# UD 7. Sistemas Distribuidos Conceptos Básicos

## Concurrencia y Sistemas Distribuidos

# Objetivos de la Unidad Didáctica

- Caracterizar a los sistemas distribuidos, detallando sus características principales.
- Identificar el principal objetivo que persiguen así como las dificultades que deben superarse.
- Identificar las técnicas y principios de diseño habituales para su construcción.

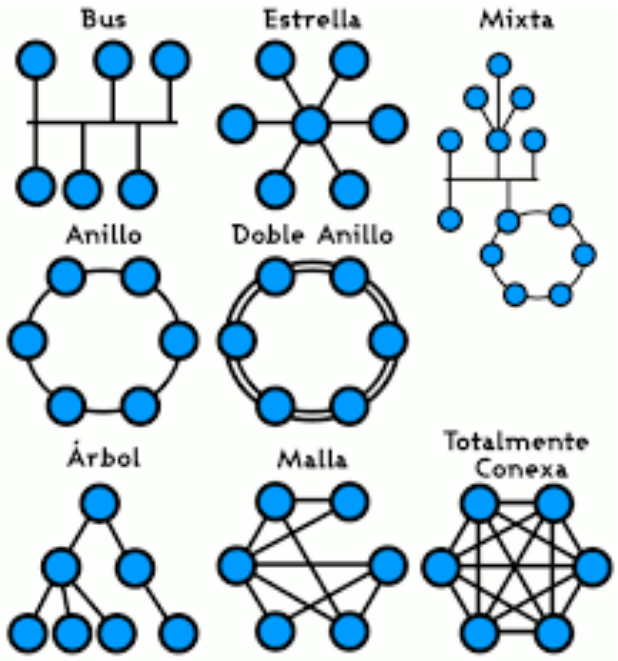
# Contenido

- Concepto de Sistema Distribuido
  - Definición de Sistema Distribuido
  - Ejemplos
  - Objetivo de los Sistemas Distribuidos
  - Middleware
- Características de los Sistemas Distribuidos
  - Transparencia
  - Disponibilidad
  - Escalabilidad
  - Seguridad

# Definición de Sistema Distribuido

- Extensión natural de los **sistemas concurrentes**, cuando estos se ejecutan en más de un ordenador.

- A cada ordenador del sistema distribuido le llamamos **nodo**
  - Los nodos se conectan unos con otros mediante canales de comunicación.
  - El resultado es un grafo, donde los ordenadores son los nodos y las aristas del grafo son los canales de comunicación.
  - Dependiendo de la forma del grafo, tendremos diferentes **topologías** (grafo totalmente conexo, grafo en anillo, etc.)



# Definición de Sistema Distribuido

- **Sistema Distribuido: conjunto de ordenadores independientes que ofrecen a sus usuarios la imagen de un sistema coherente único.**
  - A nivel de hardware: máquinas autónomas
  - A nivel de usuario: imagen de sistema único
  - A nivel interno: algoritmos distribuidos

- A nivel de hardware: **Máquinas autónomas**
  - Entre ellas no se comparte hardware (memoria, reloj, disco, etc.).
  - Son máquinas que se ejecutan y fallan de forma independiente unas de otras.
  - Se trata de ordenadores que podrían separarse físicamente, sin que exista hardware que lo impida.

# Definición de Sistema Distribuido

- A nivel de usuario: **Imagen de sistema único**
  - Los usuarios que acceden al sistema distribuido lo observan como si fuera un único ordenador.
  - También nos referimos a esta propiedad como **transparencia de distribución**: se oculta el hecho de que el sistema está distribuido entre varios ordenadores.
    - Este aspecto lo diferencia de los *sistemas en red*, donde los usuarios acceden a cada uno de los diferentes ordenadores que forman la red, de forma individual, conociendo sus nombres o direcciones de red.

# Definición de Sistema Distribuido

- A nivel interno: **Algoritmos distribuidos**
  - Cada **nodo** ejecuta una parte del algoritmo.
  - La ejecución es concurrente entre todos los nodos.
  - Un sistema distribuido puede verse como una **colección de algoritmos distribuidos** construidos con un objetivo común.
  - En cada algoritmo distribuido, los nodos se comunican entre sí y se sincronizan, habitualmente mediante el envío y recepción de mensajes.

    - Los algoritmos distribuidos tienen muchas semejanzas con los *algoritmos concurrentes*, donde la diferencia radica en disponer cada actividad en un ordenador diferente, en lugar de tratarse de hilos dentro del mismo proceso.

# Ejemplos de Sistemas Distribuidos (I)

- Sistemas de almacenamiento en la nube
  - Dropbox, Google Drive, OneDrive, etc.
- Sistemas de mensajería
  - Whatsapp, viber, line, etc.
- Sistemas de búsqueda
  - Google, Bing, yahoo, etc.
- Sistemas de directorio de recursos
  - DNS, LDAP, etc.
- Sistemas de autenticación de usuarios
  - Kerberos, sesame, etc.
- Portales de información y servicios públicos o privados
  - Poliformat.upv.es, agenciatributaria.es, valencia.es, etc.

# Ejemplos de Sistemas Distribuidos (II)

- Redes sociales
  - Facebook, twitter, etc
- Distribución de medios
  - Netflix, youtube, flickr, etc.
- Sistemas peer-to-peer (P2P)
  - BitTorrent, JXTA, Napster, Gnutella, eMule, etc.
- Sistemas Grid
  - Globus Toolkit,  Seti Project, etc.
- Juegos online, sistemas bancarios, sistemas de control, etc, etc, etc.

**Hoy en día resulta complicado encontrar sistemas que no sean distribuidos o que no tengan o utilicen a su vez alguna componente distribuida.**

# Ejemplos de sistemas NO distribuidos

- Ejemplos de sistemas no distribuidos
  - Un conjunto de ordenadores en red, donde un usuario se conecta con "ssh" a cada uno de ellos por separado, y sabe qué puede ejecutar en cada uno de ellos.
    - Sin embargo en esta red de ordenadores podrían coexistir sistemas distribuidos, como por ejemplo LDAP, acceso a DNS, sistemas de ficheros distribuidos, sistemas de impresión distribuida, etc.
  - Un conjunto de procesos dentro de un mismo ordenador.
    - Sin embargo, la mayoría de sistemas distribuidos se desarrollan en una primera etapa de prototipo dentro de un único ordenador. Pero su diseño y vocación radica en una posterior distribución física.
  - Un programa concurrente formado por varios hilos, dentro de un único proceso.

# Objetivo de los Sistemas Distribuidos

El principal objetivo de todo Sistema Distribuido es facilitar a los usuarios el acceso a recursos remotos, de forma sencilla y fiable.

- **Recurso** debe entenderse en sentido amplio
  - Ejemplos: impresoras, computadoras, dispositivos de almacenamiento, otros usuarios conectados, fotografías, vídeos, cuentas bancarias, etc.
    - Ejemplo: servicios de almacenamiento en la nube (Dropbox, Google Drive, OneDrive, etc.)
    - Ejemplo: sistemas de mensajería (whatsapp, viber, line, etc)
    - Ejemplos: Netflix, Google, Sistemas bancarios online, Facebook, Juegos online, etc.
- Ventajas:
  - Economía de recursos: los usuarios no necesitan tener localmente todos los recursos.
  - Compartición de recursos entre usuarios.

