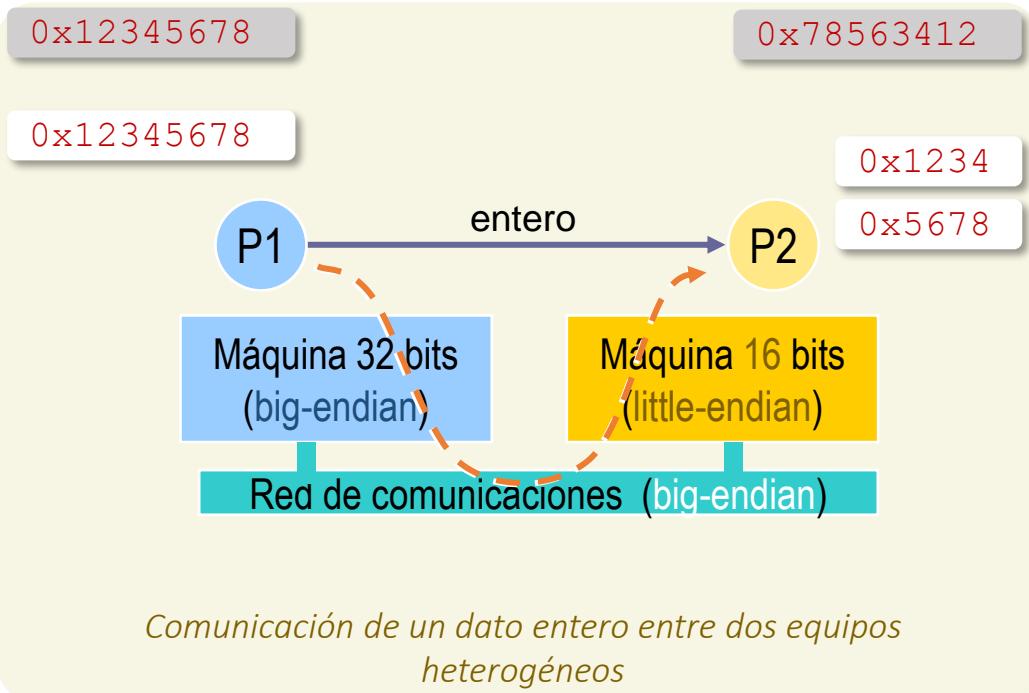
Características

Aunque a **nivel de red**

Es un flujo binario

Existe mucha **heterogeneidad**

Representación de la información



Teoría General

Características

Aunque a **nivel de red**

- Es un flujo binario

Existe mucha **heterogeneidad**

- Representación de la información

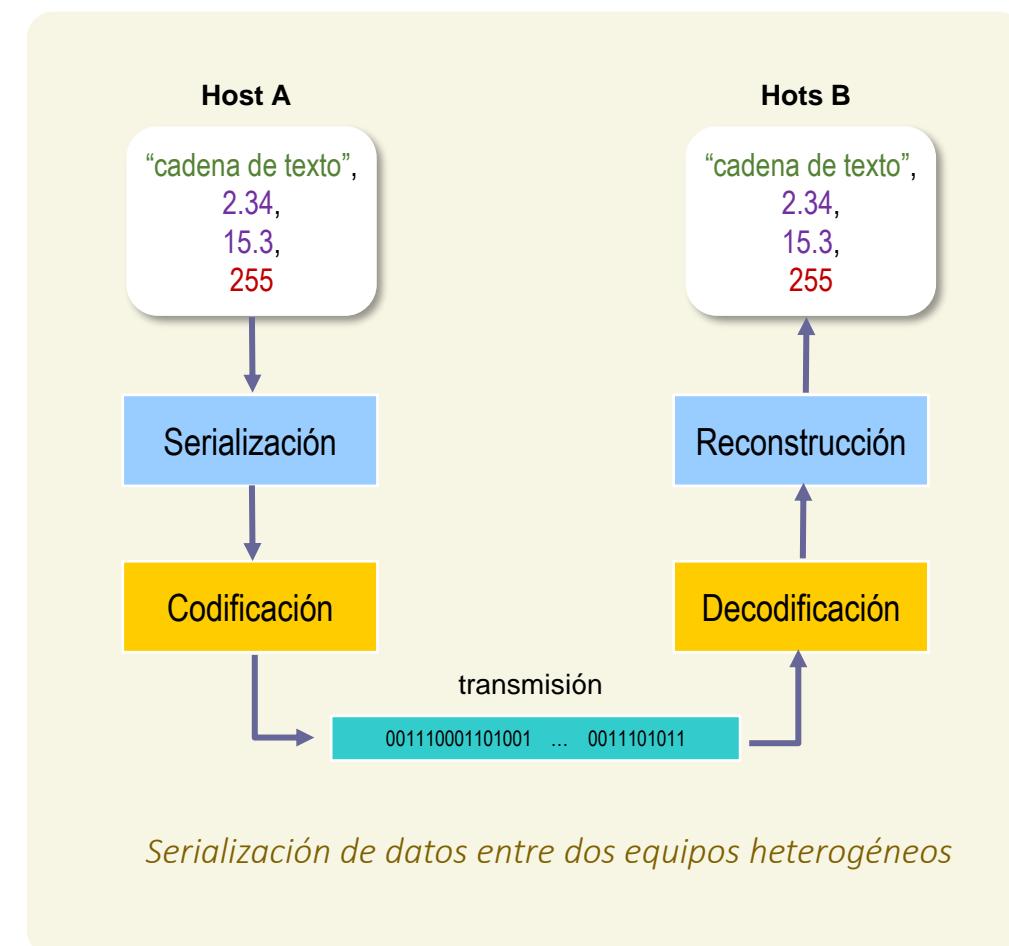
- Solución

- ✓ Emisor adapta a la representación del receptor
- ✓ Receptor adapta la representación del emisor
- ✓ Representación externa común

Empaqueamiento de datos

- Tipos complejos → Serialización

- Objetos → Marshalling



Teoría General

Características

Aunque a nivel de red

Es un flujo binario

Existe mucha heterogeneidad

Representación de la información

Solución

- ✓ Emisor adapta a la representación del receptor
- ✓ Receptor adapta la representación del emisor
- ✓ Representación externa común

Empaqueamiento de datos

Tipos complejos → Serialización

Objetos → Marshalling

Codificación de los datos

Normalización → XDR / ASN.1 / XML / JSON / Protocol Buffer / ...

XDR (External Data Representation)

```
struct Person {  
    string name;<>;  
    int age;  
    string email;<>;  
};
```

ASN.1 (Abstract Syntax Notation One)

```
Person ::= SEQUENCE {  
    name UTF8String,  
    age INTEGER,  
    email IA5String  
}
```

XML (Extensible Markup Language)

```
<Person>  
    <name>Juan López</name>  
    <age>30</age>  
    <email>jlp@ua.es</email>  
</Person>
```

JSON (JavaScript Object Notation)

```
{  
    "name": "Juan López",  
    "age": 30,  
    "email": "jlp@ua.es"  
}
```

Protocol Buffer (protobuf)

```
syntax = "proto3";  
  
message Person {  
    string name = 1;  
    int32 age = 2;  
    string email = 3;  
}
```

Teoría General

Clasificación

Basados en texto o binario



Texto: HTTP, SMTP, POP3, IMAP



Binarios: FTP

Patrón de comunicación



Petición-Respuesta: FTP, HTTP, SMTP, IMAP



Publicación-Suscripción: MQTT, XMPP, AMQP

Orientados o no a la conexión



Con conexión (TCP): HTTP, FTP, IMAP, SMTP



Sin conexión (UDP): DNS, DHCP

Con estado (*state*) o sin estado (*stateless*)



Con estado: FTP, SSH



Sin estado: HTTP

Mecanismos de Comunicación

Mecanismos de Comunicación

Paso de mensaje (Sockets)

Llamadas a procedimientos remotos (RPC)

Invocación de métodos remotos (RMI)

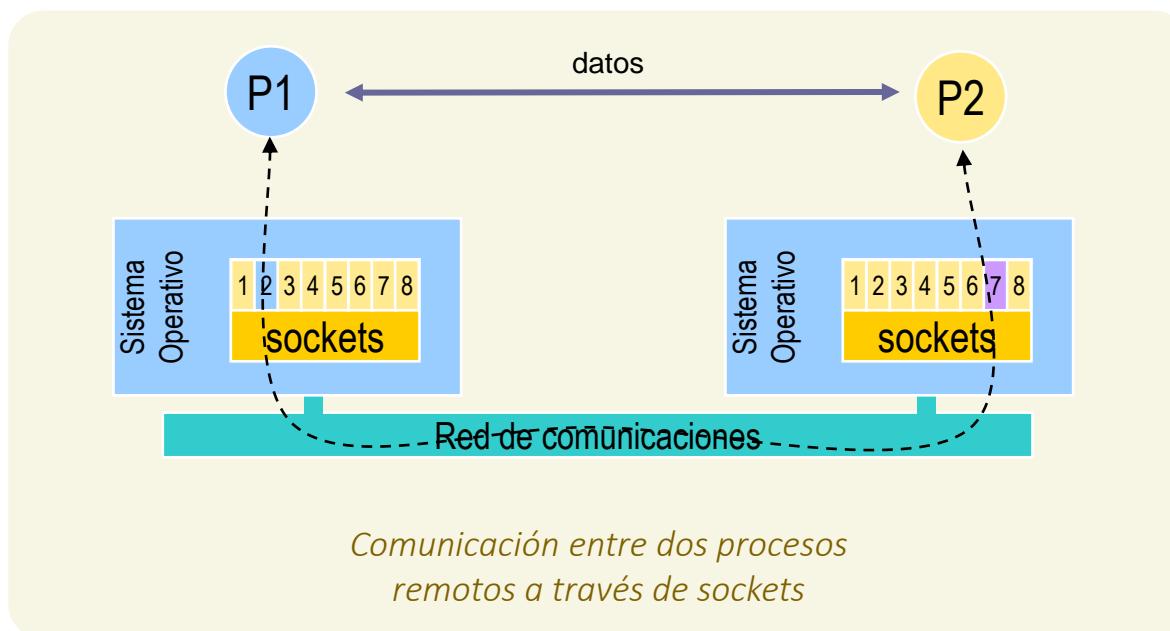
Intermediario de petición de objetos (ORB)

Paso de mensaje

Sockets

Sockets es un mecanismo IPC básico de intercambio de datos a través de un conector (socket):

Constituye un punto final de comunicación bidireccional entre procesos en una red, multiplexados mediante Puertos.



Paso de mensaje

Sockets

| **Sockets** es un mecanismo IPC básico de intercambio de datos a través de un conector (socket):

| Constituyes un punto final de comunicación bidireccional entre procesos en una red, multiplexados mediante **Puertos**.

| **API de sockets:** biblioteca de programación

| Protocolos Soportados:

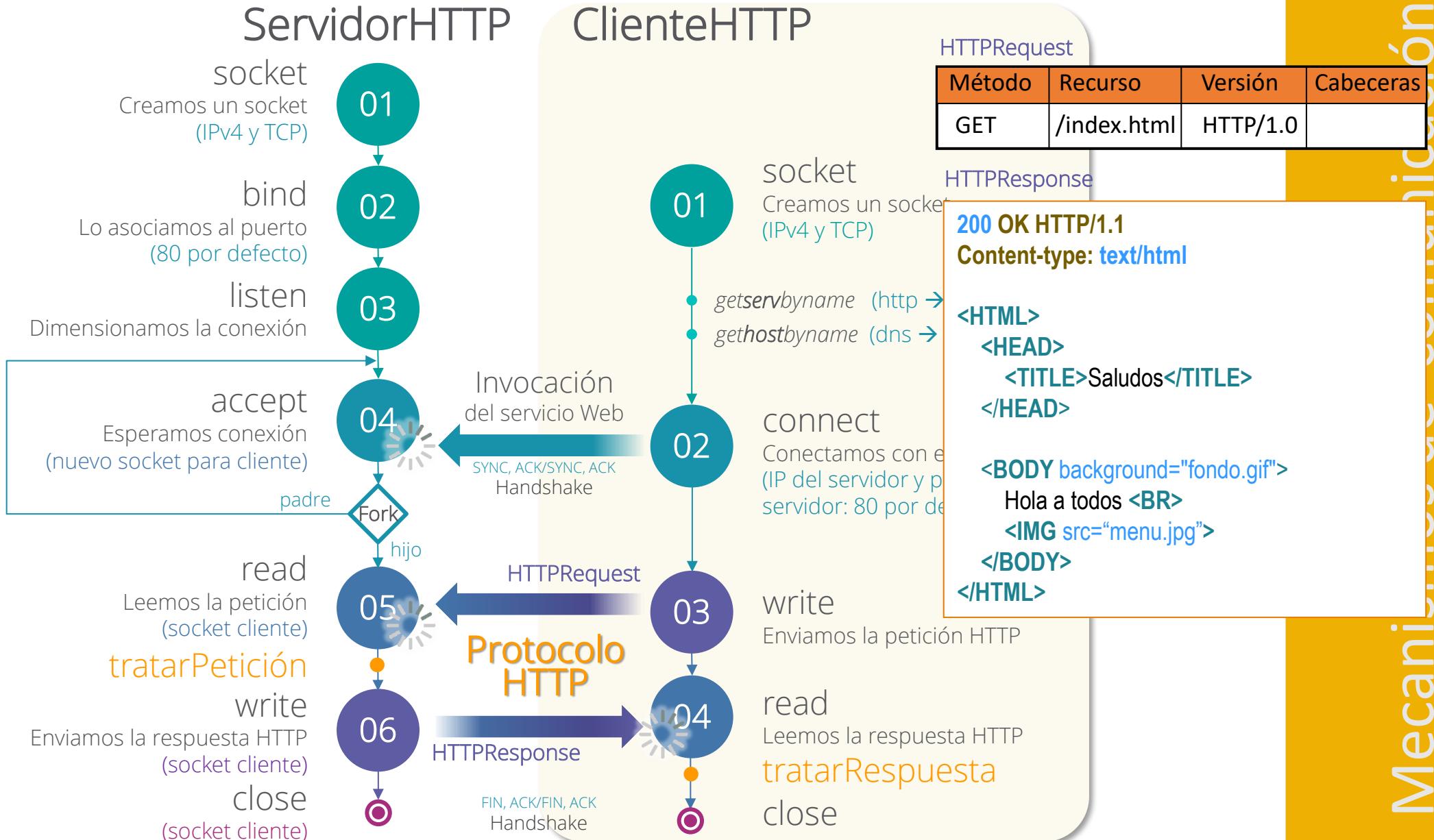
- | **TCP** (orientado a conexión): flujo de datos (paquetes), confiable, mantiene conexión.
- | **UDP** (sin conexión): datagramas, rápido, sin necesidad de conexión establecida.

| Principales llamadas al sistema:

- | `socket()`, `bind()`, `connect()`, `send()`, `recv()`, `close()`

Paso de mensaje

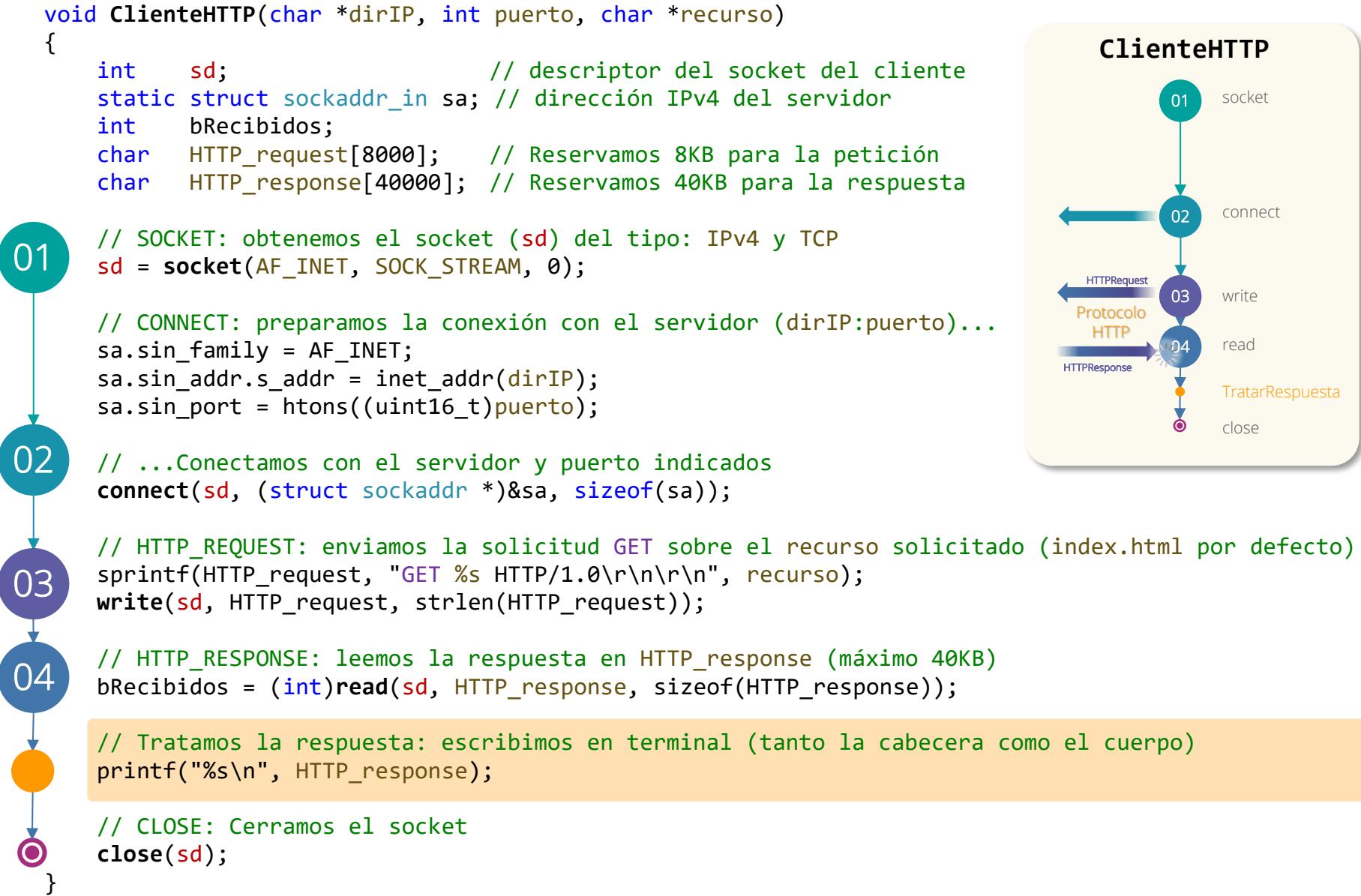
cliente/servidor HTTP con sockets



Mecanismo de transferencia de datos

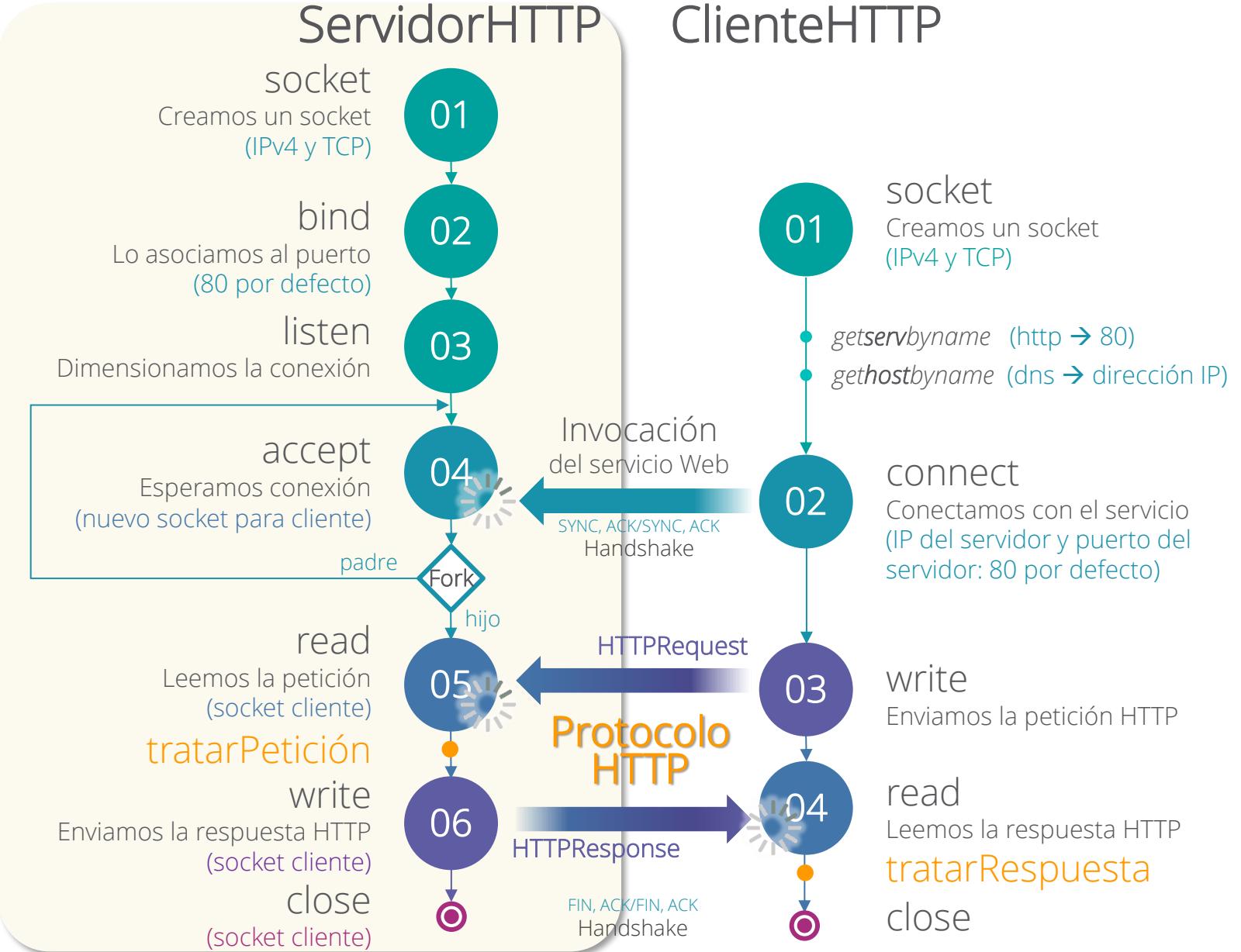
Paso de mensaje

navegador WEB de texto codificado en lenguaje C



Paso de mensaje

cliente/servidor HTTP con sockets



Mecanismos de comunicación

Paso de mensaje

servidor WEB

codificado en *pseudo-código*

servidorHTTP()

Begin

01 *sd = socket(IPv4, TCP)*

02 *bind(sd, 80)*

03 *listen(sd, 5)*

Do

04 *sdHijo = accept(sd)*

05 *While (fork() != 0)*

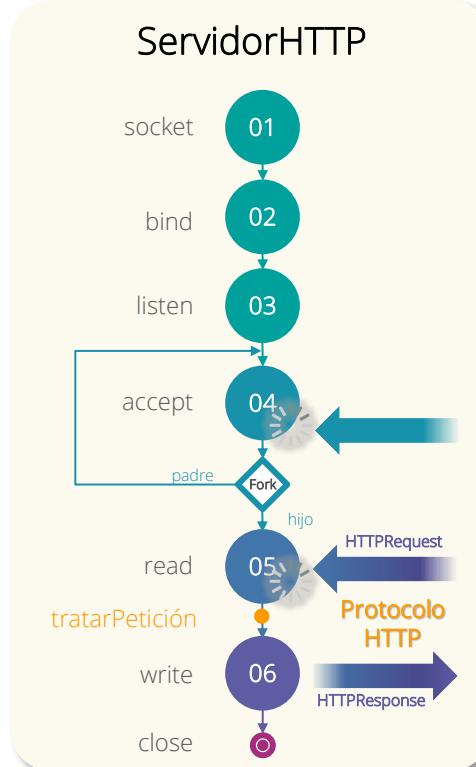
06 *request = read(sdHijo)*

tratarPetición(request, response)

write(sdHijo, response)

close(sdHijo)

End



tratarPetición(cadena req, cadena res,)

Begin

recurso = obtenerRecurso(req)

res = "200 OK HTTP/1.1<\n>" + headers + "<\n><\n>" + recurso

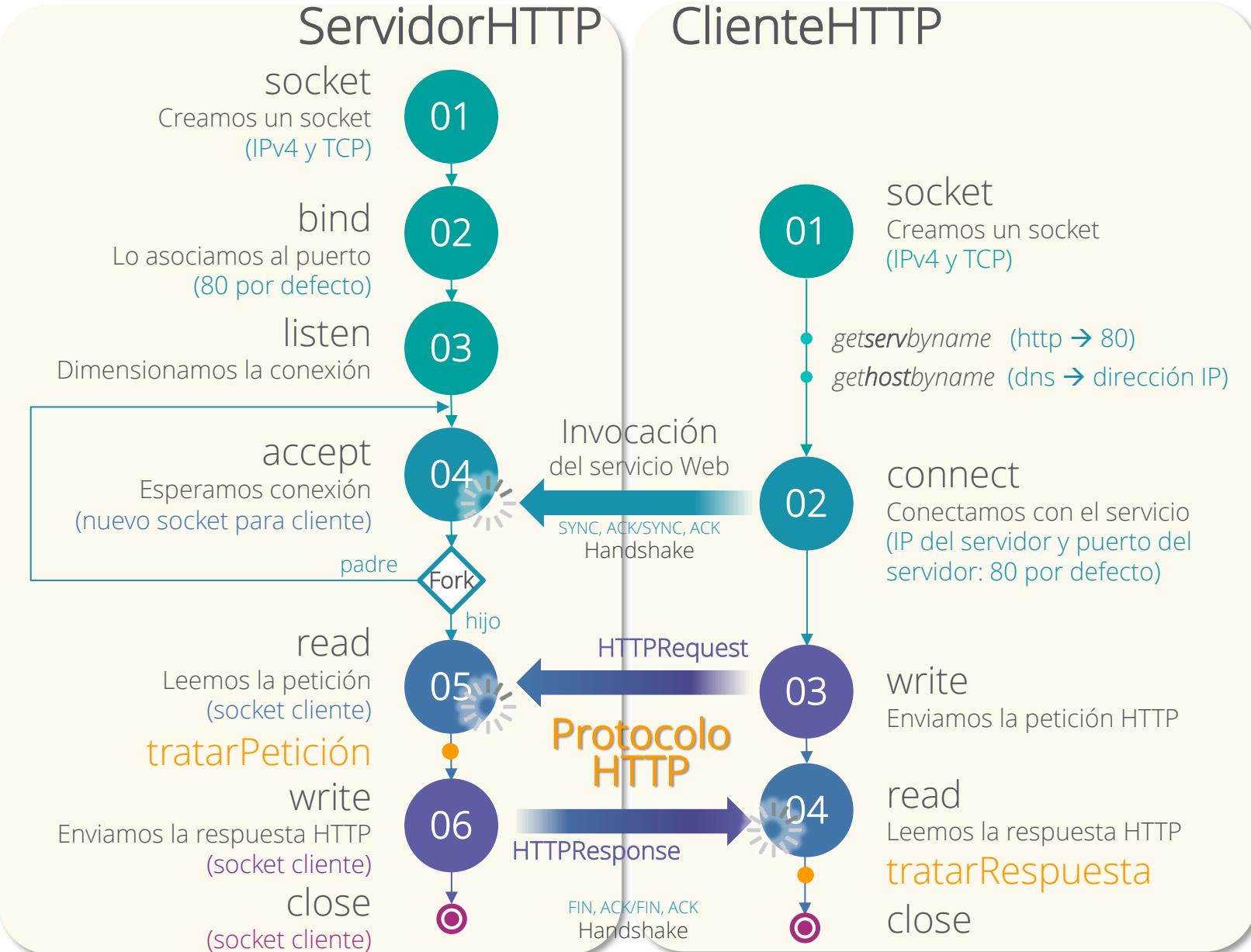
End

Versión en *pseudo-código* de un servidor Web

Mecanismos de comunicación

Paso de mensaje

cliente/servidor HTTP con sockets



Mecanismos de comunicación

Paso de mensaje

Sockets

Mecanismos de comunicación

Características:

- Comunicación Directa → Control preciso sobre el flujo de datos.

Ventajas:

- Alto rendimiento
- Flexibilidad: Adecuado para sistemas personalizados

Desventajas:

- Complejidad en manejo de errores
- Complejo abordar proyectos con grandes volúmenes de conexiones

Aplicaciones Comunes:

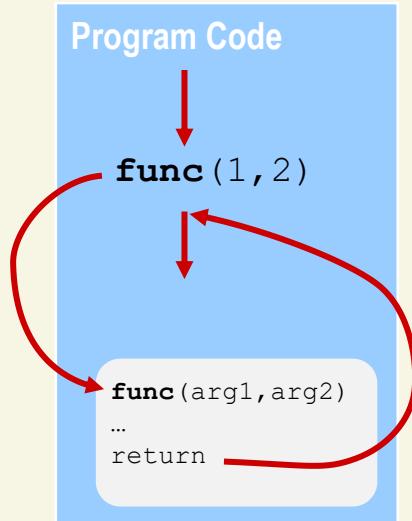
- Servidores Web (HTTP)
- Juegos en Red
- Streaming (video/audio con baja latencia)



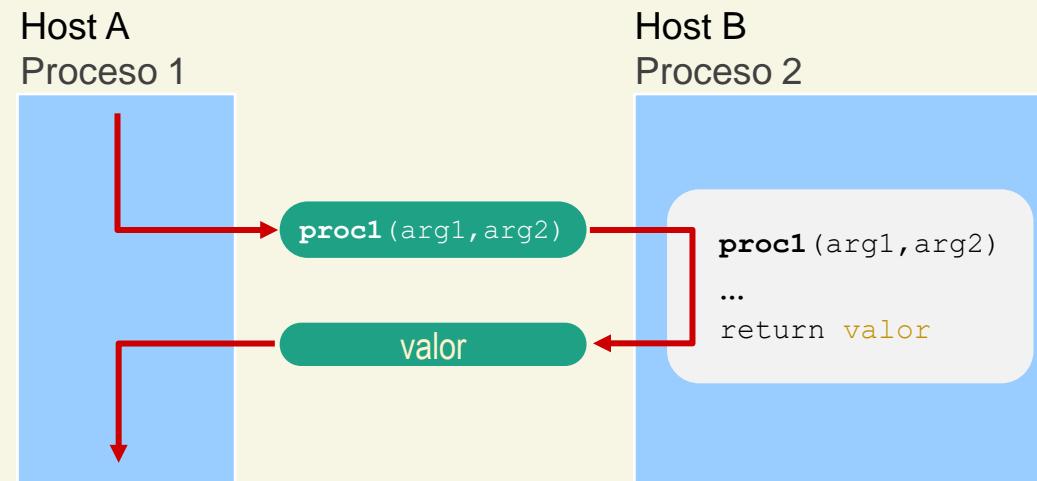
RPC (Remote Procedure Call)

Características

RPC permite ejecutar **funciones** en un **servidor remoto** como si fueran locales.



Llamada a una función o procedimiento local a un programa



Llamada a un procedimiento remoto ubicado en un proceso diferente, en un host diferente

RPC

SOCKETS

RPC (Remote Procedure Call)

Características

■ **RPC** permite ejecutar funciones en un servidor remoto como si fueran locales.

■ **Evolución:** Surgió como una mejora de la comunicación mediante sockets orientado a lenguajes procedimentales.

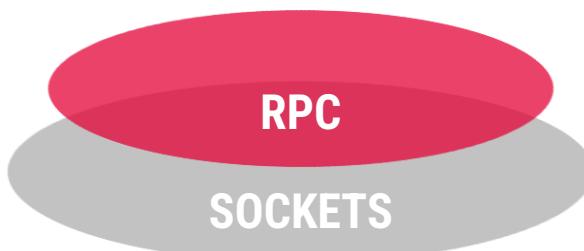
■ **Objetivo:** Simplificar la comunicación en red al abstraer detalles técnicos.

■ **Características Principales:**

■ **Transparencia:** oculta la localización del procedimiento.

■ **Abstracción de Red:** detalles de red invisibles para el desarrollador.

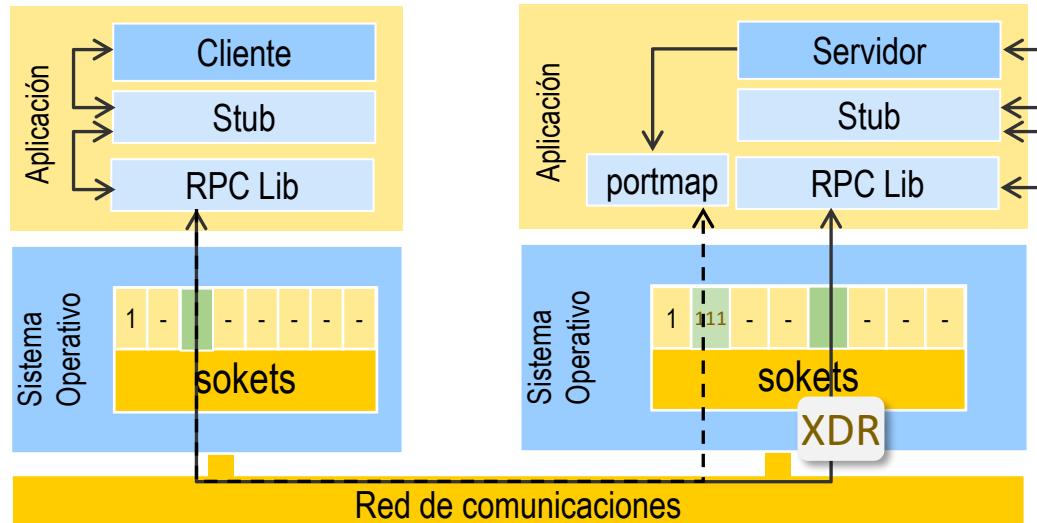
■ **Simplicidad:** facilita el desarrollo de sistemas distribuidos.



RPC (Remote Procedure Call)

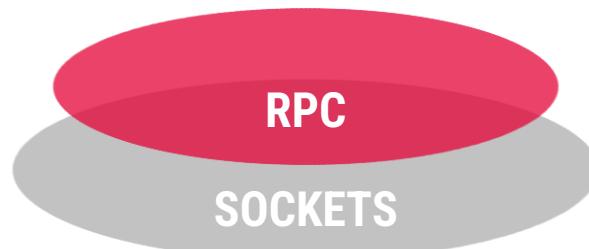
Características

Arquitectura de modelo de llamadas a procedimientos remotos



Componentes de RPC:

- **Cliente y Stub del Cliente:** Solicita y convierte las llamadas en solicitudes de red.
- **Biblioteca RPC:** Actúa como **Sistema de Transporte** transfiriendo las solicitudes y respuestas entre cliente y servidor.
- **Servidor:** Ejecuta la función y devuelve los resultados.
- **Stub del Servidor:** Deserializa la solicitud y la envía al servidor.
- **Portmap:** Servicio que registra y proporciona el **puerto** en el que estará escuchando el Servidor.
- **RPCGEN:** Compilador que genera el código a partir de una definición de interfaz (**IDL**)



RPC (Remote Procedure Call)

Ejemplo de RPC

Código de la aplicación

```
void Suma(int a, int b, int *c);
```

```
void main() {  
    int a=1, b=3, c;  
  
    Suma(a, b, &c);  
    printf("%d\n", c);  
}
```

```
void Suma(int a, int b, int *c) {  
    *c = a + b;  
}
```

RPC (Remote Procedure Call)

Ejemplo de RPC

Definición Interfaz

```
void Suma(int a, int b, int *c);
```

Código Cliente

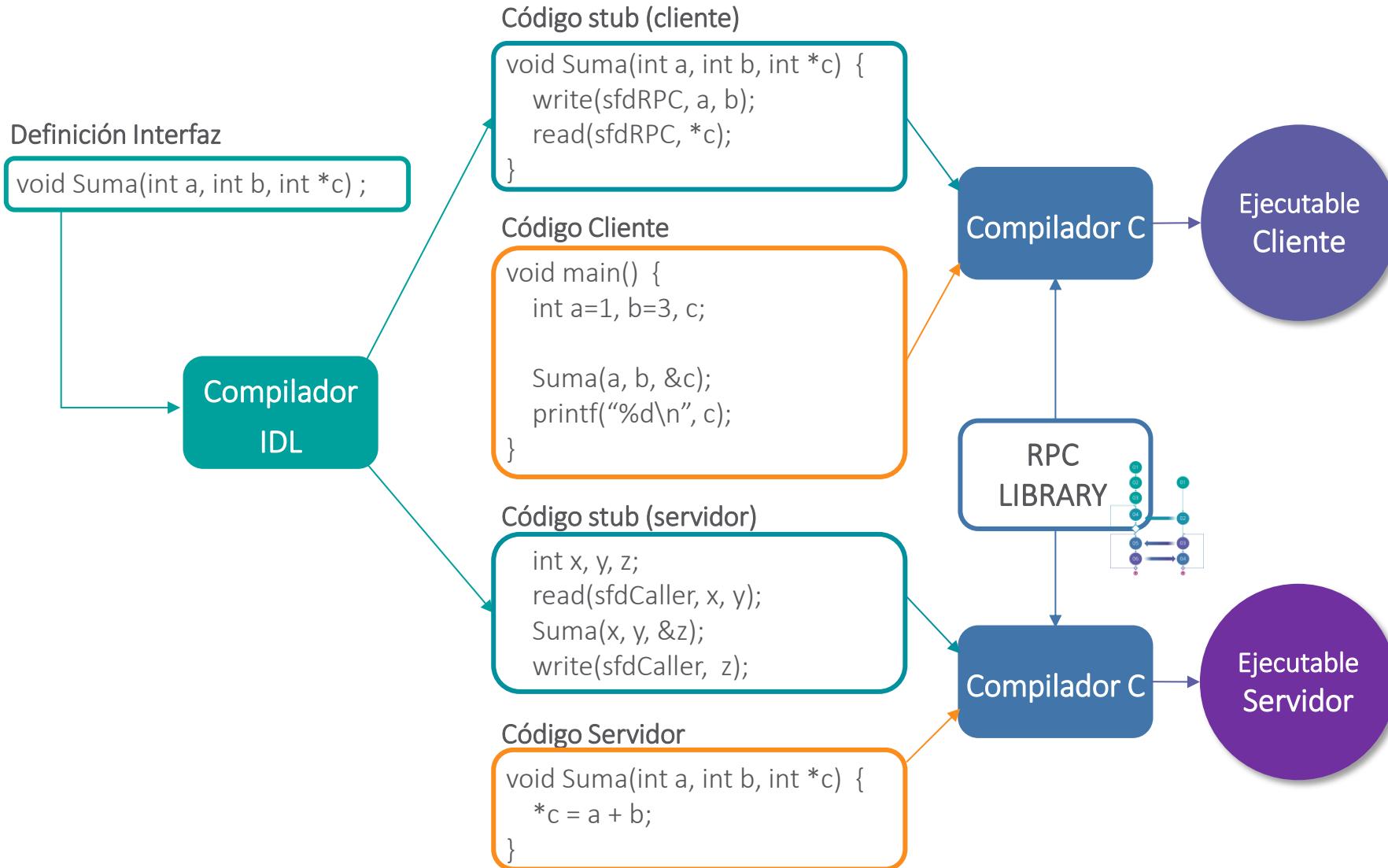
```
void main() {  
    int a=1, b=3, c;  
  
    Suma(a, b, &c);  
    printf("%d\n", c);  
}
```

Código Servidor

```
void Suma(int a, int b, int *c) {  
    *c = a + b;  
}
```

RPC (Remote Procedure Call)

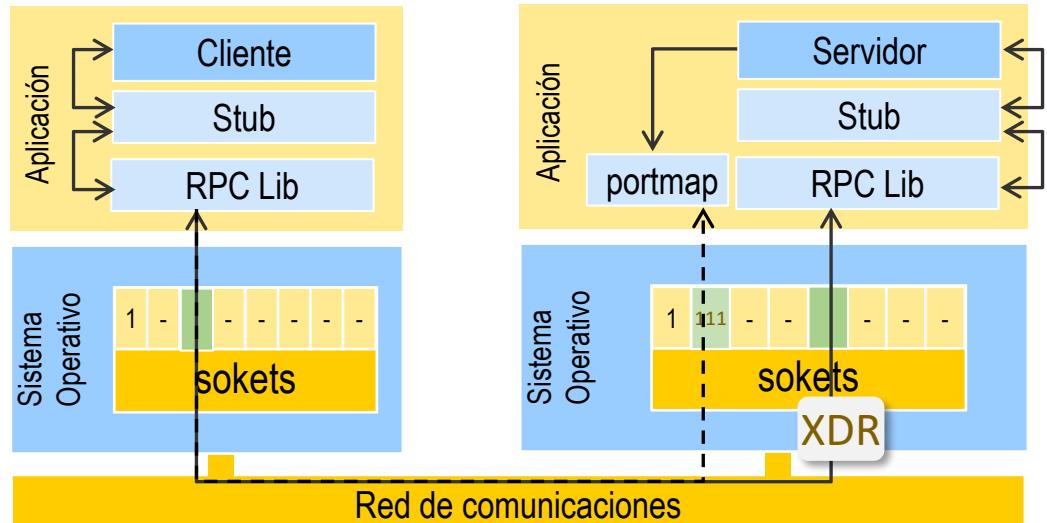
Ejemplo de RPC



RPC (Remote Procedure Call)

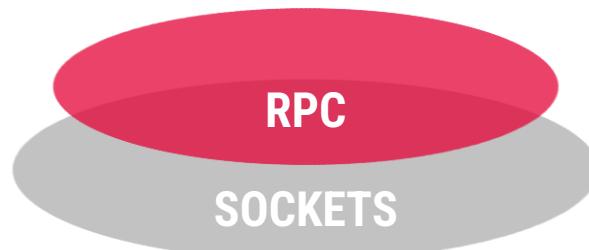
Características

Arquitectura de modelo de llamadas a procedimientos remotos



Componentes de RPC:

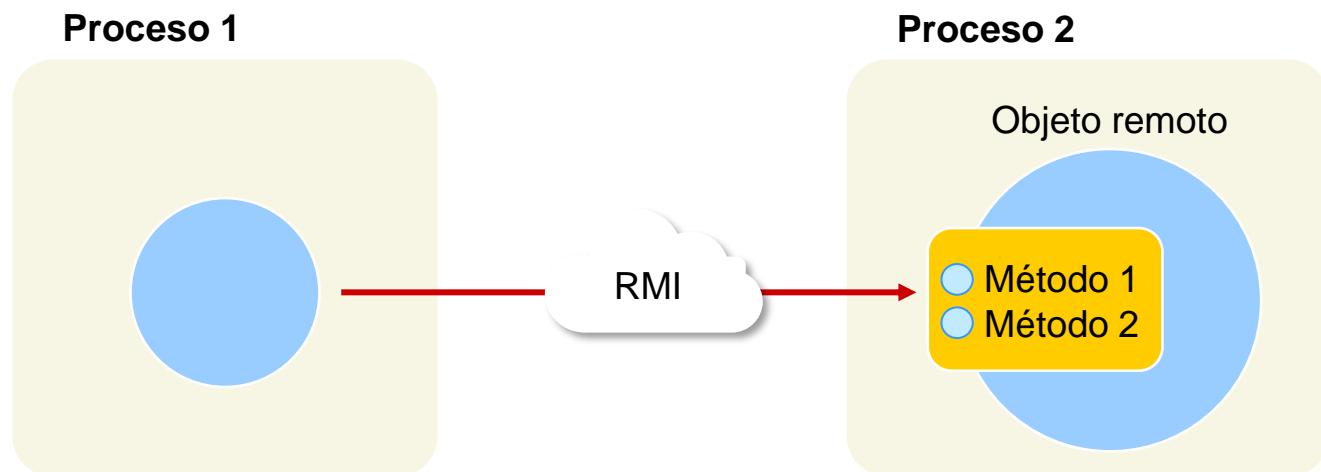
- **Cliente y Stub del Cliente:** Solicita y convierte las llamadas en solicitudes de red.
- **Biblioteca RPC:** Actúa como **Sistema de Transporte** transfiriendo las solicitudes y respuestas entre cliente y servidor.
- **Servidor:** Ejecuta la función y devuelve los resultados.
- **Stub del Servidor:** Deserializa la solicitud y la envía al servidor.
- **Portmap:** Servicio que registra y proporciona el **puerto** en el que estará escuchando el Servidor.
- **RPCGEN:** Compilador que genera el código a partir de una definición de interfaz (**IDL**)



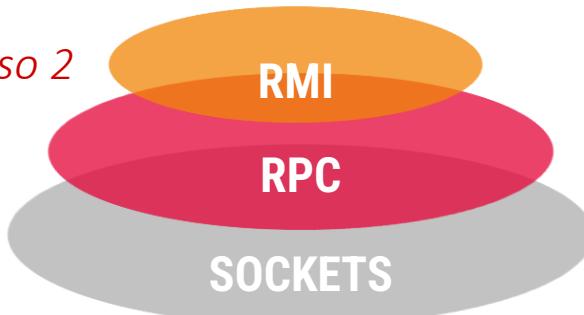
RMI (Remote Methods Invocation)

Características

RMI permite a una aplicación Java invocar métodos en objetos remotos como si fueran locales.



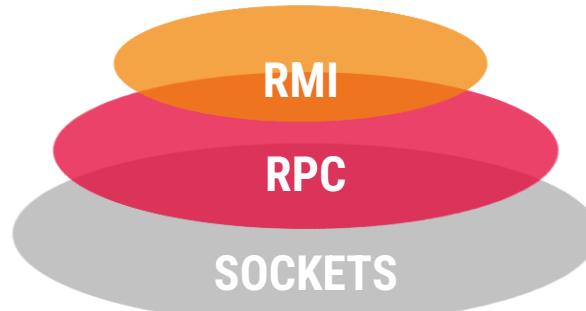
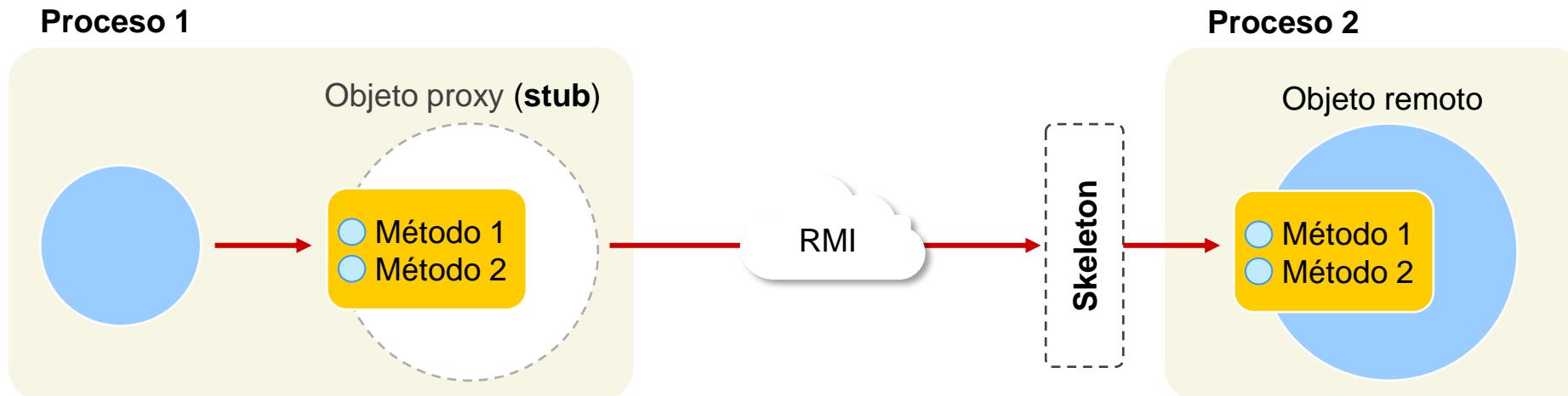
Invocación desde el proceso 1 a un método perteneciente a un objeto remoto ubicado en el proceso 2



RMI (Remote Methods Invocation)

Características

RMI permite a una aplicación Java invocar métodos en objetos remotos como si fueran locales.



RMI (Remote Methods Invocation)

Características

| RMI permite a una aplicación Java invocar métodos en objetos remotos como si fueran locales.

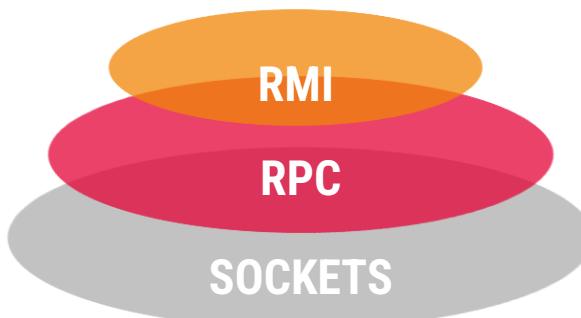
| Basado en el paradigma orientado a objetos, ideal para Java.

Características

- | Transparencia en la invocación
- | Registro de objetos remotos
- | Abstracción de la red

Novedades

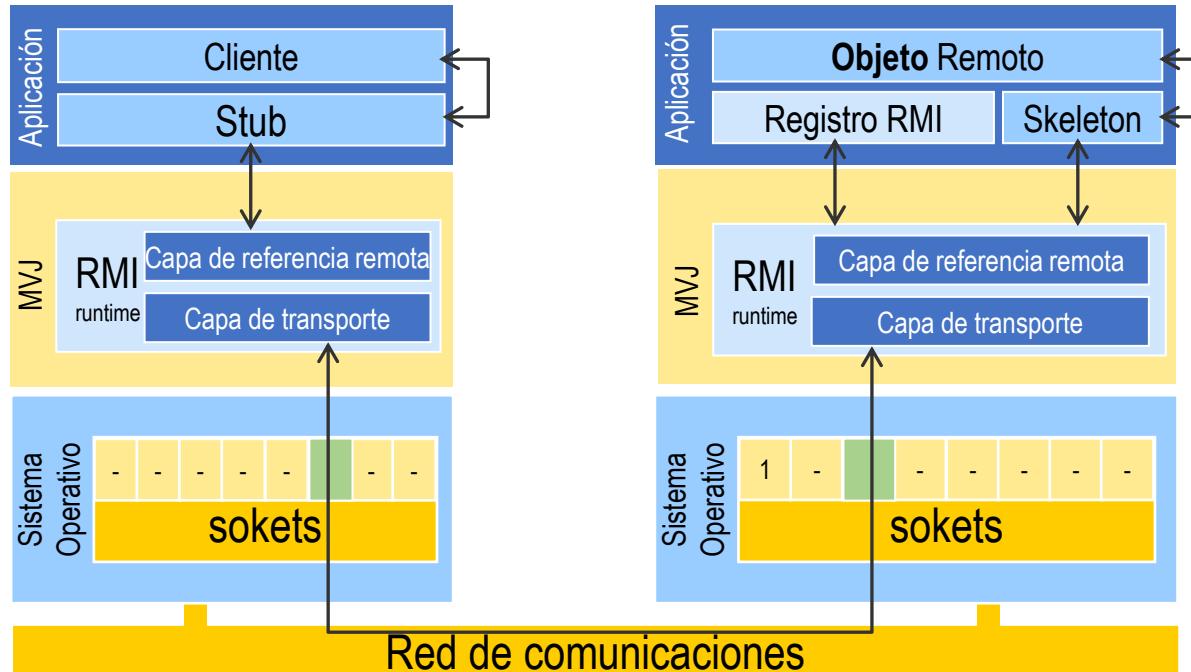
- | Orientación a objetos
- | Incorporación de servicios adicionales (Registro que sustituye a portmap)



RMI (Remote Methods Invocation)

Características

Arquitectura de modelo de invocación a métodos remotos



Componentes de RPC:

Cliente: solicita la ejecución de un método remoto.

Stub del cliente: actúa como un **proxy** local que convierte la llamada del método en una solicitud de red.

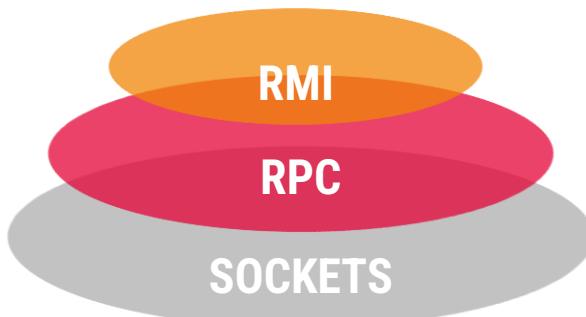
Biblioteca RMI: maneja la comunicación entre el cliente y el servidor, transfiriendo las solicitudes y respuestas de métodos remotos.

Servidor: implementa las interfaces remotas y ejecuta los métodos solicitados por el cliente, devolviendo los resultados.

Skeleton del servidor (obsoleto, sustituido por **stub**): deserializa la solicitud recibida y la envía al objeto servidor para su ejecución. Luego, serializa la respuesta para enviarla de vuelta al cliente.

Registro RMI (RMI Registry): Servicio que permite a los clientes localizar y obtener referencias a los objetos remotos registrados.

Compilador RMI (rmic): genera los stubs y skeletons necesarios a partir de las interfaces remotas definidas en el código.



RMI (Remote Methods Invocation)

Características

| Ventajas

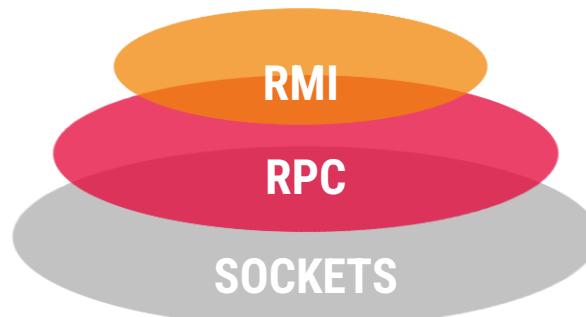
- | Facilita la programación distribuida en Java
- | Oculta la complejidad de red

| Desventajas

- | Dependencia de Java
- | Sobrecarga de red por serialización

| Casos de Uso

- | Aplicaciones empresariales



ORB (Object Request Broker)

Características

Object Request Broker (ORB)

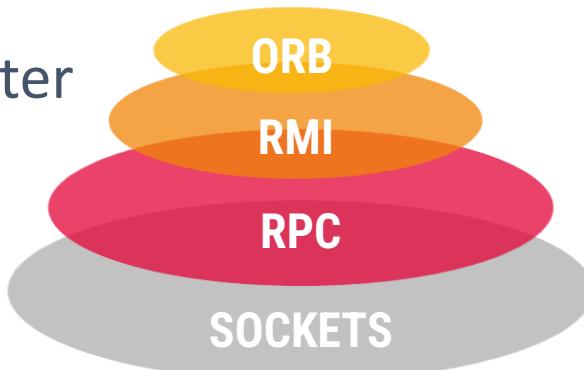
- Facilita la comunicación entre objetos distribuidos.
- Evolución de RMI para otros lenguajes e incorporación de más servicios.
- Ejemplo: **CORBA** (Common Object Request Broker Architecture).

Características

- Transparencia de localización y plataforma.
- Uso de IDL (Interface Definition Language) para definir interfaces.
- Serialización automática.

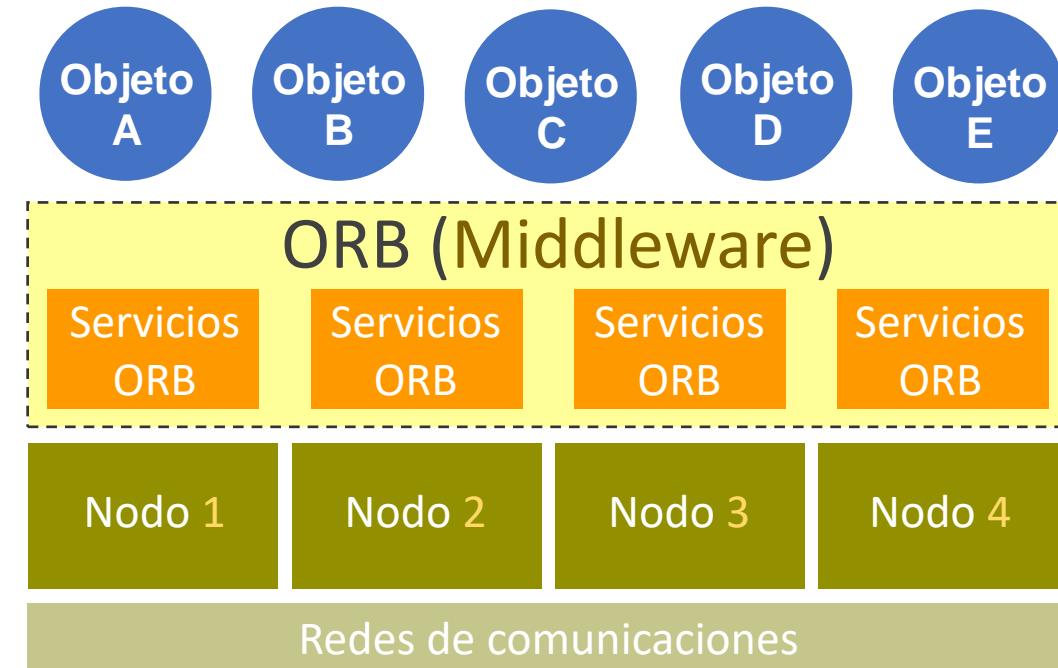
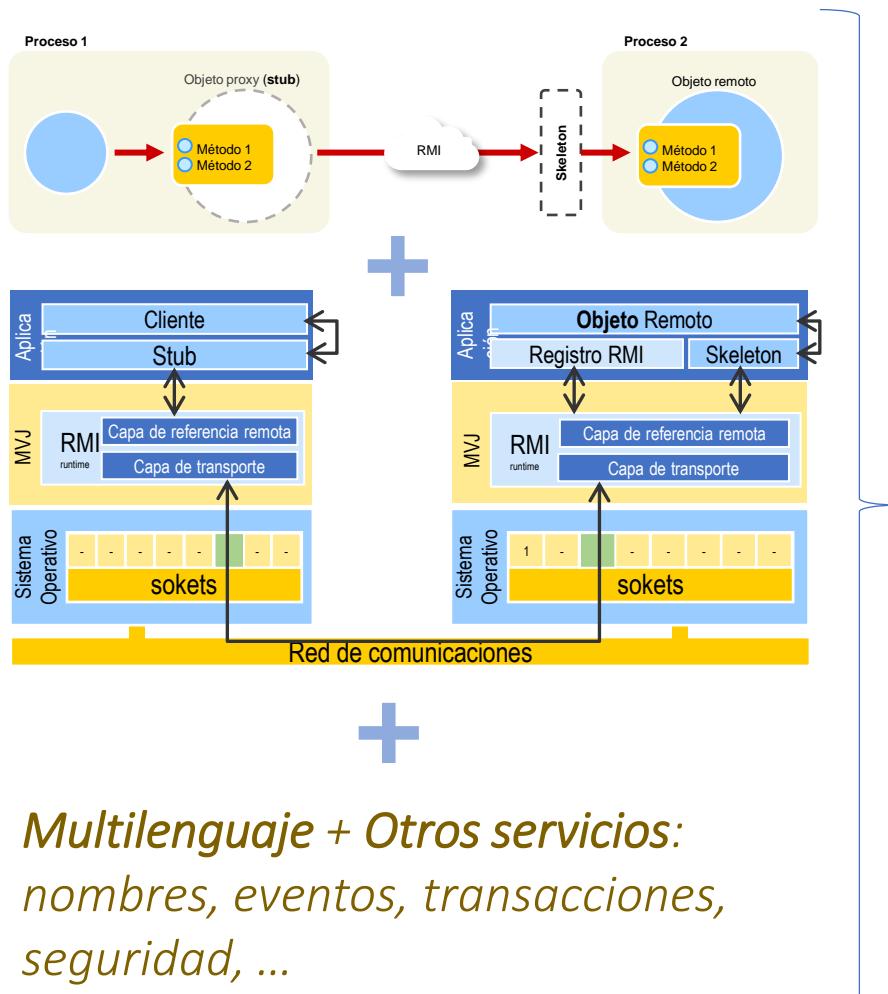
Componentes

- Cliente, Stub, ORB, Skeleton, Objeto remoto, Compilador IDL, Register
- Servicios adicionales: Name, events, transactions, security, ...



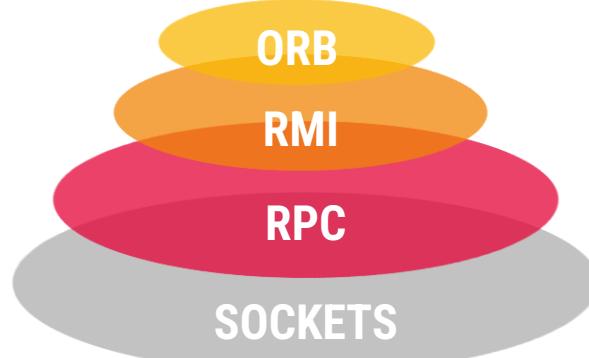
ORB (Object Request Broker)

Características



*Multilenguaje + Otros servicios:
nombres, eventos, transacciones,
seguridad, ...*

Paradigma de objetos remotos basados en Invocación a Métodos Remotos



ORB (Object Request Broker)

Características

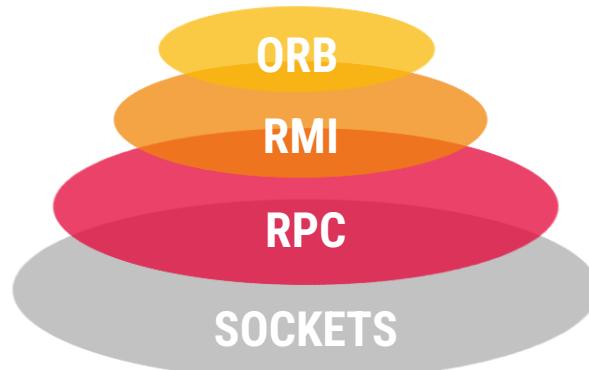
Ventajas

- Soporte multiplataforma: SO, lenguajes, hardware.
- Interoperabilidad en entornos heterogéneos.

Desventajas

- Complejidad en configuración.
- Latencia por serialización.

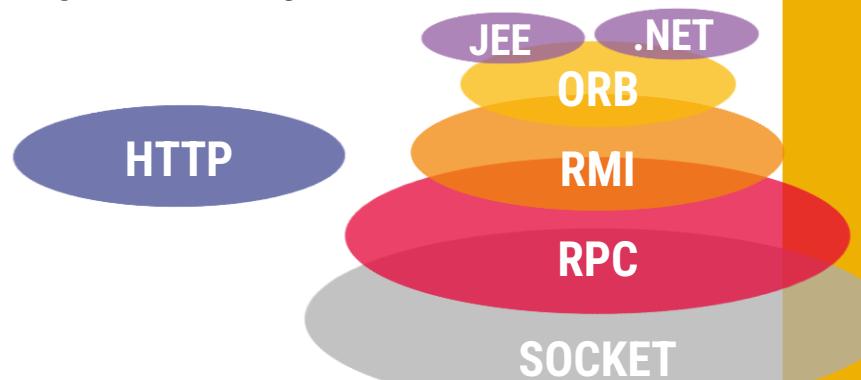
Base de concepto Middleware



Middleware

Características

- Capa de abstracción **compleja**
- Reduce esfuerzos del programador para la comunicación entre procesos
- Inconvenientes
 - Aumenta la **complejidad** y la **curva de aprendizaje** de los propios middlewares
 - Altos consumidores de **recursos**



Mecanismos de comunicación

- Modelo HTTP Básicos

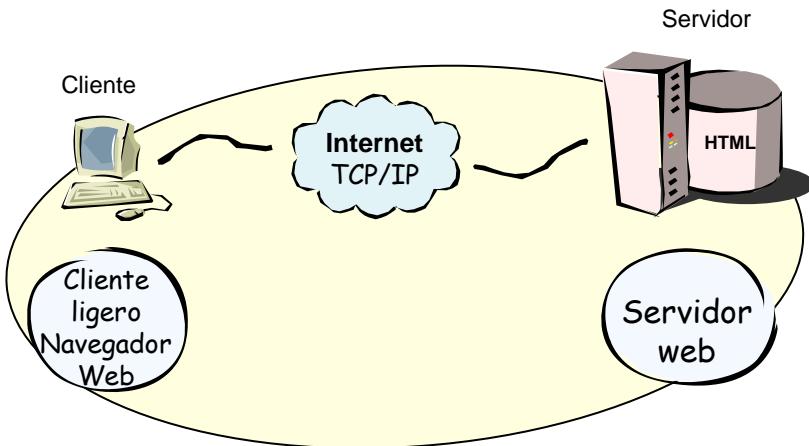
Revisión de tecnologías Web

Contenidos

Modelo HTTP Básico

Modelo HTTP Básico

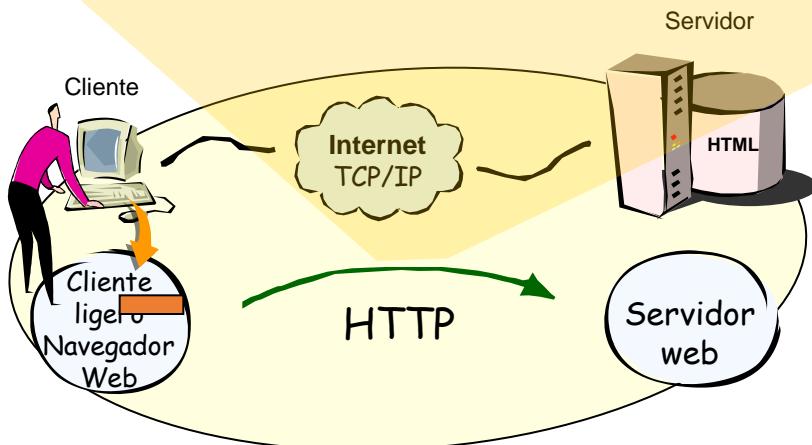
servicio HTTP básico



Modelo HTTP Básico

servicio HTTP básico

Método	Recurso	Versión	Cabeceras
GET	/index.html	HTTP/1.0	



Tecnologías Web

Servicio HTTP básico : XMLHttpRequest

Formato de una petición HTTP (HTTP Request)

Linea	Método dirección Recurso versión Protocolo
Cabecera	Etiqueta: valor Etiqeta: valor . . . Etiqueta: valor <línea en blanco>
Cuerpo	Información adicional en formato texto (opcional)

Principales métodos HTTP

- GET** Solicitar contenido
- HEAD** Solicitar únicamente la cabecera
- POST** Enviar datos al servidor (Crea nuevo contenido)
- PUT** Actualizar contenido existente
- DELETE** Eliminar contenido del servidor

Métodos HTTP: **GET, POST, PUT, DELETE, HEAD, ...**

Versión del Protocolo: HTTP/1.0, HTTP/1.1, ...

Etiquetas típicas de la **Cabecera**:

- Content-Length** indica el tamaño del cuerpo o **body**
- Content-Type** indica el tipo MIME del contenido del **body**
- Authorization** se emplea para enviar credenciales de acceso (como un token JWT)
- Connection** indica si la conexión TCP debe mantenerse abierta para otra petición HTTP

Tecnologías Web

Servicio HTTP básico : XMLHttpRequest

Cabecera

```
POST /cgi/miAplicacion.cgi HTTP/1.0
```

```
Accept: */*
Connection: Keep-Alive
User-Agent: Generic
Content-type: application/json
Content-length: 39
<línea en blanco>
```

Cuerpo

```
{"nombre":"Iren","email":"ilorenzo@dtic.ua.es"}
```

Ejemplo 1: petición HTTP tipo POST

Cabecera

```
GET /~ilorenzo/sod/index.html HTTP/1.0
```

```
Accept: */*
Connection: Keep-Alive
User-Agent: Generic
<línea en blanco>
```

Cuerpo

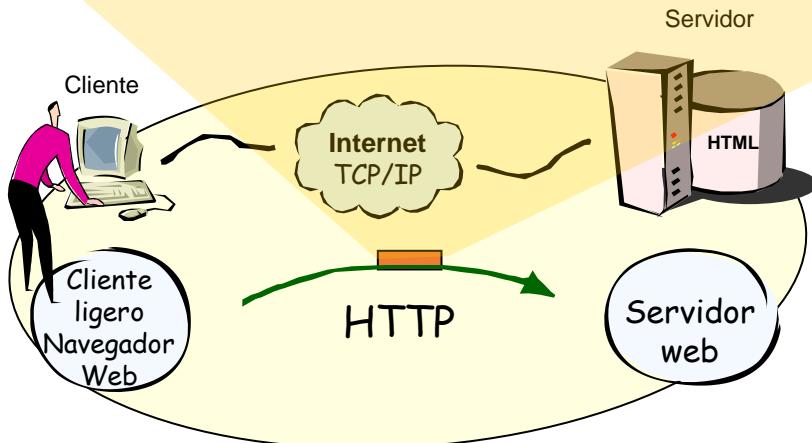
```
<vacío>
```

Ejemplo 2: petición HTTP tipo GET

Modelo HTTP Básico

servicio HTTP básico

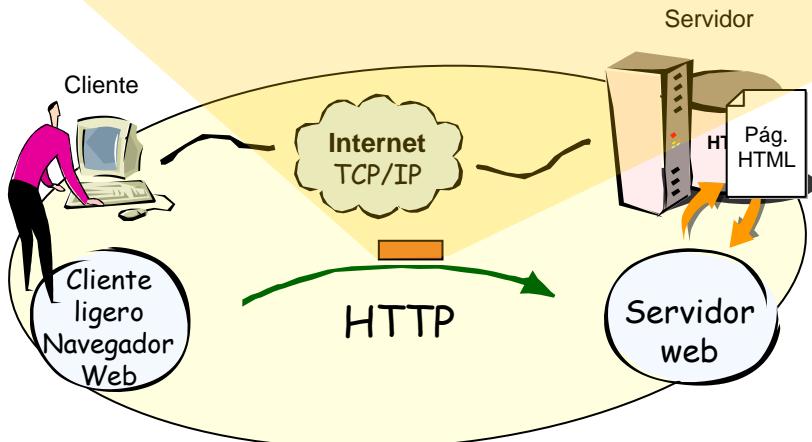
Método	Recurso	Versión	Cabeceras
GET	/index.html	HTTP/1.0	



Modelo HTTP Básico

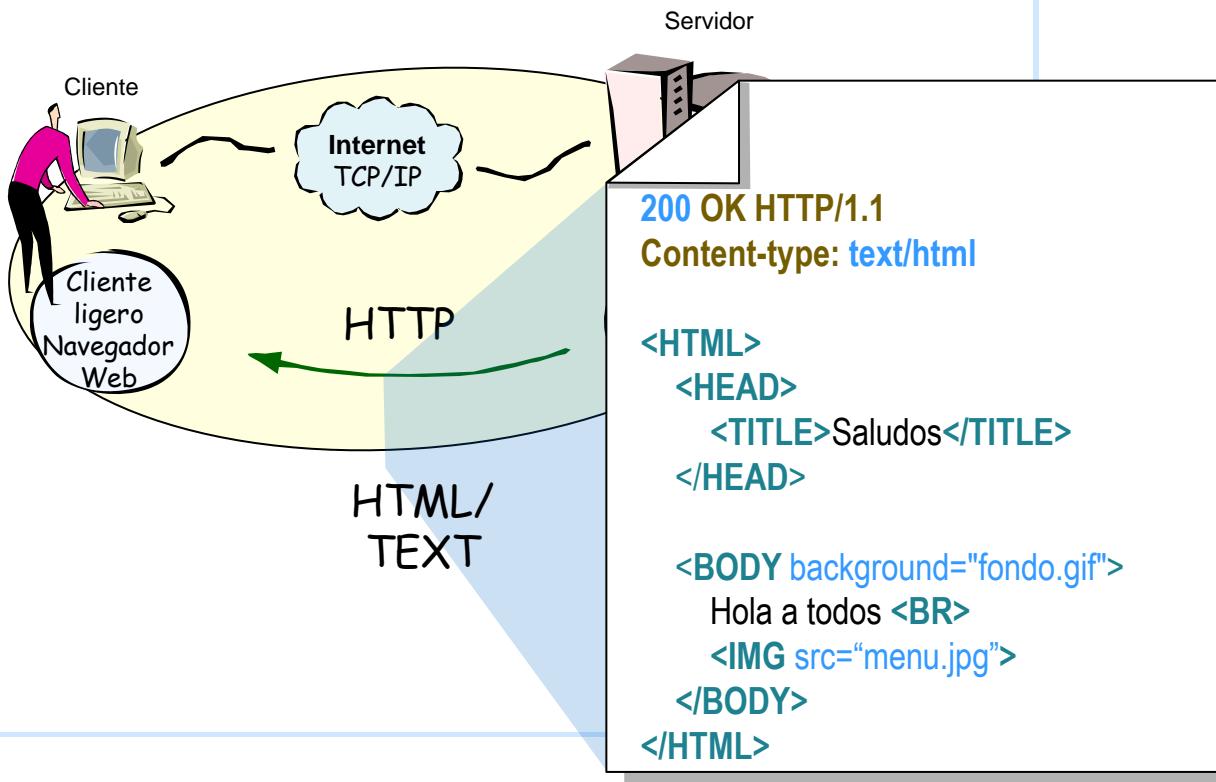
servicio HTTP básico

Método	Recurso	Versión	Cabeceras
GET	/index.html	HTTP/1.0	



Modelo HTTP Básico

servicio HTTP básico



Tecnologías Web

Servicio HTTP básico : **HTTPRespond**

Estado

HTTP/1.1 Código Estado Respuesta Mensaje Respuesta

Cabeceras

Etiqueta: valor

Etiqueta: valor

.

.

.

Etiqueta: valor

<línea en blanco que indica el final de las cabeceras>

Cuerpo

Información adicional en formato texto (el tipo se indica en la cabecera content-type)

El body o cuerpo es (opcional)

Formato de una respuesta HTTP (HTTP Response)

Según el **código de estado**, las respuestas se agrupan en cinco clases:

[100 – 199] Respuestas informativas

[200 – 299] Respuestas satisfactorias

[300 – 399] Redirecciones

[400 – 499] Errores de los clientes

[500 – 599] Errores de los servidores

Códigos de estado de respuesta HTTP

Tecnologías Web

Servicio HTTP básico : HTTPRespond

Códigos de estado de respuesta HTTP

1xx Mensajes de información	4xx Error por parte del cliente
100 Continua	400 Solicitud incorrecta
101 Cambio de protocolo	401 No autorizado
2xx Operación exitosa	402 Pago requerido
200 Ok	403 Prohibido
201 Creado	404 No encontrado
202 Aceptado	405 Método no permitido
203 Información no oficial	406 No aceptable
204 Sin Contenido	407 Proxy requerido
205 Contenido para reset	408 Tiempo de espera agotado
206 Contenido parcial	409 Conflicto
3xx Redirección hacia otro URL	410 No mpas disponible
300 Múlpiples posibilidades	411 Requiere logitud
301 Mudado permanentemente	412 Falló precondición
302 Encontrado	413 Entidad de solicitud demasiado larga
303 Véa otros	414 URI de solicitud demasiado largo
304 No modificado	415 Tipo de medio no soportado
305 Utilice un proxy	416 Rango solicitado no disponible
307 REDirección temporal	417 Falló expectativa
5xx Error por parte del servidor	
500 Error interno	
501 No implementado	
502 Pasarela incorrecta	
503 Servicio no disponible	
504 Tiempo de espera de la pasarela agotado	
505 Versión de <i>HTTP</i> no soportada	

Tecnologías Web

Servicio HTTP básico : HTTPRespond

Estado	HTTP/1.1 200 OK
Cabeceras	Date: Thu, 04 Nov 2004 17:59:15 GMT Server: Apache/2.0.40 (Red Hat Linux) Last-Modified: Wed, 03 Nov 2004 13:01:02 GMT ETag: "378029-902-3bd83f80" Accept-Ranges: bytes Content-Length: 256 Content-Type: text/html <línea en blanco>
Cuerpo	<html> <head> <title>Tecnología Informática y Computación</title> </head> <body background="fondo.gif"> Hola a todas y a todos </body> </html>

Ejemplo: respuesta HTTP

Tecnologías Web

Servicio HTTP básico : HTTPRespond

Content-Type: text/html

HTML (HiperText Markup Language)

- ① Documento de texto
- ② Etiquetas de formato
 - <etiqueta [arg1 arg2 ...]> ... [</etiqueta>]
- ③ Referencias cruzadas
 - texto descriptivo

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
<title>Mi Página Web</title>
</head>
<body>
  <h1>Bienvenidos a mi página web</h1>
  <p>Esta es una página web de ejemplo creada con HTML.</p>
</body>
</html>
```

Tecnologías Web

Servicio HTTP básico

: HTTPRespond

Estado	HTTP/1.1 200 OK
Cabeceras	Date: Thu, 04 Nov 2004 17:59:15 GMT Server: Apache/2.0.40 (Red Hat Linux) Last-Modified: Wed, 03 Nov 2004 13:01:02 GMT ETag: "378029-902-3bd83f80" Accept-Ranges: bytes Content-Type: application/json Content-Length: 158 Connection: close <línea en blanco>
Cuerpo	{ Nombre: "José", Apellido1: "Pérez", Apellido2: "Jiménez", Edad: "22", Direccion: "C/ General Mola, 7", CP: "04302", EMail: "jperez@gmail.com", }

Ejemplo: respuesta HTTP enviando un objeto JSON

Tecnologías Web

Servicio HTTP básico : **MIME**

Content-Type: **MIME** (Multipurpose Internet Mail Extension)

Texto sin formato: text/plain

- **HTML:** text/html
- **JSON:** application/json
- **XML:** application/xml
- **JavaScript:** application/javascript
- **CSS:** text/css

Imágenes:

- image/jpeg
- image/png
- image/gif
- image/webp
- image/svg+xml

Audio:

- audio/mpeg
- audio/ogg
- audio/wav

Archivos:

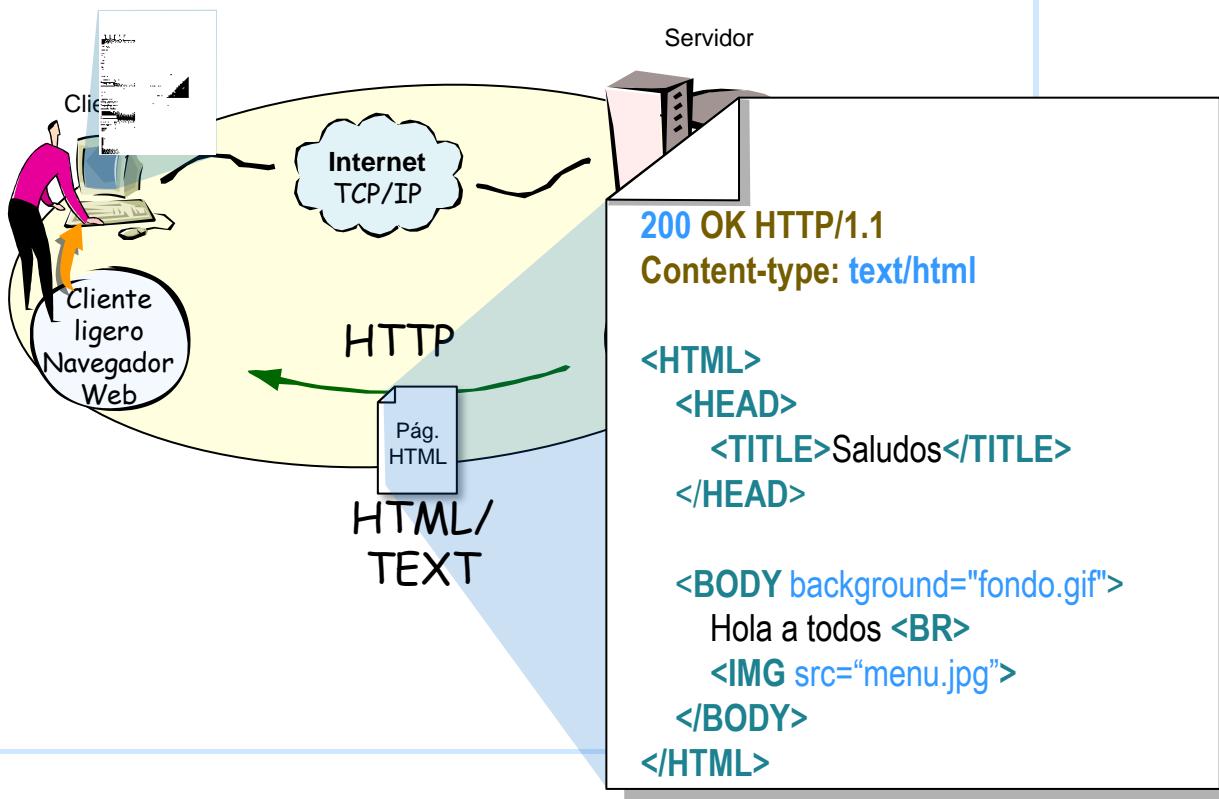
- application/pdf
- application/zip
- application/vnd.ms-excel (para archivos Excel)
- application/msword (para archivos Word)

Video:

- video/mp4
- video/webm
- video/ogg

Modelo HTTP Básico

servicio HTTP básico



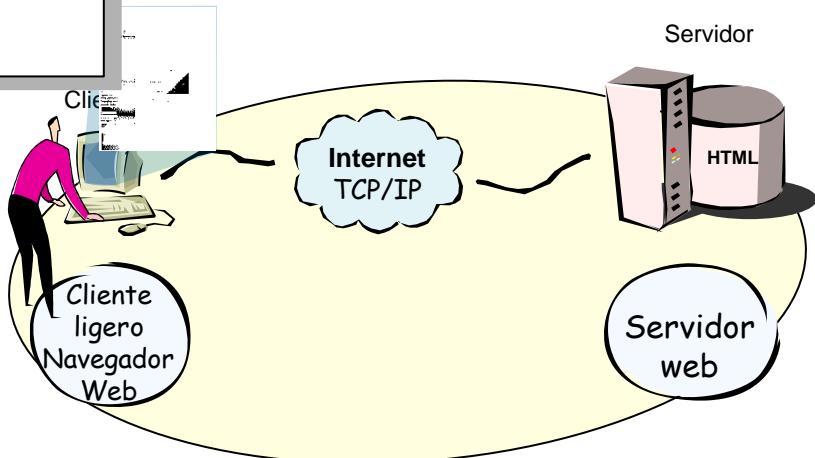
Modelo HTTP Básico

200 OK HTTP/1.1
Content-type: text/html