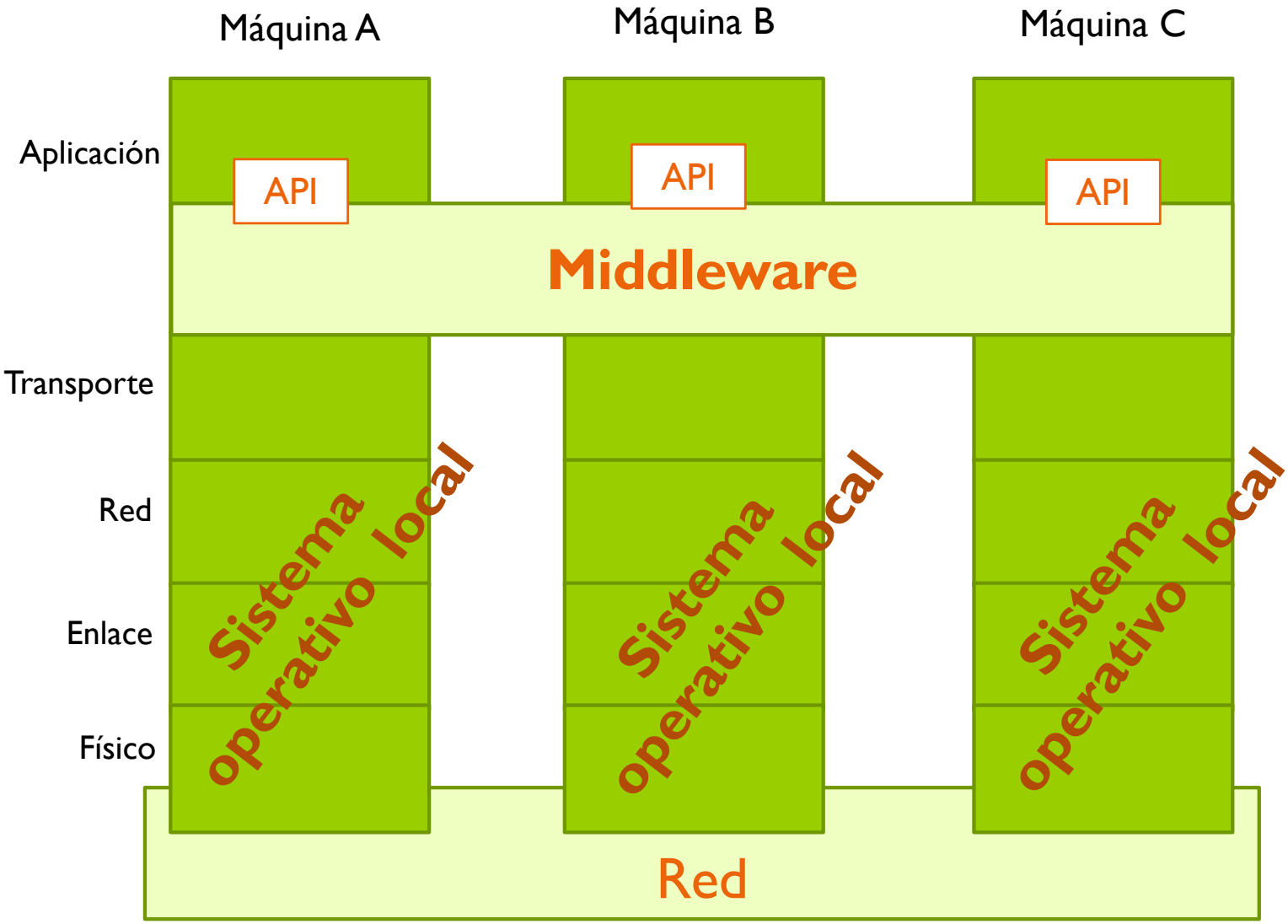
# Middleware

**Se define como una capa de software por encima del sistema operativo, pero bajo el programa de aplicación que proporciona una abstracción común de programación a lo largo de un sistema distribuido**

Bakken, D. 2001

- Desde el punto de vista del programador de aplicaciones, se trata de un simple **API** con el que acceder a servicios distribuidos.
- El uso del API desencadena un trabajo distribuido entre los diferentes nodos del sistema, que colaborarán para ofrecer el servicio distribuido
- El uso de servicios middleware permite el desarrollo de **aplicaciones distribuidas**, evitando al programador el desarrollo de parte de su complejidad: transparencia, disponibilidad, escalabilidad o seguridad.
- Asimismo, al desarrollar un sistema distribuido, será conveniente ofrecer los servicios en forma de middleware, idealmente mediante interfaces que empleen **estándares abiertos**.

# Middleware




Ayuda al desarrollo de aplicaciones distribuidas, simplificando al programador gran parte de su complejidad.

# Middleware

- Los sistemas distribuidos (SD) son complejos
  - SD puro = hardware (nodos + red) → variado, complejo, heterogéneo
  - **SD extendido** = SD puro + **Middleware** → mucho más fácil de programar
    - abstracciones (ej. canales, bus de eventos, objetos remotos, etc.)
    - gestión de réplicas, soporte para transparencia de fallos, etc.
    - portabilidad, interoperabilidad

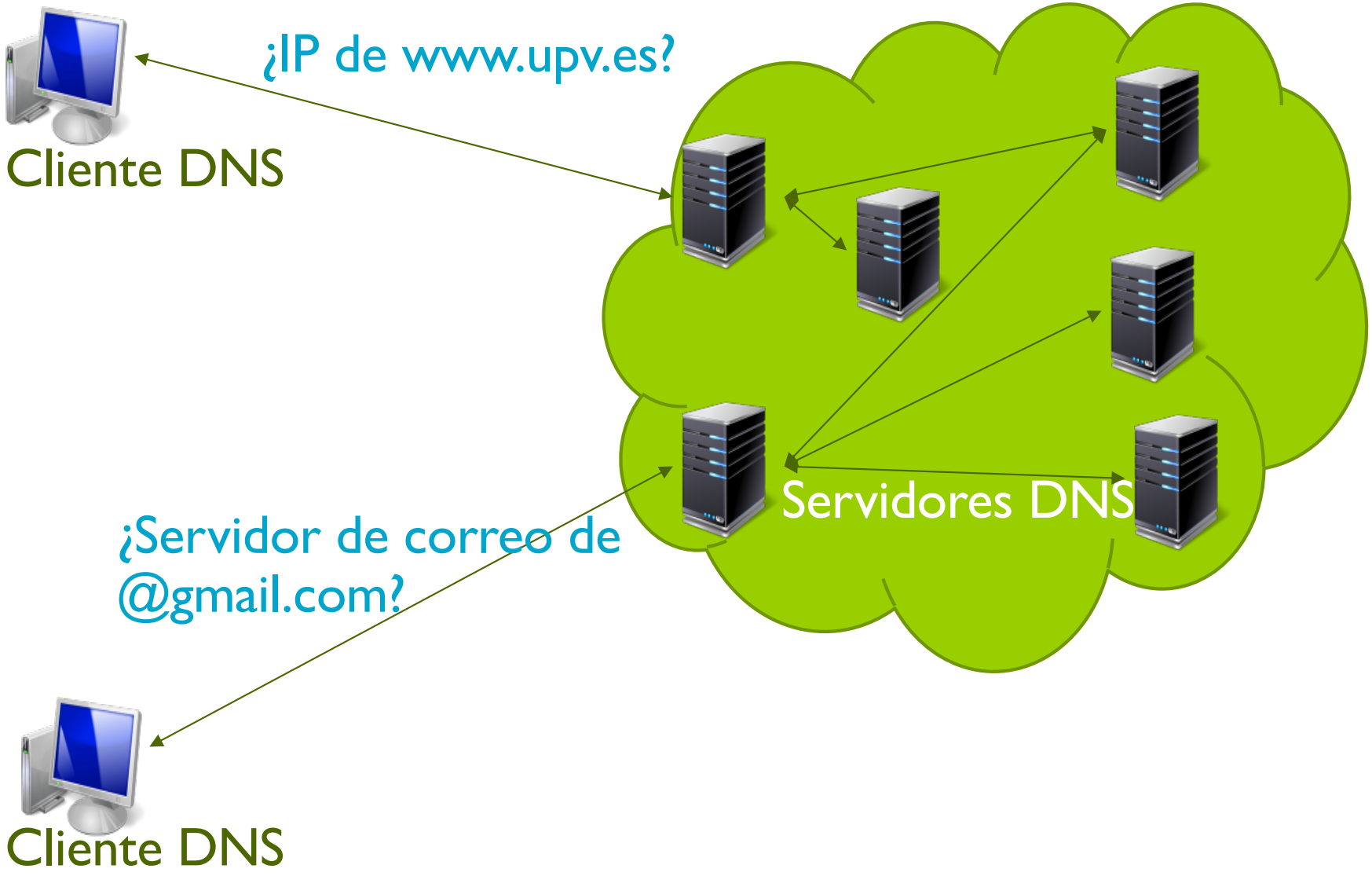
# Middleware

- Existen distintos tipos de Middleware: cada uno crea SD extendido distinto. Ejemplos:
  - Orientados a objetos (RMI = Remote Method Invocation)
    - Objeto Remoto = objeto que no reside en el mismo espacio de direcciones (ej. en otro nodo)
    - Permite programar invocaciones de objetos como si todos fuesen locales
  - Orientados a comunicaciones: elementos de comunicación intermedios (canales, temas, ...).
  - Orientados a eventos: emisores y oyentes de eventos, publicación/suscripción, ...