```
<HTML>
  <HEAD>
    <TITLE>Saludos</TITLE>
  </HEAD>

  <BODY background="fondo.gif">
    Hola a todos <BR>
    <IMG src="menu.jpg">
  </BODY>
</HTML>
```

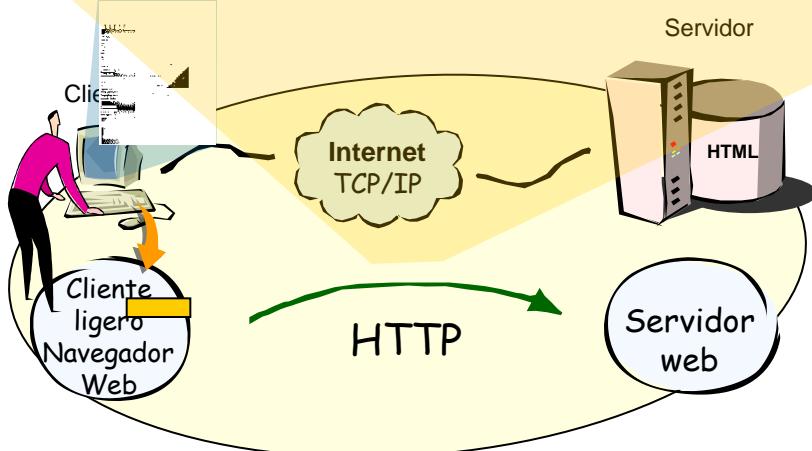
HTTP básico



Modelo HTTP Básico

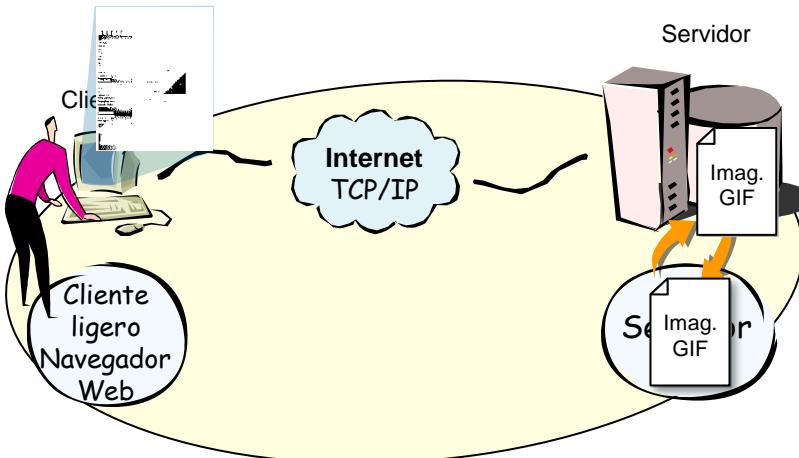
servicio HTTP básico

Método	Recurso	Versión	Cabeceras
GET	/fondo.gif	HTTP/1.0	



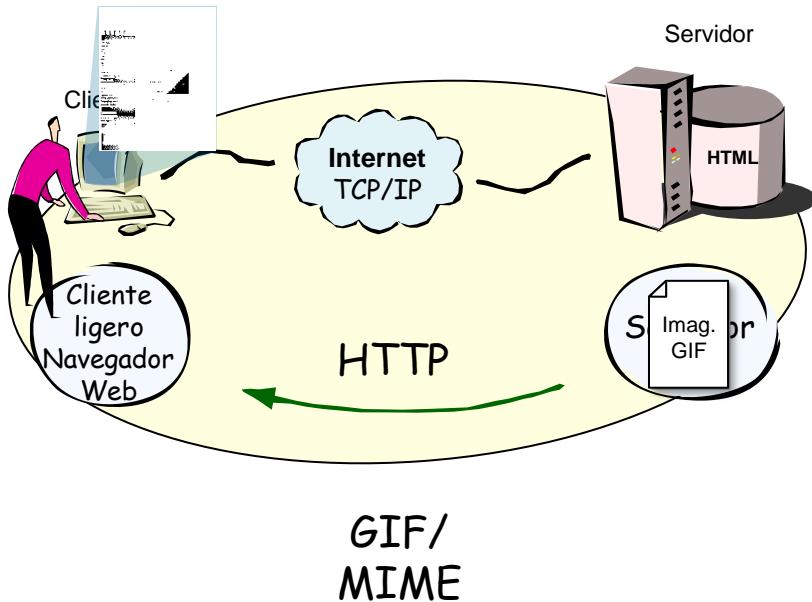
Modelo HTTP Básico

servicio HTTP básico



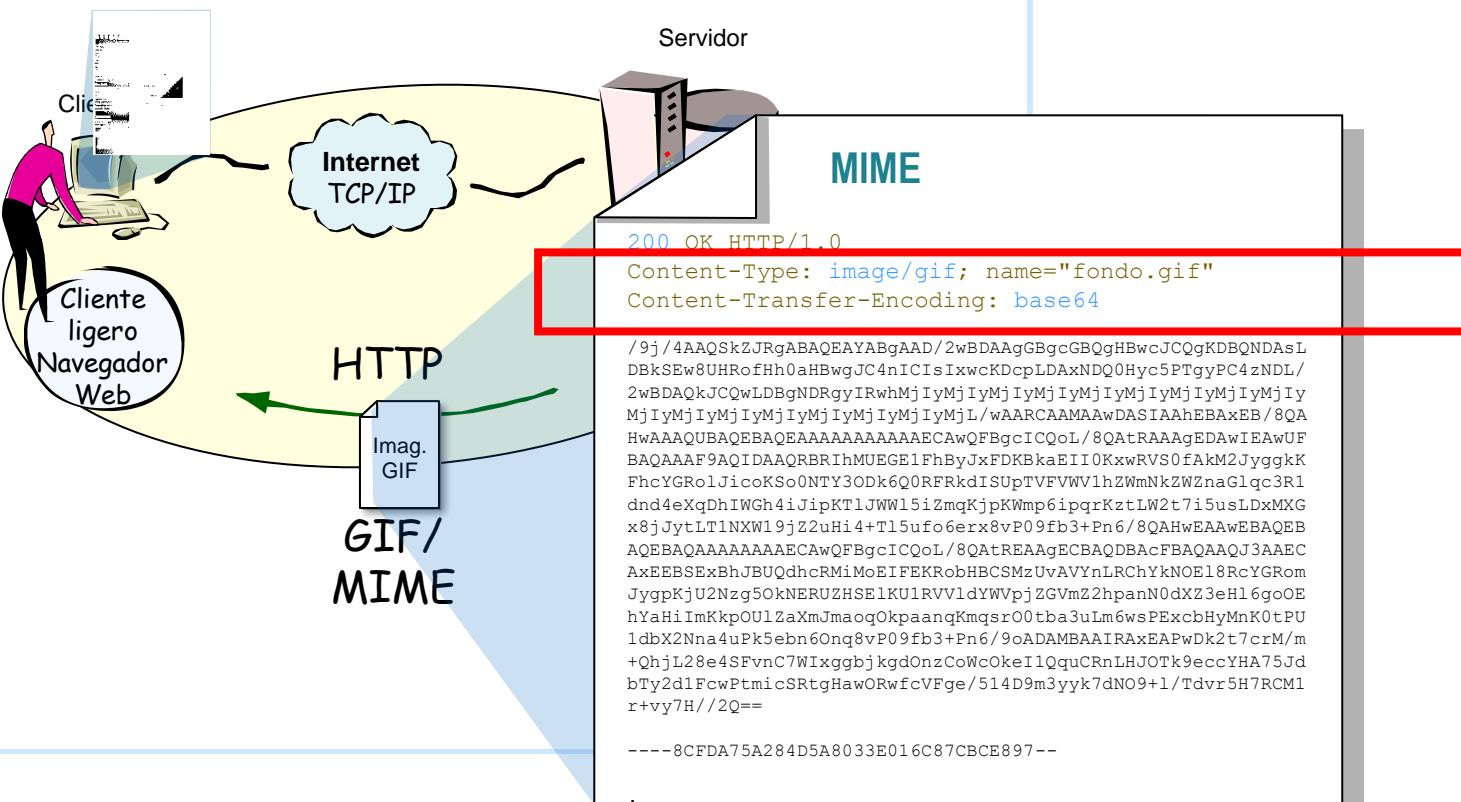
Modelo HTTP Básico

servicio HTTP básico



Modelo HTTP Básico

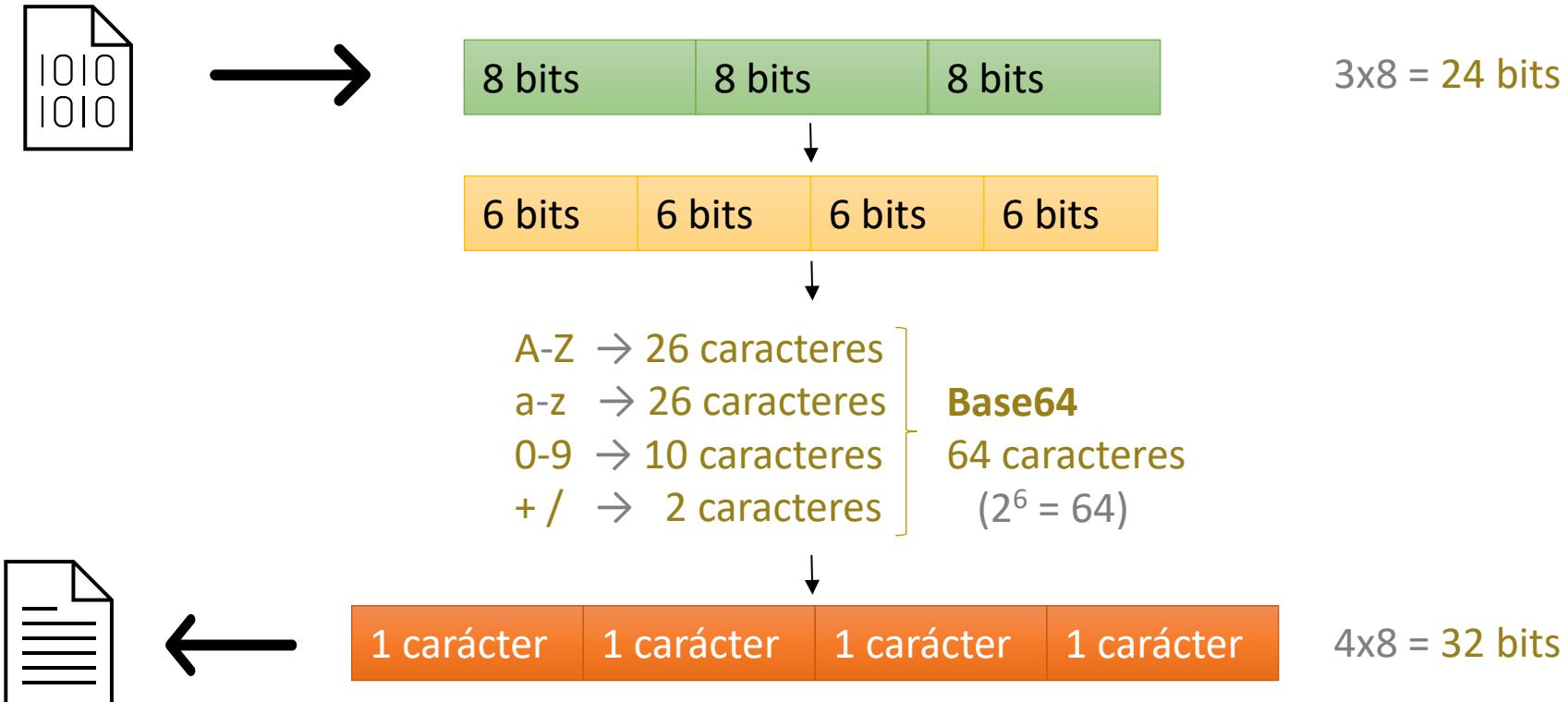
servicio HTTP básico



05 Tecnologías Web

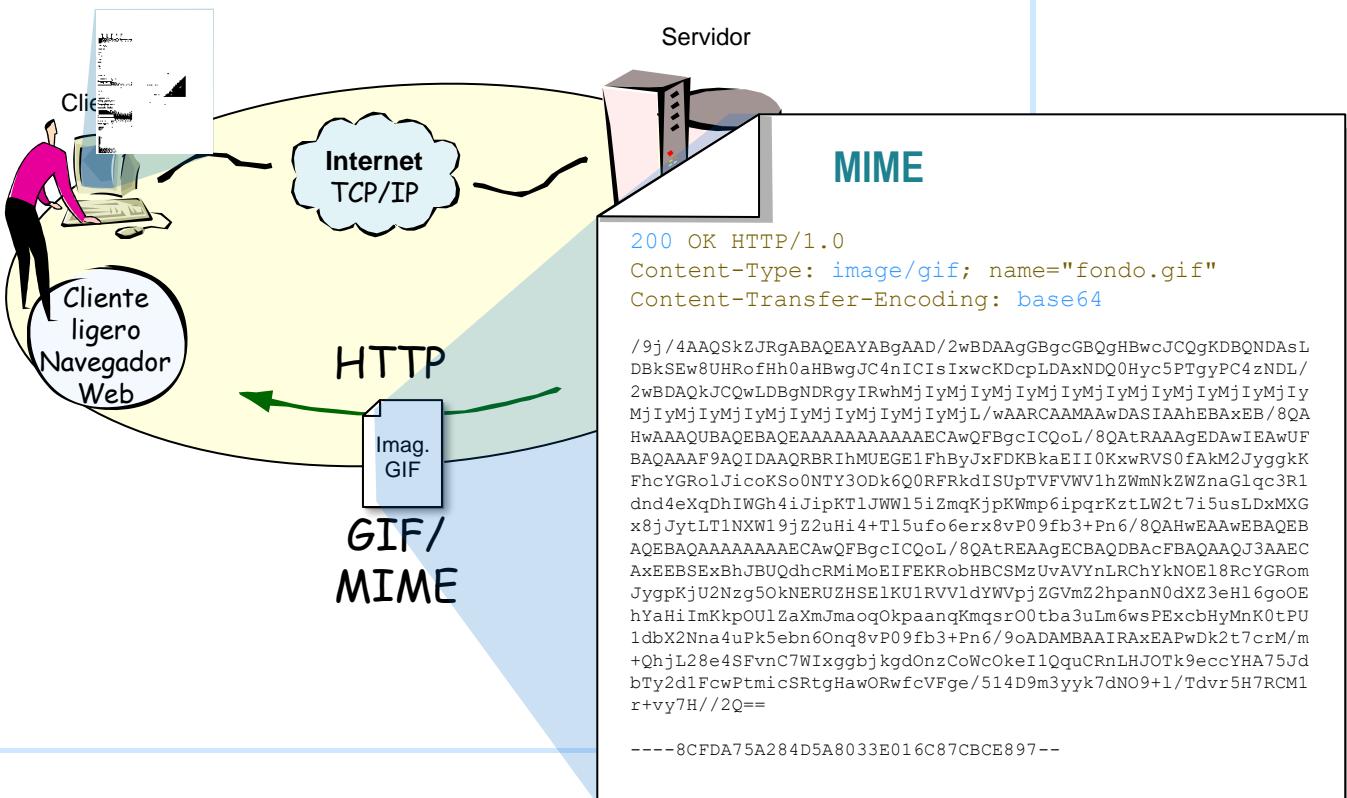
Modelos Básicos → Servicio HTTP básico

MIME (Multipurpose Internet Mail Extension)



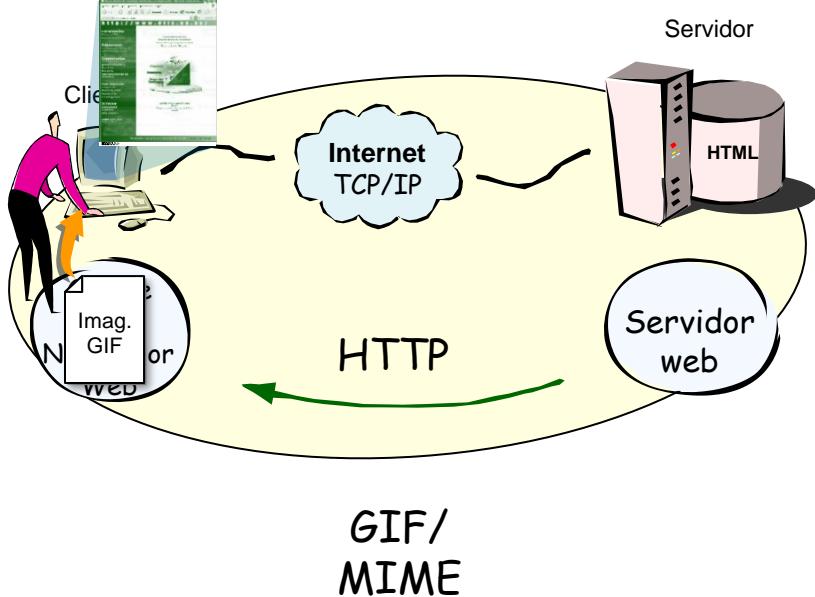
Modelo HTTP Básico

servicio HTTP básico



Modelo HTTP Básico

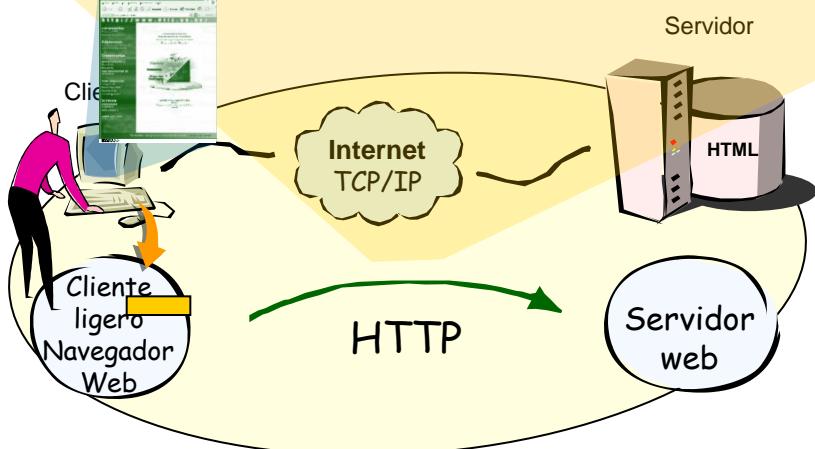
servicio HTTP básico



Modelo HTTP Básico

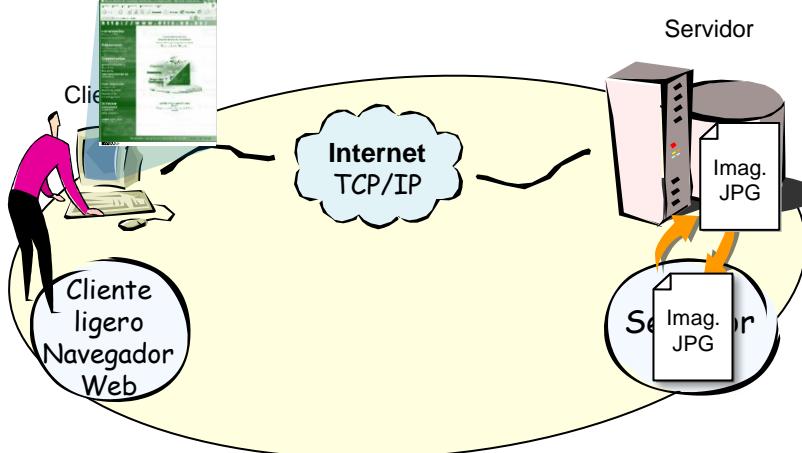
servicio HTTP básico

Método	Recurso	Versión	Cabeceras
GET	/menu.jpg	HTTP/1.0	



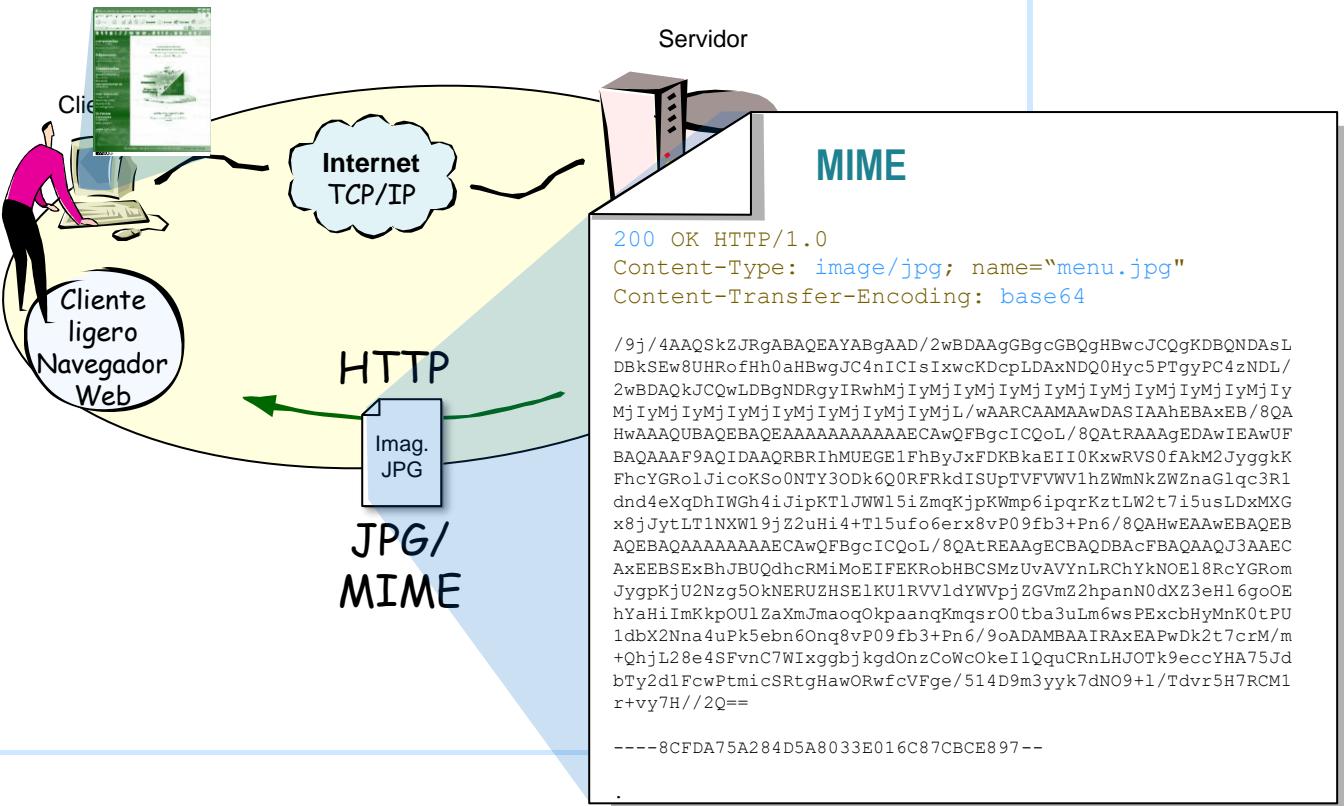
Modelo HTTP Básico

servicio HTTP básico



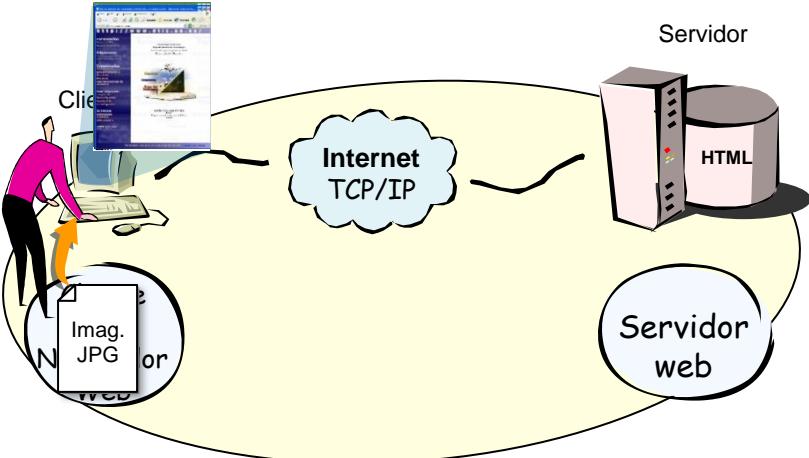
Modelo HTTP Básico

servicio HTTP básico

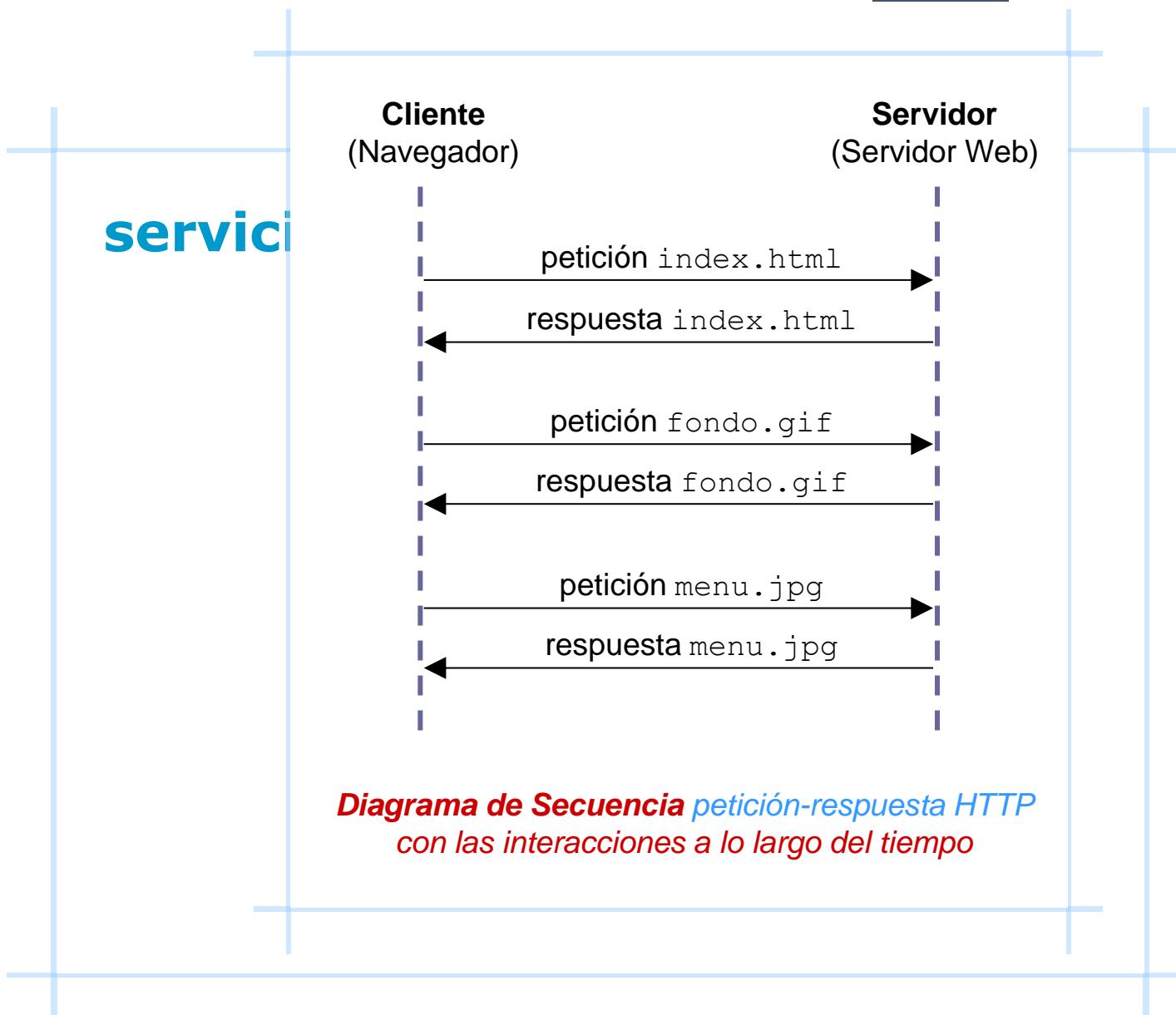


Modelo HTTP Básico

servicio HTTP básico

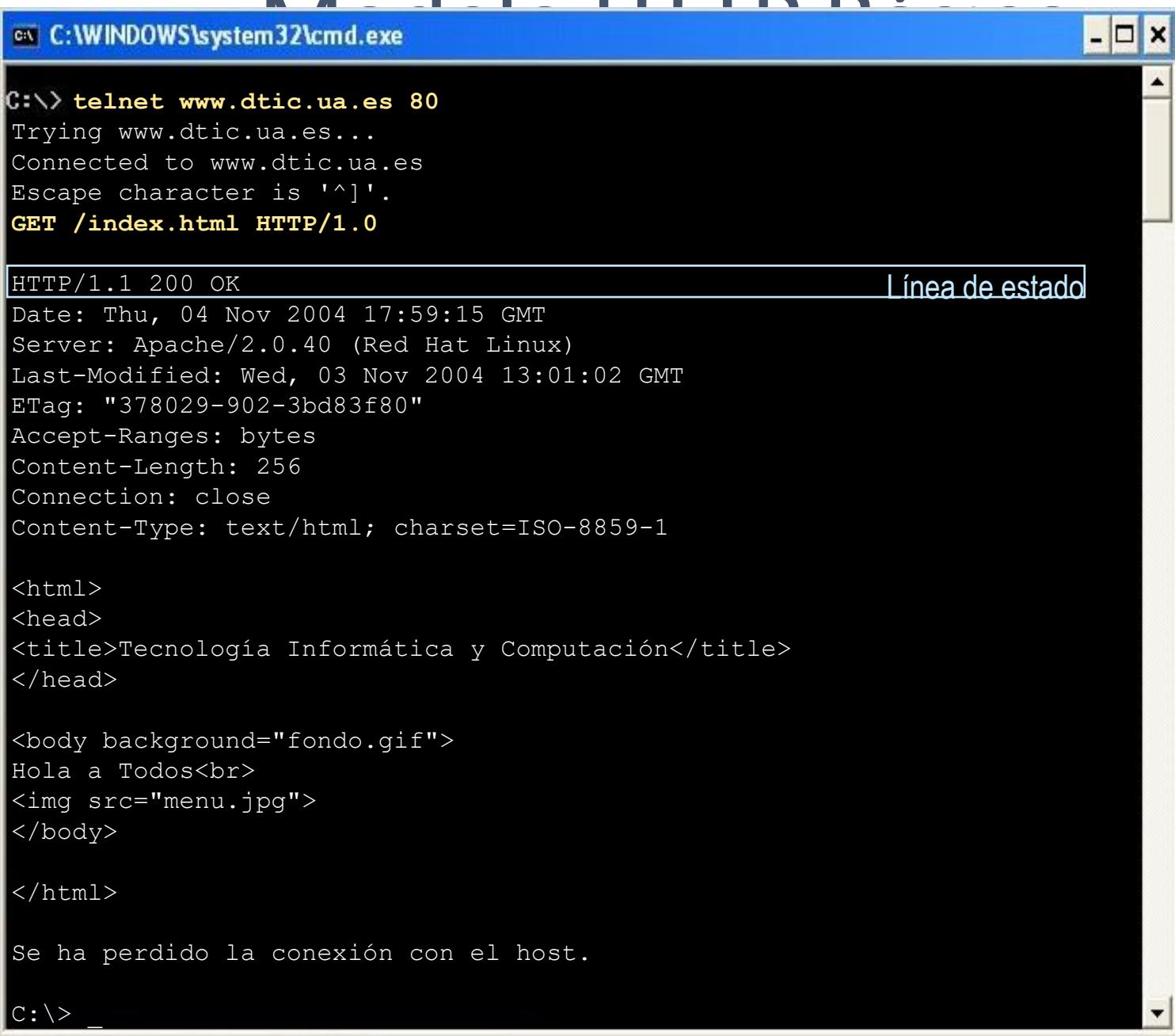


Modelo HTTP Básico



Revisión de tecnologías Web

serv



The screenshot shows a Windows command-line window titled "C:\WINDOWS\system32\cmd.exe". The user has run the command "telnet www.dtic.ua.es 80". The server responded with an "HTTP/1.1 200 OK" status line. The "Línea de estado" (status line) is highlighted in yellow. The response body contains the HTML code for the website, including the title "Tecnología Informática y Computación" and some introductory text. The session ends with a message about losing the connection.

```
C:\> telnet www.dtic.ua.es 80
Trying www.dtic.ua.es...
Connected to www.dtic.ua.es
Escape character is '^]'.
GET /index.html HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 04 Nov 2004 17:59:15 GMT
Server: Apache/2.0.40 (Red Hat Linux)
Last-Modified: Wed, 03 Nov 2004 13:01:02 GMT
ETag: "378029-902-3bd83f80"
Accept-Ranges: bytes
Content-Length: 256
Connection: close
Content-Type: text/html; charset=ISO-8859-1

<html>
<head>
<title>Tecnología Informática y Computación</title>
</head>

<body background="fondo.gif">
Hola a Todos<br>

</body>

</html>

Se ha perdido la conexión con el host.

C:\>
```

Revisión de tecnologías Web

serv

```
C:\> telnet www.dtic.ua.es 80
Trying www.dtic.ua.es...
Connected to www.dtic.ua.es
Escape character is '^]'.
GET /index.html HTTP/1.0

HTTP/1.1 200 OK                                Línea de estado
Date: Thu, 04 Nov 2004 17:59:15 GMT
Server: Apache/2.0.40 (Red Hat Linux)
Last-Modified: Wed, 03 Nov 2004 13:01:02 GMT
ETag: "378029-902-3bd83f80"
Accept-Ranges: bytes
Content-Length: 256
Connection: close
Content-Type: text/html; charset=ISO-8859-1

<html>
<head>
<title>Tecnología Informática y Computación</title>
</head>

<body background="fondo.gif">
Hola a Todos<br>

</body>

</html>

Se ha perdido la conexión con el host.

C:\>
```

Revisión de tecnologías Web

serv

```
C:\> telnet www.dtic.ua.es 80
Trying www.dtic.ua.es...
Connected to www.dtic.ua.es
Escape character is '^]'.
GET /index.html HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 04 Nov 2004 17:59:15 GMT
Server: Apache/2.0.40 (Red Hat Linux)
Last-Modified: Wed, 03 Nov 2004 13:01:02 GMT
ETag: "378029-902-3bd83f80"
Accept-Ranges: bytes
Content-Length: 256
Connection: close
Content-Type: text/html; charset=ISO-8859-1
[línea en blanco]

<html>
<head>
<title>Tecnología Informática y Computación</title>
</head>

<body background="fondo.gif">
Hola a Todos<br>

</body>

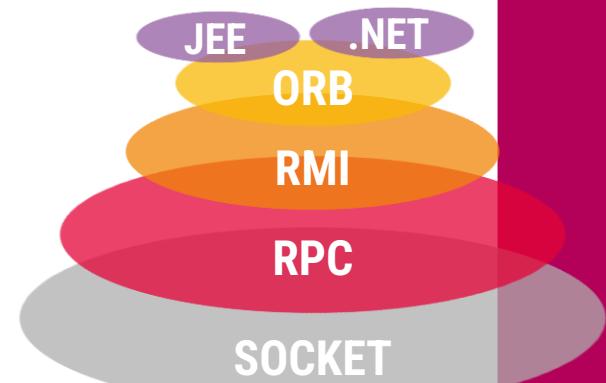
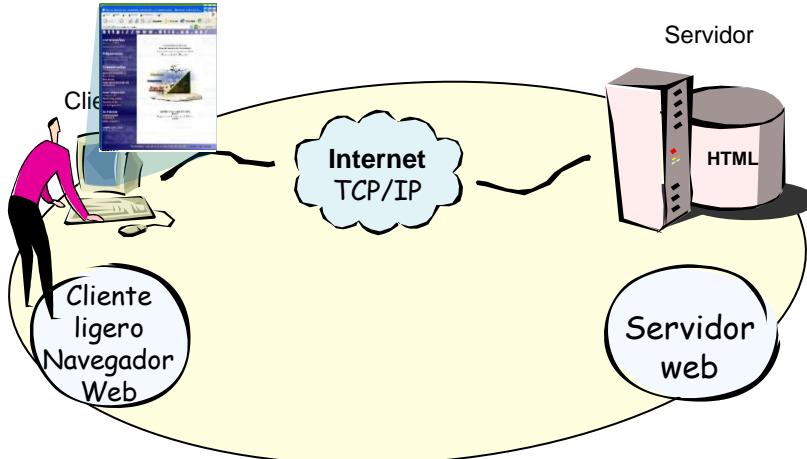
</html>

Se ha perdido la conexión con el host.