# Middleware

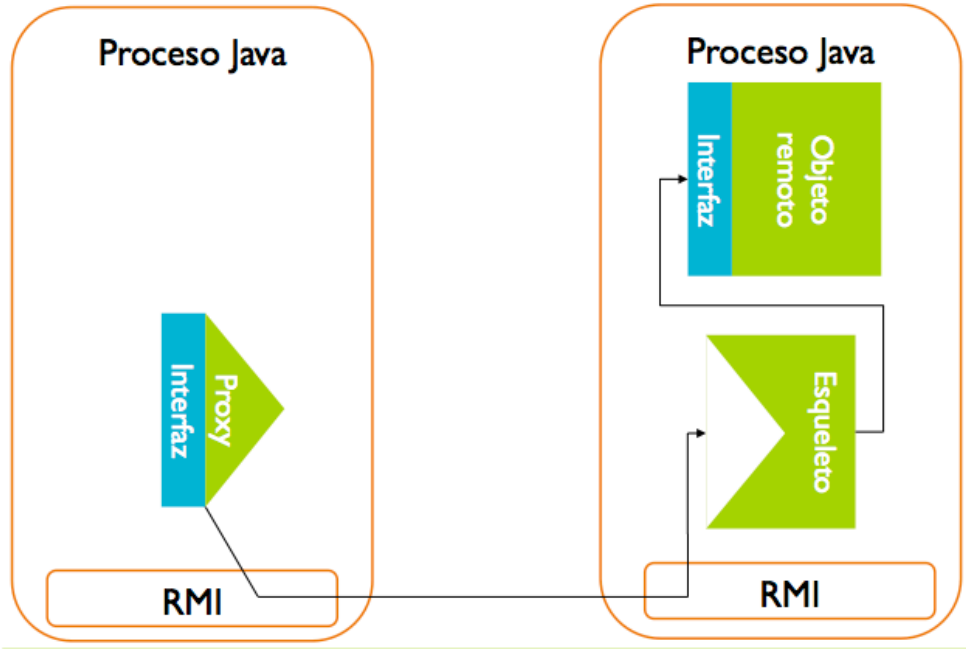
- Ejemplos de Middleware:
  - Java Remote Method Invocation (RMI)
  - Java Message Service (JMS)
  - Domain Name System (DNS)
  - Open Database Conectivity (ODBC)
  - Lightweight Directory Access Protocol (LDAP)

# Ejemplo de middleware: DNS



# Ejemplo de middleware: Java RMI

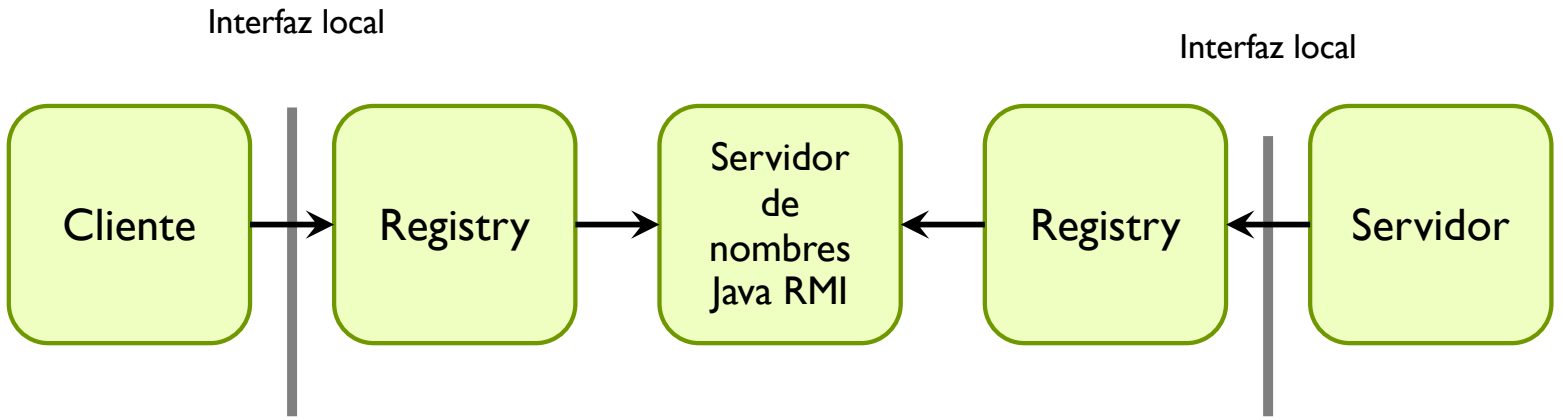
- **Java RMI** (*Remote Method Invocation*) es un **middleware** de comunicación orientado a **objetos**
  - Permite **invocar** métodos de objetos Java de otra JVM, y **pasar objetos** Java como argumentos cuando se invocan dichos métodos




- Para cada objeto remoto, RMI crea dinámicamente un objeto llamado **esqueleto**
- Para invocar un objeto remoto desde otro proceso se utiliza un **Proxy**
  - Permite localizar al esqueleto del objeto remoto

# Ejemplo de middleware: Java RMI

- El **servidor de nombres** permite registrar objetos remotos, para que puedan ser localizados.
  - Puede residir en cualquier nodo y es accedido desde el cliente o el servidor usando la interfaz local llamada **`Registry`**
- En las distribuciones Oracle de Java, el servidor de nombres se lanza usando la orden **`rmiregistry`**



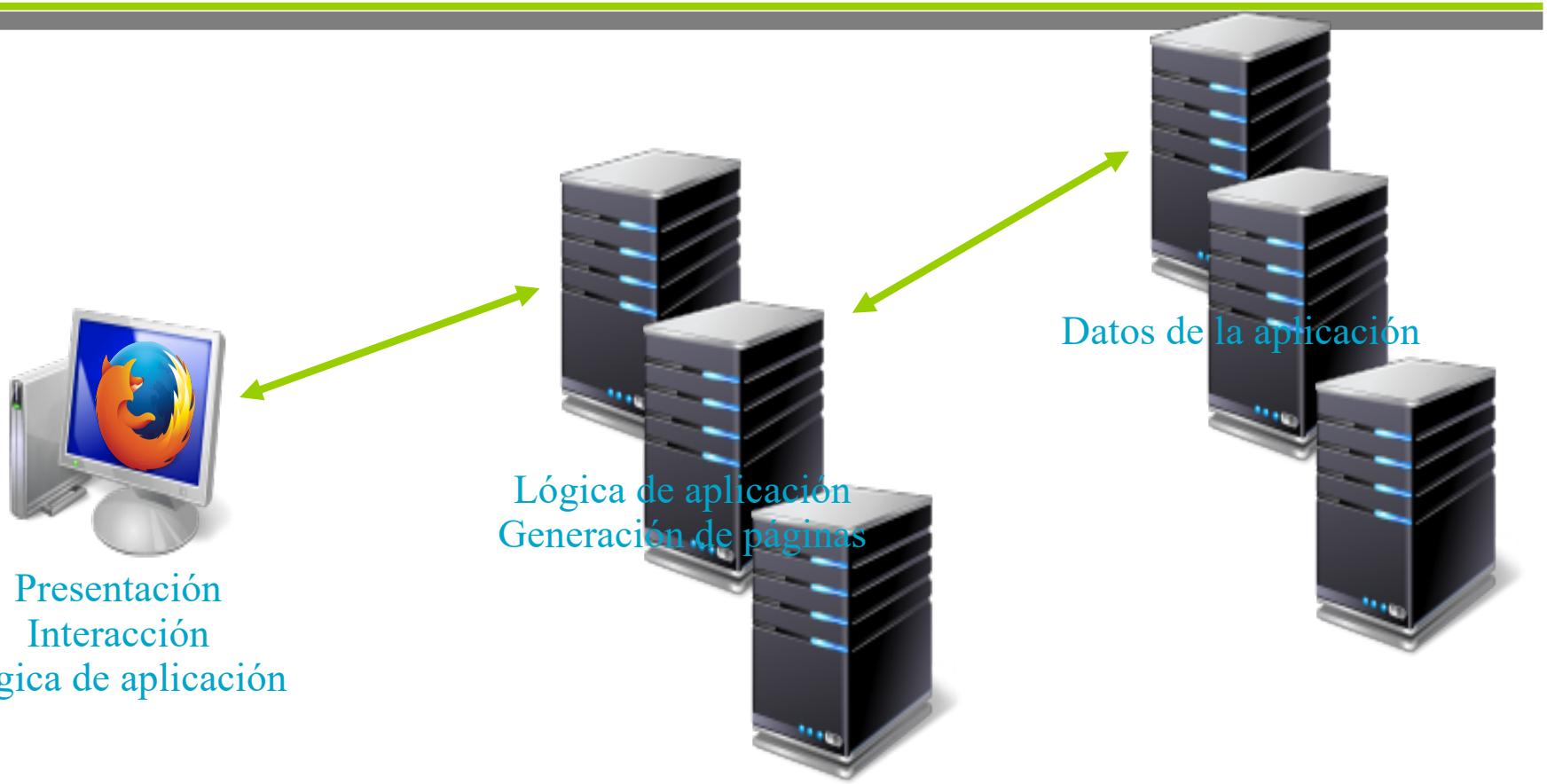
# Contenido

- Concepto de Sistema Distribuido
  - Definición de Sistema Distribuido
  - Ejemplos
  - Objetivo de los Sistemas Distribuidos
  - Middleware
- Características de los Sistemas Distribuidos
  - Transparencia
  - Disponibilidad
  - Escalabilidad
  - Seguridad