C:\>
```

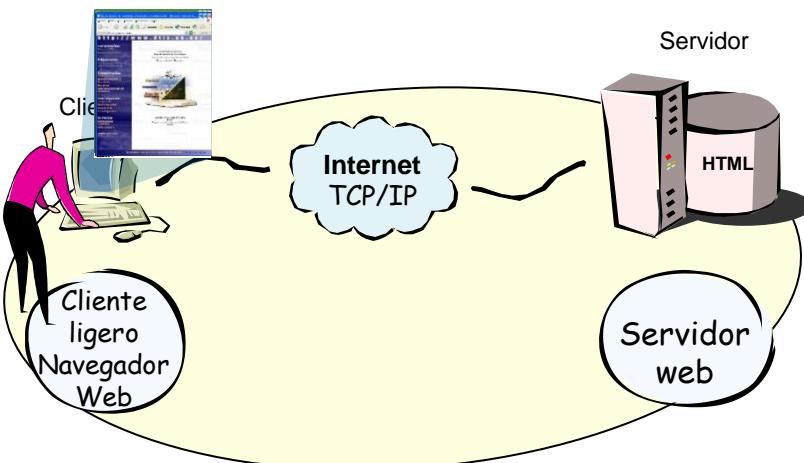
Modelo HTTP Básico

servicio HTTP básico

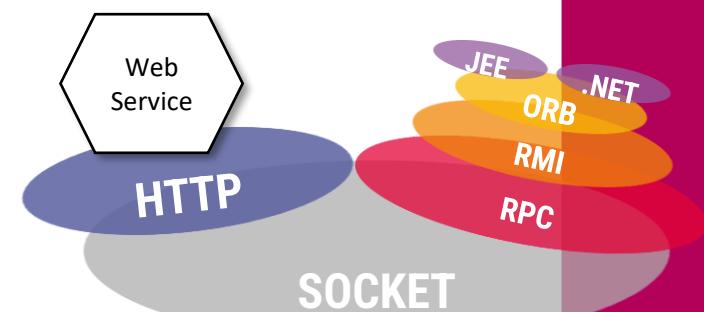


Modelo HTTP Básico

servicio HTTP básico



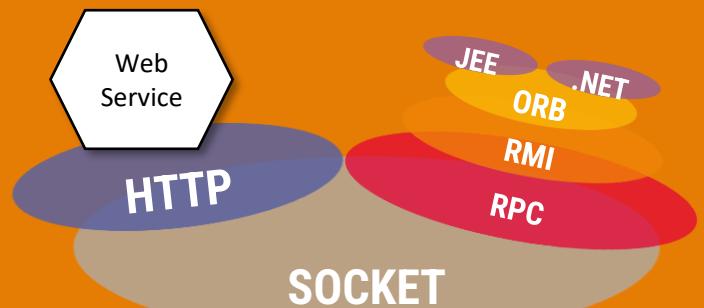
- Uso como **protocolo de transporte**
- **Aprovecha** las arquitecturas de red basadas en **firewalls y proxis**
- Orientado a **conexión**
- **Sin estado** (caché y balanceadores)



- Definición
- SOA
- REST
- Otras tecnologías

Servicios Web

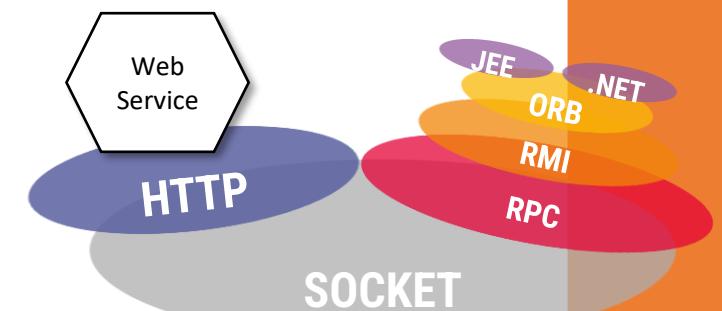
Contenidos



Servicios WEB

Características

- **Evolución** natural de los **sistemas distribuidos** y la necesidad de comunicación entre aplicaciones en diferentes plataformas



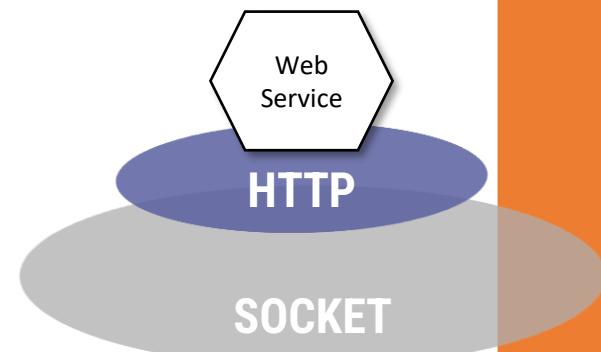
Servicios Web

Servicios WEB

Características

- **Evolución** natural de los **sistemas distribuidos** y la necesidad de comunicación entre aplicaciones en diferentes plataformas

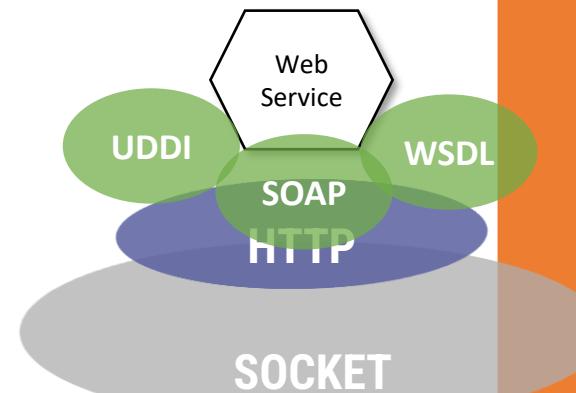
Servicios Web



Servicios WEB

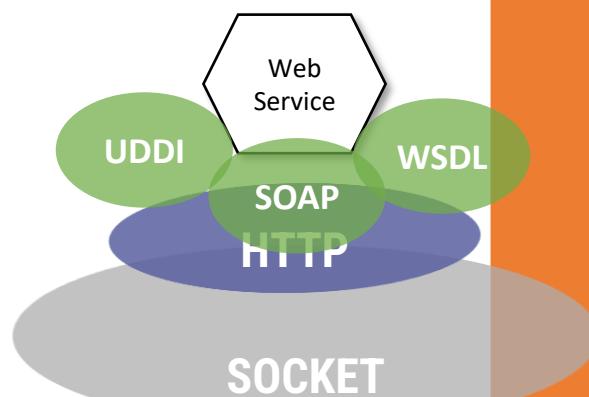
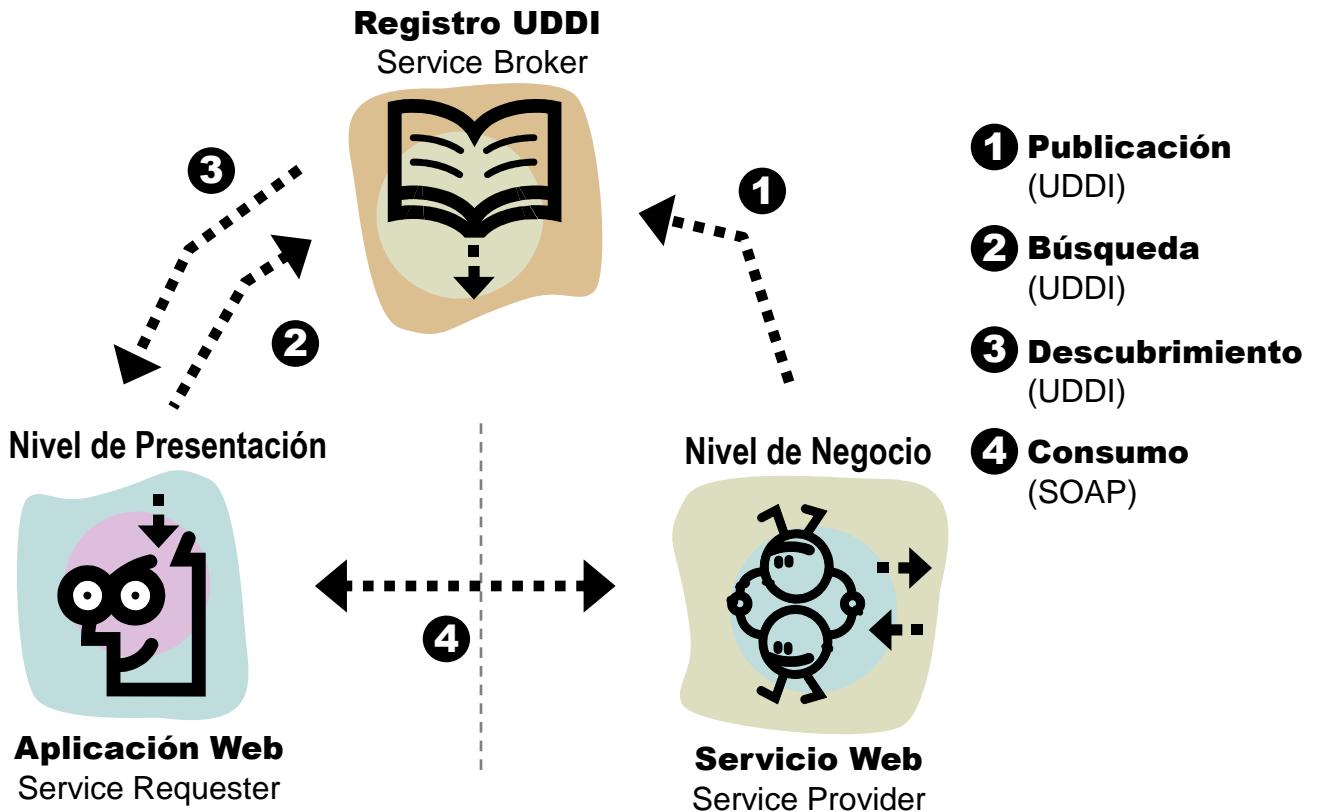
Características

- **Evolución** natural de los **sistemas distribuidos** y la necesidad de comunicación entre aplicaciones en diferentes plataformas
- **SOAP** (**Protocolo de acceso a objetos simples**) y el lenguaje de descripción de servicios web (**WSDL**) fue lo que realmente impulsó la adopción generalizada de los servicios web
- Se **consolidó** aún más con **UDDI** (Universal Description, Discovery, and Integration) para el **registro y descubrimiento** de servicios web



Servicios WEB

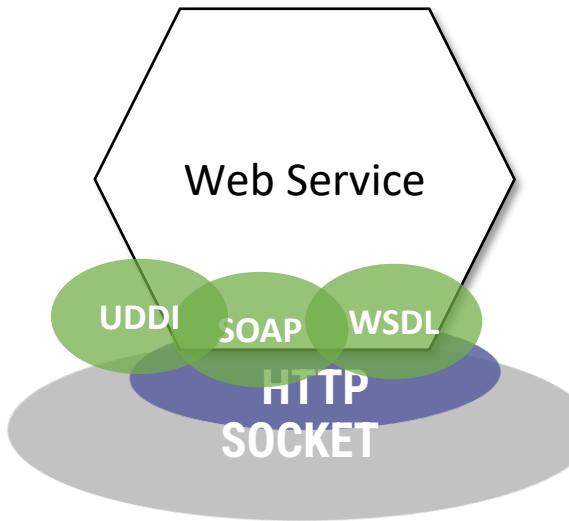
SOA



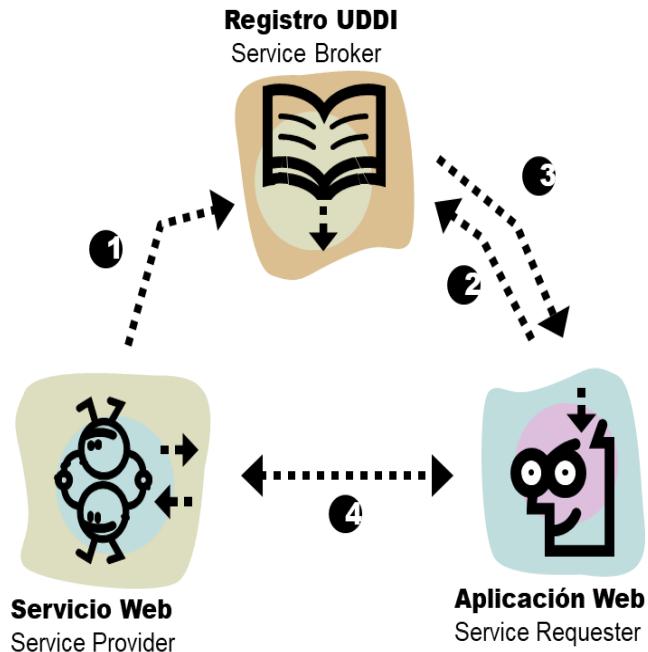
Servicios Web

Servicios WEB

Características



SOA

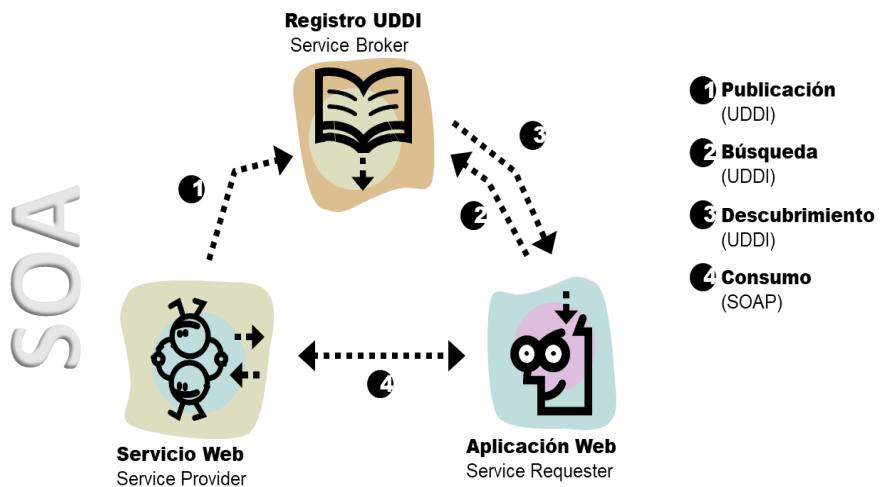
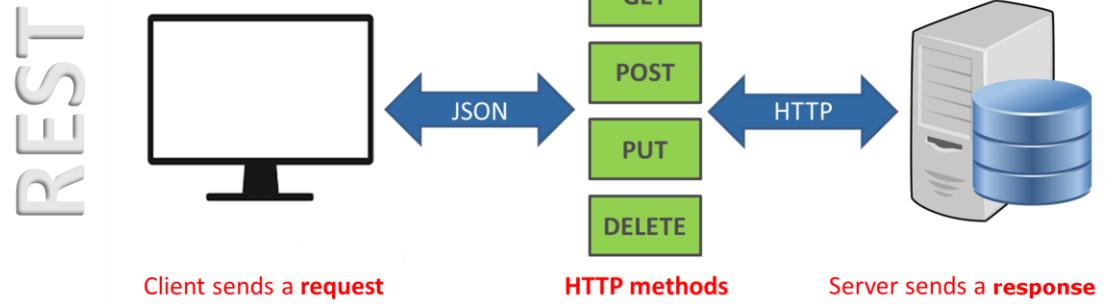
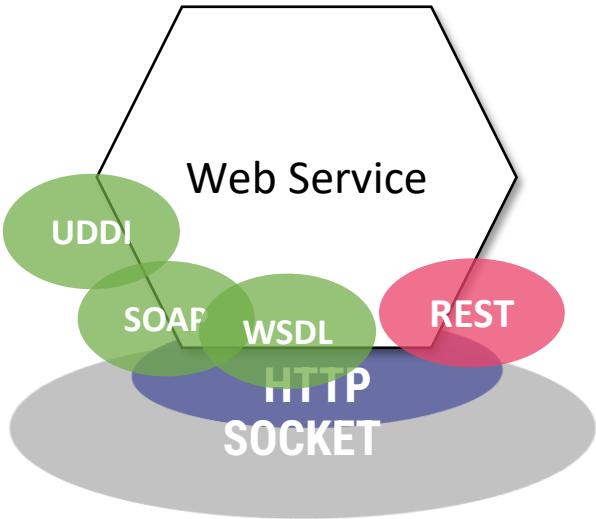


- 1 **Publicación** (UDDI)
- 2 **Búsqueda** (UDDI)
- 3 **Descubrimiento** (UDDI)
- 4 **Consumo** (SOAP)

Servicios Web

Servicios WEB

Características



Servicios Web

REST

REST (Representational State Transfer)

- p.v. Programador: conjunto de **principios y buenas prácticas** para la **interacción** entre distintos **componentes**.

01

Alternativa a SW tradicionales

Alternativa más ligera a Servicios Web tradicionales (basados en: RPC, SOAP o WSDL) para mantener los principios de SOA (Arquitecturas Orientadas a Servicios).

02

Protocolo

El protocolo más usado que cumple esta definición es **HTTP**

- Toda aplicación web bajo HTTP es una aplicación REST.
- Aunque no necesariamente son servicios web RESTful.

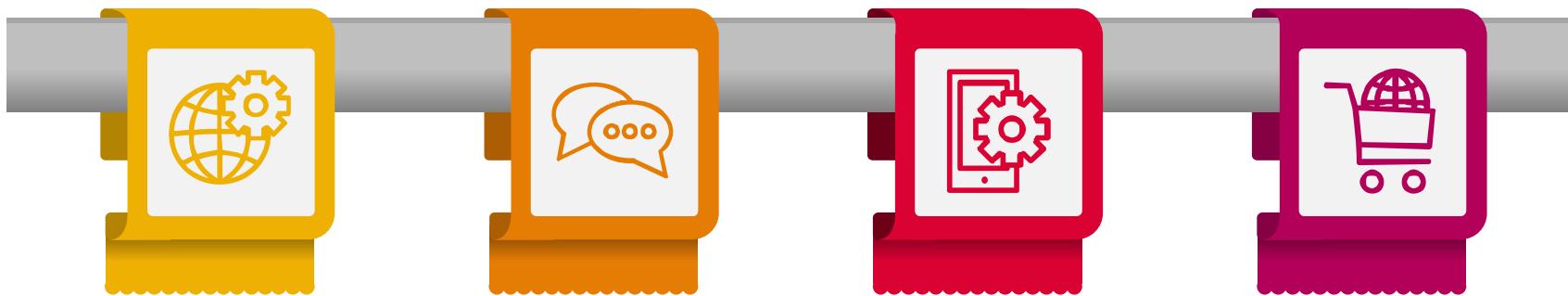
03

Selección

La **decisión** sobre la **tecnología** dependerá de las **características** del proyecto y requiere un análisis.

REST

Reglas básicas de una aplicación REST para alcanzar objetivos



Arquitectura cliente-servidor

Separación e independencia entre los 2 agentes básicos que intervienen: el cliente y el servidor

Stateless

El servidor no tiene por qué almacenar datos del cliente para mantener su estado.

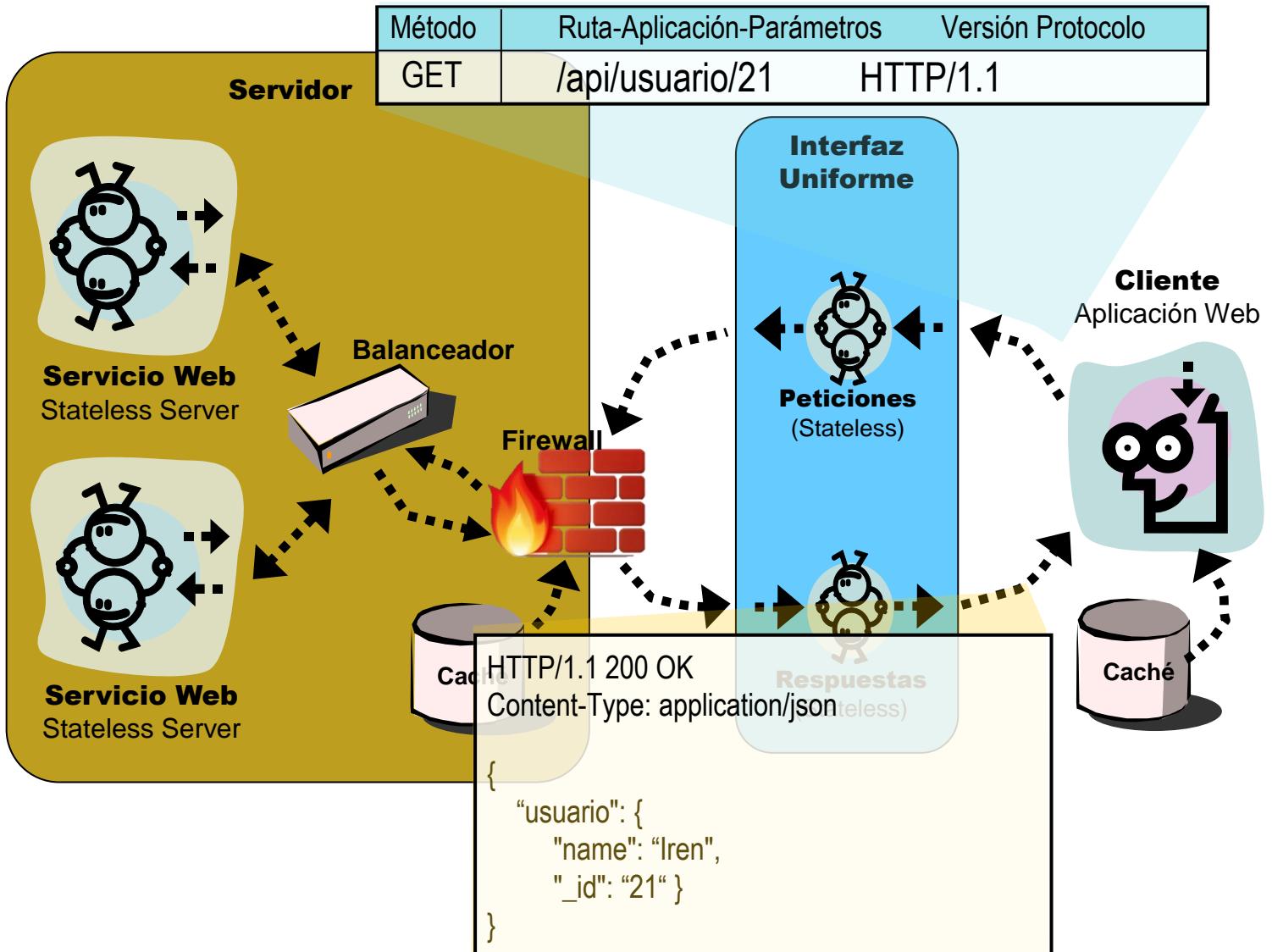
“Cacheable”

El servidor que sirve las peticiones del cliente debe definir algún modo de cachear dichas peticiones para aumentar el rendimiento, escalabilidad, etc. HTTP implementa esto con la cabecera Cache-control, mediante varios parámetros

Interfaz uniforme

Todos los servicios REST compartirán una forma de invocación y métodos uniforme utilizando los métodos GET, DELETE,...

Arquitectura REST



Servicio Web RESTful

¿Cómo diseñar un servicio Web REST?

Crear una URL para cada recurso. Los recursos deberían ser nombres **no verbos** (acciones).



URL



Identificar

Identificar todas las **entidades** conceptuales que se desea exponer como servicio

Servicio Web RESTful

¿Cómo diseñar un servicio Web REST?

GET
Todos los recursos accesibles mediante **GET** **no** deberían tener **efectos** secundarios. Los recursos deberían devolver la representación del recurso. Por tanto, invocar al recurso **no** debería ser el **resultado de modificarlo**.



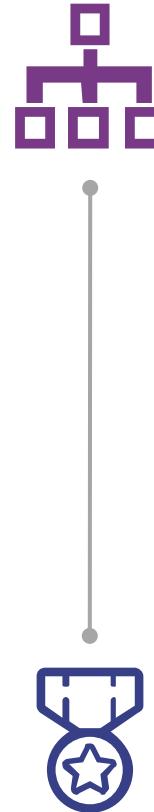
Categorizar

Categorizar los **recursos** de acuerdo con si los clientes pueden obtener una **representación** del recurso o si pueden **modificarlo**. Para el primero, debemos hacer los recursos accesibles utilizando un HTTP GET. Para el último, debemos hacer los recursos accesibles mediante los verbos HTTP: POST, PUT y DELETE.

Servicio Web RESTful

— ¿Cómo diseñar un servicio Web REST?

Descripción
Describir cómo ha de ser invocado nuestro servicio mediante **OpenAPI**



Hipervínculo

Ninguna representación debería estar **aislada**. Es recomendable poner hipervínculos dentro de la representación de un recurso para permitir a los clientes obtener **más información**.

Servicio Web RESTful

REST, API REST y API RESTful

REST

Estilo de arquitectura de software que define un conjunto de **restricciones** y **principios** para el diseño de servicios web:

- Comunicación **sin estado** entre el cliente y el servidor
- Capacidad de **cachear** datos
- Uso de operaciones **CRUD** (Crear, Leer, Actualizar, Borrar) sobre recursos identificables mediante URLs

API REST

Implementación de una **API** que sigue los principios de **REST**.

El API debe cumplir con todas las restricciones de REST:

- **Métodos HTTP** de manera apropiada (GET, POST, PUT, DELETE, etc.)
- Manipulación de recursos a través de representaciones de estado,
- Hipermedios para la navegación entre recursos

Servicios Web

Servicio Web RESTful

REST, API REST y API RESTful

REST

Estilo de arquitectura de software que define un conjunto de **restricciones** y **principios** para el diseño de servicios web:

- Comunicación **sin estado** entre el cliente y el servidor
- Capacidad de **cachear** datos
- Uso de operaciones **CRUD** (Crear, Leer, Actualizar, Borrar) sobre recursos identificables mediante URLs

API RESTful

Implementación de una **API** que sigue los principios de **REST**.

El API debe cumplir con todas las restricciones de REST:

- **Métodos HTTP** de manera apropiada (GET, POST, PUT, DELETE, etc.)
- Manipulación de recursos a través de representaciones de estado,
- Hipermedios para la navegación entre recursos

Servicios Web

Servicio Web RESTful

REST, API REST y API RESTful

REST

Estilo de arquitectura de software que define un conjunto de **restricciones** y **principios** para el diseño de servicios web:

- Comunicación **sin estado** entre el cliente y el servidor
- Capacidad de **cachear** datos
- Uso de operaciones **CRUD** (Crear, Leer, Actualizar, Borrar) sobre recursos identificables mediante URLs

API RESTful

Implementación de una **API** que sigue los principios de **REST**.

El API debe cumplir con todas las restricciones de REST:

- **Métodos HTTP** de manera apropiada (GET, POST, PUT, DELETE, etc.)
- Manipulación de recursos a través de representaciones de estado,
- Hipermedios para la navegación entre recursos

API REST

Implementación de una **API** que **NO** sigue **estrictamente** todos los principios de **REST**.

Servicios Web

Servicio Web RESTful

Ejemplo de implementación

Un ejemplo de una **API REST** sería una API que proporciona acceso a una base de datos de libros con estos **endpoints**:

- **GET /obtenerLibros**: Para obtener una lista de todos los libros.
- **GET /obtenerLibro/{id}**: Para obtener detalles sobre un libro específico.
- **POST /crearLibro**: Para agregar un nuevo libro a la base de datos.
- **POST /actualizarLibro/{id}**: Para actualizar la información de un libro existente.
- **GET /eliminarLibro/{id}**: Para eliminar un libro de la base de datos.

Endpoints en una **API RESTful** que por tanto sigue más de cerca los principios de REST y hace una definición correcta:

- **GET /libros**: Para obtener una lista de todos los libros.
- **GET /libros/{id}**: Para obtener detalles sobre un libro específico.
- **POST /libros**: Para agregar un nuevo libro a la base de datos.
- **PUT /libros/{id}**: Para actualizar la información de un libro existente.
- **DELETE /libros/{id}**: Para eliminar un libro de la base de datos.

API REST

VS

API RESTful

Otras Tecnologías

Otras Tecnologías

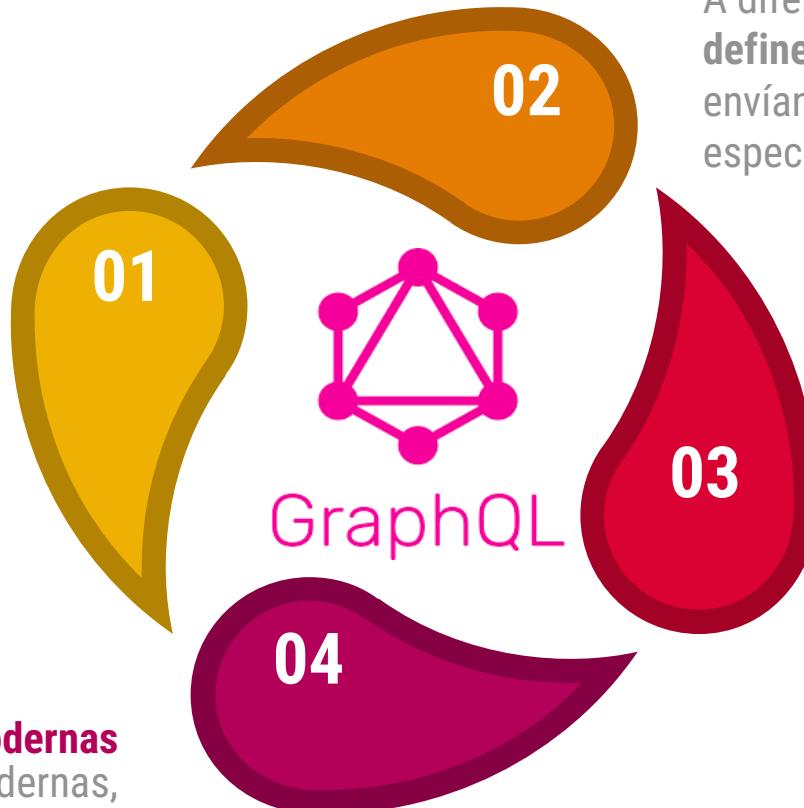
GraphQL

Alternativa

Alternativa relativamente nueva (2015) que permite a los clientes solicitar solo los datos que necesitan

Aplicaciones modernas

Popular en aplicaciones modernas, especialmente en entornos donde se necesita una recuperación de datos eficiente y personalizada



Especificar datos

A diferencia de REST, donde el **servidor define la estructura de los datos** que se envían, en GraphQL, los clientes pueden especificar qué datos desean recuperar

Reduce tráfico

Esto **reduce el exceso de datos** transmitidos y permite una mayor flexibilidad en la comunicación entre aplicaciones

Otras Tecnologías

Websocket

Comunicación

Proporciona un canal de comunicación **bidireccional** (cliente y servidor pueden iniciar comunicación), **full-duplex**

Alternativa a REST (en realidad a HTTP)

- Protocolo de más bajo nivel
- Protocolo **con estado**
- Protocolo **más ligero**



TCP

Utiliza una conexión **TCP** única (frente a REST)

Protocolo de comunicación

WebSocket es un **protocolo de comunicaciones** mientras que REST es un patrón arquitectónico basado en HTTP

Otras Tecnologías

gRPC

Uso

Ha ganado popularidad en **aplicaciones distribuidas y microservicios**

Streaming Bidireccional

Admite el **streaming bidireccional**, lo que permite la transmisión de datos de manera eficiente en ambas direcciones entre el cliente y el servidor..



Soporte Multilenguaje

Utiliza **Protocol Buffers** para definir sus servicios y mensajes, lo que permite una **serialización binaria altamente eficiente** y un tipado fuerte en la comunicación entre servicios

Comunicación

Especialmente diseñado para entornos donde se necesitan **comunicaciones eficientes** y de **baja latencia**

Protocolo

Utiliza **HTTP/2** como protocolo subyacente, lo que permite la transmisión de datos de manera más eficiente que HTTP/1.x utilizado por REST y GraphQL

Selección de Tecnología

Resumen

	Para aplicaciones donde la interoperabilidad y la simplicidad son prioritarias, especialmente para operaciones CRUD	Aplicaciones web y móviles que requieren acceso a recursos a través de HTTP Ejemplos: APIs de redes sociales como Twitter, Facebook , APIs de servicios de pago como Stripe
	Para aplicaciones complejas que requieren flexibilidad en las consultas y manejo eficiente de datos complejos.	Aplicaciones con múltiples vistas de datos Ejemplos: APIs de GitHub, Yelp , APIs internas de Facebook . Suele ser utilizado de cara a la aplicación cliente.
	Para aplicaciones que requieren alta velocidad, eficiencia y comunicación entre microservicios .	Aplicaciones con requisitos de rendimiento y escalabilidad. Comunicación entre microservicios en un entorno de microservicios. Ejemplos: Servicios internos de Google Suele ser utilizado para comunicación servicios backend.
	Para aplicaciones que requieren comunicación bidireccional persistente y tiempo real .	Aplicaciones de chat en tiempo real, Juegos multijugador en tiempo real , Aplicaciones de seguimiento en tiempo real. Ejemplos: Aplicaciones de chat en tiempo real como Slack , aplicaciones de juegos en línea, sistemas de seguimiento de flotas en tiempo real

Tecnologías para los Sistemas Distribuidos

Contenidos

Mecanismos de comunicación distribuida

IPC, Sockets, RPC, RMI, ORB

Revisión de tecnologías Web

Modelo HTTP Básico

Servicios Web

Definición, SOA, REST

Resumen general

Tecnologías de Comunicación → Evolución



Sistemas Operativos y Distribuidos

Iren Lorenzo Fonseca
iren.fonseca@.ua.es



TEMA 3. Sistemas Distribuidos.

Tecnologías para los
Sistemas Distribuidos

Sistemas Operativos y Distribuidos

Iren Lorenzo Fonseca
iren.fonseca@.ua.es

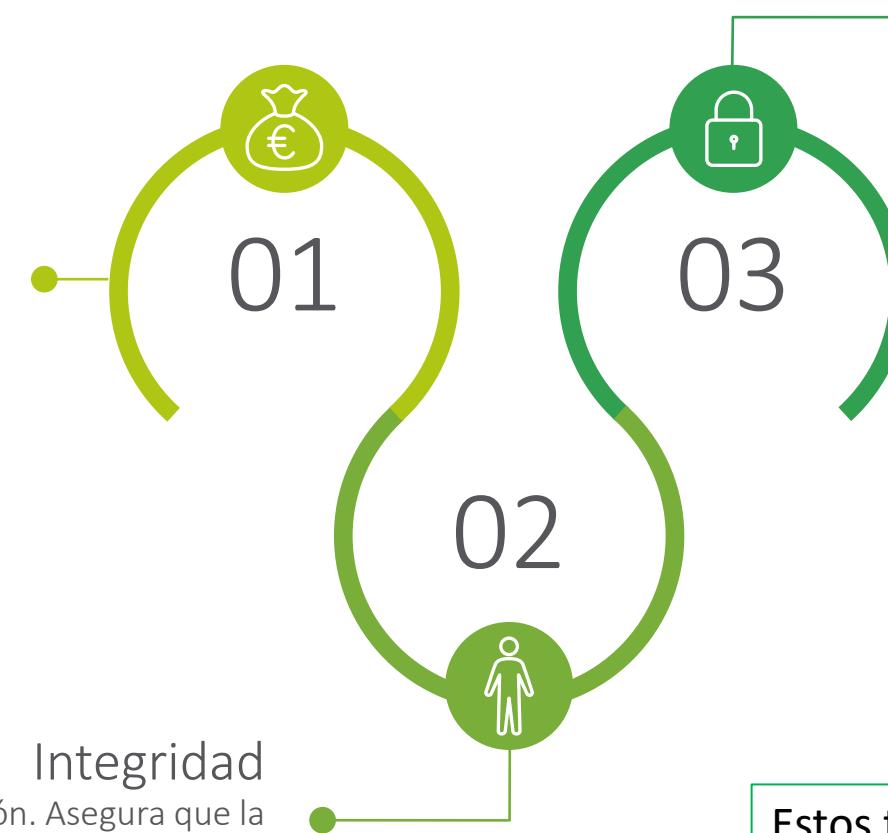


TEMA 3. Sistemas Distribuidos.

Seguridad en Sistemas
Distribuidos

Seguridad en Sistemas Distribuidos

Confidencialidad
Protección de la información contra accesos no autorizados. Es decir, solo las personas o entidades autorizadas deben poder acceder a la información.



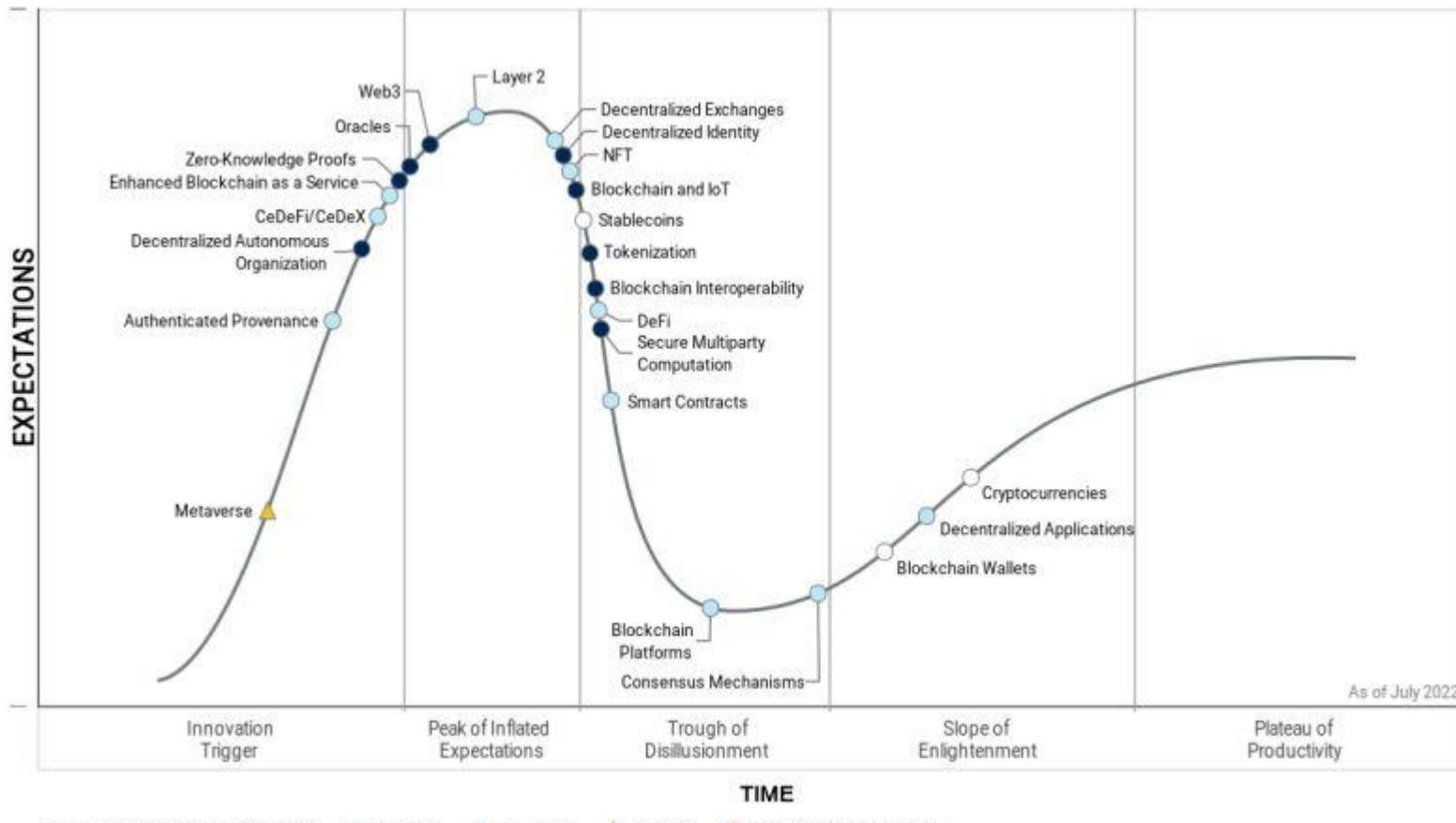
Integridad
Exactitud y completitud de la información. Asegura que la información no haya sido alterada de manera no autorizada y que cualquier modificación sea realizada solo por personas autorizadas.

Disponibilidad
Asegura que la información y los recursos están accesibles y utilizables por los usuarios autorizados cuando sea necesario

Estos tres principios están interrelacionados y son esenciales para un enfoque integral de la seguridad de la información. La falta de cualquiera de estos elementos puede comprometer la seguridad global del sistema

Seguridad en Sistemas Distribuidos

Gartner blockchain, web3 hype cycle 2022



Blockchain
La tecnología detrás del Bitcoin

Fuente: Gartner

Introducción

Requerimientos de Bitcoin

- | No depender de **terceros**
- | La moneda digital debe ser **única** y **no** debe poder **duplicarse**
- | Las **transacciones** realizadas con esta moneda y mediante este sistema **no** deben poder **alterarse**
- | Poseer un **mecanismo** para involucrar, **motivar** y compensar a la **comunidad** de forma que **proporcione** los **recursos** que precisa la red de blockchain
- | La moneda debe tener **valor**
- | Debe ser **sencillo** poder realizar **transacciones** comerciales con esta moneda y ser capaz de **adaptarse** al **entorno cambiante**
- | Las **transacciones** deben estar **libres de comisiones** o, al menos, que sean lo más **reducidas** posible

Definición de Bitcoin

El término hace referencia a dos elementos:

01 Una **divisa electrónica**, criptomoneda o moneda digital [**bitcoin**]

02 Un **sistema de pago electrónico** [**Bitcoin Core**]



Definición de la divisa electrónica

Bitcoin Core

Problema: se requiere que no se pueda falsificar: evitar el problema del “doble gasto”

Solución: la divisa se representa como un apunte de cada operación:
lista de transacciones

Ordenante	Beneficiario	Cantidad
Juan	José	10 ₿
José	María	6 ₿
María	Ana	2 ₿

* **No precisa un TOKEN especial**

→ No puede duplicarse. El saldo se calcula sobre todas las operaciones



Definición de la divisa electrónica

Bitcoin

Problema: se requiere que no exista dependencia de terceros (bancos, entidades financieras, sistemas de pago, ...)

Solución:

- 01 Que exista un único registro de transacciones
- 02 Que este registro lo tengan todos los participantes

Red
Peer

Ordenante	Beneficiario	Cantidad
Juan	José	10 ₿
José	María	6 ₿
María	Ana	2 ₿



Red P2P



Red P2P



Red P2P



Red P2P



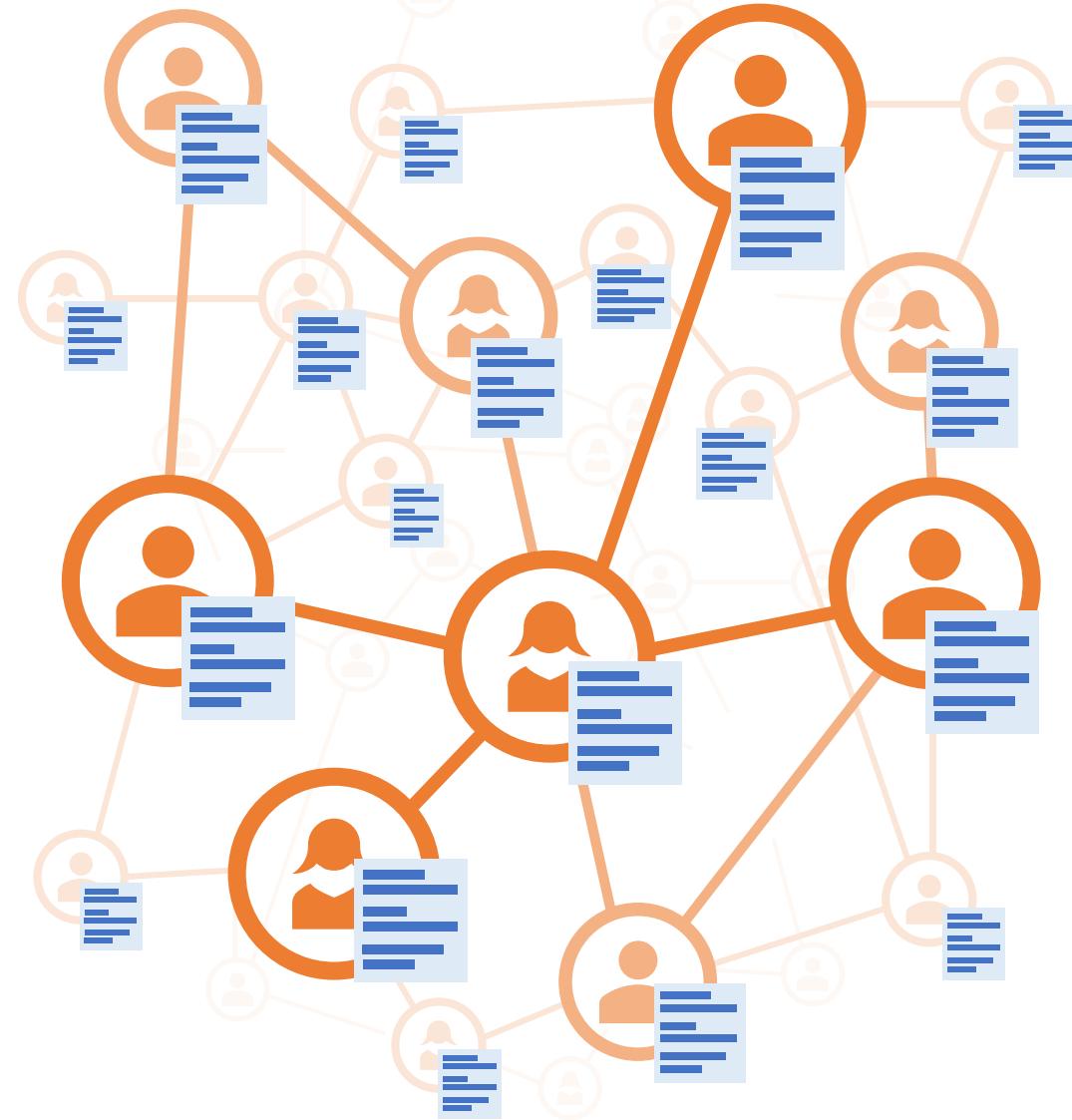
Red P2P



Red P2P



Red P2P



Cadena de Bloques (Blockchain)

Registro único de transacciones

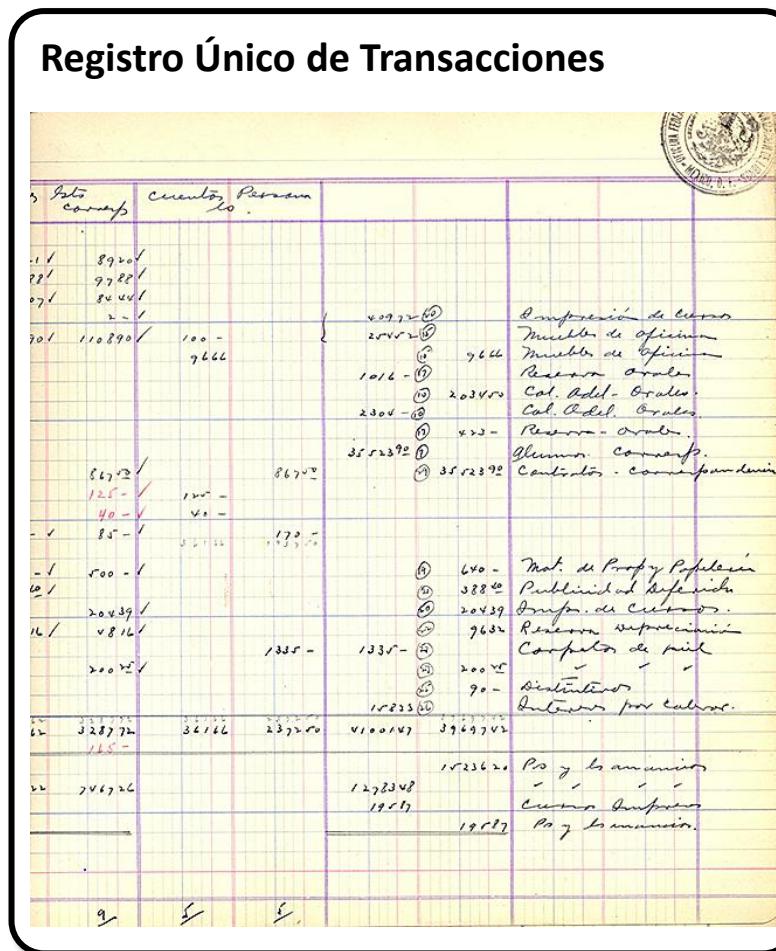
Libro Diario Contable

S. Gato conways		Cuentas Peterson lo.					
11 /	8920 /			409720 (6)		Compra de leños	
22 /	9782 /			203452 (6)		Mueble de oficina	
07 /	8444 /			(6)	9644	Mueble de oficina	
	2 - /			1016 - (6)		Reserva - orales	
30 /	110890 /	100 -		(6) 203452		Cal. Adel. - orales	
		9666		2304 - (6)		Cal. Adel. - orales	
				(6) 472 -		Reserva - orales	
				3552392 (6)		Gomer. conways.	
				(6) 3552392		Cambios - conway	
	6672 /		86720				
	125 - /	120 -					
	40 - /	40 -					
- ✓	85 - /	85 -					
		32166	170 -				
- ✓	500 - /			(6) 600 -		Mati. de Propy. Poblacion	
02 /	20439 /			(6) 38840		Publicidad Diferencia	
12 /	8716 /			(6) 24239		Difus. de celos.	