# Características de los Sistemas Distribuidos

- Las dificultades propias de la construcción de sistemas distribuidos, se convierten en retos a superar y finalmente en características intrínsecas que podemos encontrar en todo sistema distribuido, en mayor o menor grado:
  - **Transparencia:** se oculta el hecho en sí mismo de la distribución, las diferencias entre las diferentes máquinas y la complejidad de los mecanismos de comunicación.
  - **Disponibilidad**: los servicios que ofrecen deben estar siempre disponibles. Se trata de sistemas intrínsecamente tolerantes a fallos y donde las tareas de mantenimiento no deben interrumpir el servicio.
  - **Escalabilidad**: deben ser relativamente sencillos de ampliar, tanto en usuarios, como en recursos y en número de nodos.
  - **Seguridad**: los recursos y usuarios deben coexistir respetando las reglas y restricciones de acceso y políticas de seguridad que dependerán de cada sistema.

# Ejemplo de Sistema Distribuido: PoliformaT




Transparencia Disponibilidad Escalabilidad Seguridad

# Contenido

- Concepto de Sistema Distribuido
  - Definición de Sistema Distribuido
  - Ejemplos
  - Objetivo de los Sistemas Distribuidos
  - Middleware
- Características de los Sistemas Distribuidos
  - Transparencia
  - Disponibilidad
  - Escalabilidad
  - Seguridad

# Transparencia

- **Transparencia de distribución** (imagen de sistema único): ocultar al usuario el hecho que los procesos y los recursos están físicamente distribuidos sobre diferentes ordenadores.
  - Logrando transparencia, se ofrecen **servicios sencillos** al usuario
    - Se le oculta la complejidad de los algoritmos que se ejecutan, los fallos que puedan suceder, el número de ordenadores involucrados, su ubicación y todas aquellas características que sean irrelevantes para el usuario final.
  - Para lograr transparencia de distribución se deben ocultar diferentes aspectos específicos, llamados **Ejes de la transparencia de distribución**

| Ejes de la transparencia de distribución |
|---|
| Transparencia de ubicación |
| Transparencia de fallos |
| Transparencia de replicación |
| Otros ejes de transparencia (persistencia, concurrencia, migración, transacción, etc.) |

# Ejes de la Transparencia de Distribución

- **Transparencia de ubicación:** se oculta al usuario la ubicación de los recursos.
  - **Ejemplos**

    - Los usuarios de Dropbox, no necesitan saber en qué ordenador están las copias de sus archivos.
    - Los usuarios del servicio de mapas de Google no necesitan saber en qué ordenador se encuentran los mapas.
    - Los usuarios de páginas web raramente quieren saber dónde están los recursos donde acceden. Les basta con poder clicar en enlaces para obtener más información o recursos específicos.
  - **Mecanismos**
    - Para lograr transparencia de ubicación, los recursos suelen estar identificados con **nombres simbólicos únicos**.
      - Cuando los usuarios desean acceder a los recursos, utilizarán los nombres simbólicos.
      - El sistema traducirá "de forma transparente" al usuario el nombre simbólico a su verdadera ubicación.
    - **Servicios de nombres**, servicios de localización y búsqueda de recursos, servicios de directorio, etc.

# Ejes de la Transparencia de Distribución

- **Transparencia de fallos**: se oculta el hecho de que los componentes de un sistema distribuido fallen.
  - Cuantos más ordenadores forman parte de un sistema distribuido, más alta es la probabilidad de que falle alguno de ellos. El usuario no está interesado en saber cuándo falla algún ordenador.
  - **Ejemplos**
    - Se estima (Wikipedia, 2017) que el servicio de búsqueda de Google cuenta con 2.000.000 ordenadores. Suponiendo una tasa de fallo conservadora, de un fallo en cada ordenador cada 5 años, implica que fallan cerca de 1.000 ordenadores cada día. Cerca de 1 fallo por minuto !!
    - El usuario no percibe estos fallos.
  - **Mecanismos**:
    - **Detectores de fallos** → los nodos están siendo monitorizados continuamente
    - **Replicación** → todos los recursos están replicados en más de un nodo. Si se detecta el fallo de un nodo, otro nodo será encargado de continuar dando el mismo servicio sobre la copia del recurso.
    - **Algoritmos tolerantes a fallos** → todos los algoritmos que se ejecutan en el sistema deben reaccionar adecuadamente al fallo de un nodo, ofreciendo el mismo servicio antes y después del fallo.

# Ejes de la Transparencia de Distribución

- **Transparencia de replicación:** se oculta el hecho que los recursos están replicados en más de un nodo.
  - Los sistemas se replican para lograr sistemas disponibles (tolerancia a fallos) y para aumentar la escalabilidad (más usuarios accediendo al sistema simultáneamente, más recursos, más nodos).

  - **Ejemplos**
    - Al subir una fotografía a un perfil de Facebook, esa fotografía se almacena en varios ordenadores. De esta forma, en caso de fallos, otro ordenador la puede servir. Adicionalmente, varios ordenadores diferentes sirven copias de la misma fotografía a diferentes usuarios que visiten dicho perfil.
    - El usuario no está interesado en saber cuántos ordenadores tienen copia de la fotografía, ni qué réplica es la que le ofrece el servicio, ni siquiera está interesado en saber que existe tal replicación.
  - **Mecanismos:**
    - Mapeo de los nombres simbólicos de recursos a la ubicación de diferentes réplicas.
    - Algoritmos de balanceo de carga, para elegir qué réplica sirve cada petición sobre cada recurso.
    - Algoritmos de replicación que ofrezcan **consistencia entre las réplicas**: (este concepto se verá en el apartado de **disponibilidad**)

# Ejes de la Transparencia de Distribución

- **Otros ejes de la transparencia de distribución**
  - Existen múltiples tipos de transparencia que podemos encontrar en la literatura. Todos ellos con la vocación de colaborar en el objetivo común de lograr transparencia de distribución y proporcionar servicios sencillos de utilizar.
  - Están presentes en mayor o menor medida en determinados sistemas y tendrán importancia en función del tipo de sistema distribuido que estemos estudiando.
  - **Transparencia de persistencia**
    - Se oculta el hecho de que los recursos están almacenados en disco.
  - **Transparencia de concurrencia**
    - Se oculta el hecho de que el sistema está siendo utilizado por múltiples usuarios al mismo tiempo.
  - **Transparencia de migración**
    - Se oculta el hecho de que los recursos pueden moverse.
  - **Transparencia de acceso, de transacción, de reubicación, etc.**

# Consideraciones sobre la transparencia

- Ofrecer transparencia tiene un **coste elevado**
  - Coste en mayor número de ordenadores, en ordenadores y redes de mayores prestaciones, en mayor coste de desarrollo y mantenimiento de los sistemas, y en mayor coste algorítmico (espacial, temporal, número de mensajes).
  - A mayor transparencia → mayor coste y mayor calidad observada por usuarios
- A veces no interesa transparencia total en alguno de los ejes. La conveniencia depende de cada sistema en particular.
  - Por ejemplo: dominios ".es" (conviene saber que son páginas de España)
- A veces la transparencia total es imposible de lograr o tendría demasiado coste para determinado sistema.
  - Por ejemplo: "poliformaT" se ubica en el campus de la universidad. Un fallo general que afecte a las redes o suministro de toda la universidad, provocaría una violación a la transparencia de fallos → el coste de ofrecer mayor transparencia de fallos puede ser inabordable de momento.