	2002 /		1335 -	(6) 9632		Reserva vegetación	
			1335 -	(6) 20045		Carpeta de piel	
				(6) 90 -		Distintivos	
				10233 (6)		Indumentos por cultivo.	
22	32872	32166	232250	V100187	39627V2		
	165 -						
22	74626			1278348	1522620	Ps y los anuncios	
				19573		✓	
					19573	Cierre Empres	
						Ps y los anuncios.	

Blockchain

Cadena de Bloques (Blockchain)

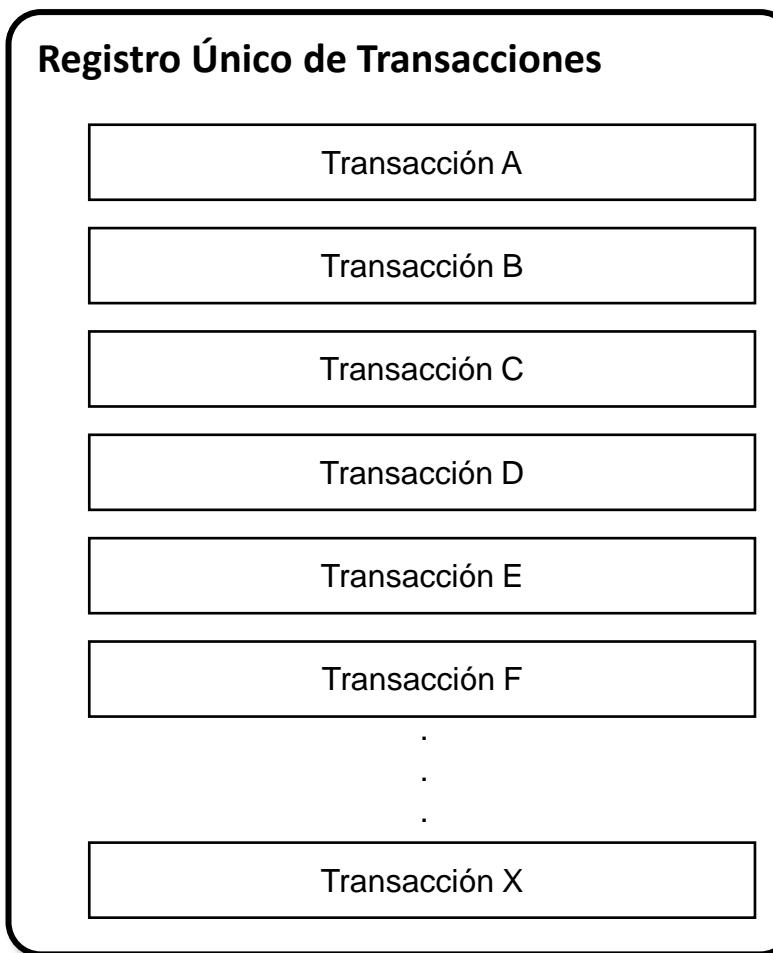
Registro único de transacciones



Blockchain

Cadena de Bloques (Blockchain)

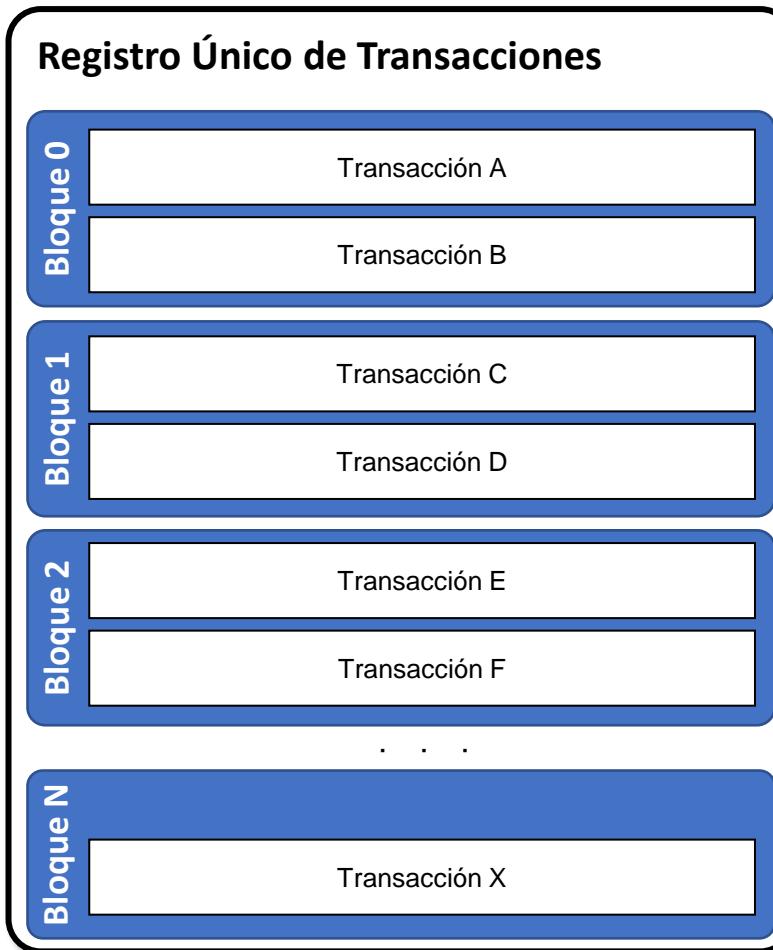
Registro único de transacciones



Blockchain

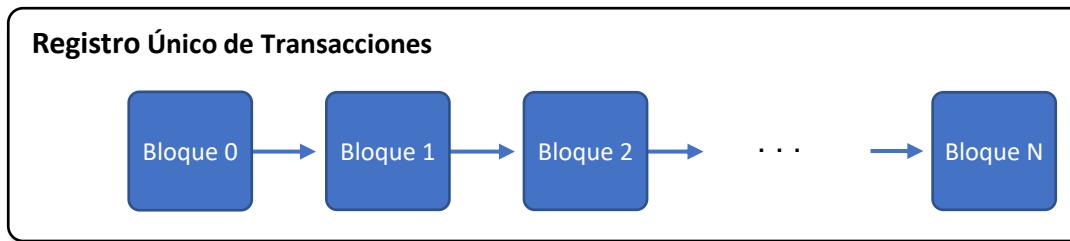
Cadena de Bloques (Blockchain)

Lista de bloques



Cadena de Bloques (Blockchain)

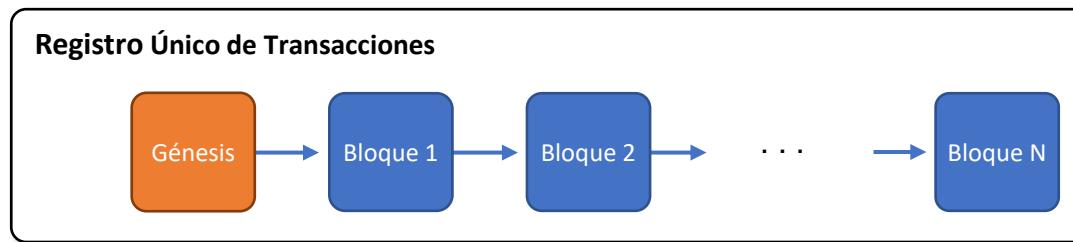
Lista de bloques



Blockchain

Cadena de Bloques (Blockchain)

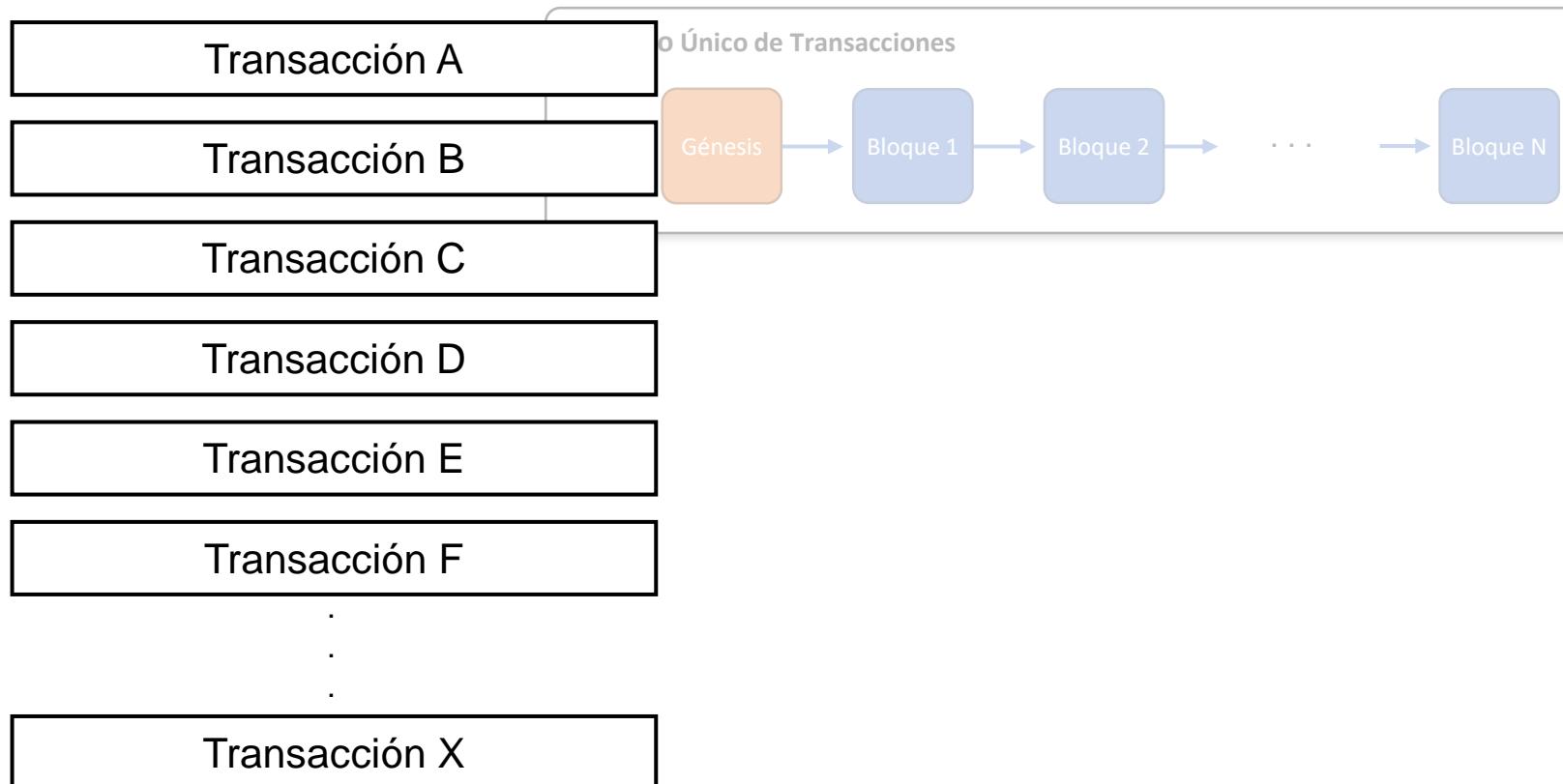
Bloque génesis



Ordenante	Beneficiario	Cantidad
<hr/>		
	Juan	50 ₩
Juan	José	10 ₩
José	María	6 ₩
María	Ana	2 ₩

Cadena de Bloques (Blockchain)

Nuevo bloque ($n+1$)

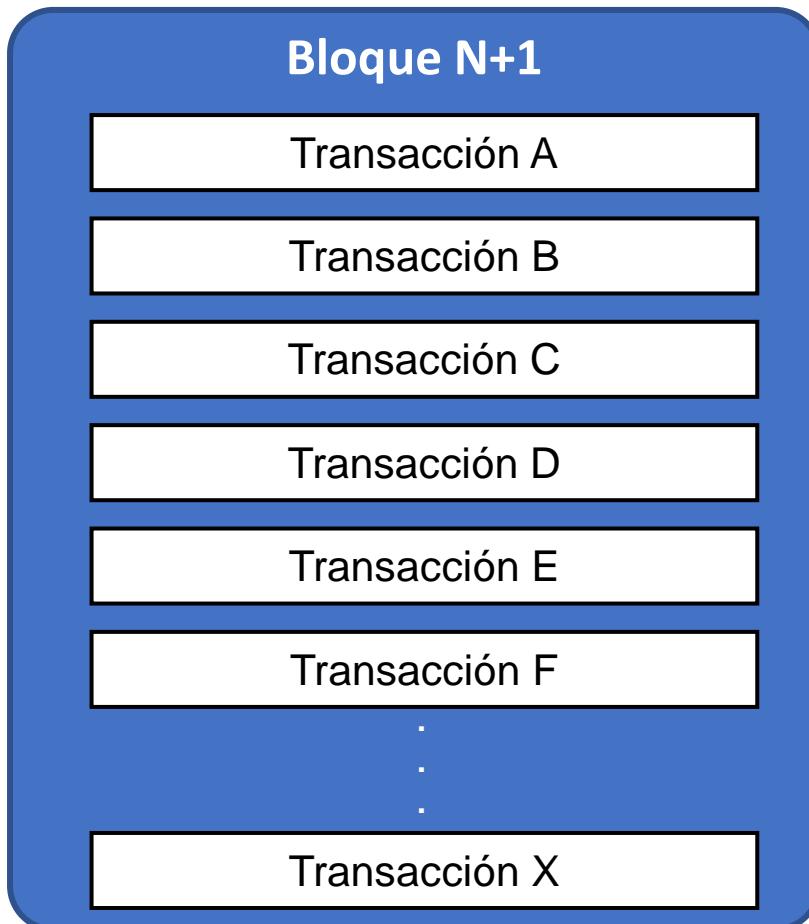


Blockchain



Cadena de Bloques (Blockchain)

Nuevo bloque (n+1)



co de Transacciones

esis

Bloque
Bloque 1 → ... → Bloque N

01

01 Bloque Bitcoin de transacciones

- 1.56 MB de tamaño máximo
- Entre 1.000 y 6.200 transacciones x bloque
- Cada minero va montando bloques a partir de las transacciones pendientes
- Cada transacción debe verificarse previamente



Cadena de Bloques (Blockchain)

Nuevo bloque (n+1)



Cadena de Transacciones

esis

Bloque 1 → Bloque 2 → ... → Bloque N

02

Transacción Coinbase

| Recompensa por minar + comisiones

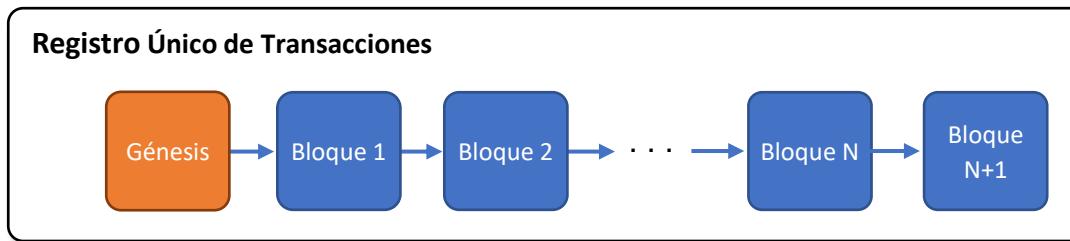
| Inicialmente 50 BTC (actualmente 3,125 BTC)

| Halving: reducción a la mitad cada 210.000 bloques (cada 4 años aproximadamente)

| Sobre 2140 se minará el último BTC de un total de 21 millones BTC

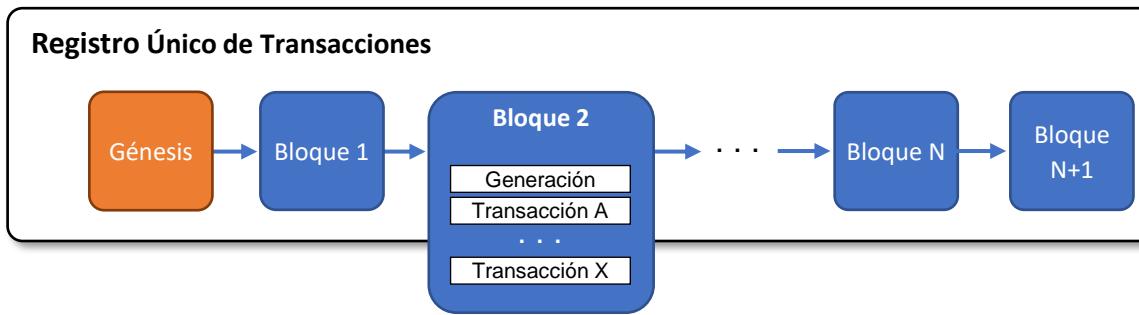
Cadena de Bloques (Blockchain)

Nuevo bloque (n+1)



Cadena de Bloques (Blockchain)

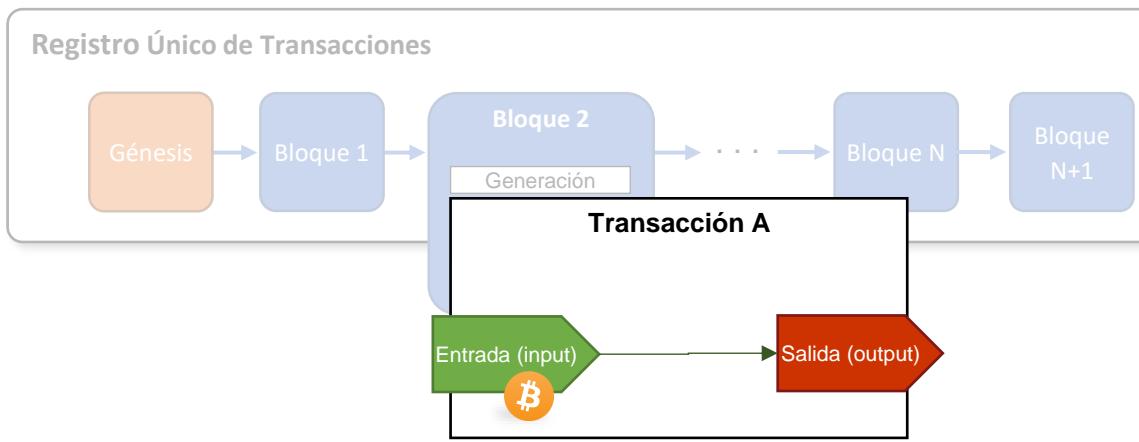
Transacciones Tx



Blockchain

Cadena de Bloques (Blockchain)

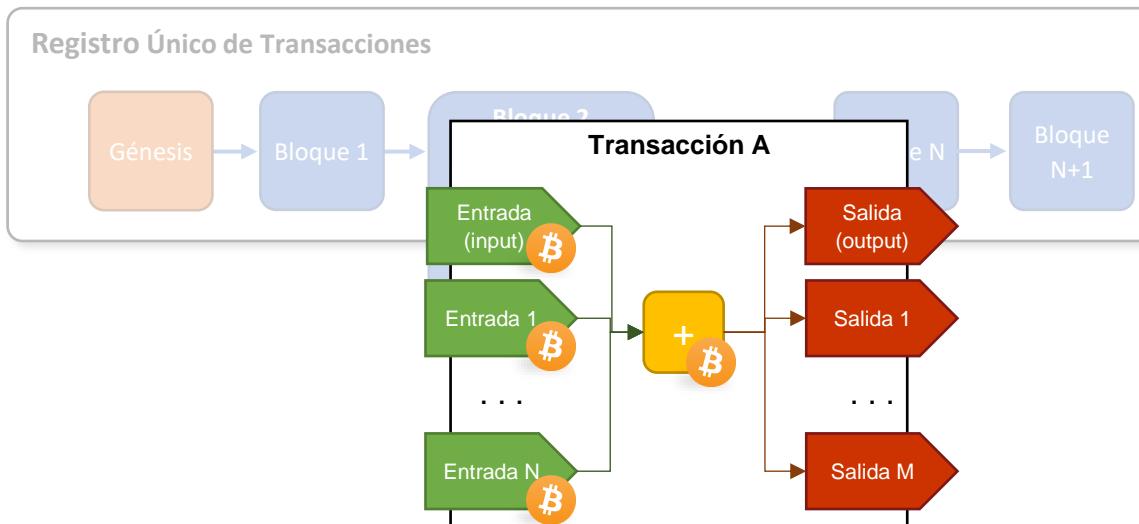
Transacciones Tx



$1 \text{ BTC} (\text{bitcoin}) == 100.000.000 \text{ satoshi}$
 $1 \text{ satoshi} = 1/100.000.000 \text{ BTC} = 0,00000001 \text{ BTC}$

Cadena de Bloques (Blockchain)

Transacciones Tx

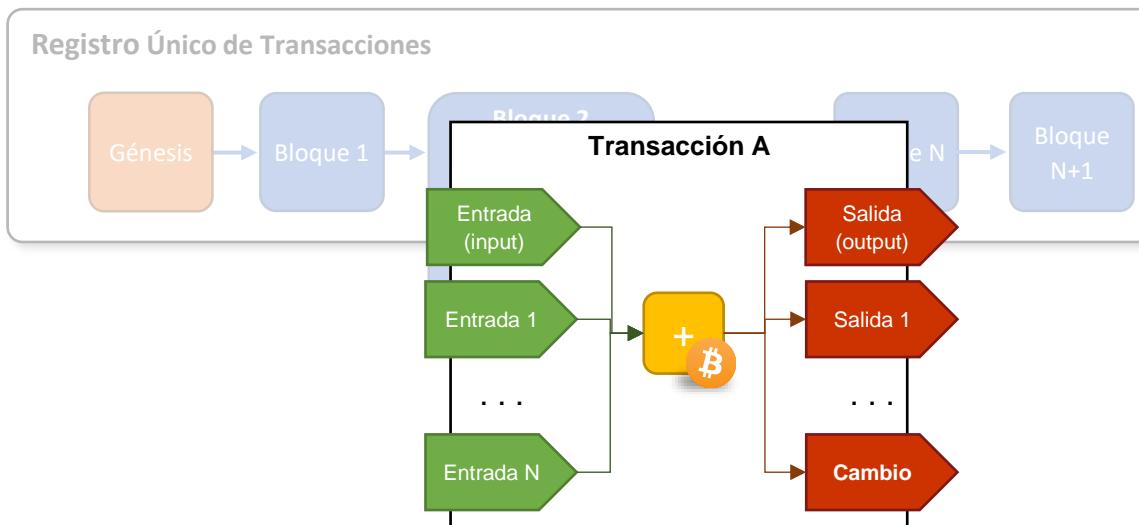


$$\sum_{i=0}^N Input_i$$

Blockchain

Cadena de Bloques (Blockchain)

Transacciones Tx



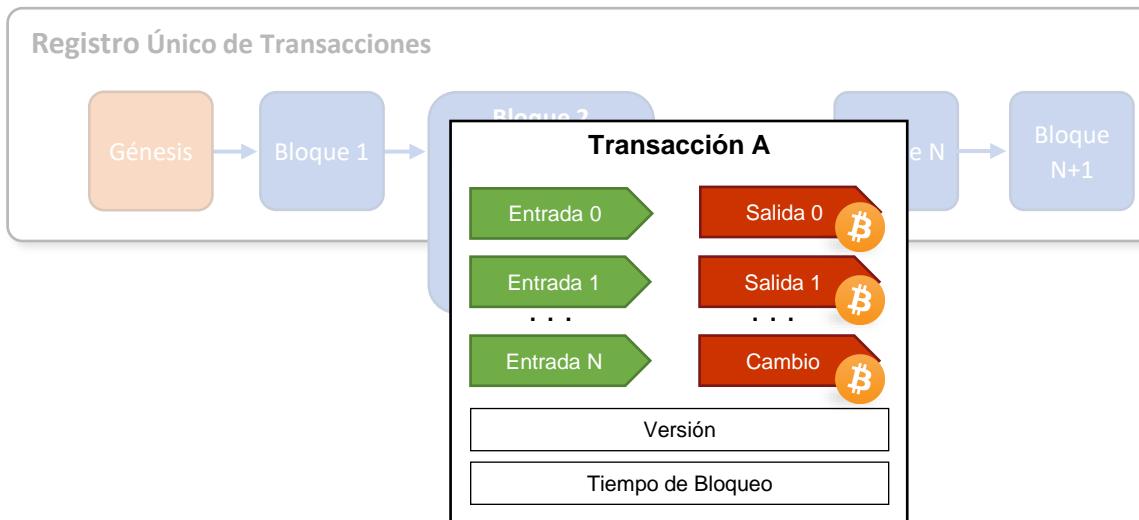
$$\sum_{i=0}^N Input_i \geq \sum_{j=0}^M Output_j$$

$$\sum_{i=0}^N Input_i - \sum_{j=0}^M Output_j = \text{Comisión}$$

Blockchain

Cadena de Bloques (Blockchain)

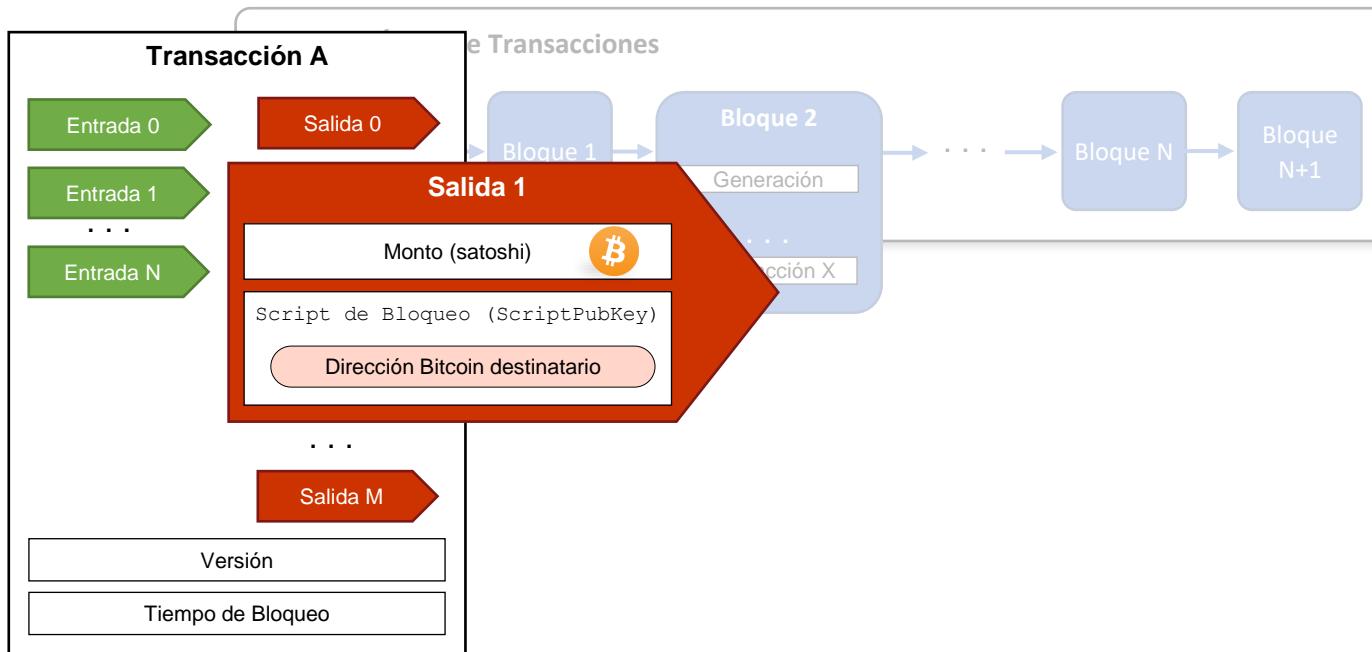
Transacciones Tx



Blockchain

Cadena de Bloques (Blockchain)

Transacciones Tx



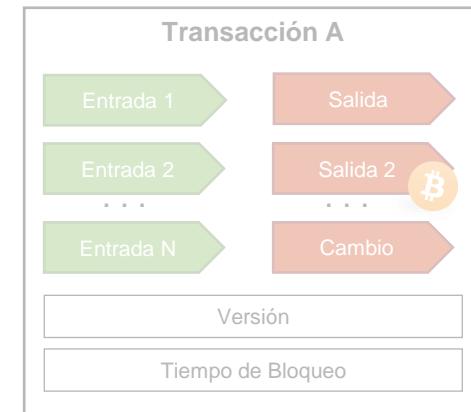
Blockchain

Criptografía básica para blockchain

01 Función Hash

02 Función Hash Criptográfica

03 Criptografía Asimétrica



Criptografía básica para blockchain

Función Hash y Hash Criptográfica

01 Función Hash

Comprime una cadena de entrada en otra de tamaño fijo

Texto entrada (p.ej.: Transacciones):	Juan	José	10 ₿
	José	María	6 ₿
	María	Ana	2 ₿

Texto salida (Hash): 

02 Función Hash Criptográfica (SHA-256 y RIPEMD-160)

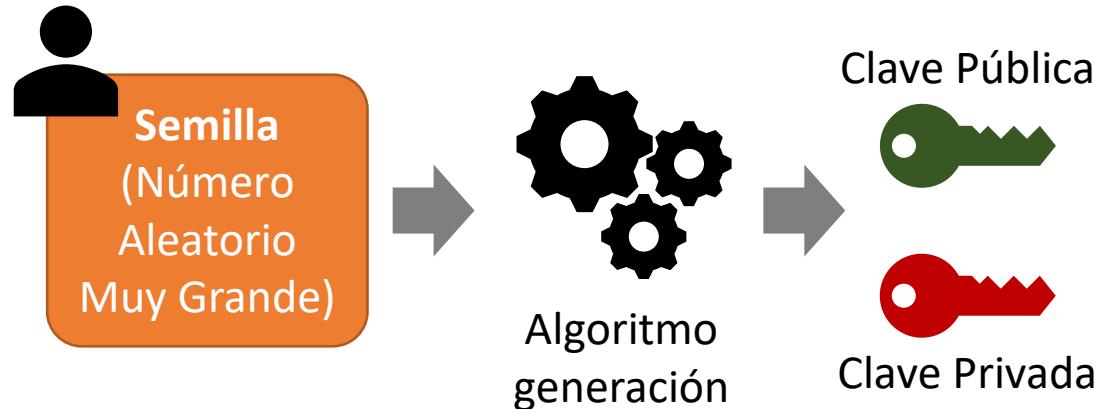
Hash útiles para criptografía → fácil calcular hash, muy difícil deducir la fuente

Criptografía básica para blockchain

Criptografía Asimétrica

03 Criptografía Asimétrica

Utiliza un par de claves para encriptar y desencriptar una información



Principalmente lo utilizamos para :
03A Cifrado de clave pública

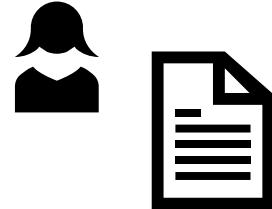
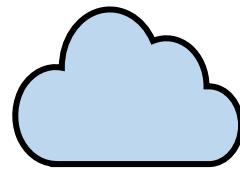
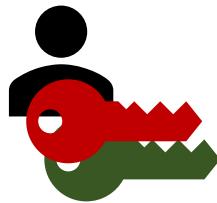
Criptografía básica para blockchain

Criptografía Asimétrica

03ACifrado de clave pública

Utiliza la clave pública para cifrar un mensaje:

María desea enviar un archivo que sólo pueda ver Juan.



Juan tiene una clave privada y su correspondiente clave pública

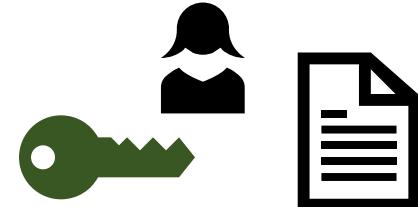
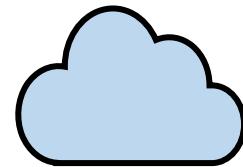
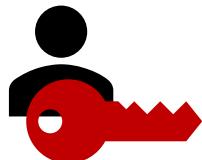
Criptografía básica para blockchain

Criptografía Asimétrica

03ACifrado de clave pública

Utiliza la clave pública para cifrar un mensaje:

María desea enviar un archivo que sólo pueda ver Juan.



Juan envía su clave pública a María

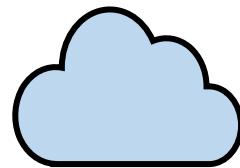
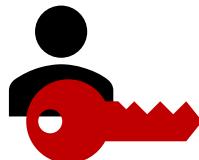
Criptografía básica para blockchain

Criptografía Asimétrica

03ACifrado de clave pública

Utiliza la clave pública para cifrar un mensaje:

María desea enviar un archivo que sólo pueda ver Juan.



... es como enviar un sobre abierto, con cerradura

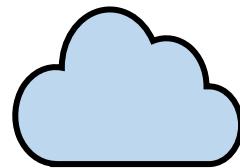
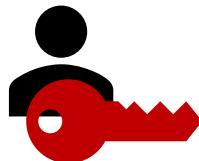
Criptografía básica para blockchain

Criptografía Asimétrica

03ACifrado de clave pública

Utiliza la clave pública para cifrar un mensaje:

María desea enviar un archivo que sólo pueda ver Juan.



María mete el mensaje o el archivo (o los datos en general) dentro del sobre abierto

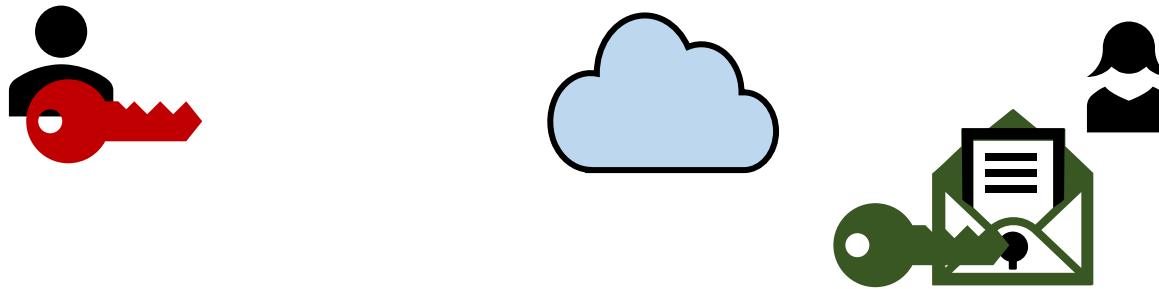
Criptografía básica para blockchain

Criptografía Asimétrica

03ACifrado de clave pública

Utiliza la clave pública para cifrar un mensaje:

María desea enviar un archivo que sólo pueda ver Juan.



María cierra (**encripta**) el sobre con la clave pública de Juan

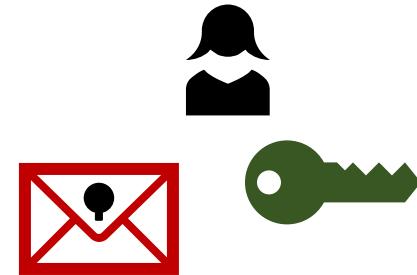
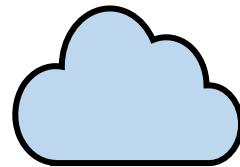
Criptografía básica para blockchain

Criptografía Asimétrica

03ACifrado de clave pública

Utiliza la clave pública para cifrar un mensaje:

María desea enviar un archivo que sólo pueda ver Juan.



María ya no podrá abrir (**desencriptar**) nuevamente el sobre !!!

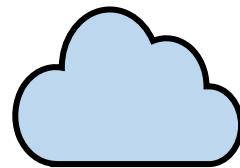
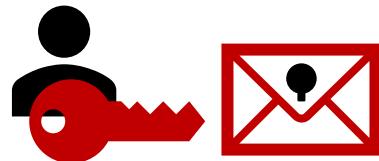
Criptografía básica para blockchain

Criptografía Asimétrica

03ACifrado de clave pública

Utiliza la clave pública para cifrar un mensaje:

María desea enviar un archivo que sólo pueda ver Juan.



María envía la información encriptada y Juan recibe el sobre cerrado

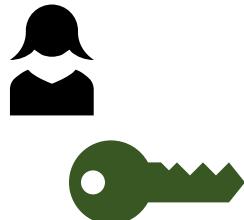
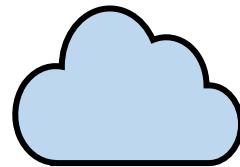
Criptografía básica para blockchain

Criptografía Asimétrica

03ACifrado de clave pública

Utiliza la clave pública para cifrar un mensaje:

María desea enviar un archivo que sólo pueda ver Juan.



Juan puede abrir (**desencriptar**) el sobre porque tiene la clave privada

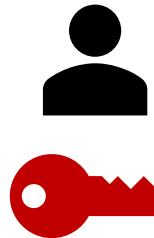
Criptografía básica para blockchain

Criptografía Asimétrica

03ACifrado de clave pública

Utiliza la clave pública para cifrar un mensaje:

María desea enviar un archivo que sólo pueda ver Juan.



El sobre solo pudo abrirse (**desencriptarse**) con la clave privada asociada
a la clave pública con la que se cerró (**encriptó**)

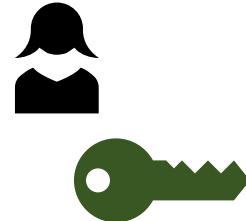
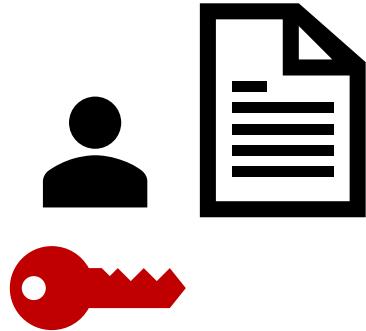
Criptografía básica para blockchain

Criptografía Asimétrica

03ACifrado de clave pública

Utiliza la clave pública para cifrar un mensaje:

María desea enviar un archivo que sólo pueda ver Juan.



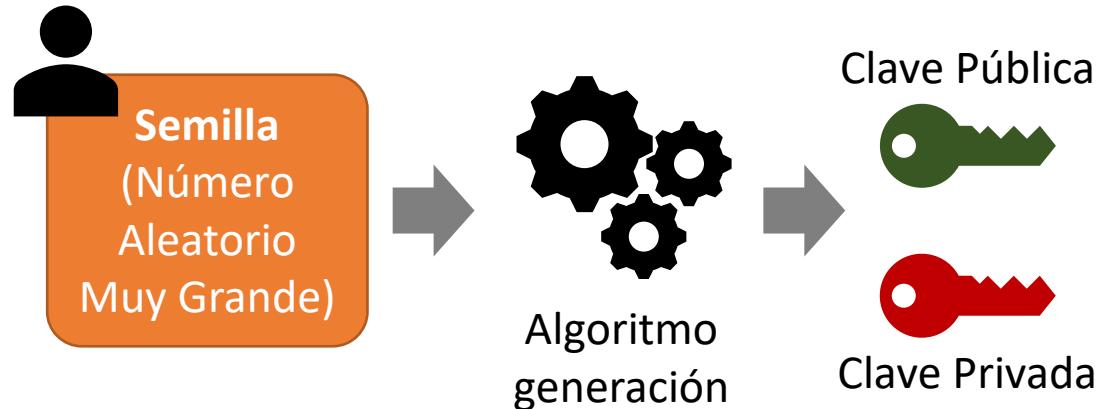
Juan puede acceder al mensaje cifrado

Criptografía básica para blockchain

Criptografía Asimétrica

03 Criptografía Asimétrica

Utiliza un par de claves para encriptar y desencriptar una información



Principalmente lo utilizamos para :

03A Cifrado de clave pública

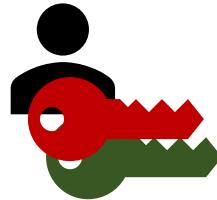
03B Firma digital

Criptografía básica para blockchain

Criptografía Asimétrica

03B Firma digital

- | Utiliza la clave privada para firmar un hash del mensaje



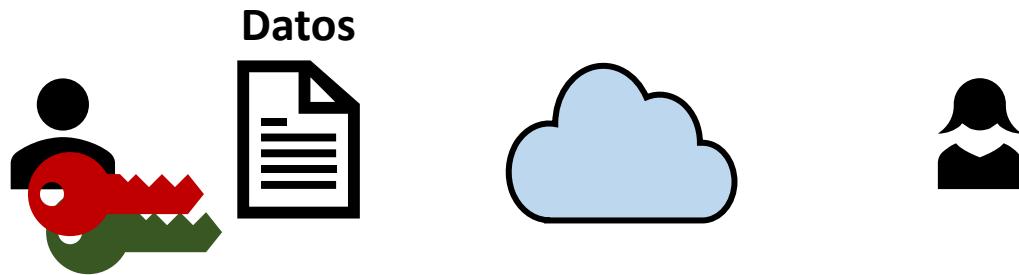
Juan posee una clave privada y su correspondiente clave pública

Criptografía básica para blockchain

Criptografía Asimétrica