# Contenido

- Concepto de Sistema Distribuido
  - Definición de Sistema Distribuido
  - Ejemplos
  - Objetivo de los Sistemas Distribuidos
  - Middleware
- Características de los Sistemas Distribuidos
  - Transparencia
  - Disponibilidad
  - Escalabilidad
  - Seguridad

# Disponibilidad

- Podemos definir **disponibilidad** como la probabilidad de que un determinado sistema ofrezca sus servicios a los usuarios.
  - Se habla de sistemas altamente disponibles, cuando la probabilidad es mayor que el 99.999%
  - En sistemas críticos, se tratan probabilidades mayores y se habla de sistemas ultra-altamente disponibles.
- Hay 3 factores que afectan a la disponibilidad:
  - Fallos
  - Tareas de mantenimiento
  - Ataques maliciosos

En este apartado nos centramos en los conceptos generales relacionados con **tolerancia a fallos**

# Factores que afectan a la Disponibilidad

- 1) Fallos
  - Fallan los nodos y fallan las redes.
  - Se deben diseñar sistemas que continuen ofreciendo servicio en presencia de fallos → **sistemas tolerantes a fallos**.
- 2) Tareas de mantenimiento
  - Todo sistema requiere mantenimiento.
    - Tareas tales como sustituir discos, alterar la configuración de los sistemas, realizar copias de seguridad, ampliar sistemas, o incluso cambiar el sistema por completo.
  - Se deben diseñar sistemas que permitan realizar el mantenimiento al mismo tiempo que los usuarios acceden a los servicios.

# Factores que afectan a la Disponibilidad

- 3) Ataques maliciosos
  - Todo sistema es objeto potencial de ataques maliciosos.
  - Desde intrusos que alteran, eliminan, interceptan o suplantan recursos hasta a ataques distribuidos de denegación de servicio.
  - Se debe diseñar sistemas que sean resistentes a los ataques de forma que no se interrumpa el servicio ofrecido → **seguridad informática**

# Disponibilidad

En este apartado nos centramos en los conceptos generales relacionados con **tolerancia a fallos**

- **Todo sistema distribuido debe ser tolerante a fallos**. El sistema debe seguir funcionando y proporcionando servicio en presencia de fallos.
- El mecanismo básico para lograr tolerancia a fallos en sistemas distribuidos es la **replicación**
  - Un servicio se configura como un conjunto de nodos, llamados **réplicas**, de tal forma que el fallo de una réplica no impida al servicio seguir funcionando.

# Disponibilidad

- La replicación introduce el problema de la **consistencia**: el grado de similitud/diferencia entre las diferentes réplicas.
  - Sistemas con mucha consistencia, o **consistencia fuerte**:
    - Idealmente todas las réplicas son iguales entre sí en todo momento.
    - Este objetivo es imposible de lograr, pero se pueden lograr sistemas con consistencia bastante fuerte, donde los usuarios del servicio obtienen la misma respuesta independientemente de la réplica que les atienda.
  - Sistemas con poca consistencia, o **consistencia débil**:
    - Las réplicas pueden divergir, de forma que cada una de ellas puede dar respuestas diferentes en cierto instante.

NOTA: *El estudio de la consistencia, se verá en materias de cursos próximos. En este curso hablaremos de* ***consistencia fuerte (réplicas casi idénticas)*** *o* ***consistencia débil (réplicas potencialmente diferentes)***.

# Disponibilidad: tipos de fallos

- **Tipos de fallos:**
  - Los fallos pueden ser **simples**, afectando a un único nodo o a un único canal de comunicación, o **compuestos**, afectando a varios nodos y canales simultaneamente.
  - Los fallos pueden ser **detectables** o **indetectables**.
    - Son detectables si otro nodo es capaz de observar el fallo.
    - Son indetectables en caso contrario.

# Disponibilidad: tipos de fallos

- **Fallos simples detectables**
  - Un fallo es detectable si otro nodo es capaz de observar el fallo.
    - **Fallo de parada**
      - El nodo falla deteniéndose.
      - Otro nodo lo puede detectar mediante monitorización continua, "pings" periódicos.
    - **Fallo de temporización**
      - El nodo falla, tardando demasiado en responder.
      - Otro nodo lo puede detectar con temporizadores asociados a cada petición, también mediante "pings" periódicos.
    - **Fallo de respuesta detectable**
      - El nodo falla, proporcionando una respuesta equivocada, detectable como tal.
      - Otro nodo lo puede detectar aceptando únicamente rangos válidos de respuesta.

# Disponibilidad: tipos de fallos

- **Fallos simples detectables**
  - En general todos los fallos simples detectables se pueden tratar de forma parecida.
    - Se emplea **replicación**.
    - Se detecta el fallo de una réplica, se expulsa a la réplica que ha fallado y el resto de réplicas continuan ofreciendo el servicio.
    - Nótese que se expulsa a la réplica aunque ésta siga funcionando (quizás funciona, pero tardó mucho en contestar, o lo hizo de forma errónea). Para evitar interferencias de esta réplica, se ignorarán los mensajes que esta réplica pueda enviar en el futuro.

# Disponibilidad: tipos de fallos

- **Fallos simples indetectables**
  - También llamados **fallos bizantinos**
  - Un nodo falla exhibiendo un comportamiento arbitrario o proporcionando una respuesta que no puede detectarse como fallo.
  - Los fallos bizantinos pueden deberse a diferentes causas: errores en el software, errores en el hardware y ataques maliciosos.
  - Ejemplo:
    - Tenemos un sistema que mide la temperatura ambiente. Si la temperatura supera cierta medida (>25 grados), pone en marcha la refrigeración. El rango de temperaturas admisible es 10-40 grados.
    - Un nodo mide la temperatura, ofrece como servicio consultar temperatura. Otro nodo usa el servicio y decide si poner en marcha la refrigeración. Si el nodo no contesta, o contesta tarde, o contesta fuera de rango → fallo detectable.
    - ¿Qué ocurre si la temperatura ambiente son 15 grados, pero el nodo contesta diciendo 30 grados? → fallo indetectable.

# Disponibilidad: tipos de fallos

- **Fallos simples indetectables**
  - Los fallos bizantinos se tratan de diferente forma a los fallos detectables.
    - Se puede emplear replicación, pero de forma diferente al caso de fallos detectables:
      - Cada petición se envía a todas las réplicas y todas contestan. Se elige la respuesta mayoritaria. A estos algoritmos también se les llama **algoritmos de quorum**. Al mismo tiempo, se genera una alarma sobre la falta de unanimidad → esquema usado en sistemas ultra-altamente disponibles.
    - Hay otros sistemas que emplean mecanismos basados en criptografía para tolerar fallos bizantinos: → ejemplos en cripto-monedas, y en criptografía de umbral.

# Disponibilidad: tipos de fallos

- **Fallos compuestos**
  - Fallan simultaneamente varios nodos o varios canales de comunicación.
  - En la mayoría de casos, se tratan de igual forma a la aparición de varios fallos simples de forma consecutiva.
    - Varios fallos de parada → se trata cada uno de forma independiente y consecutiva.
    - Varios fallos bizantinos → se tratan suponiendo que habrá mayoría de nodos no afectados por fallos y la mayoría correcta podrá seguir ofreciendo el servicio.

# Disponibilidad: tipos de fallos

- **Fallos compuestos**
  - Un tipo de fallo compuesto merece atención especial: **las particiones**.
    - Se producen varios fallos en nodos o canales de comunicación que dejan al sistema dividido en 2 o más subgrupos.
    - Problema relevante en los grandes sistemas actuales (cloud computing)
    - **Teorema CAP**: (Consistency, Availability, Partitions) → Es imposible lograr un sistema que ofrezca al mismo tiempo Consistencia fuerte, Disponibilidad elevada y puedan ocurrir Particiones.
      - Existen sistemas CA, CP y AP
        - **CP:** Sistemas con consistencia fuerte y particiones (sin disponibilidad elevada)
        - **CA:** Sistemas con consistencia fuerte y disponibilidad elevada (sin particiones)
        - **AP:** Sistemas con disponibilidad elevada y particiones (sin consistencia fuerte)

# Disponibilidad

- **Mecanismos para lograr tolerancia a fallos** (suponiendo sólo fallos simples detectables, que no causen particiones):
  - Detectores de fallos
  - Servicio de pertenencia a grupo
  - Replicación

# Disponibilidad

## Detectores de fallos

- La detección de fallos suele integrarse en un módulo de detección de fallos incorporada a cada nodo. Este módulo se encarga de monitorizar a otro nodo o a varios nodos y emitir "sospechas de fallo".
- Más que detectar un fallo, se suele indicar que el nodo **sospecha** del fallo de otro nodo
  - Nótese que puede haber un canal con fallo entre ambos nodos y por tanto ambos se sospecharán mutuamente.
- En caso de sospechar un fallo, el módulo de detección de fallos lo notifica al servicio de pertenencia a grupo.

# Disponibilidad

## Servicio de pertenencia a grupo

- Servicio encargado de establecer un acuerdo entre los nodos vivos, sobre qué nodos han fallado.
- Ante la sospecha de un fallo, o varias sospechas, este servicio inicia una **fase de acuerdo**, para determinar qué nodo o nodos han fallado (generalmente desde el punto de vista de la mayoría).
- Si se acuerda el fallo de un nodo, el servicio de pertencia lo expulsará y notificará a todos los nodos que permanecen "vivos" informándoles del fallo.
  - Los nodos que reciben la notificación de fallo, ignorarán los posibles mensajes que provengan de ese nodo y se reconfigurarán para seguir funcionando sin ese nodo.

# Disponibilidad

- **Servicio de pertenencia a grupo**
  - Nótese que es posible que ante cierta sospecha, se determine que el nodo que falla es el que emitió la sospecha, pues los demás nodos acuerdan que la sospecha es infundada.
    - Quizás el nodo sospechó porque él mismo está funcionando demasiado despacio.
  - Esta forma de trabajo la podemos enunciar diciendo: **todos los fallos simples detectables se convierten a fallos de parada**
    - Ocurra el fallo que ocurra, si lo detectamos y se acuerda el fallo, se expulsa a algún nodo y los demás nodos siguen funcionando, ignorando al nodo que fue expulsado.

# Disponibilidad

En sistemas distribuidos, cuando **decimos que un nodo ha fallado**, estamos diciendo que dicho nodo **ha sido expulsado del sistema** por parte de cierto servicio de pertenencia. Nótese que es posible que el nodo siga funcionando.

Todos los nodos que permanezcan vivos, recibirán la notificación del fallo, antes o después.

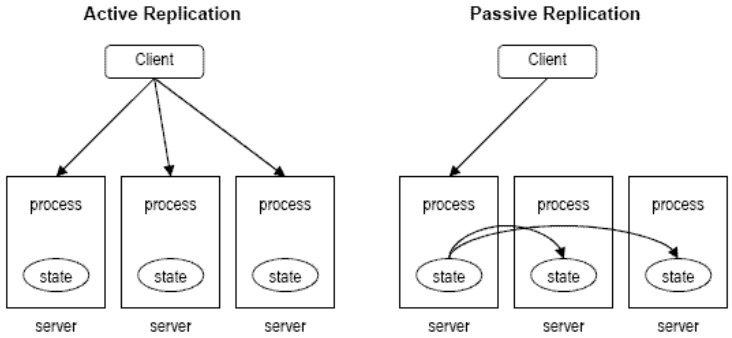
Cuando un nodo reciba la notificación del fallo de otro nodo, se **reconfigurará** para seguir trabajando sin el nodo expulsado:

- Todos los algoritmos distribuidos en los que intervenga el nodo se reconfigurarán
- Todos los servicios presentes en el nodo, se reconfigurarán

# Disponibilidad

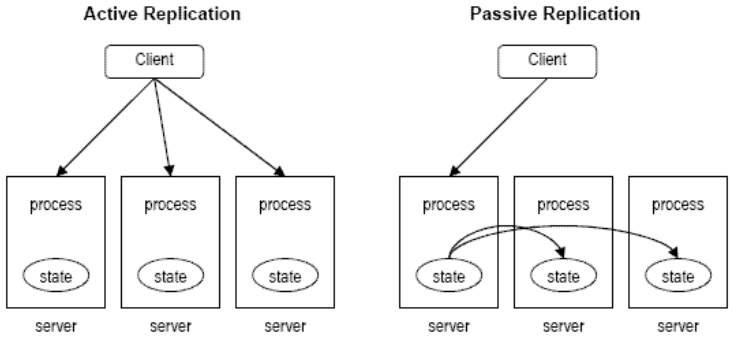
- **Replicación**
  - Cada servicio se configura con más de una réplica, de forma que ante fallos, las réplicas que permanecen "vivas", se reconfiguran y continuan ofrenciendo servicio.
  - Los clientes del servicio acceden al servicio con **transparencia de replicación**, observando como única diferencia, una mayor disponibilidad.

# Disponibilidad




- **Esquemas de Replicación:**
  - **Replicación pasiva**
    - Entre el grupo de réplicas, una única réplica será la réplica **primaria**. El resto de réplicas son las **secundarias**.
    - El nombre de "replicación pasiva" viene derivado del rol de las réplicas secundarias, que no procesan peticiones. Son pasivas.
      - La réplica primaria es la única réplica "activa". La única que trabaja.
      - También se llama a este esquema **replicación primario-secundario**.
  - **Replicación activa**
    - Todas las réplicas son iguales. Todas reciben las peticiones, todas procesan las peticiones y todas contestan al cliente.
  - **Replicación semi-activa, semi-pasiva**: esquemas híbridos, mezclando ambas aproximaciones.

# Disponibilidad




- **Esquemas de Replicación:**
  - **Replicación pasiva**
    - Entre el grupo de réplicas, una única réplica será la réplica **primaria**. El resto de réplicas son las **secundarias**.
    - El nombre de "replicación pasiva" viene derivado del rol de las réplicas secundarias, que no procesan peticiones. Son pasivas.
      - La réplica primaria es la única réplica "activa". La única que trabaja.
      - También se llama a este esquema **replicación primario-secundario**.
  - **Replicación activa**
    - Todas las réplicas son iguales. Todas reciben las peticiones, todas procesan las peticiones y todas contestan al cliente.
  - **Replicación semi-activa, semi-pasiva**: esquemas híbridos, mezclando ambas aproximaciones.

# Disponibilidad

- **Esquemas de Replicación: Replicación pasiva (I)**
  - Entre el grupo de réplicas, una única réplica será la réplica **primaria**. El resto de réplicas son las **secundarias**.
  - La réplica primaria recibe todas las peticiones por parte de los clientes del servicio, las procesa y responde a los clientes.
  - Ante cada petición que implique un cambio de estado, la réplica primaria difundirá un mensaje de actualización de estado (**checkpoint**) a las secundarias.
    - Si el mensaje de *checkpoint* se envía a las secundarias, esperando la correspondiente confirmación, antes de reponder al cliente, se tendrá un **sistema replicado con consistencia fuerte**.
    - Si el mensaje de *checkpoint* se envía a las secundarias más tarde, tendremos un **sistema replicado con consistencia débil**.

# Disponibilidad

- **Esquemas de Replicación: Replicación pasiva (II)**
  - Reconfiguración en caso de fallos:
    - En caso de fallo de una réplica secundaria, el trabajo de reconfiguración es pequeño → la réplica primaria enviará un mensaje menos de checkpoint.
    - En caso de fallo de la réplica primaria, el trabajo de reconfiguración puede ser elevado:
      - Elegir a una réplica secundaria para que asuma el rol de réplica primaria. Todas las réplicas deberán estar de acuerdo.
      - Asegurarse que la réplica elegida tiene el estado más reciente posible.
      - Asegurarse que los clientes pueden encontrar adecuadamente al nuevo primario cuando detecten que el antiguo primario no responde.
      - Durante la reconfiguración, es muy posible que el servicio no esté disponible.

# Disponibilidad

- **Esquemas de Replicación: Replicación pasiva (III)**
  - Ventajas sobre replicación activa:
    - Sólo una réplica trabaja: más eficiente durante el tiempo sin fallos.
    - Se pueden replicar servicios cuya implementación no sea determinista → la mayoría.
    - Si para procesar la petición del cliente fuese necesario acceder a datos externos a las réplicas, sólo lo haría la réplica primaria, por lo que no es necesario utilizar exclusión mutua distribuida.

# Disponibilidad

- **Esquemas de Replicación: Replicación activa (I)**
  - El nombre de "replicación activa" viene derivado del rol de todas las réplicas. Todas son activas, pues todas procesan peticiones.
  - También se llama a este esquema **replicación mediante máquina de estados.**
    - Este nombre viene derivado del hecho que suponemos que todas las réplicas se comportan de forma **determinista**
      - Modelo de ejecución muy restrictivo y que implica en la práctica que las réplicas no tendrán concurrencia interna.
  - Implica el uso de "**algoritmos de difusión**" que proporcionen la misma secuencia de mensajes a todas las réplicas.
    - Estos algoritmos tienen coste elevado.