03B Firma digital

- | Utiliza la clave privada para firmar un hash del mensaje



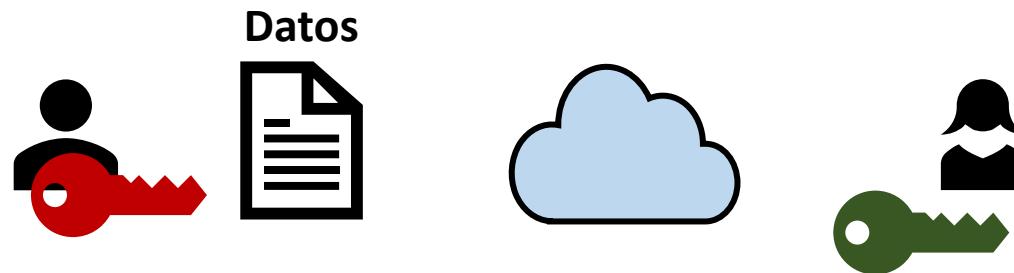
Juan desea enviar un mensaje a María de forma que ella pueda saber si lo envió él

Criptografía básica para blockchain

Criptografía Asimétrica

03B Firma digital

- | Utiliza la clave privada para firmar un hash del mensaje



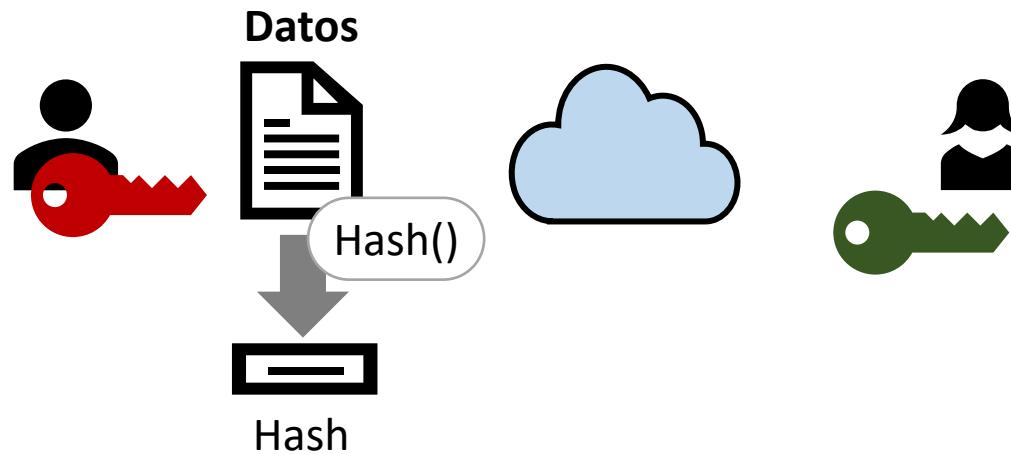
Para que María se asegure de que el emisor es Juan, este le envía su clave pública

Criptografía básica para blockchain

Criptografía Asimétrica

03B Firma digital

- Utiliza la clave privada para firmar un hash del mensaje



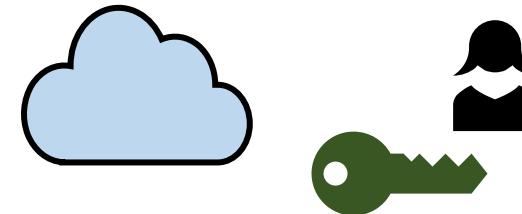
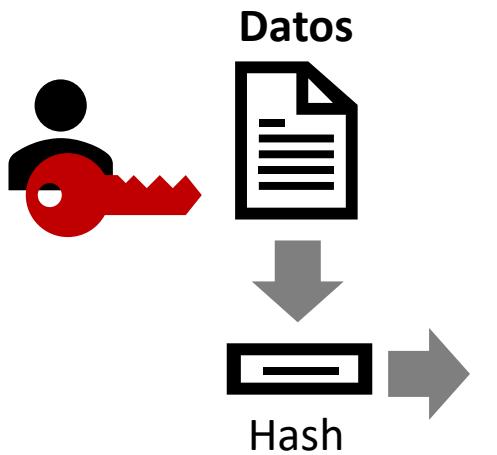
Juan calcula un hash de su mensaje

Criptografía básica para blockchain

Criptografía Asimétrica

03BFirma digital

- | Utiliza la clave privada para firmar un hash del mensaje



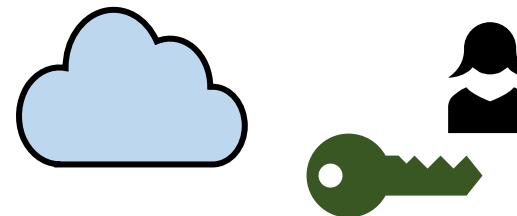
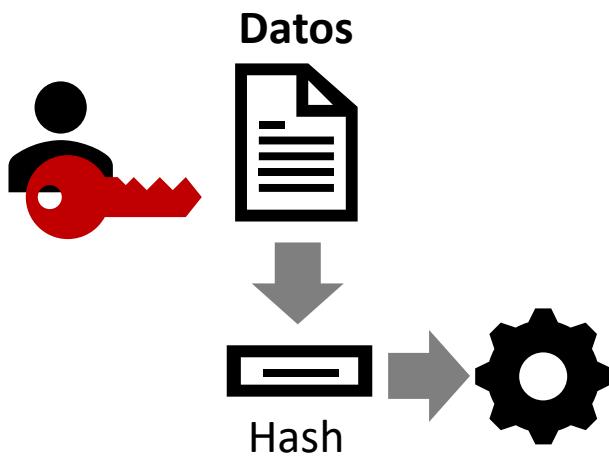
Juan calcula un hash de su mensaje

Criptografía básica para blockchain

Criptografía Asimétrica

03BFirma digital

- | Utiliza la clave privada para firmar un hash del mensaje



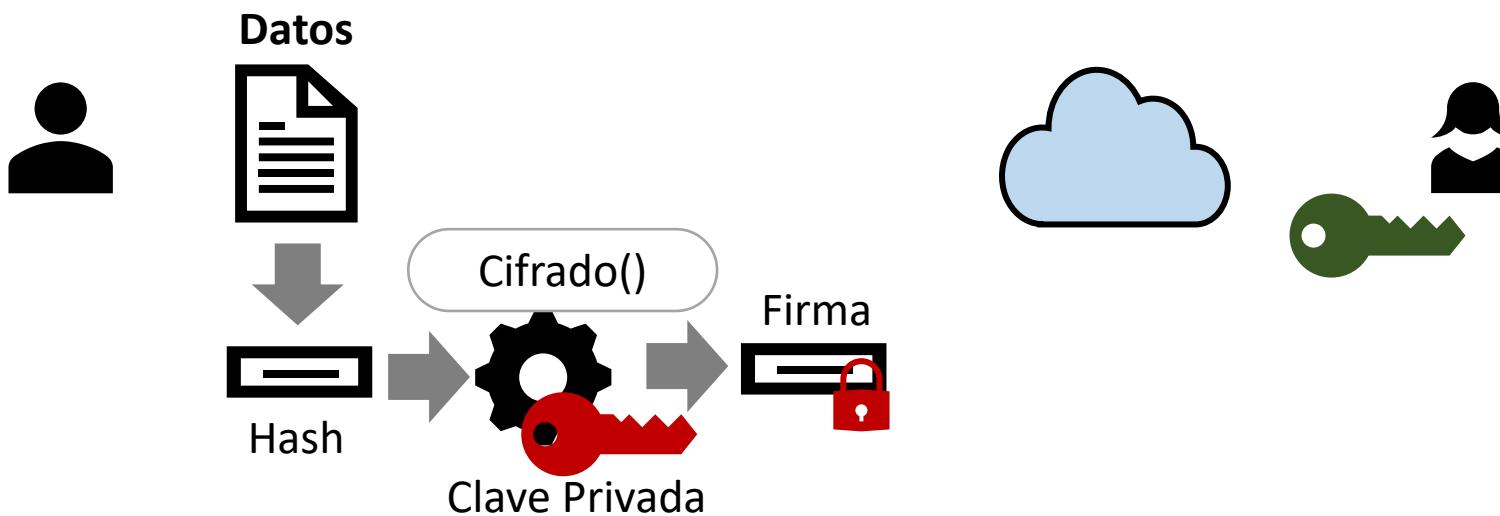
Y utiliza su clave privada para cifrarlo

Criptografía básica para blockchain

Criptografía Asimétrica

03B Firma digital

- Utiliza la clave privada para firmar un hash del mensaje



Obteniendo una firma digital mediante su clave privada

Criptografía básica para blockchain

Criptografía Asimétrica

03B Firma digital

- | Utiliza la clave privada para firmar un hash del mensaje



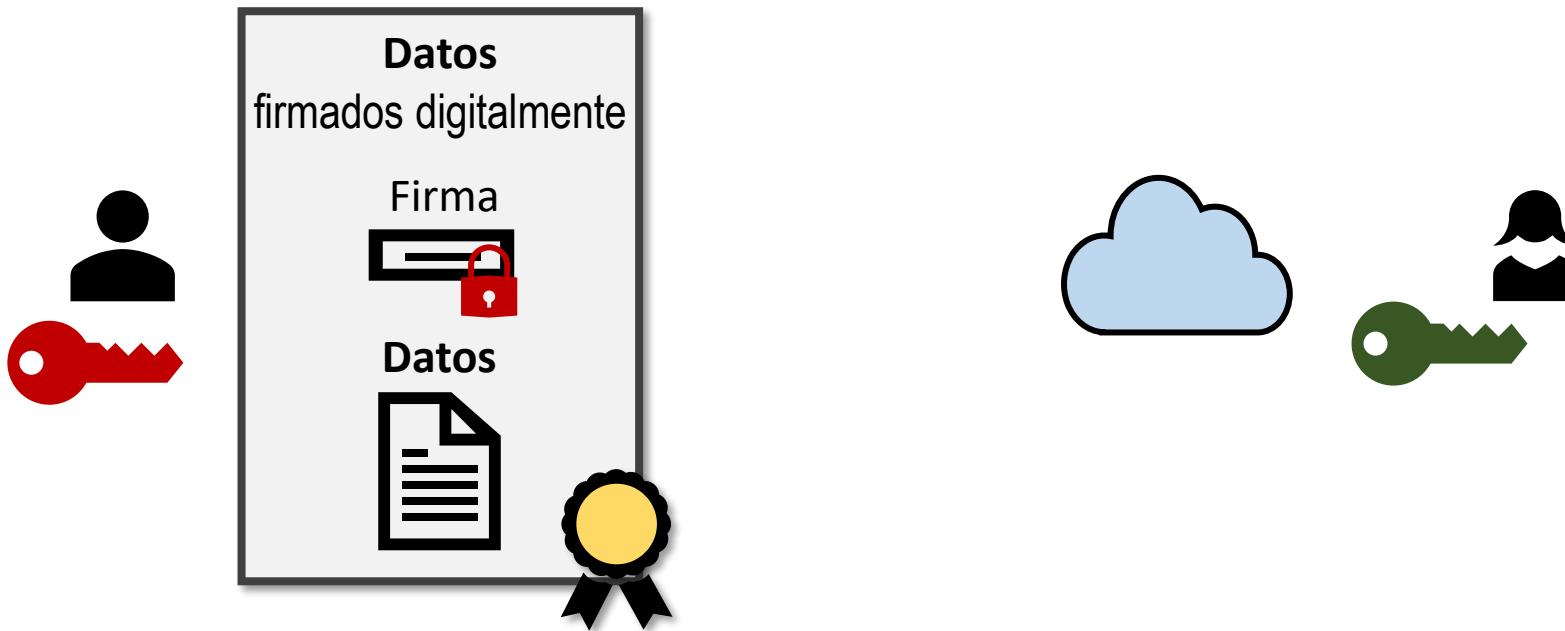
Obteniendo una firma digital mediante su clave privada

Criptografía básica para blockchain

Criptografía Asimétrica

03B Firma digital

- Utiliza la clave privada para firmar un hash del mensaje



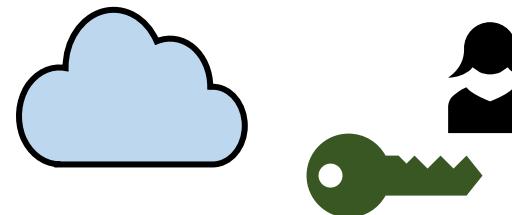
Empaque su mensaje junto con la firma digital obtenida

Criptografía básica para blockchain

Criptografía Asimétrica

03B Firma digital

- Utiliza la clave privada para firmar un hash del mensaje



Envía el mensaje firmado digitalmente a María

Criptografía básica para blockchain

Criptografía Asimétrica

03B Firma digital

- | Utiliza la clave privada para firmar un hash del mensaje



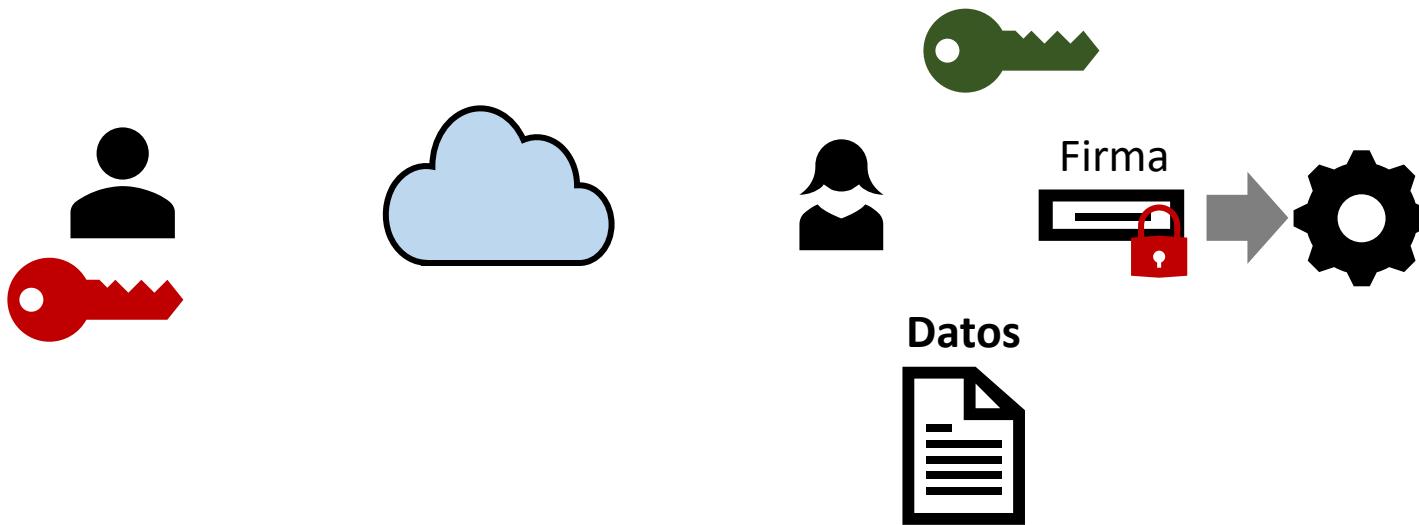
María tiene ahora el mensaje, la firma digital y la clave pública de Juan

Criptografía básica para blockchain

Criptografía Asimétrica

03B Firma digital

- | Utiliza la clave privada para firmar un hash del mensaje



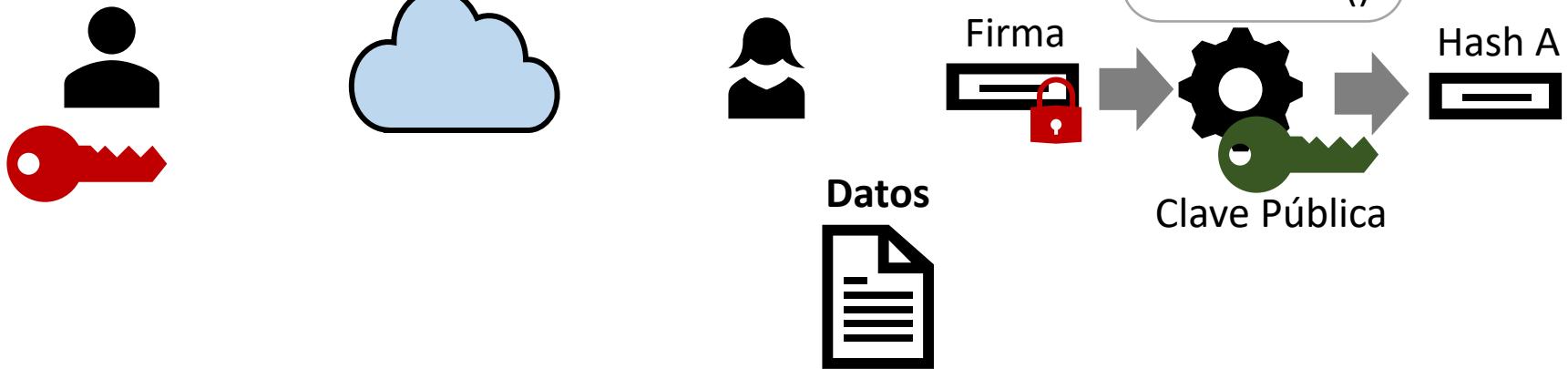
María tiene ahora el mensaje, la firma digital y la clave pública de Juan

Criptografía básica para blockchain

Criptografía Asimétrica

03B Firma digital

- Utiliza la clave privada para firmar un hash del mensaje



Mediante la clave pública de Juan, descifra la firma y obtiene de nuevo el hash original

Criptografía básica para blockchain

Criptografía Asimétrica

03BFirma digital

- Utiliza la clave privada para firmar un hash del mensaje



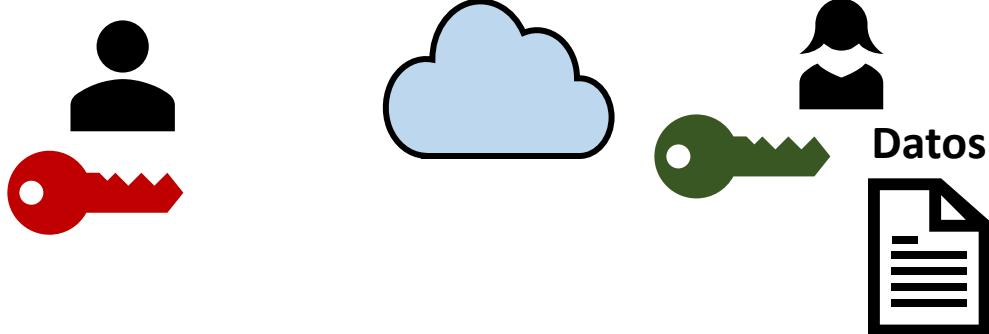
Calcula el hash del documento directamente

Criptografía básica para blockchain

Criptografía Asimétrica

03B Firma digital

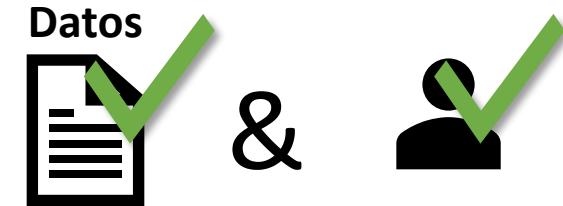
- | Utiliza la clave privada para firmar un hash del mensaje



Compara ambos hash... si coinciden...

If Hash A == Hash B

Then

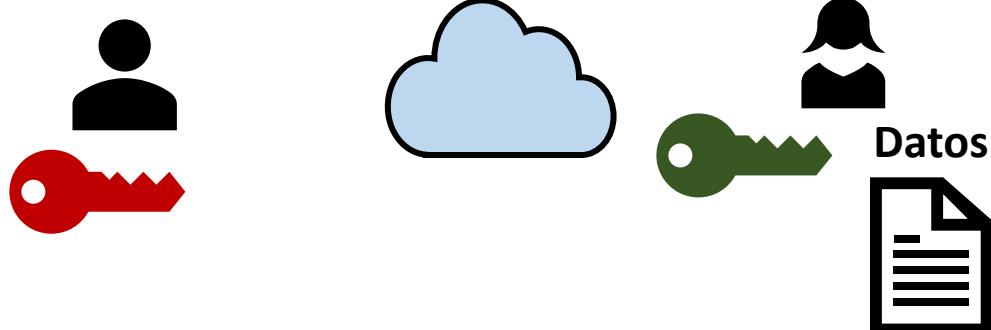


Criptografía básica para blockchain

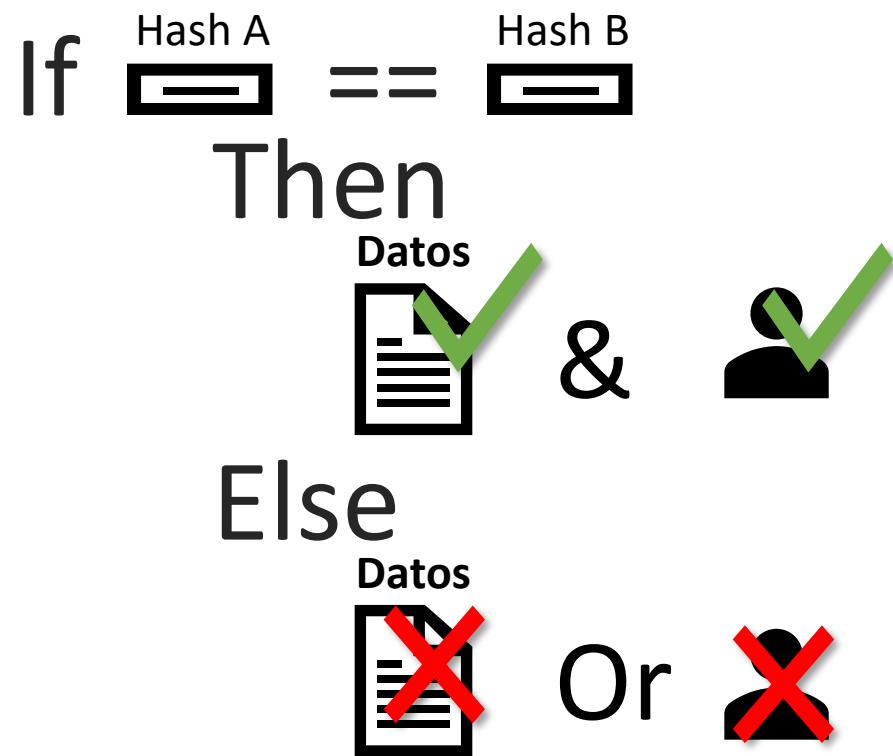
Criptografía Asimétrica

03B Firma digital

- Utiliza la clave privada para firmar un hash del mensaje



...Si no coinciden...

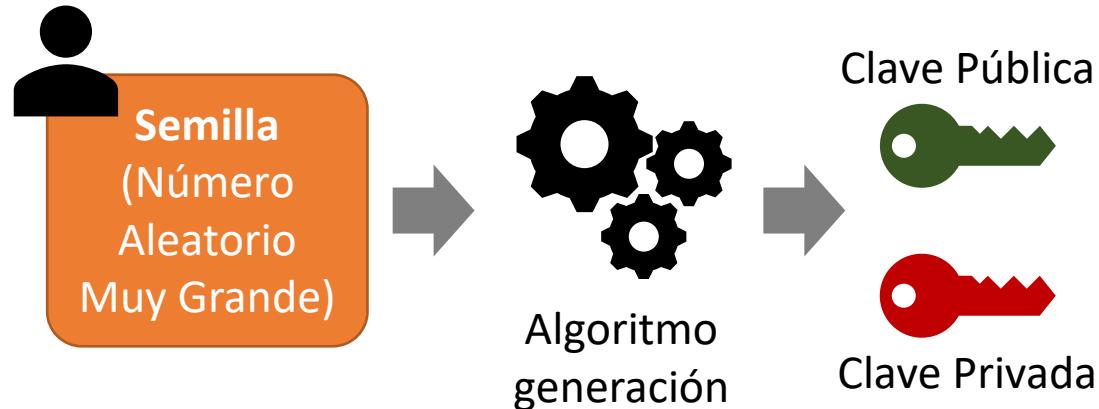


Criptografía básica para blockchain

Criptografía Asimétrica

03 Criptografía Asimétrica

Utiliza un par de claves para encriptar y desencriptar una información



Principalmente lo utilizamos para :

03A Cifrado de clave pública

03B Firma digital

Criptografía básica para blockchain

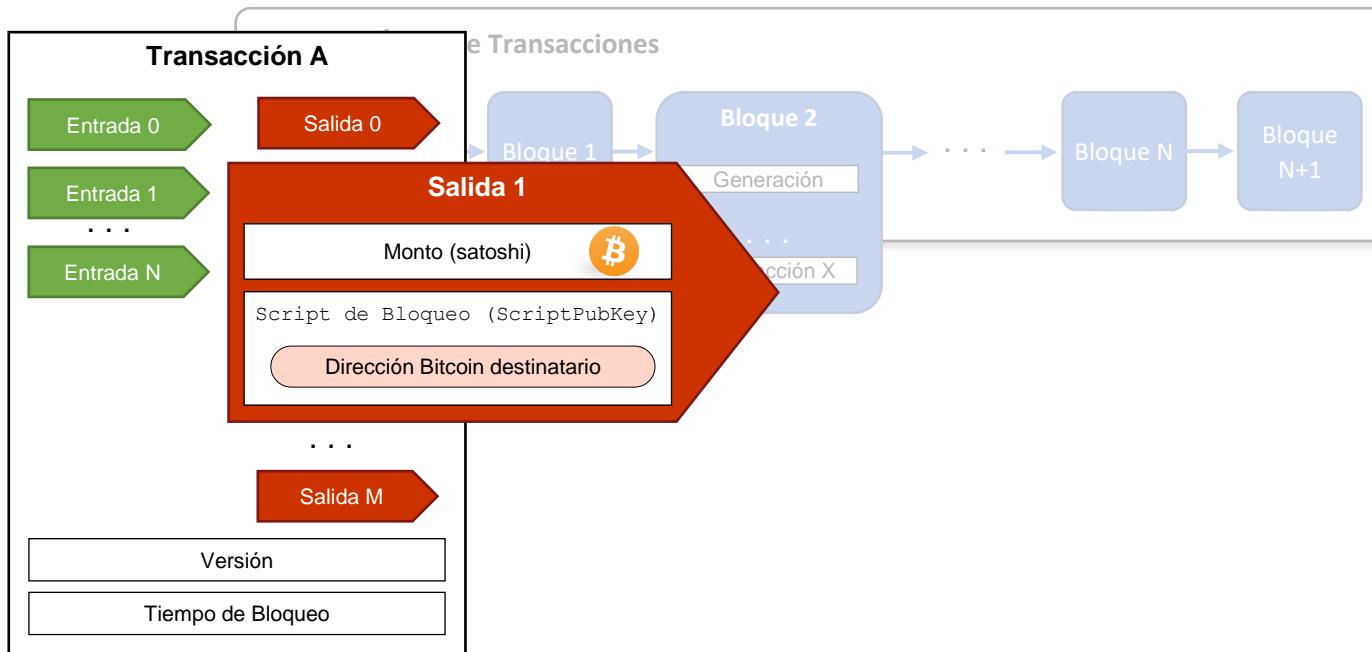
Criptografía Asimétrica

03 Criptografía Asimétrica

- | Utiliza un par de claves para encriptar y desencriptar una información
- | Algoritmos tradicionales más utilizados: RSA, DSA
- | Limitaciones:
 - Tiempo de proceso elevado
 - Claves muy largas
 - El mensaje cifrado ocupa más espacio que el original
- | Criptografía de Curva Elíptica (ECDSA)
 - Variante de criptografía asimétrica más rápida y con claves más cortas

Cadena de Bloques (Blockchain)

Transacciones Tx

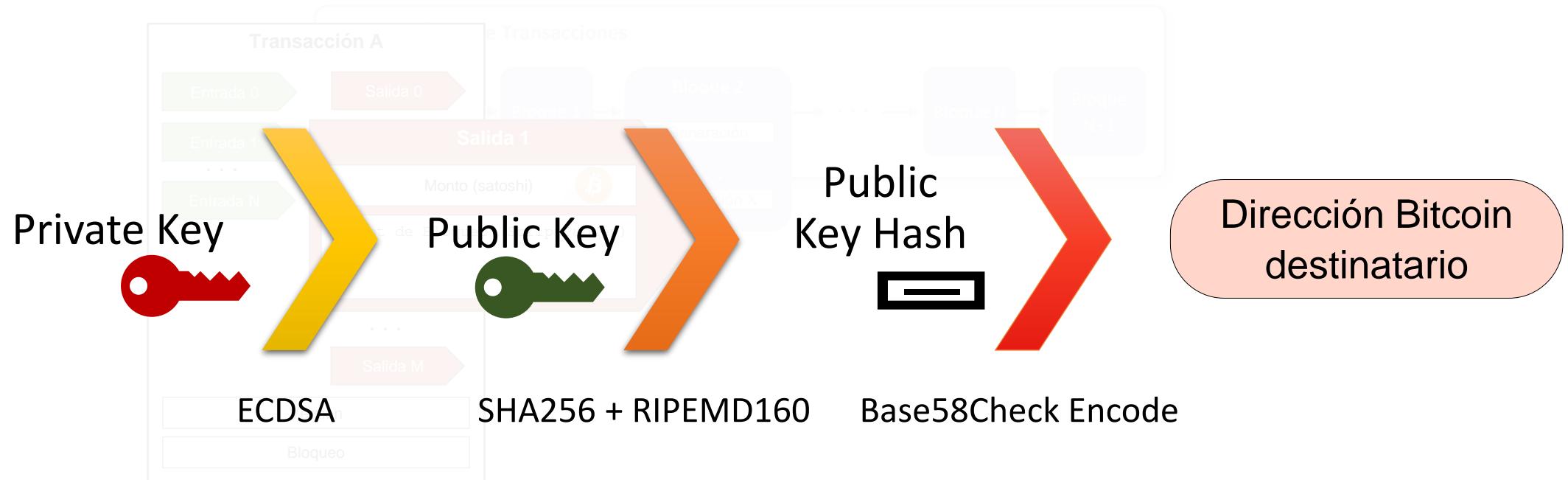


Blockchain



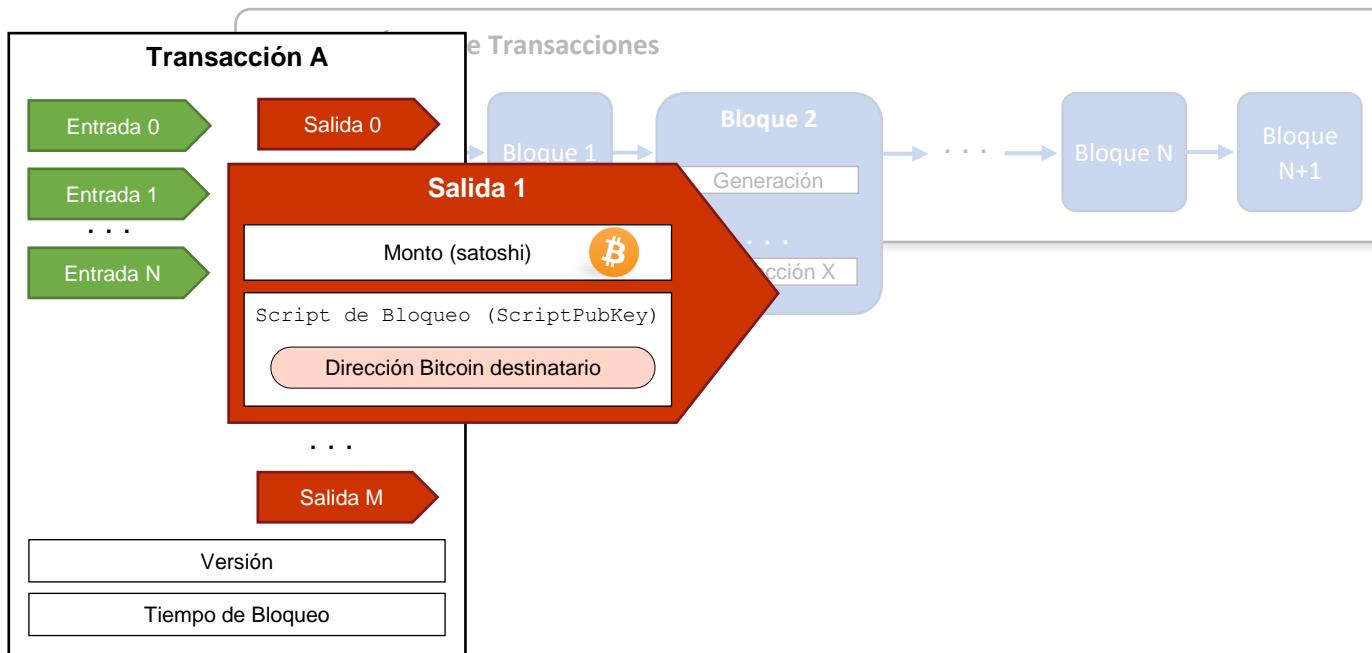
Cadena de Bloques (Blockchain)

Dirección Bitcoin



Cadena de Bloques (Blockchain)

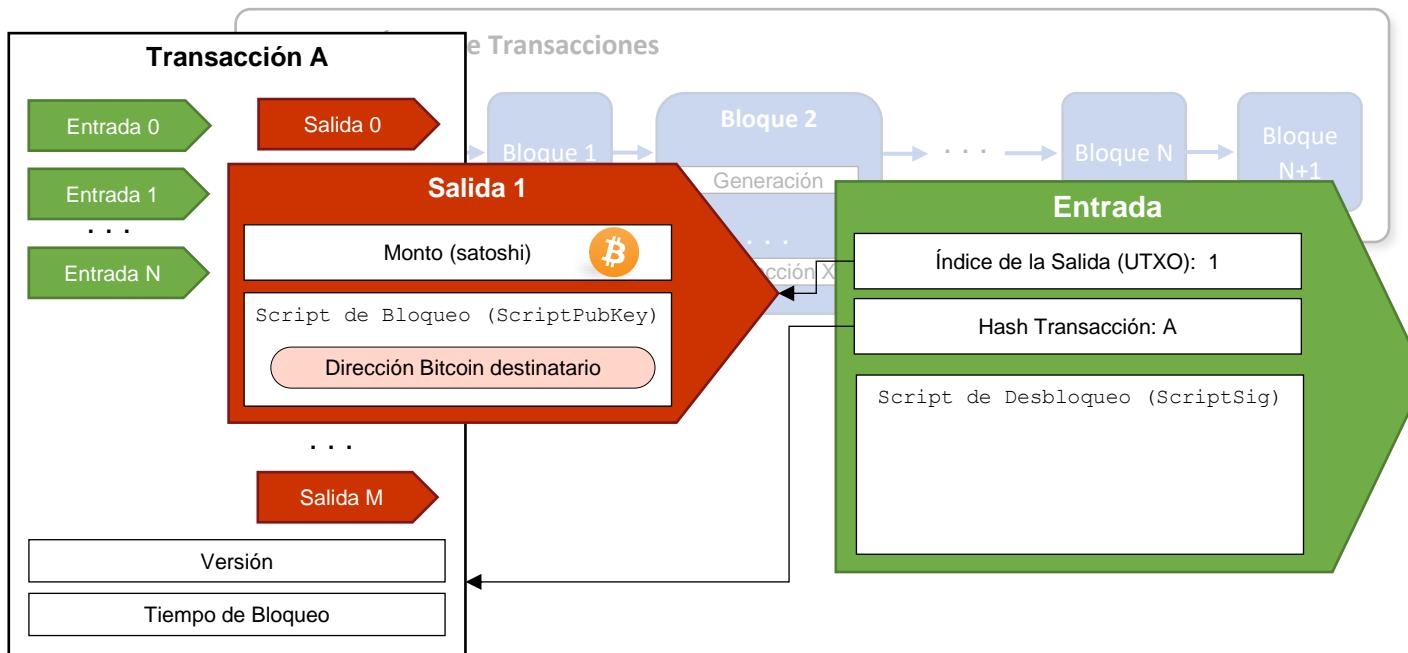
Transacciones Tx



Blockchain

Cadena de Bloques (Blockchain)

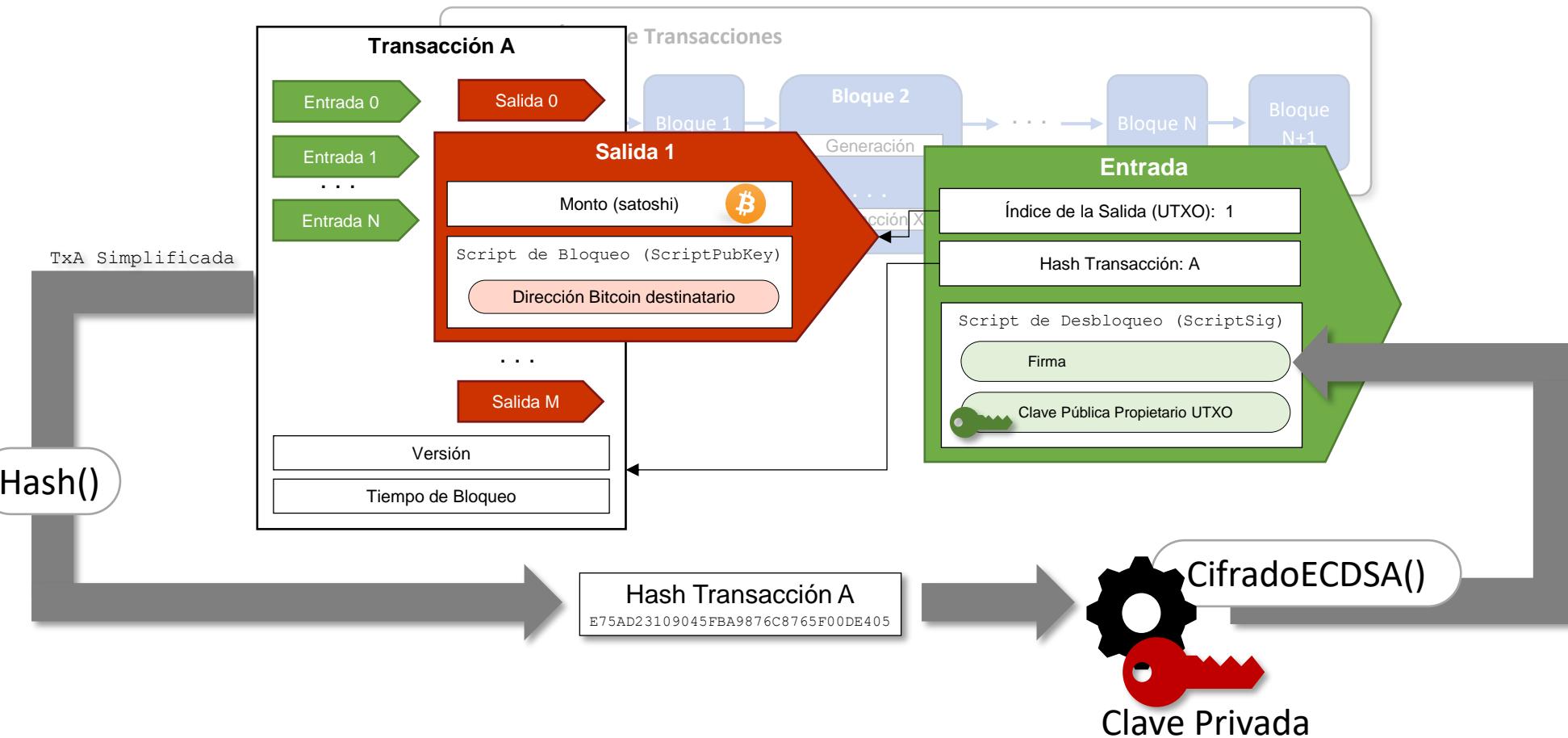
Transacciones Tx



Blockchain

Cadena de Bloques (Blockchain)

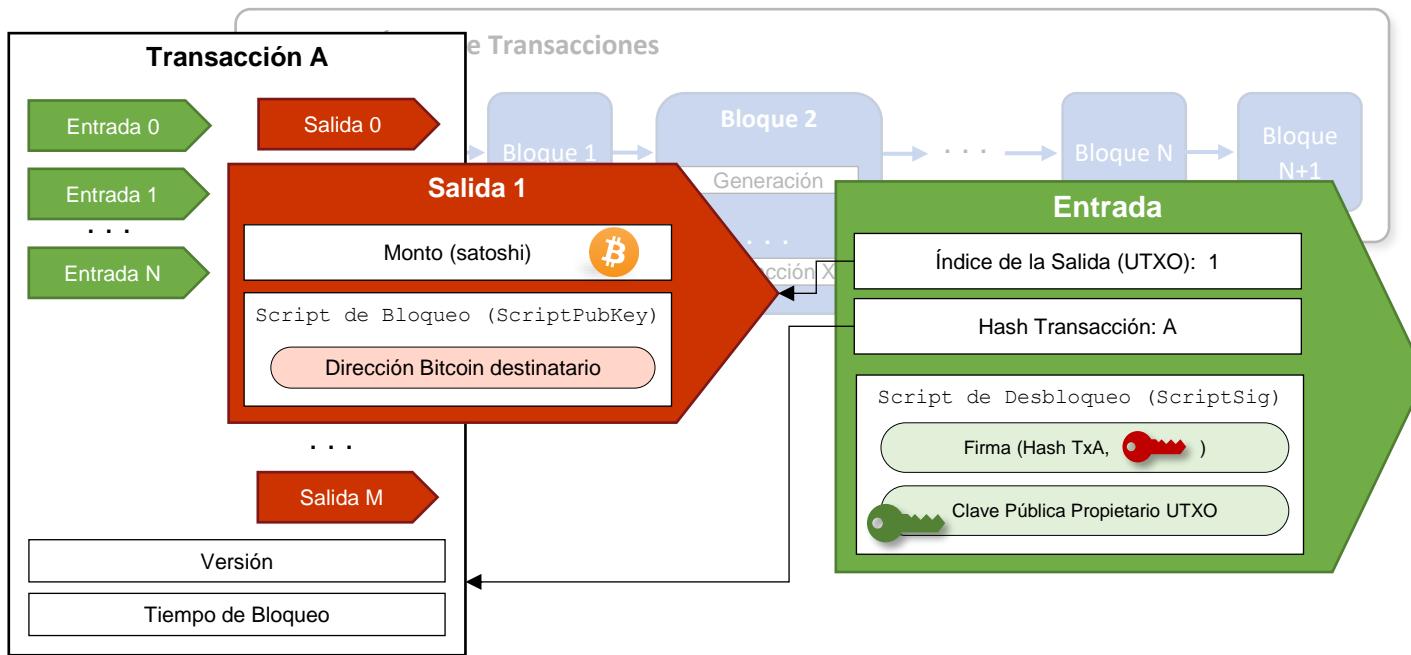
Transacciones Tx



Blockchain

Cadena de Bloques (Blockchain)

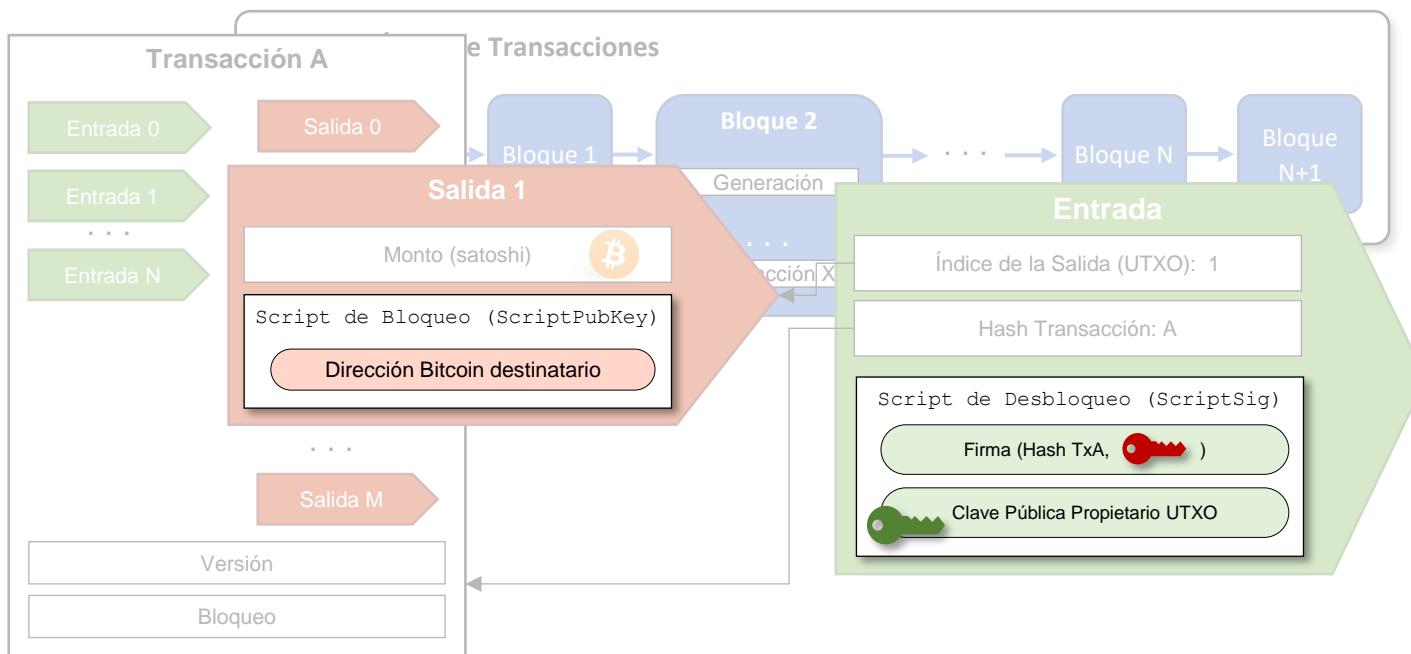
Transacciones Tx



Blockchain

Cadena de Bloques (Blockchain)

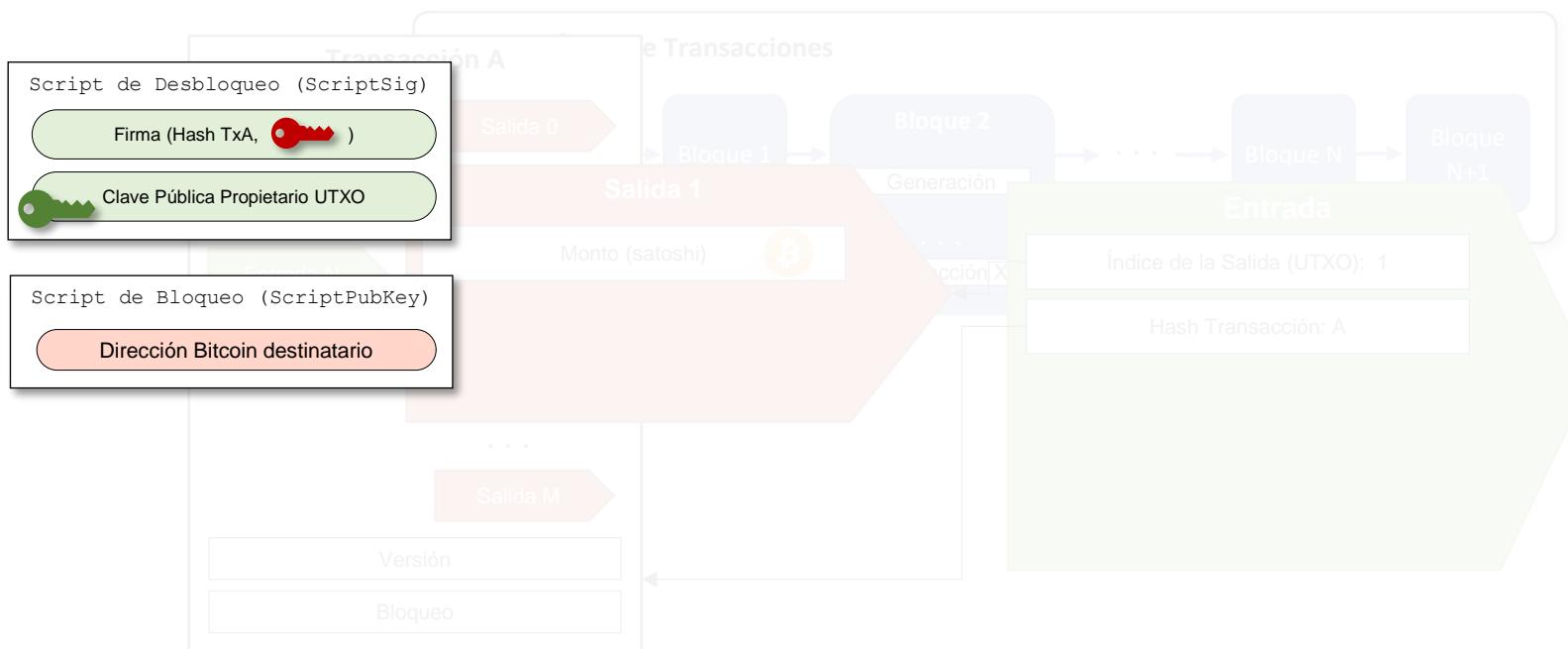
Smart-Contracts Script Verification



Blockchain

Cadena de Bloques (Blockchain)

Smart-Contracts Script Verification

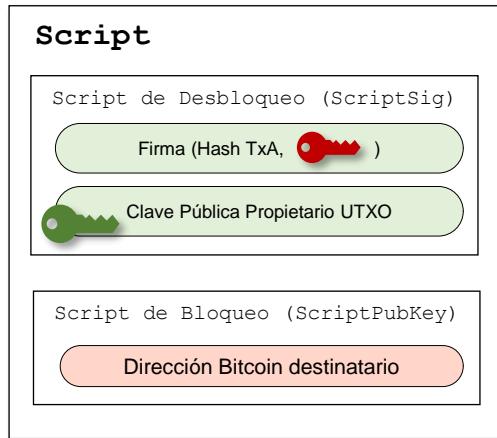


Blockchain



Cadena de Bloques (Blockchain)

Smart-Contracts Script Verification



01 Script de Desbloqueo

El usuario que desea acceder al UTXO debe proporcionar las **credenciales** necesarias para **demostrar que le pertenece**

02 Script de Bloqueo

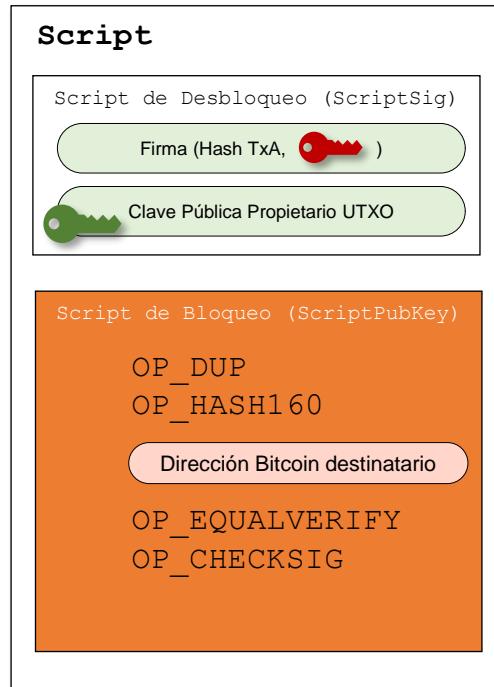
Especifica qué **condiciones** se deben **cumplir** para que alguien pueda acceder al UTXO

demostrar que la dirección bitcoin del UTXO le pertenece



Cadena de Bloques (Blockchain)

Smart-Contracts Script Verification



01 Script de Desbloqueo

El usuario que desea acceder al UTXO debe proporcionar las credenciales necesarias para demostrar que le pertenece

02 Script de Bloqueo

Especifica qué condiciones se deben cumplir para que alguien pueda acceder al UTXO

demostrar que la dirección bitcoin del UTXO le pertenece



Cadena de Bloques (Blockchain)

Smart-Contracts Script Verification

00 Cargamos credenciales en la pila
| Cargamos la Firma





Cadena de Bloques (Blockchain)

Smart-Contracts Script Verification



00 Cargamos credenciales en la pila

| Cargamos la Firma

| Cargamos la Clave Pública



Cadena de Bloques (Blockchain)

Smart-Contracts Script Verification

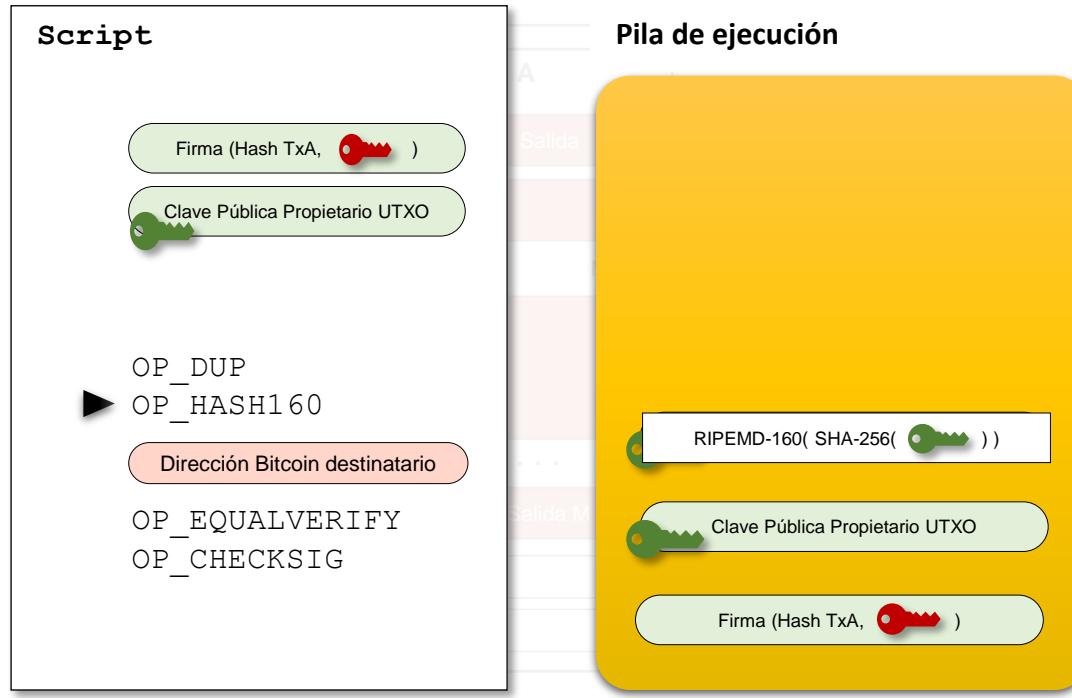


- 00 Cargamos credenciales en la pila
- 01 Verificar si la Clave Pública corresponde a la Dirección
| Duplicamos la Clave Pública

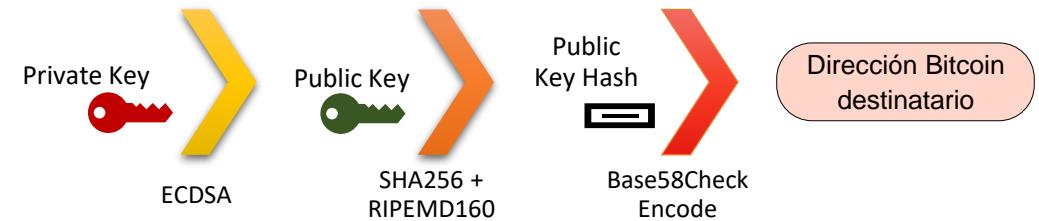


Cadena de Bloques (Blockchain)

Smart-Contracts Script Verification



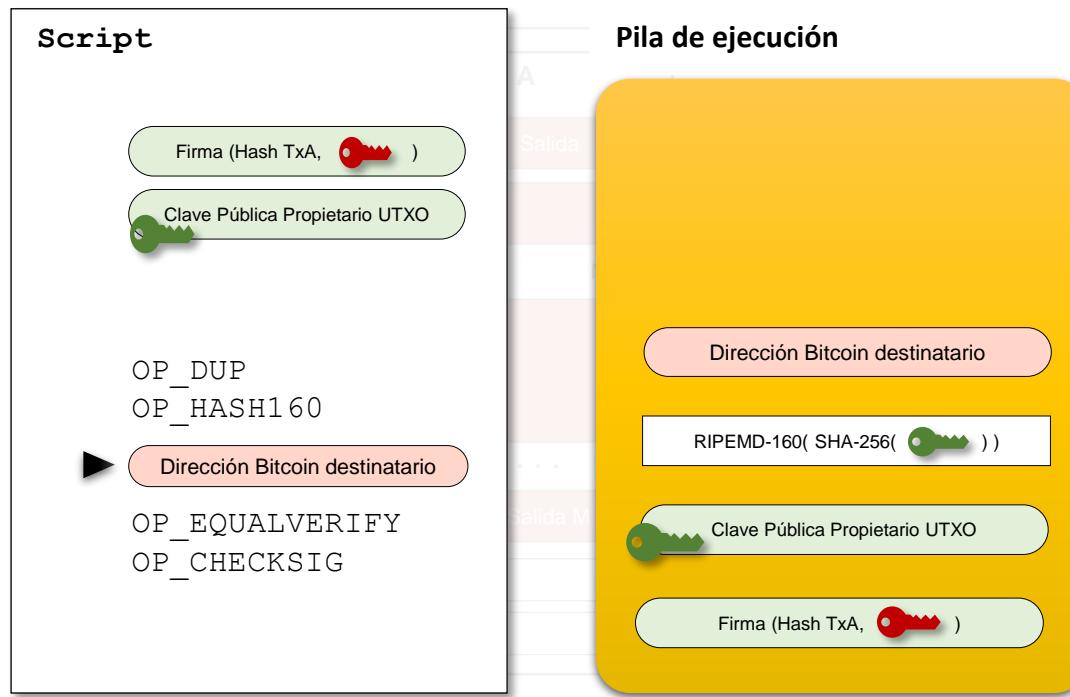
- 00 Cargamos credenciales en la pila
- 01 Verificar si la Clave Pública corresponde a la Dirección
- | Duplicamos la Clave Pública
 - | Le aplicamos un Hash



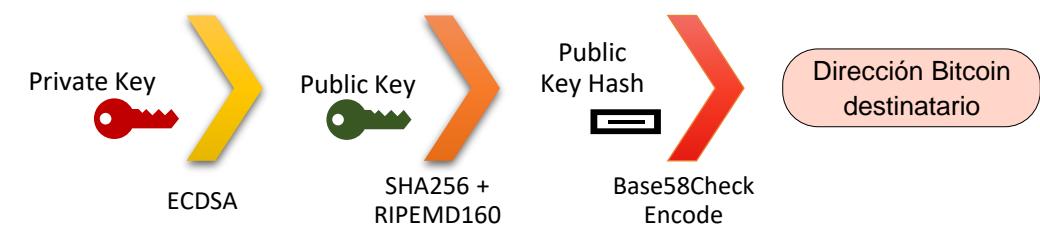
Cadena de Bloques (Blockchain)



Smart-Contracts Script Verification



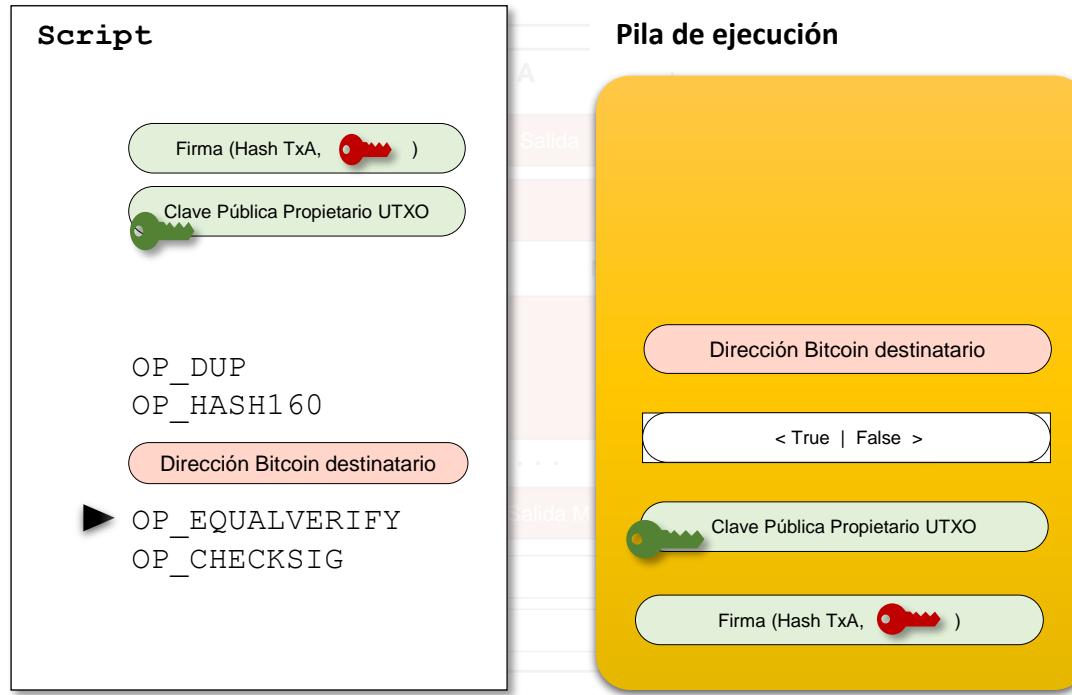
- 00 Cargamos credenciales en la pila
- 01 Verificar si la Clave Pública corresponde a la Dirección
 - | Duplicamos la Clave Pública
 - | Le aplicamos un Hash
 - | Cargamos la Dirección



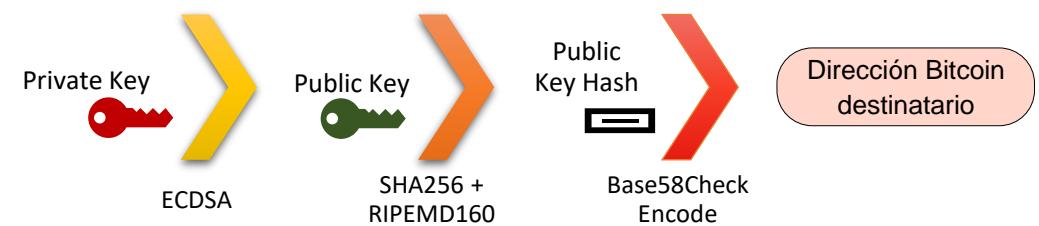


Cadena de Bloques (Blockchain)

Smart-Contracts Script Verification



- 00 Cargamos credenciales en la pila
- 01 Verificar si la Clave Pública corresponde a la Dirección
 - | Duplicamos la Clave Pública
 - | Le aplicamos un Hash
 - | Cargamos la Dirección
 - | Verificamos si son equivalentes





Cadena de Bloques (Blockchain)

Smart-Contracts Script Verification

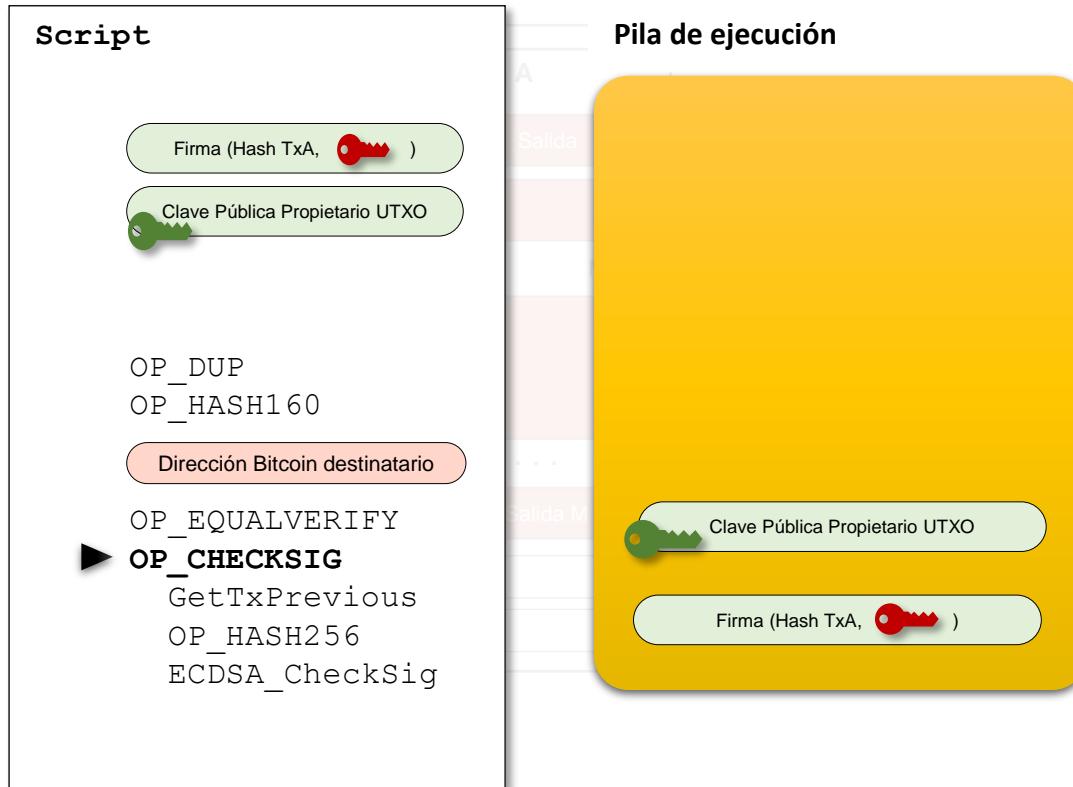


- 00 Cargamos credenciales en la pila
- 01 Verificar si la Clave Pública corresponde a la Dirección
- 02 Verificar si la **Firma** es correcta (si se generó con la Clave Privada)

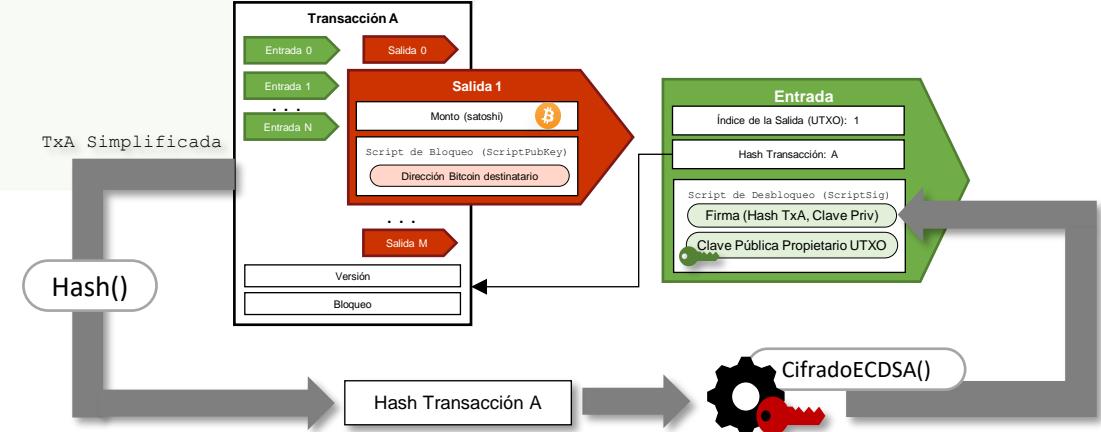


Cadena de Bloques (Blockchain)

Smart-Contracts Script Verification



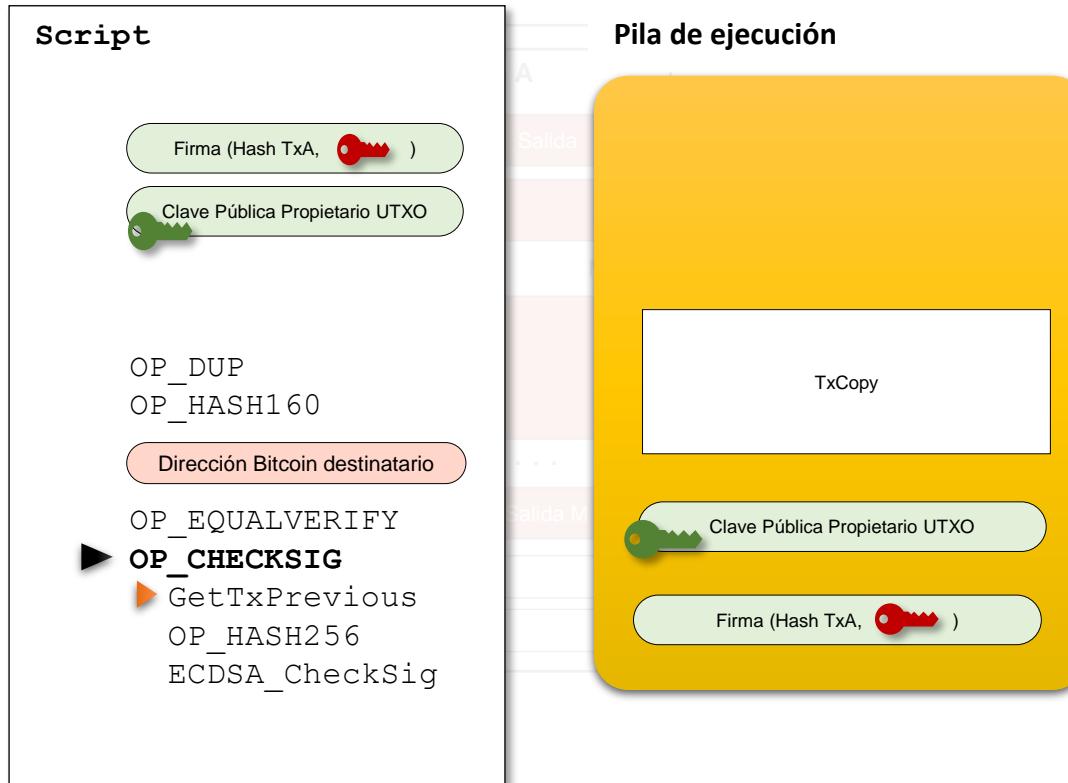
- 00 Cargamos credenciales en la pila
- 01 Verificar si la Clave Pública corresponde a la Dirección
- 02 Verificar si la **Firma** es correcta (si se generó con la Clave Privada)



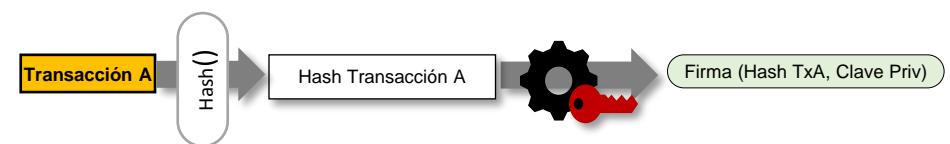
Cadena de Bloques (Blockchain)



Smart-Contracts Script Verification



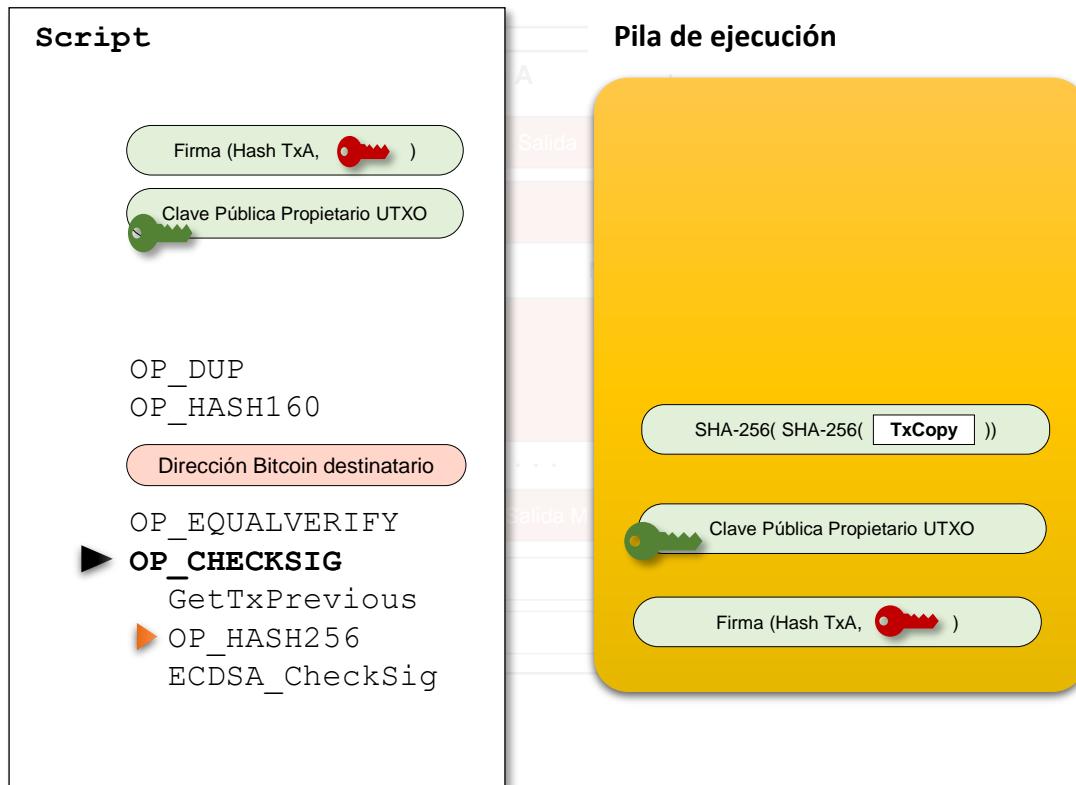
- 00 Cargamos credenciales en la pila
- 01 Verificar si la Clave Pública corresponde a la Dirección
- 02 Verificar si la **Firma** es correcta (si se generó con la Clave Privada)
| Obtenemos la transacción original



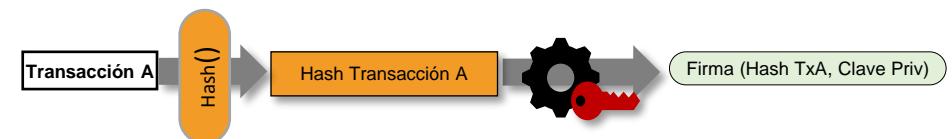
Cadena de Bloques (Blockchain)



Smart-Contracts Script Verification



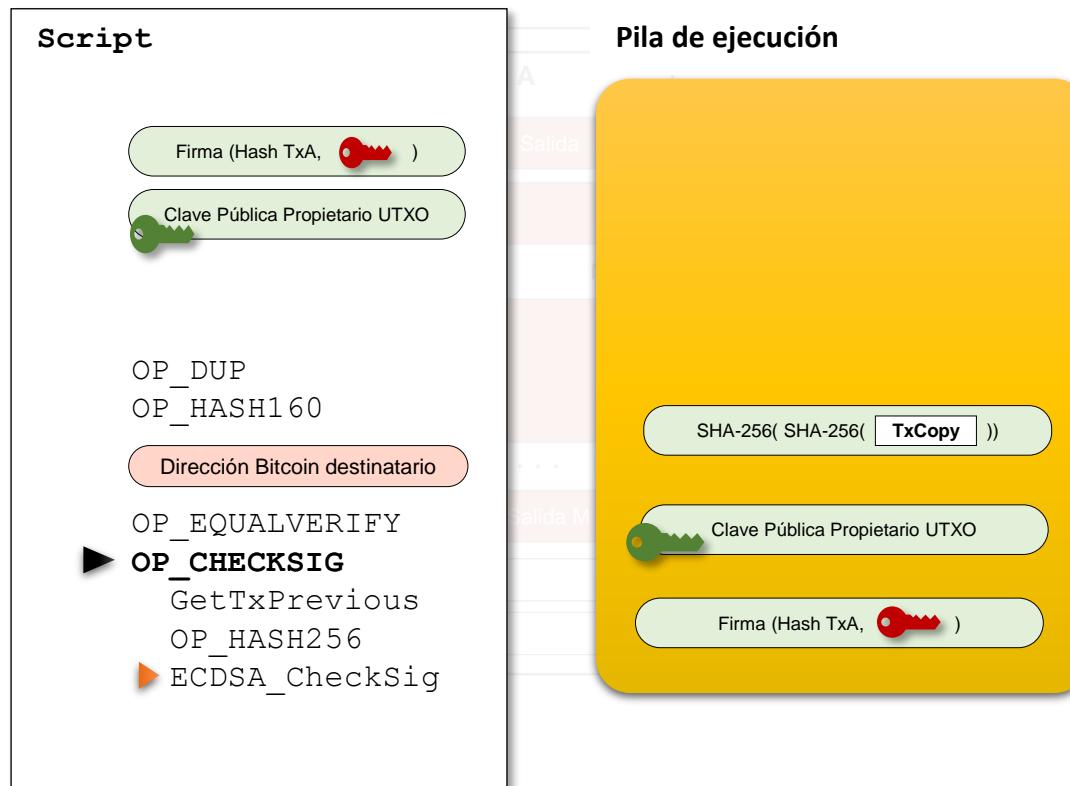
- 00 Cargamos credenciales en la pila
- 01 Verificar si la Clave Pública corresponde a la Dirección
- 02 Verificar si la **Firma** es correcta (si se generó con la Clave Privada)
 - | Obtenemos la transacción original
 - | Le aplicamos un Hash



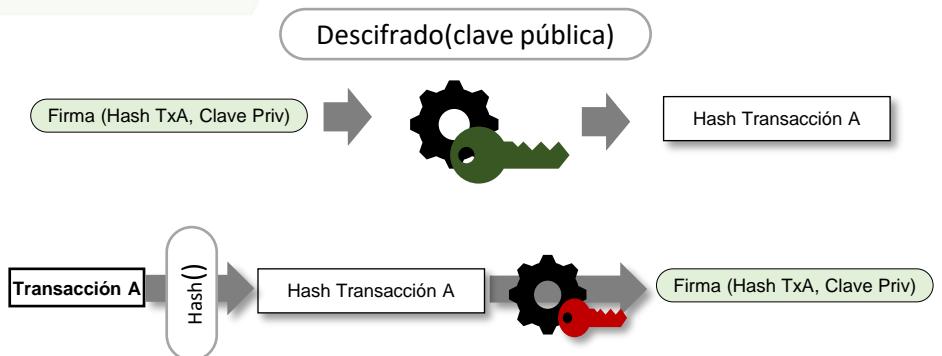
Cadena de Bloques (Blockchain)



Smart-Contracts Script Verification



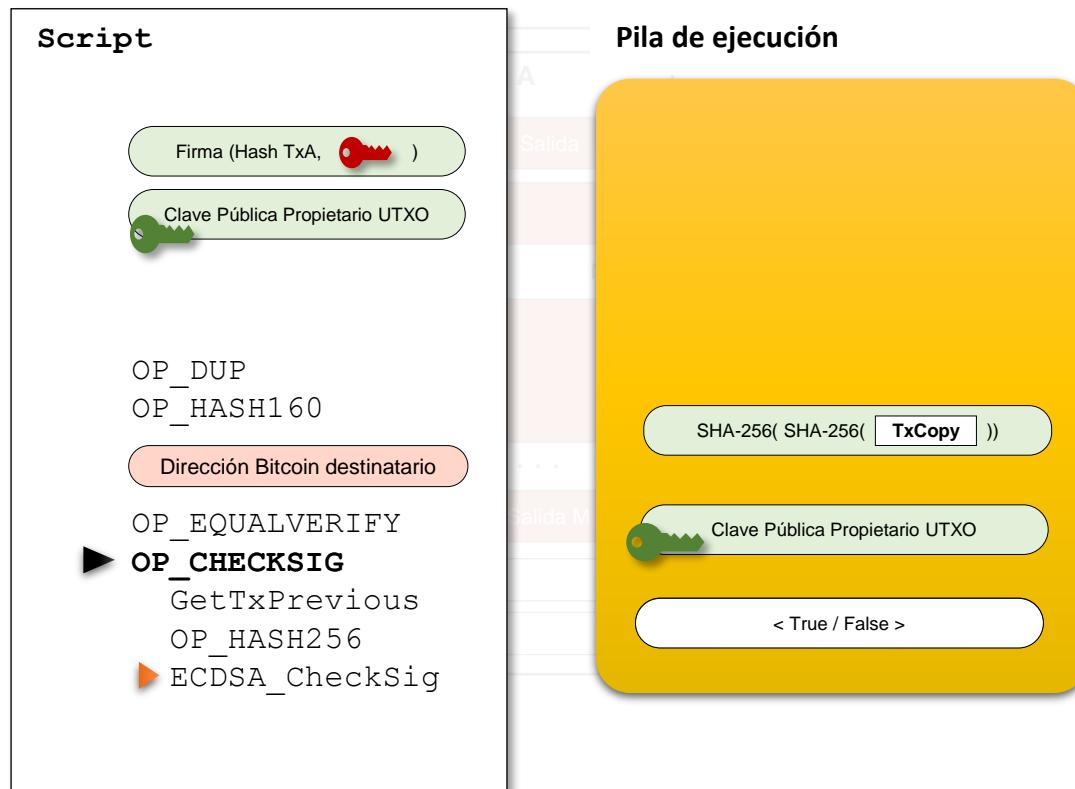
- 00 Cargamos credenciales en la pila
- 01 Verificar si la Clave Pública corresponde a la Dirección
- 02 Verificar si la **Firma** es correcta (si se generó con la Clave Privada)
 - | Obtenemos la transacción original
 - | Le aplicamos un Hash
 - | Verificamos que la firma es correcta



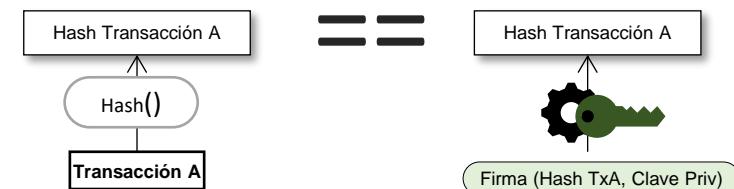
Cadena de Bloques (Blockchain)



Smart-Contracts Script Verification



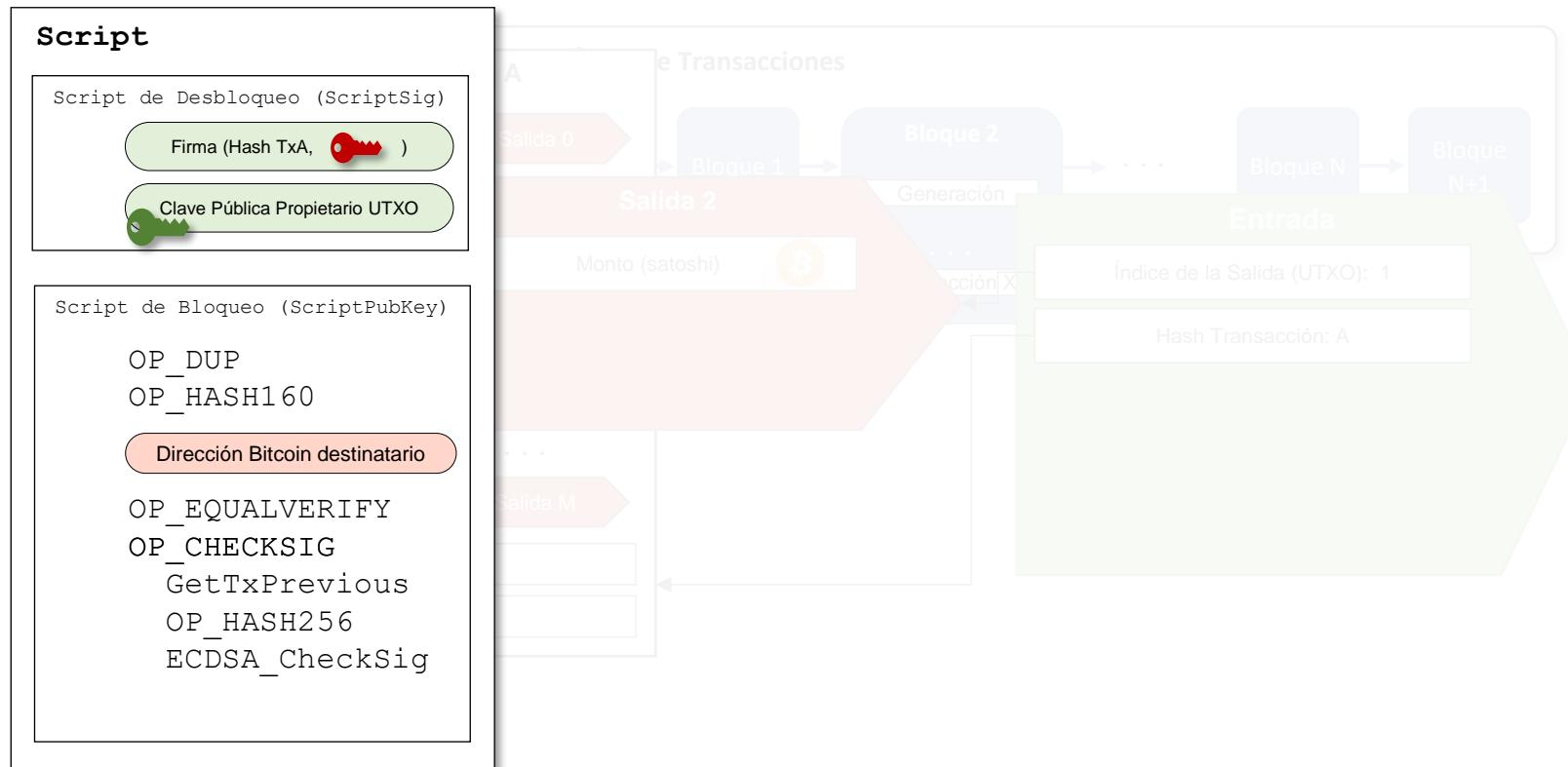
- 00 Cargamos credenciales en la pila
- 01 Verificar si la Clave Pública corresponde a la Dirección
- 02 Verificar si la **Firma** es correcta (si se generó con la Clave Privada)
 - | Obtenemos la transacción original
 - | Le aplicamos un Hash
 - | Verificamos que la firma es correcta
 - | El resultado será valor de la cima de la pila: TRUE o FALSE





Cadena de Bloques (Blockchain)

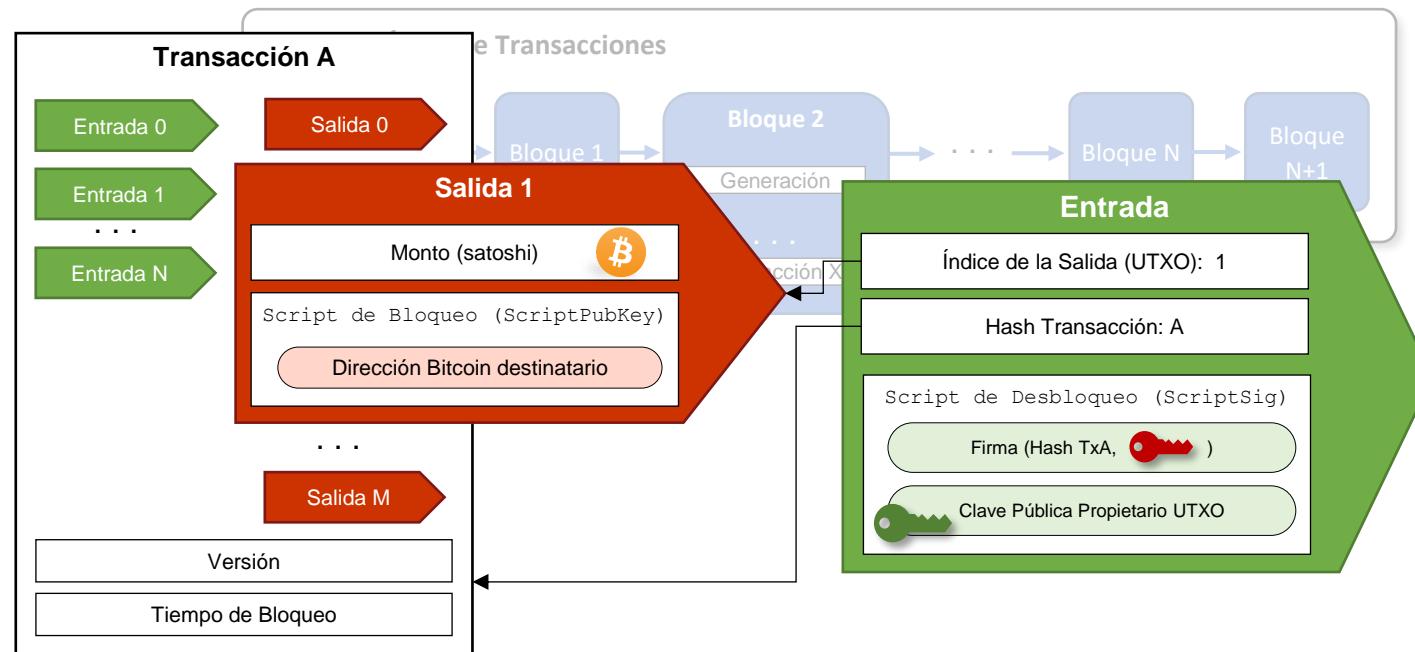
Smart-Contracts Script Verification



Cadena de Bloques (Blockchain)



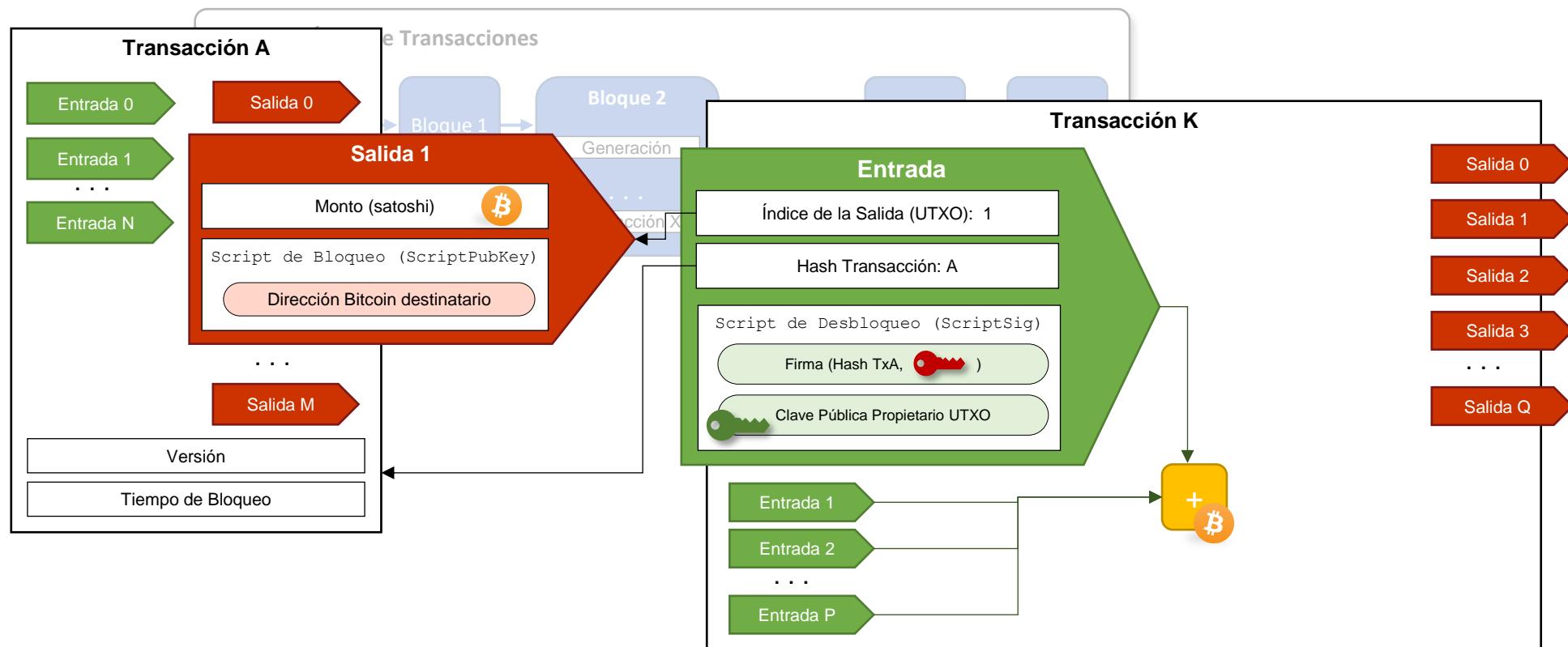
Transacciones Tx



Cadena de Bloques (Blockchain)



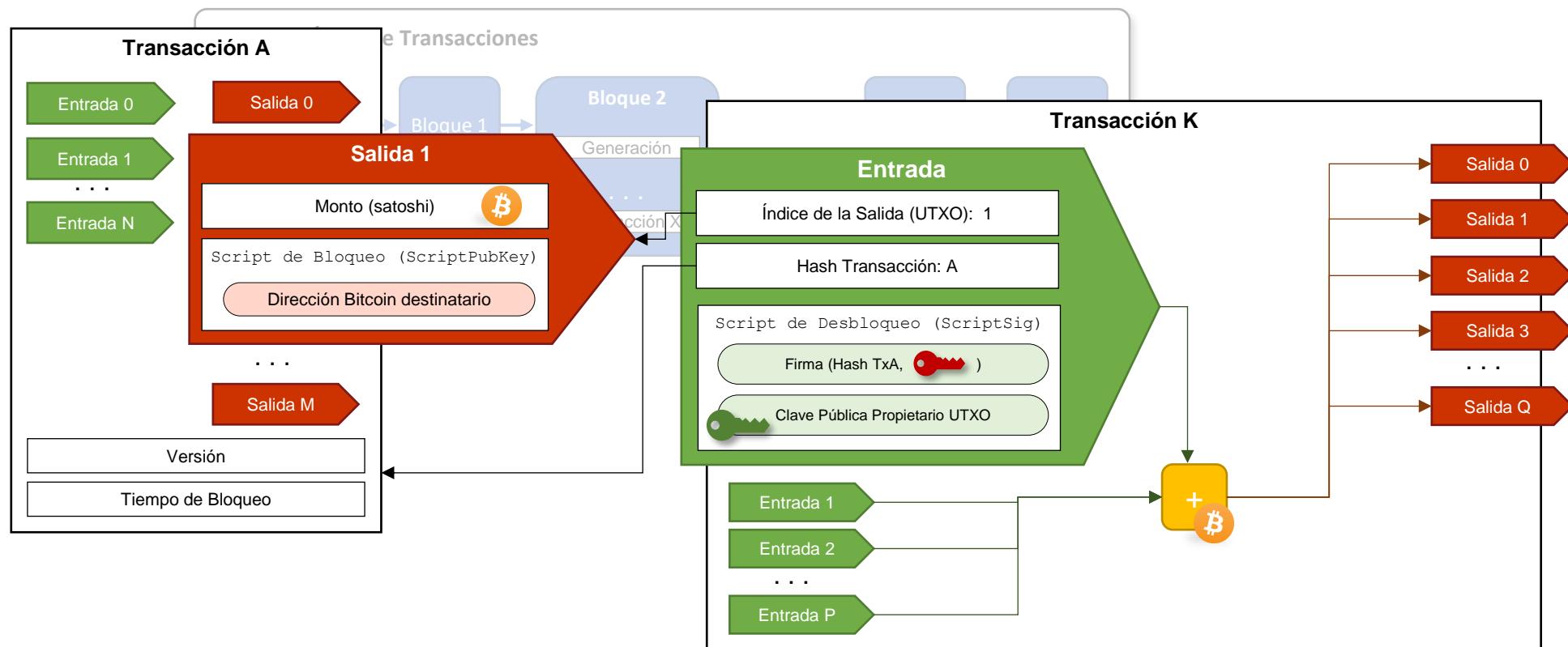
Transacciones Tx



Cadena de Bloques (Blockchain)



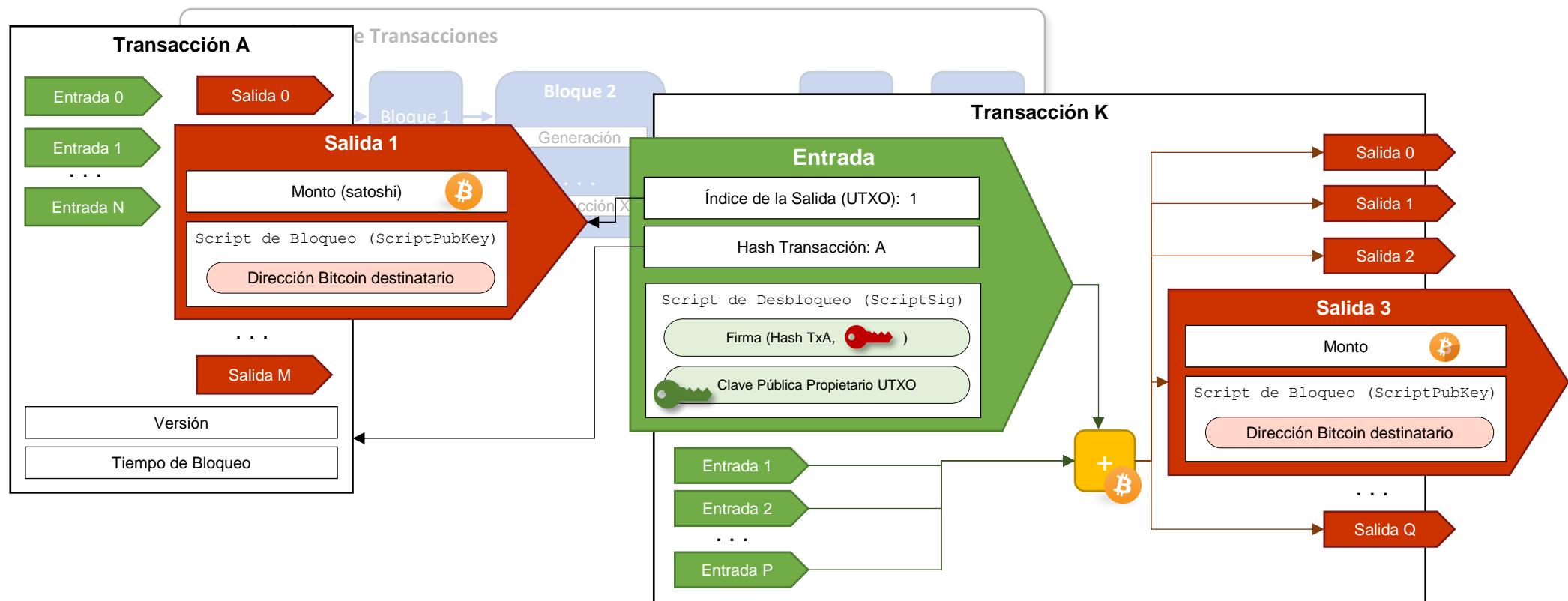
Transacciones Tx



Cadena de Bloques (Blockchain)



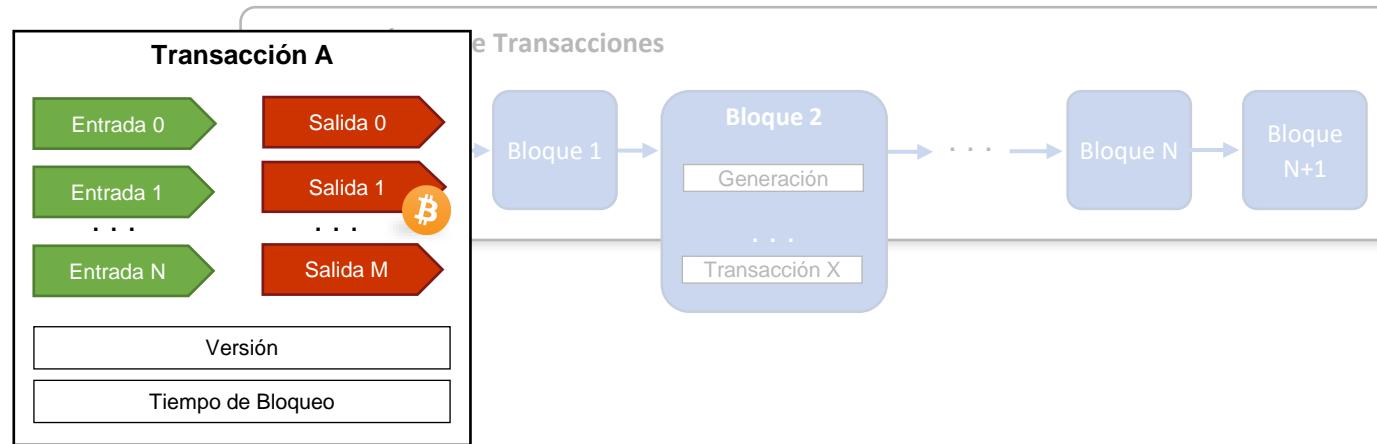
Transacciones Tx



Cadena de Bloques (Blockchain)



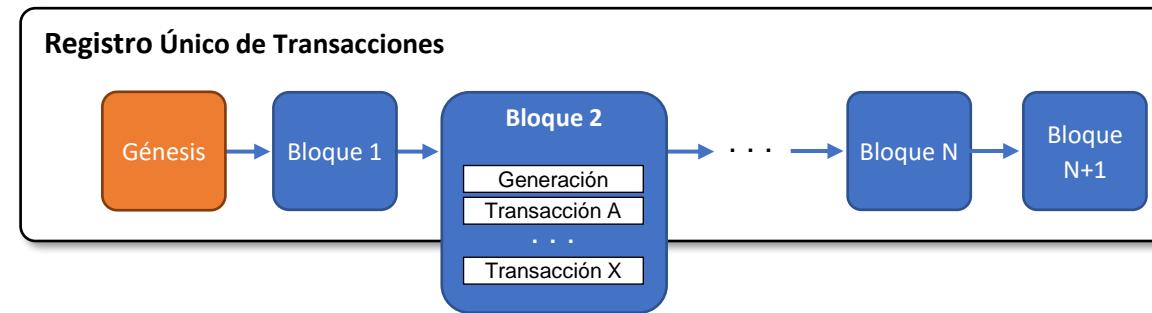
Transacciones Tx



Cadena de Bloques (Blockchain)



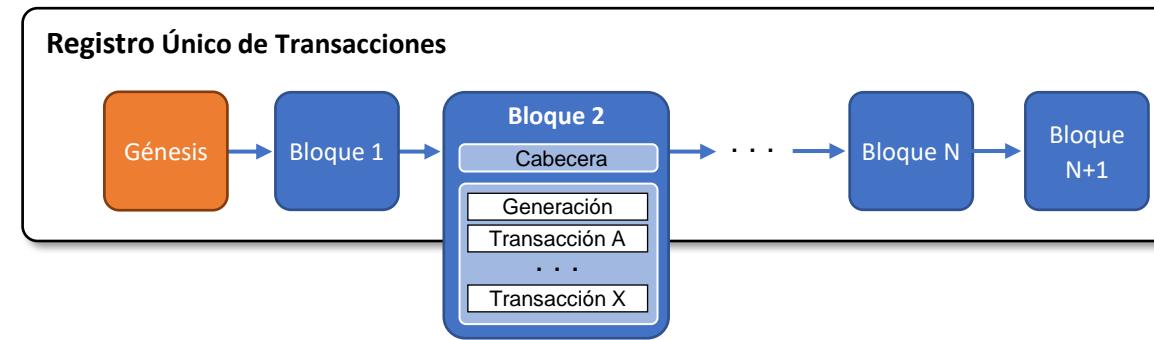
Transacciones Tx



Cadena de Bloques (Blockchain)



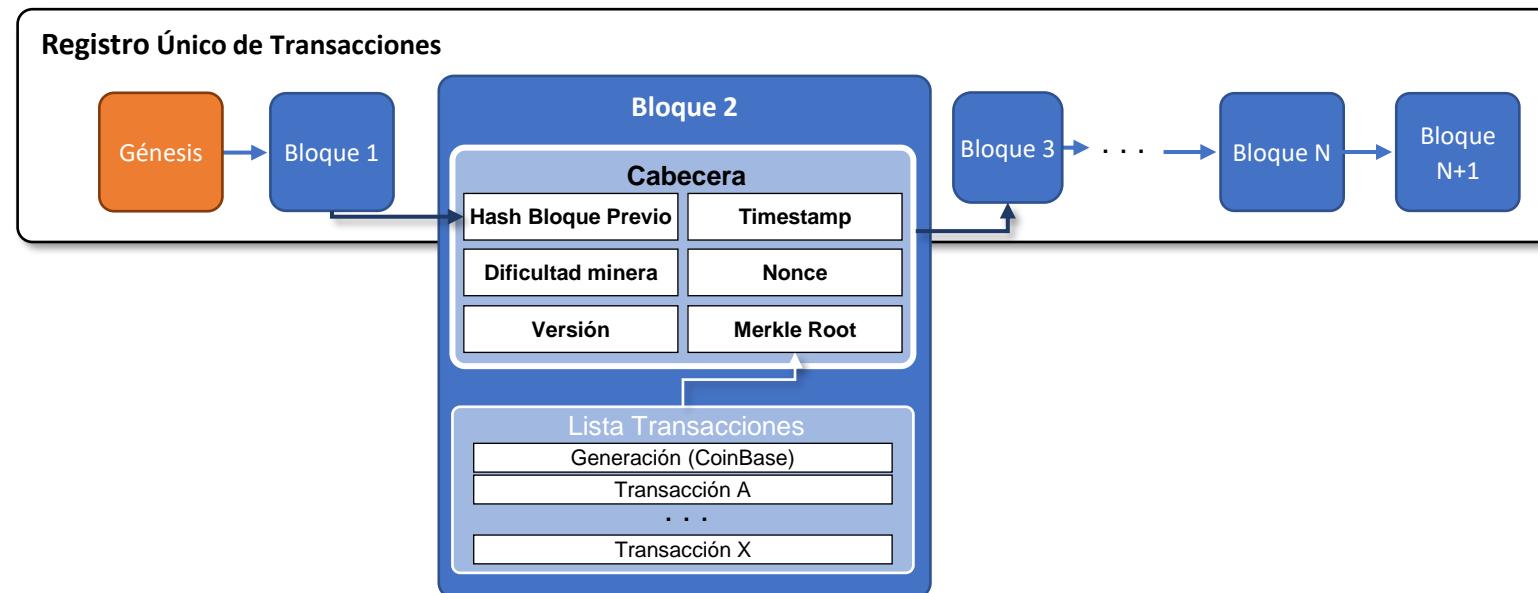
Cabecera de bloque



Cadena de Bloques (Blockchain)



Cabecera de bloque



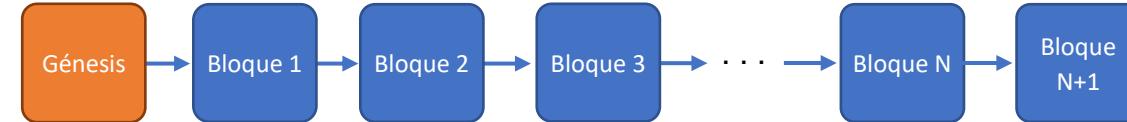


Cadena de Bloques (Blockchain)

Cabecera de bloque



Registro Único de Transacciones



Red P2P





Red P2P Bitcoin

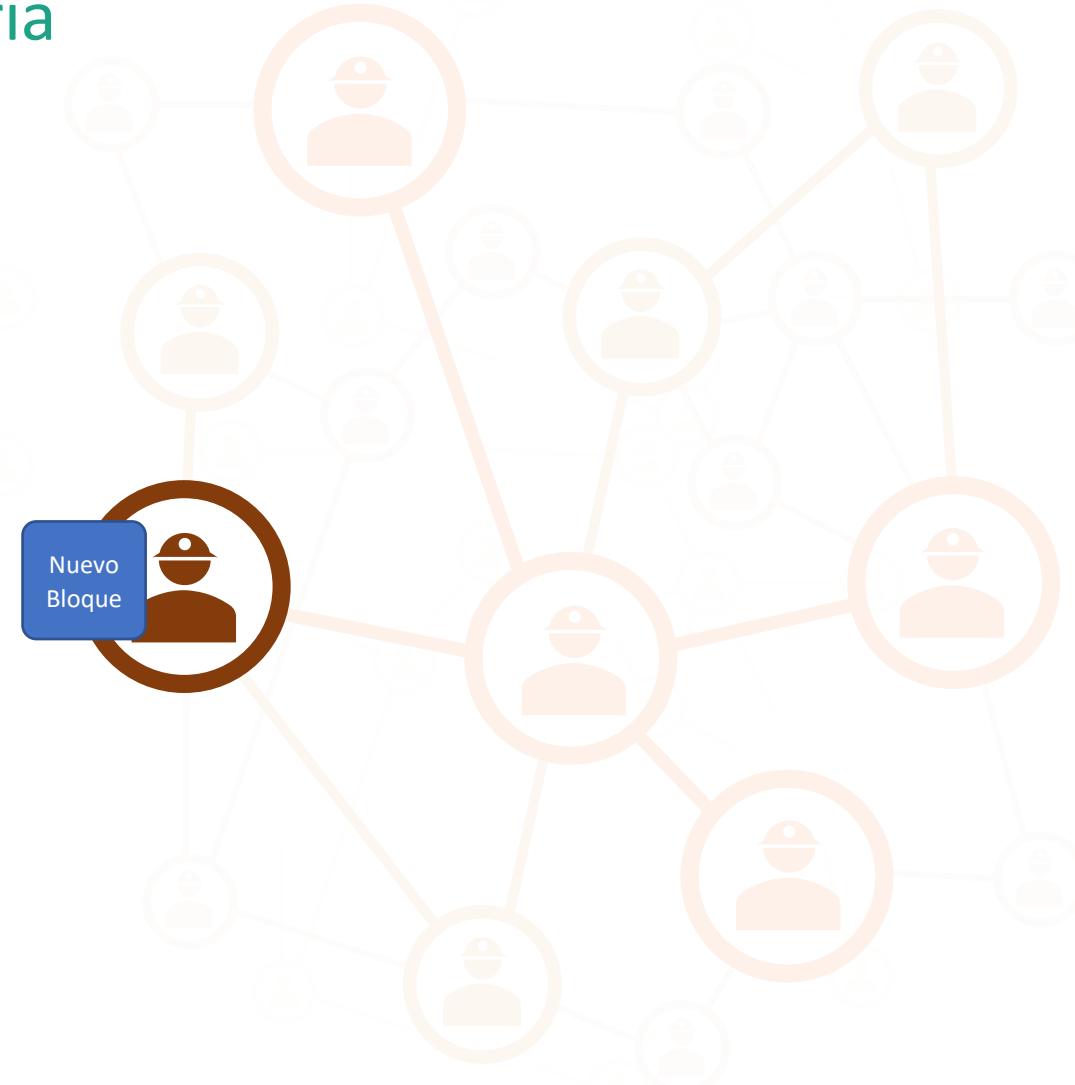
Mineros y Minería





Red P2P Bitcoin

Mineros y Minería





Red P2P Bitcoin

Mineros y Minería



01 PC / Workstation

- | Arquitectura de propósito general
- | Poco eficiente
- | Alto consumo

02 GPU

- | Arquitectura para gráficos
- | Eficiente en operaciones matemáticas
- | Consumo medio/alto

03 ASIC

- | Arquitectura a medida
- | Muy eficiente
- | Medio/bajo consumo





Red P2P Bitcoin

Mineros y Minería



03 ASIC

- | Arquitectura a medida
- | Muy eficiente
- | Medio/bajo consumo



04 Clusters de nodos

- | Cientos o miles de nodos
- | Alto ratio de Hash (TH/s)
- | Enorme consumo y calor

05 Consorcio de mineros (Pools)

- | Trabajo colaborativo
- | Reparto de los beneficios
- | Estrategias más complejas

Nuevo
Bloque



Red P2P Bitcoin

Mineros y Minería



03 ASIC

- Arquitectura a medida
- Muy eficiente
- Medio/bajo consumo

Bitcoin Core

- Software nodo de red bitcoin
- Gratis y de código abierto
- Proporciona una **billetera** de bitcoin que verifica completamente los pagos



Red P2P Bitcoin

Mineros y Minería

01 Cada nuevo nodo se conecta (puerto TCP 8333) al menos a otro nodo de la red

- | Mediante direcciones IP establecidas en el código
- | Mediante consulta al nodo dnsseed.bluematt.me que proporciona IPs actualizadas

02 Tras la conexión

- | Descubre nuevos nodos para asegurar conectividad: intercambio de IPs de vecinos entre nodos
- | Debe descargar toda la cadena y validarla (proceso lento y consume muchos recursos)

03 Comunicación entre nodos por inundación. Por cada nueva transacción o bloque recibidos:

- | Validar
- | Transmitirla a nodos vecinos





Red P2P Bitcoin

Mineros y Minería



03 ASIC

- Arquitectura a medida
- Muy eficiente
- Medio/bajo consumo

Bitcoin Core

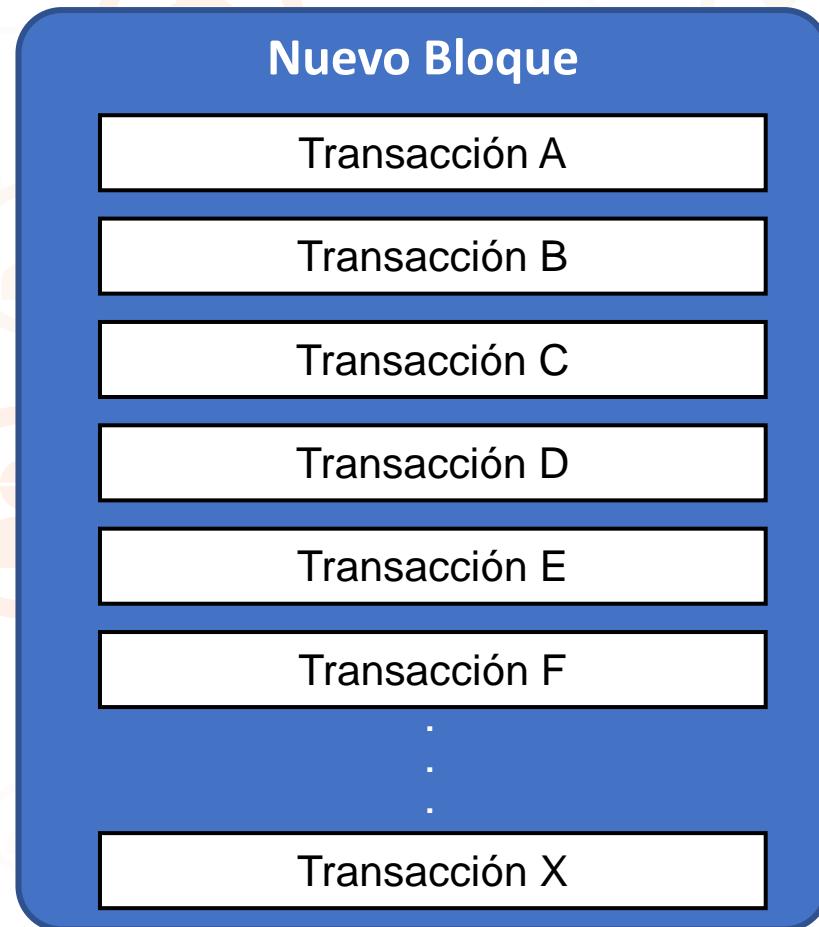
- Software nodo de red bitcoin
- Gratis y de código abierto
- Proporciona una **billetera** de bitcoin que verifica completamente los pagos





Red P2P Bitcoin

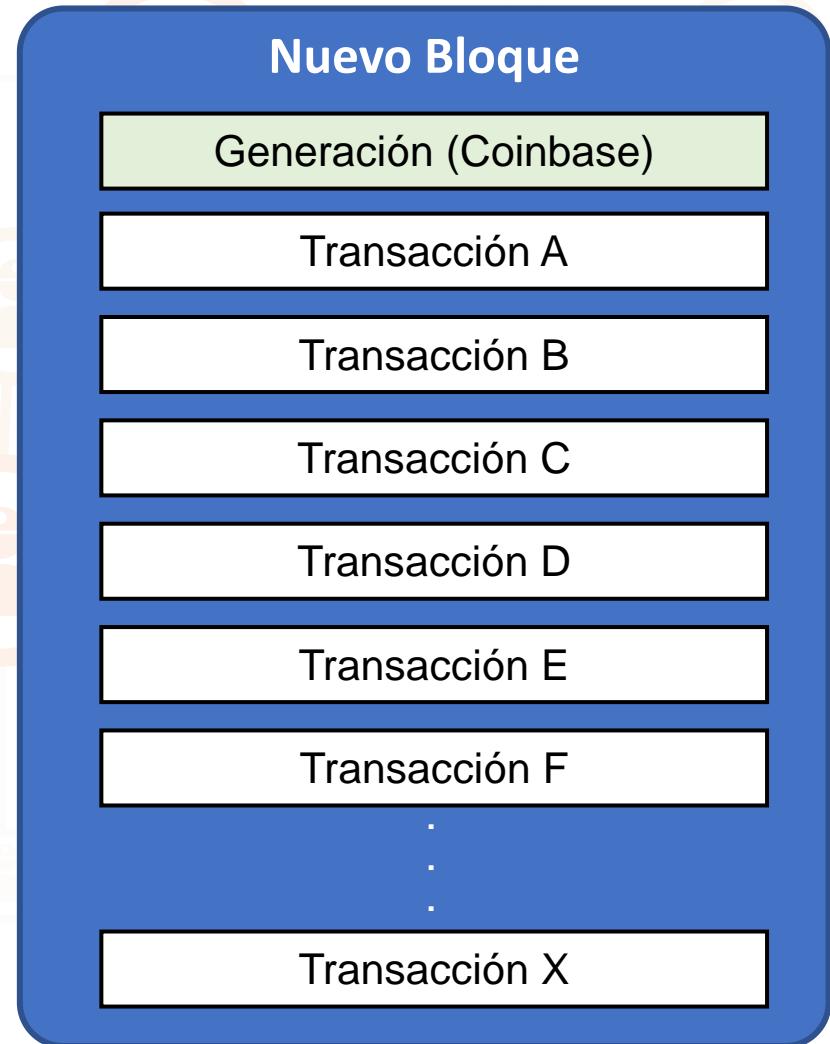
Mineros y Minería





Red P2P Bitcoin

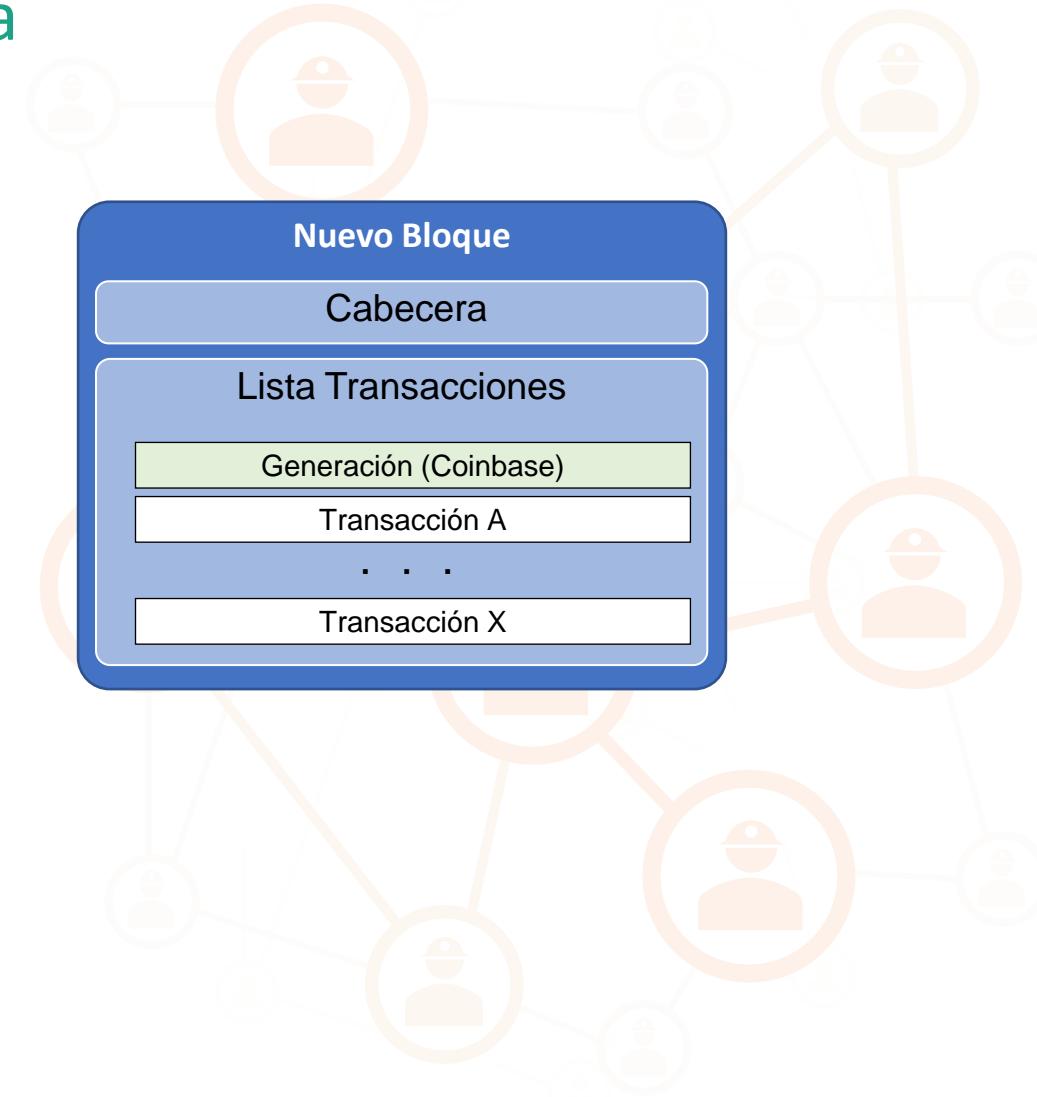
Mineros y Minería





Red P2P Bitcoin

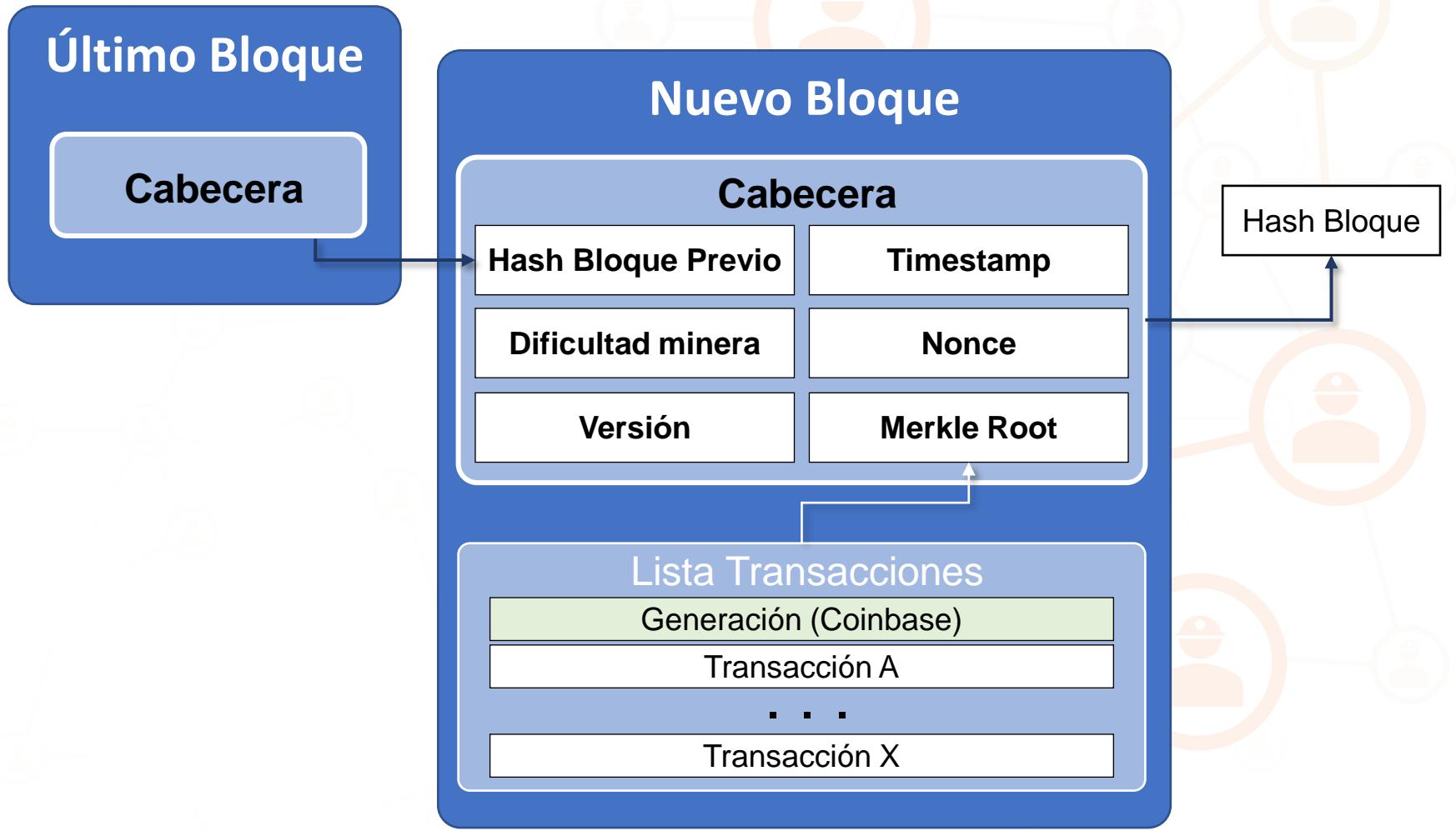
Mineros y Minería





Red P2P Bitcoin

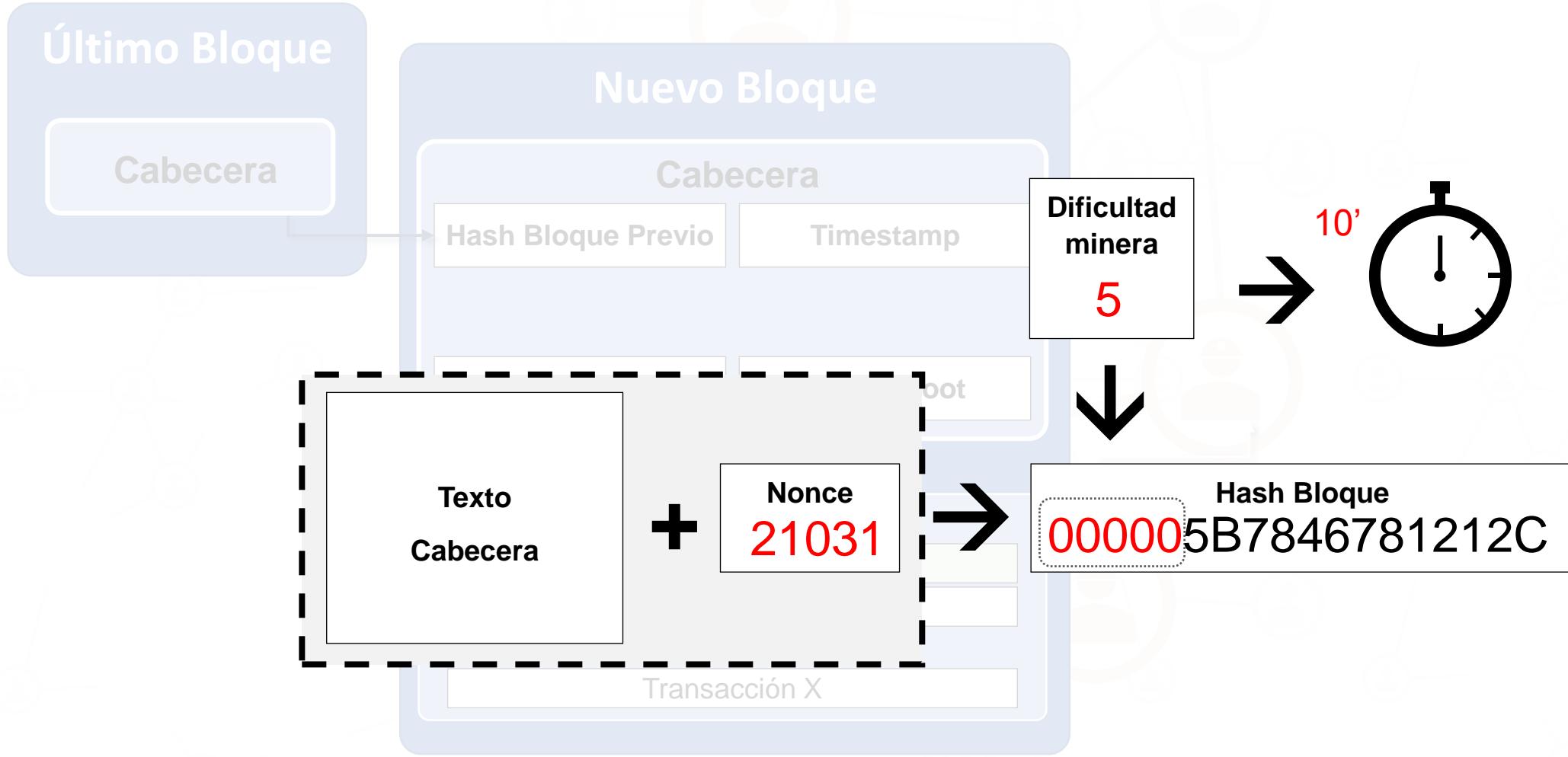
Minando un Bloque





Red P2P Bitcoin

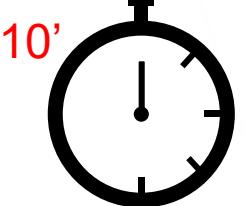
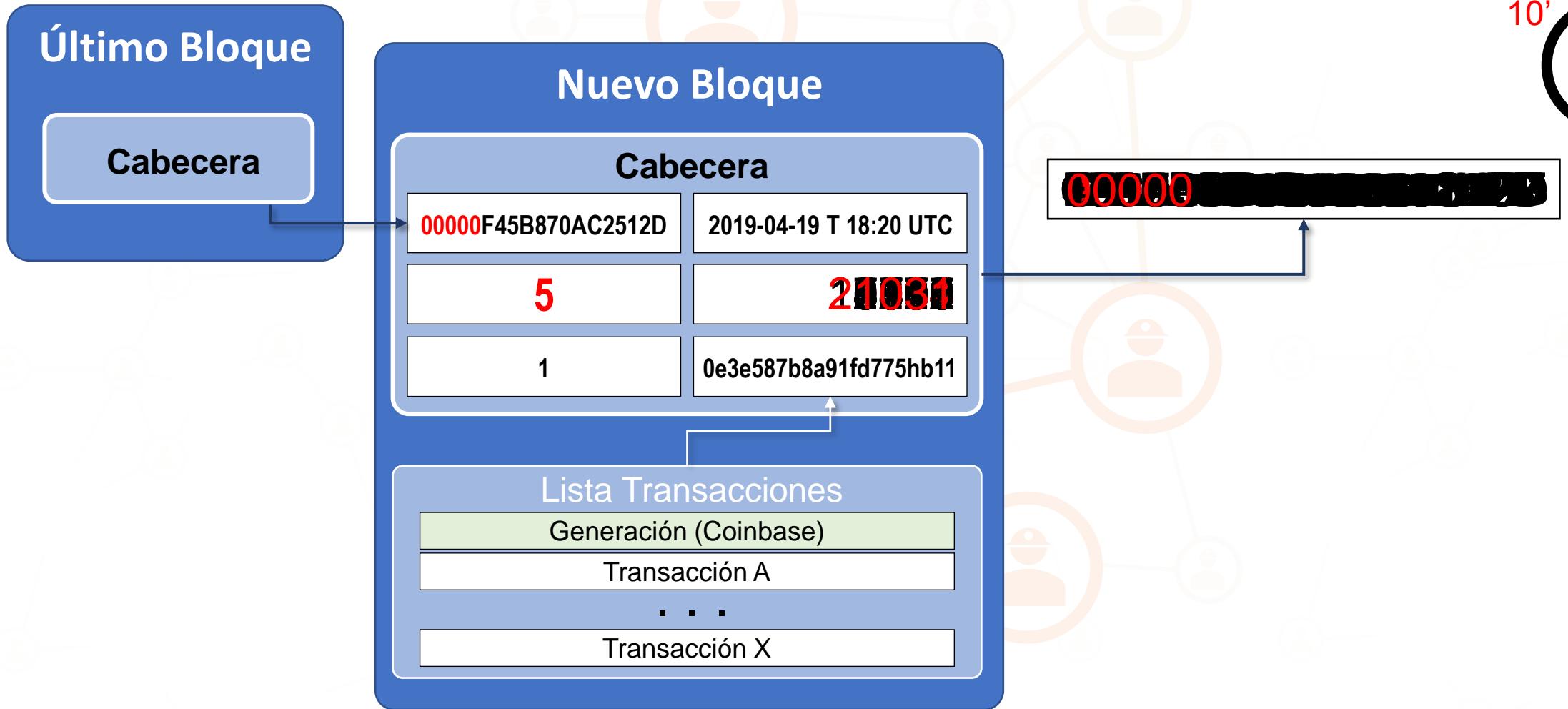
Prueba de trabajo Proof-Of-Work system





Red P2P Bitcoin

Prueba de trabajo Proof-Of-Work system





Red P2P Bitcoin

Mineros y Minería





Red P2P Bitcoin

Mineros y Minería





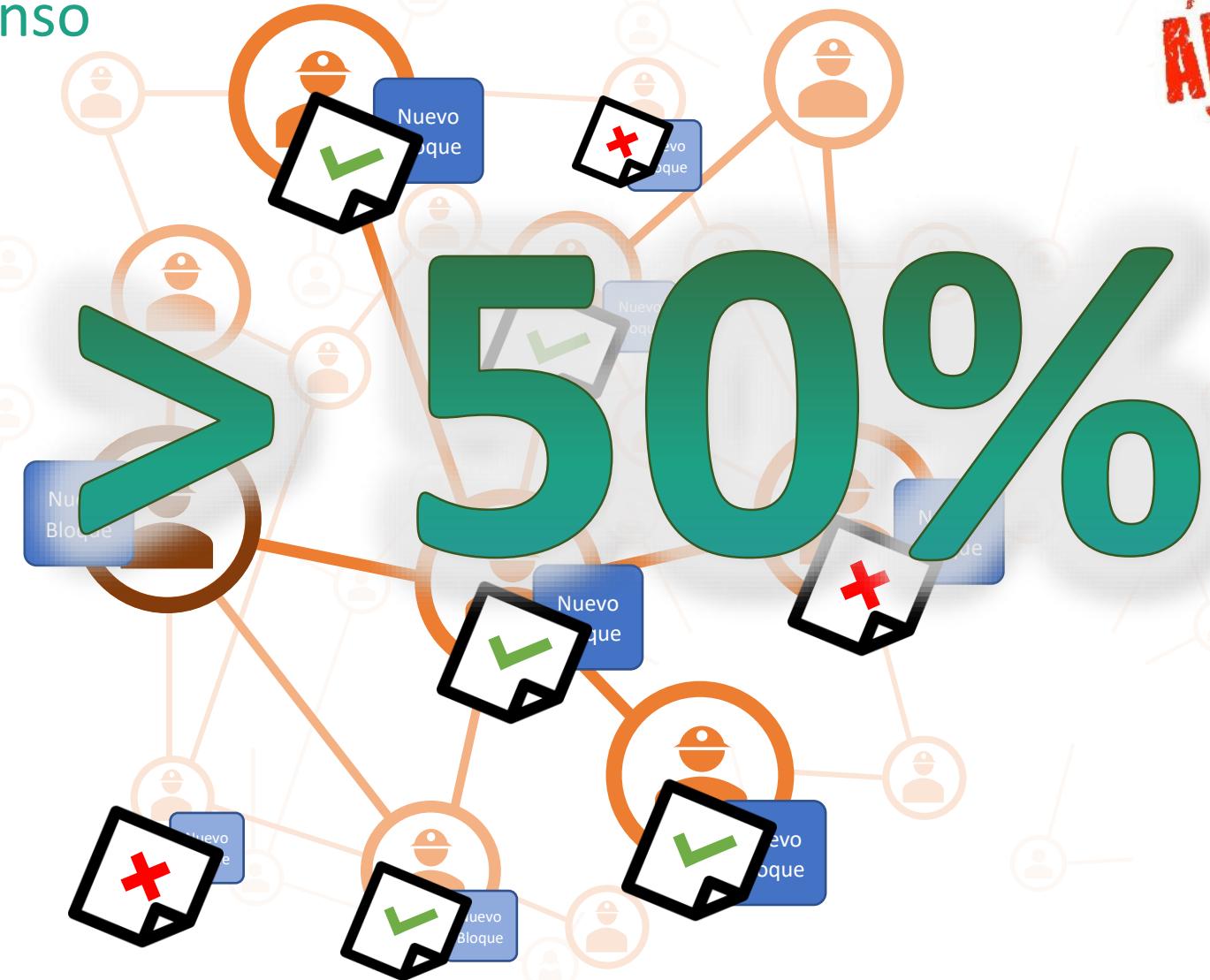
Red P2P Bitcoin

Minería y Consenso



Red P2P Bitcoin

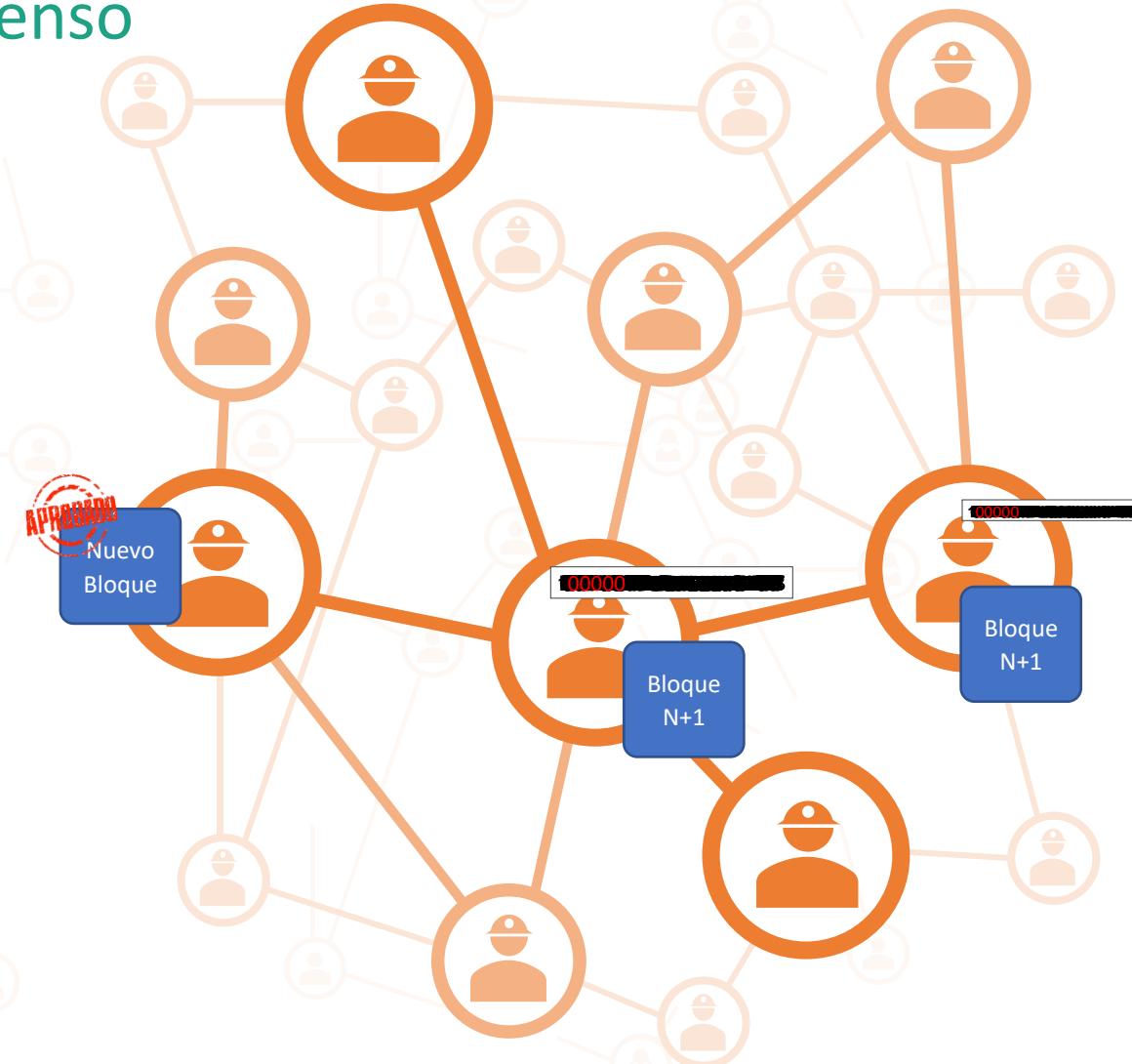
Minería y Consenso





Red P2P Bitcoin

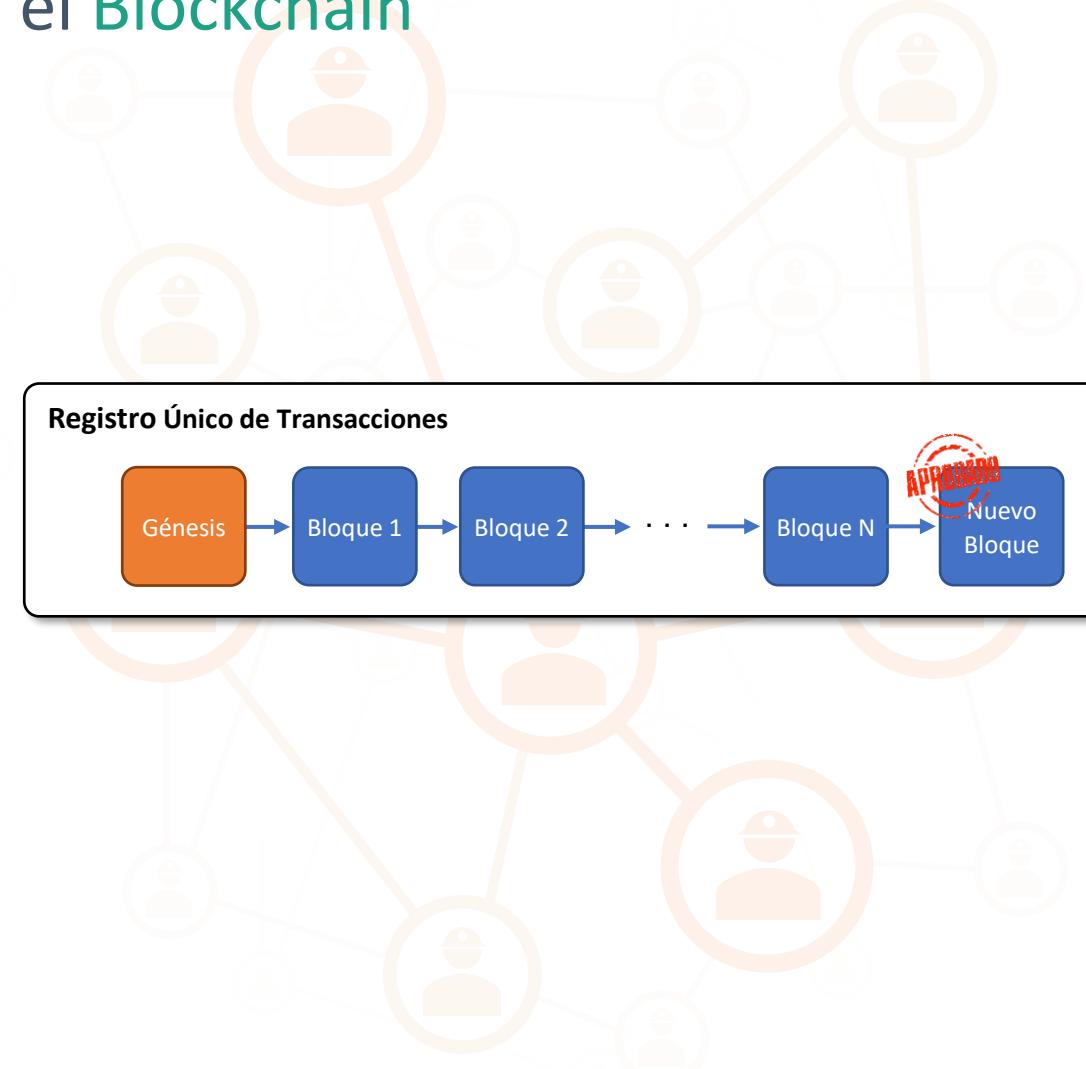
Minería y Consenso





Red P2P Bitcoin

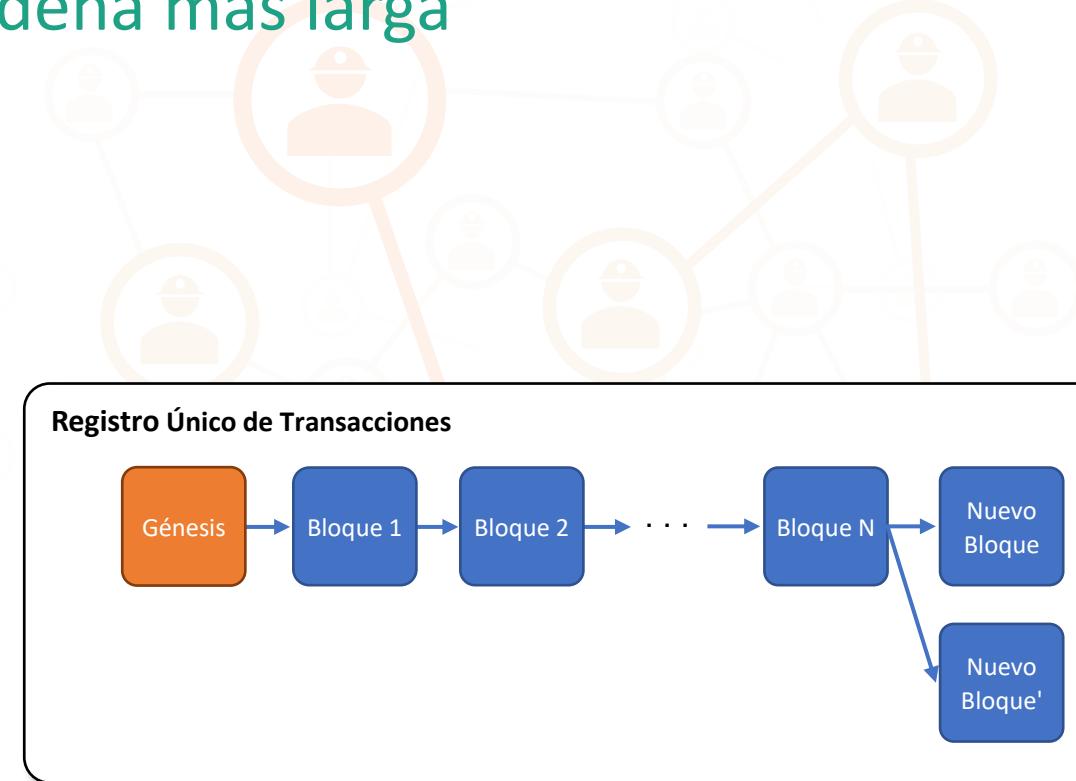
Nuevo Bloque en el Blockchain





Red P2P Bitcoin

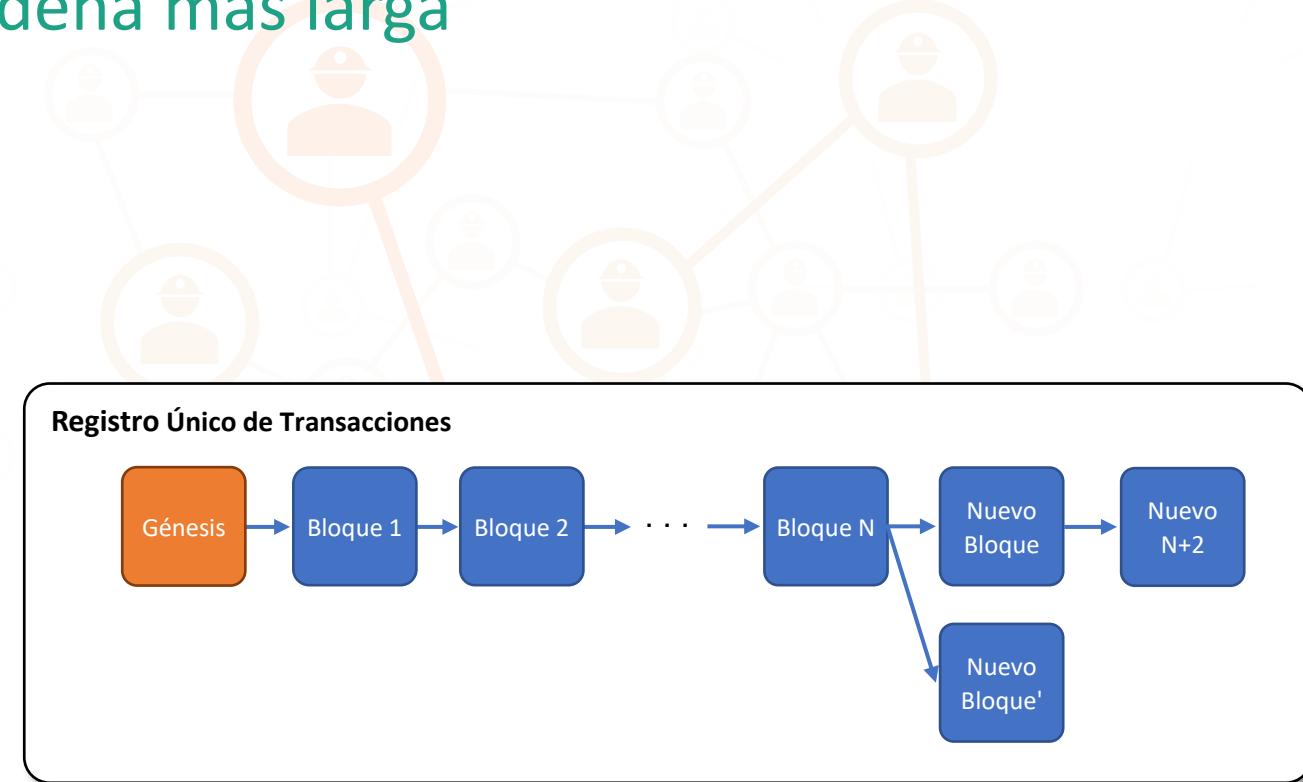
Conflictos y la Cadena más larga





Red P2P Bitcoin

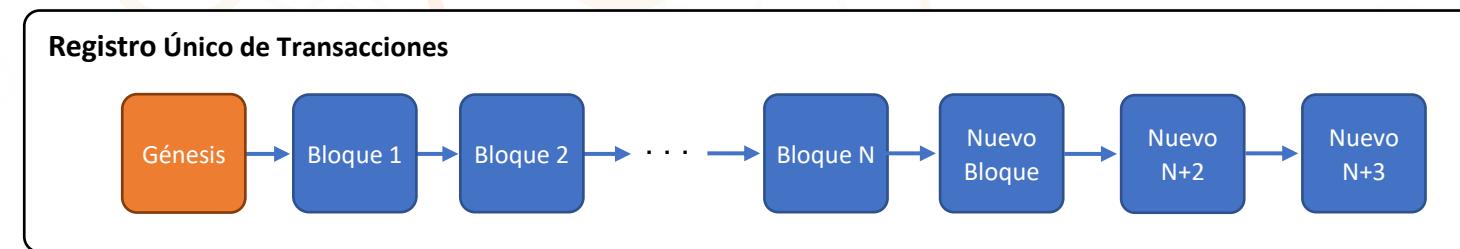
Conflictos y la Cadena más larga



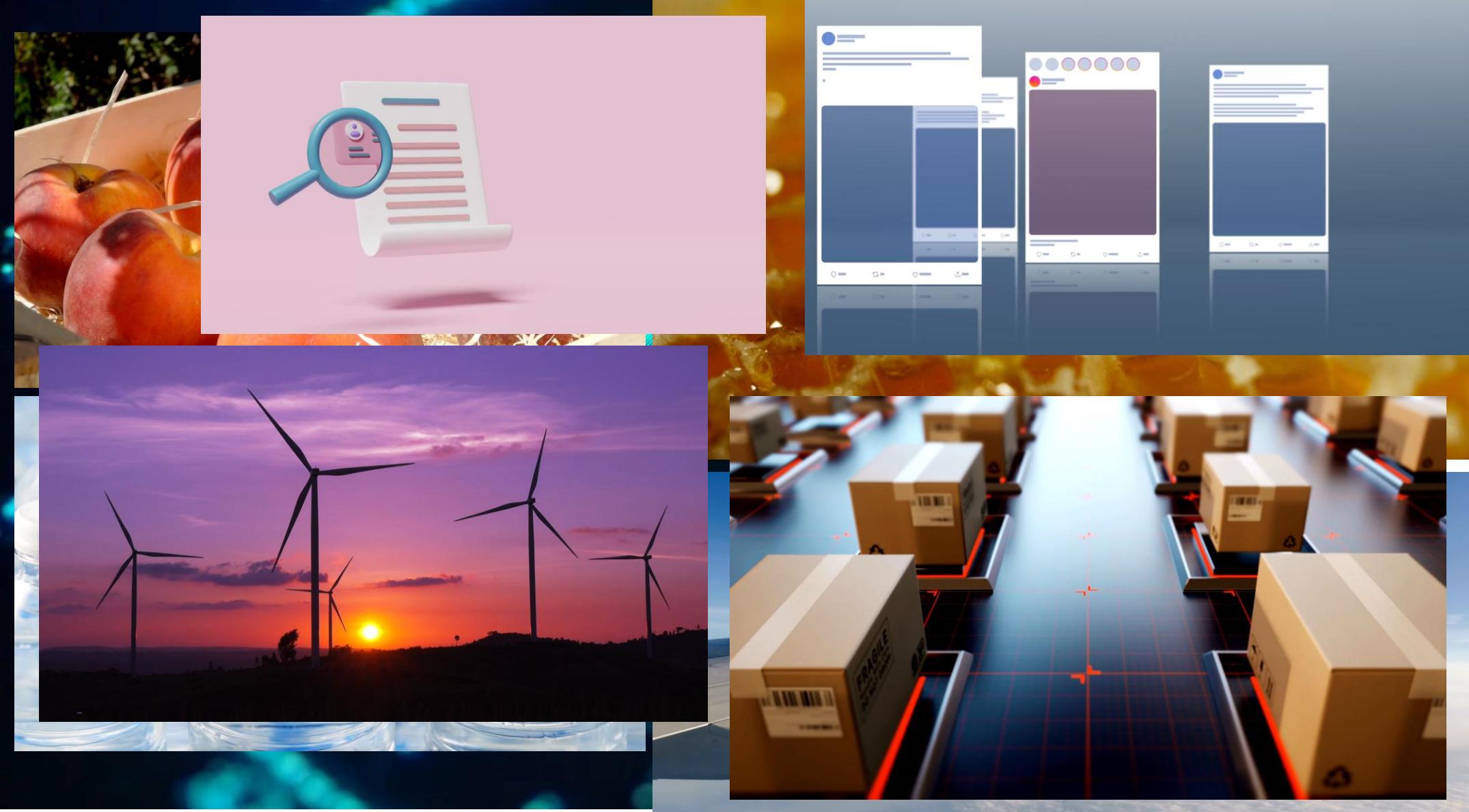


Red P2P Bitcoin

Conflictos y la Cadena más larga



Aplicaciones de Blockchain



Blockchain

Conclusiones

- 01 La **arquitectura descentralizada** de blockchain aumenta la resiliencia del sistema al **eliminar puntos únicos de fallo**
- 02 Blockchain utiliza **técnicas criptográficas avanzadas** para asegurar la **integridad** y la **confidencialidad** de los datos
- 03 Los **mecanismos de consenso**, como Proof of Work (PoW) o Proof of Stake (PoS), aseguran que todos los participantes en la red **acuerden el estado actual** de la cadena de bloques, lo que previene fraudes y manipulaciones

Conclusiones

- 04 Una vez que un **bloque** es **añadido** a la cadena de bloques, es **extremadamente difícil modificar la información** contenida sin alterar todos los bloques subsecuentes, lo que proporciona una **alta integridad** de los datos
- 05 A pesar de sus beneficios, blockchain enfrenta **desafíos** significativos en términos de **escalabilidad** y **rendimiento**.
- 06 Blockchain no solo es útil para las criptomonedas; su **aplicación** se extiende a diversas áreas como la **gestión de identidades**, el seguimiento de la **cadena de suministro**, el voto electrónico, y la gestión de registros médicos, ofreciendo **seguridad y transparencia**

Sistemas Operativos y Distribuidos

Iren Lorenzo Fonseca
iren.fonseca@.ua.es



TEMA 3. Sistemas Distribuidos.

Seguridad en Sistemas
Distribuidos