# Disponibilidad

- **Esquemas de Replicación: Replicación activa (II)**
  - Reconfiguración en caso de fallos:
    - En caso de fallo de una réplica no es necesario mucho trabajo, tan sólo eliminar las referencias a tal réplica de los clientes que traten de acceder al servicio.
    - Todas las réplicas tienen el mismo estado, por tanto no es necesario trabajo para mantener la consistencia.
  - Ventajas sobre replicación pasiva:
    - Reconfiguración sencilla en caso de fallos.
    - No es necesario implementar dos tipos de réplicas, todas ejecutan el mismo software.

# Disponibilidad

## Disponibilidad en sistemas a gran escala (I)

- En sistemas a gran escala, las **particiones** ocurren.
- Por el **teorema CAP** sabemos que se debe sacrificar disponibilidad o consistencia.
- Muchos sistemas de hoy en día a gran escala se diseñan para proporcionar alta disponibilidad reduciendo la consistencia.
- La disponibilidad resulta imprescindible para la mayoría de sistemas
  - Ej: sistemas comerciales que pierden negocio si no están disponibles.
- Las diferentes particiones seguirán ofreciendo servicio, y por tanto irán divergiendo.
  - Se sacrifica habitualmente la consistencia.

# Disponibilidad

## Disponibilidad en sistemas a gran escala (II)

- Resulta especialmente relevante la **consistencia eventual**
  - Antes o después, cuando la partición desaparezca, las diferentes réplicas convergerán.
  - Se deberán ejecutar **algoritmos de convergencia** de estado en las réplicas al desaparecer la partición.

- Es vital un estudio pormenorizado de las diferentes operaciones que modifican el estado de las réplicas para lograr que puedan converger al desaparecer la partición.

# Contenido

- Concepto de Sistema Distribuido
  - Definición de Sistema Distribuido
  - Ejemplos
  - Objetivo de los Sistemas Distribuidos
  - Middleware
- Características de los Sistemas Distribuidos
  - Transparencia
  - Disponibilidad
  - Escalabilidad
  - Seguridad

# Escalabilidad

- Los sistemas distribuidos deben diseñarse para que sigan proporcionando servicio conforme crezcan.
- Decimos que un sistema es **escalable** si el servicio que ofrece no sufre alteraciones de rendimiento y disponibilidad desde el punto de vista del usuario al aumentar:
  - El número de usuarios
  - El número de recursos
  - El número de nodos
  - El número de peticiones de servicio simultáneas


# Escalabilidad

- Cada sistema puede tener limitaciones plausibles a la escalabilidad, que dependen de cada sistema.
  - Ejemplo
    - Al diseñar un sistema como "poliformaT", no es necesario hacerlo pensando que lo utilizará todo el mundo. Su escalabilidad queda ceñida al ámbito de la universidad.
    - Sin embargo, deberá ser escalable dentro de sus limitaciones para poder servir un creciente número de peticiones simultáneas, creciente cantidad de recursos, creciente cantidad de nodos, creciente número de usuarios.
- A los sistemas con objetivos de crecimiento a escala global se les llama **sistemas altamente escalables**.
  - Ejemplos: Google, Facebook, Netflix, Whatsapp, DNS, correo electrónico, etc.

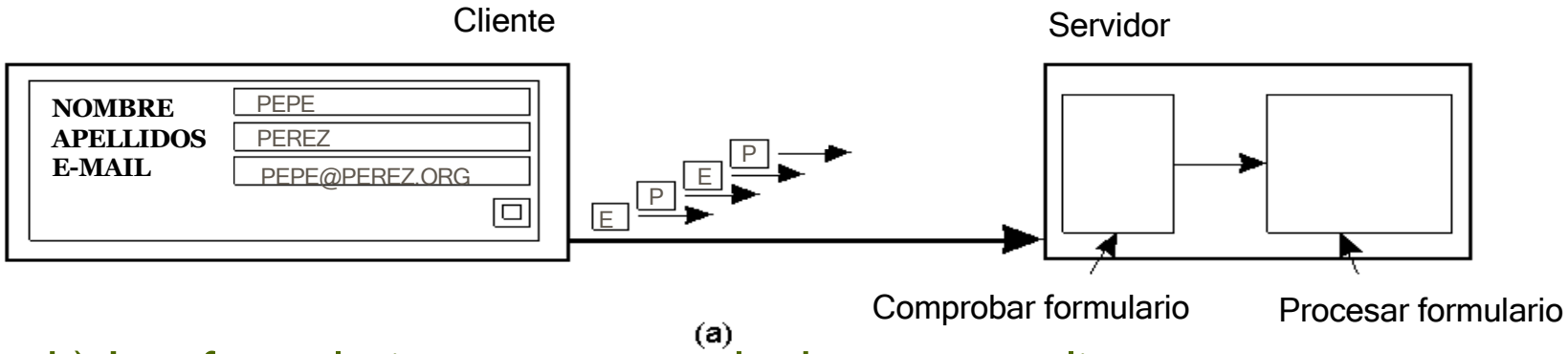
# Escalabilidad

- En general, la escalabilidad se ve amenazada cuando se adoptan estrategias centralizadas para gestionar los servicios, datos o algoritmos en un sistema distribuido.
- Las **técnicas** más importantes para aumentar la escalabilidad implican aumentar la distribución eliminando centralización.
  - **Distribuir la carga:** distribuir el procesamiento que realiza el servicio a diferentes nodos (incluso a los clientes)
  - **Distribuir los datos:** distribuir los recursos en diferentes nodos, de forma que cada nodo sirva una parte de los recursos del sistema.
    - También se conoce a esta técnica con el nombre de **particionado de los datos**
  - **Replicación:** replicar los recursos para permitir que cada réplica atienda parte del total de peticiones sobre el mismo recurso.
    - Se emplean balanceadores de carga para repartir la carga entre las diferentes réplicas.
  - **Caching:** caso particular de la replicación, donde disponemos de una copia del recurso en el propio cliente.

# Escalabilidad

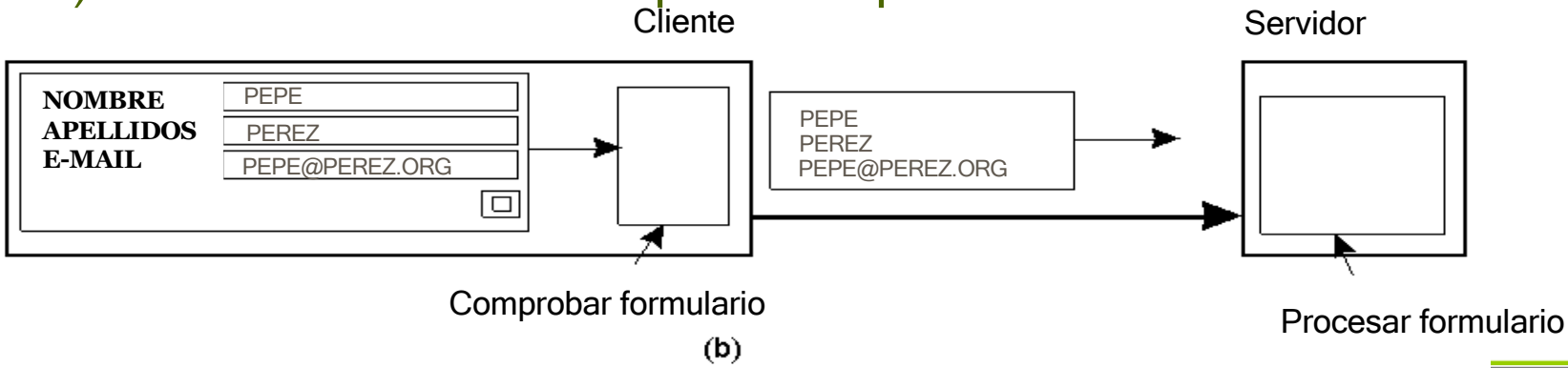
- Ejemplo de mejora de escalabilidad por distribución de la carga → mover parte de los cálculos que realiza un sistema Web a los clientes.

a) Los formularios son comprobados por un servidor




(a)

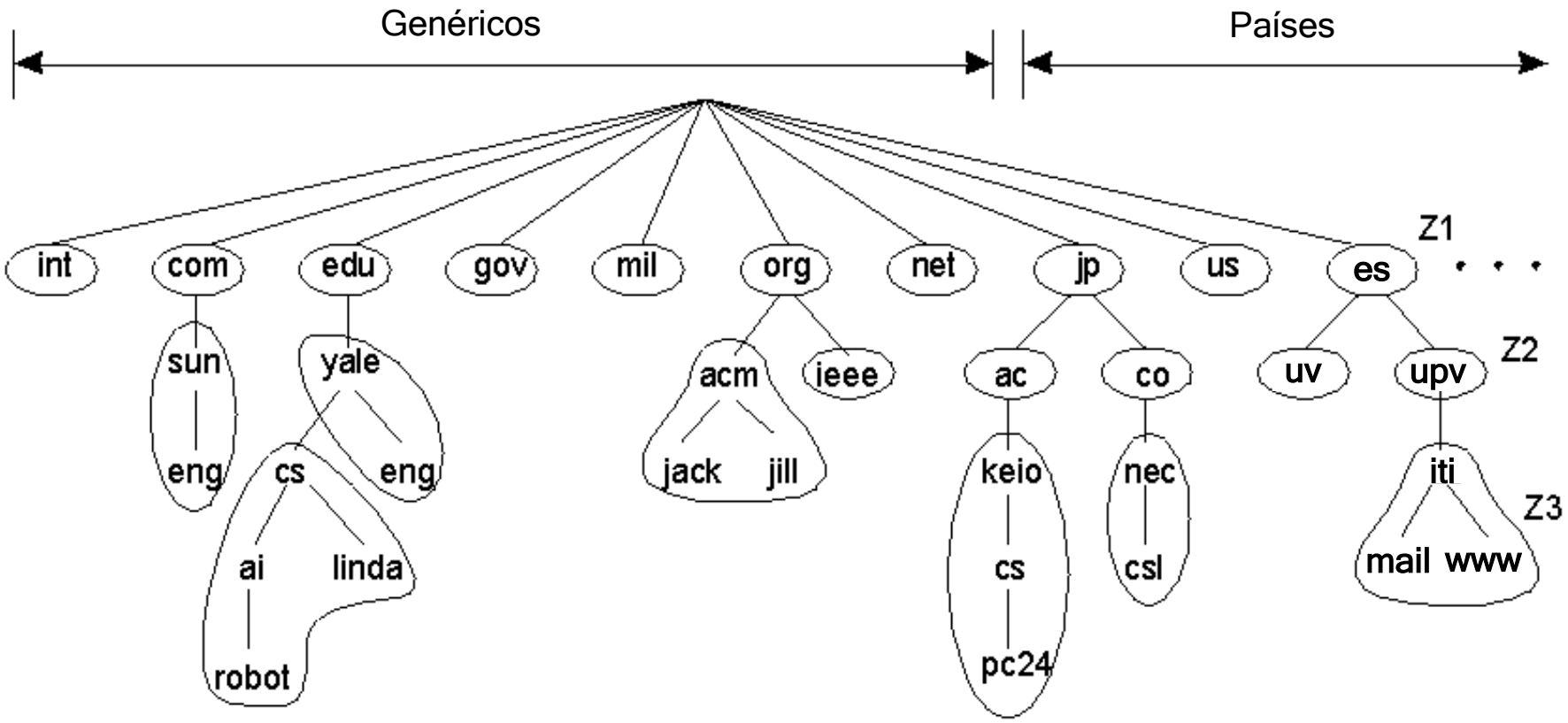
b) Los formularios son comprobados por un cliente




(b)

# Escalabilidad

- Ejemplo de mejora de escalabilidad por distribución de los datos → División de los DNS en zonas



# Escalabilidad

- **Replicación (I)**
  - La replicación resulta esencial en sistemas distribuidos para **aumentar la disponibilidad**.
  - También resulta esencial para **aumentar la escalabilidad**, sobre todo en sistemas altamente escalables.
  - Resulta muy eficaz en sistemas donde la mayoría de peticiones son de sólo lectura:
    - Ejemplos: Google, Facebook, Netflix, DNS, sistemas peer-to-peer de compartición de archivos, etc.

# Escalabilidad

- **Replicación (II)**
  - En **sistemas altamente escalables**, donde hay operaciones de modificación de los datos, se suele sacrificar la consistencia
    - Debido a que se pretende alta disponibilidad y pueden existir particiones.
    - Ejemplo: los comentarios que alguien introduce a una foto de Facebook, pueden tardar tiempo en estar visibles al resto de usuarios que visiten ese perfil.
      - Esto es debido a que diferentes usuarios acceden a réplicas diferentes y no todas se actualizan al mismo tiempo.

# Escalabilidad

- **Replicación (III)**
  - En sistemas donde hay mayoría de operaciones de escritura y se pretende **consistencia fuerte**, la replicación no escala bien
  - Se deben mantener mutuamente consistentes las réplicas y resulta necesario emplear algoritmos que bloquean el acceso al recurso replicado, mientras se realizan los cambios.
    - En sistemas que demanden consistencia fuerte, se suele optar por un minucioso **particionado de los datos**
      - Ejemplo: PayPal → las cuentas se mantienen en nodos diferentes. La replicación se emplea para tolerar fallos y no tanto para aumentar la escalabilidad.

# Escalabilidad

- **Caching**
  - Consiste en recordar las últimas versiones de la información accedida en cada componente de una aplicación distribuida.
  - Para accesos repetidos sobre un mismo elemento, se obtiene el valor de la copia guardada localmente, sin necesidad de acceder al servicio remoto.
  - Es un caso particular de la replicación, donde **el mismo cliente mantiene una réplica**.
    - Esta réplica tendrá **consistencia débil** respecto al servicio y por tanto será adecuado para aquellos servicios que no requieran consistencia fuerte y para recursos de sólo lectura.
  - Ejemplos:
    - Caches de Web.
    - Caches de ficheros.
    - Caches de fotografías de Facebook.

# Contenido

- Concepto de Sistema Distribuido
  - Definición de Sistema Distribuido
  - Ejemplos
  - Objetivo de los Sistemas Distribuidos
  - Middleware
- Características de los Sistemas Distribuidos
  - Transparencia
  - Disponibilidad
  - Escalabilidad
  - Seguridad

# Seguridad

La mayoría de conceptos sobre seguridad son objeto de estudio en materias específicas de seguridad informática.

- **Todo sistema distribuido debe ofrecer un servicio disponible y correcto a los usuarios legítimos del sistema y sólo a ellos.**
- Para ello, debe garantizar:
  - **Autenticación**: los usuarios y las peticiones que éstos realizan deben identificarse adecuadamente.
  - **Integridad**: el sistema debe mantener los datos sin modificaciones maliciosas o no autorizadas.
  - **Confidencialidad**: sólo los usuarios autenticados y autorizados pueden acceder a determinados recursos.
  - **Disponibilidad**: el servicio no debe sufrir interrupciones en todo o parte.

# Resultados de aprendizaje de la Unidad Didáctica

- Al finalizar esta unidad, el alumno deberá ser capaz de:
  - Identificar las ventajas y problemas que comporta el desarrollo de aplicaciones y sistemas distribuidos.
  - Caracterizar los sistemas distribuidos, en comparación a sistemas centralizados, paralelos, sistemas en red y sistemas concurrentes no distribuidos.
  - Comprender la problemática de la transparencia de distribución
  - Identificar los diferentes mecanismos necesarios para obtener disponibilidad en un sistema distribuido
  - Distinguir las diferentes mecanismos para lograr escalabilidad de un sistema distribuido.
  - Conocer los conceptos básicos de seguridad informática, especialmente relevante en sistemas distribuidos.

# Bibliografía

## Sistemas Distribuidos


- **Capítulo 7:** Francisco Muñoz, Estefanía Argente, Agustín Espinosa, Pablo Galdámez, Ana García-Fornes, Rubén de Juan, Juan Salvador Sendra. ***Concurrencia y Sistemas Distribuidos***. Editorial Universidad Politécnica de València. , ISBN: 978-84-8363-986-3.

- Andrew S. Tanenbaum y Maarten van Steen. **Sistemas Distribuidos: Principios y Paradigmas**. Pearson, 2a edición, 2008. ISBN 9789702612803.


## Middleware


- Philip A. Bernstein**. Middleware: A model for Distributed System Services**. Communications of the ACM, Vol. 39(2), pp. 86-98. 1996.