# PROGRAMACION CONCURRENTE

## VI.2: Introducción a los sistemas distribuido: Paradigma cliente/servidor

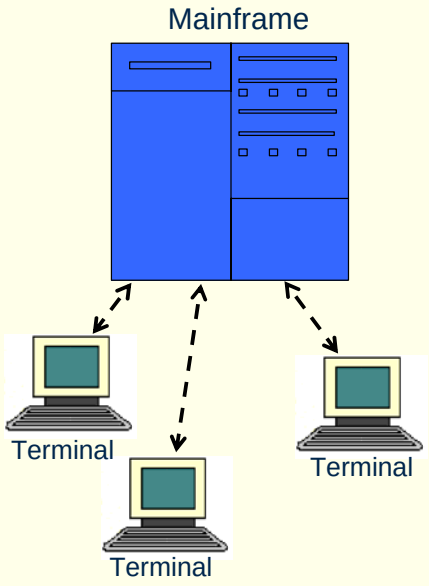
José M. Drake

Notas:

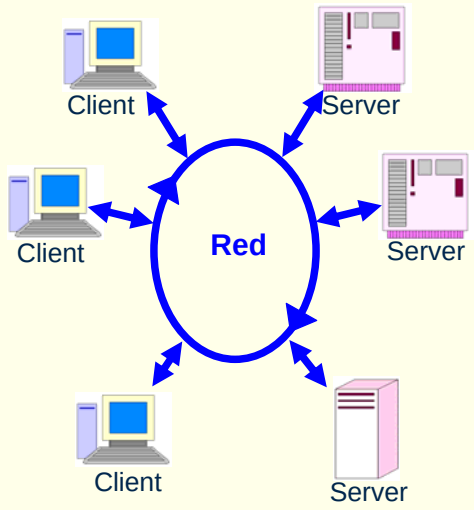
**Posibilidades que ofrece Java para la comunicación en red: Socket,RMI y URL.**

# Antiguos y nuevos tiempos en arquitecturas

Arquitectura centralizada



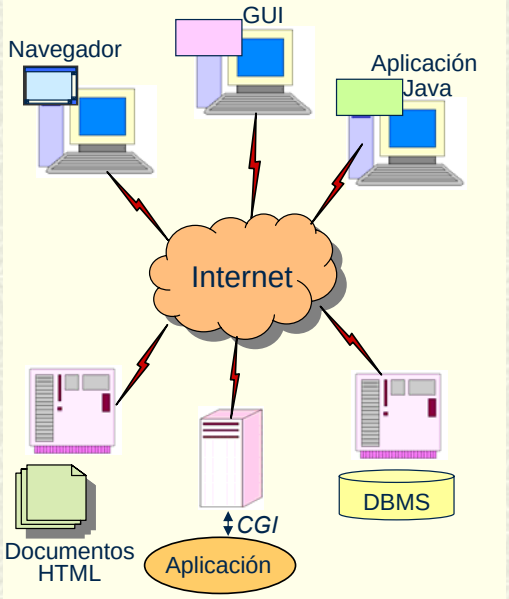

Arquitectura distribuida

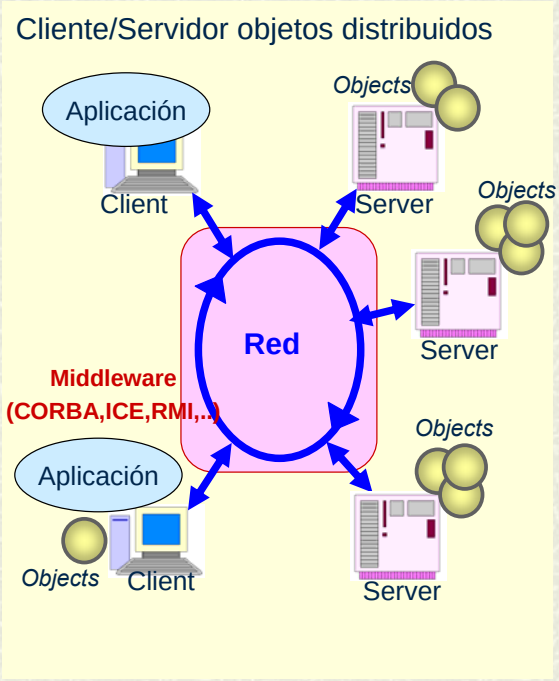





Notas:

# Tipos de arquitecturas distribuidas






Notas:

## Metodología Cliente/Servidor

- La metodología Cliente/Servidor es un paradigma de organización de los elementos que constituyen una aplicación distribuida, para que colaborando conjuntamente implemente la funcionalidad especificada a la aplicación:
  - Clientes: elementos activos que dirigen las actividades que deben ejecutarse para implementar la tarea requerida por la aplicación. Requiere de los servidores que ejecuten algunas de esas actividades.
  - Servidores: Elemento pasivos especializados en realizar ciertas tareas bajo requerimientos de los clientes. Habitualmente representan elementos que son compartidos por múltiples clientes, de una o varias aplicaciones.
- Proporciona un marco de referencia sencillo, flexible y abierto para distribuir la ejecución de una aplicación en múltiples nudos de una plataforma. En él la mezcla y el acoplamiento es la norma.



Notas:

## Características de la arquitectura Cliente/Servidor (1)

- **Servicios**: Facilita la colaboración de procesos que se ejecutan en diferentes máquinas, a través de intercambios de servicios. Los procesos servidores proveen los servicios, los clientes los consumen.
- **Recursos compartidos**: Los servidores pueden ser invocados concurrentemente por los clientes, y una de sus principales funciones es arbitrar el acceso a recursos compartidos que son gestionados por el propio servidor.
- **Protocolos asimétricos**: Un servidor puede atender a múltiples clientes. El cliente conoce el servidor que invoca. El servidor no necesita conocer el cliente que atiende.
- **Independencia de la ubicación**: La ubicación de los servidores es irrelevante. Se utilizan servicios de localización definidos a nivel de plataforma para que los clientes encuentren a los de servidores.
- **Compatibilidad de clientes y servidores**: Los mecanismos de interacción entre clientes y servidores son independientes de las plataformas. Un middleware independiza la aplicación de la plataforma.



Notas:

## Características de la arquitectura Cliente/Servidor (2)

- **Comunicación basada en intercambio de mensajes**: Los clientes y servidores son elementos acoplados de forma muy libre. Interaccionan a través de intercambios de mensajes, con los se implementan las invocaciones de los servicios y las respuestas de los servicios.
- **Encapsulación de los servicios**: Los servicios son elementos especializados, que tienen declarados públicamente los servicios que puede servir. Sin embargo, la forma que implementa el servicio es sólo propia de él, y no puede afectar a los clientes que los requieren.
- **Escalabilidad**: Las aplicaciones basadas en clientes/servidores son fácilmente escalables. Hay dos tipos de escalado:
  - Escalado vertical: Los sistemas pueden crecer por un incremento del número de clientes y servidores.
  - Escalado horizontal: Los servidores pueden descomponenrse en grupos de servidores que ofrezcan servicios desacoplados mas específicos.
- **Integridad**: La información es administrada por el servidor de forma unificada, dando lugar un mantenimiento mas sencillo y seguro. El middleware de distribución garantiza la seguridad en los accesos a los servicios y en la integridad de los datos.



Notas:

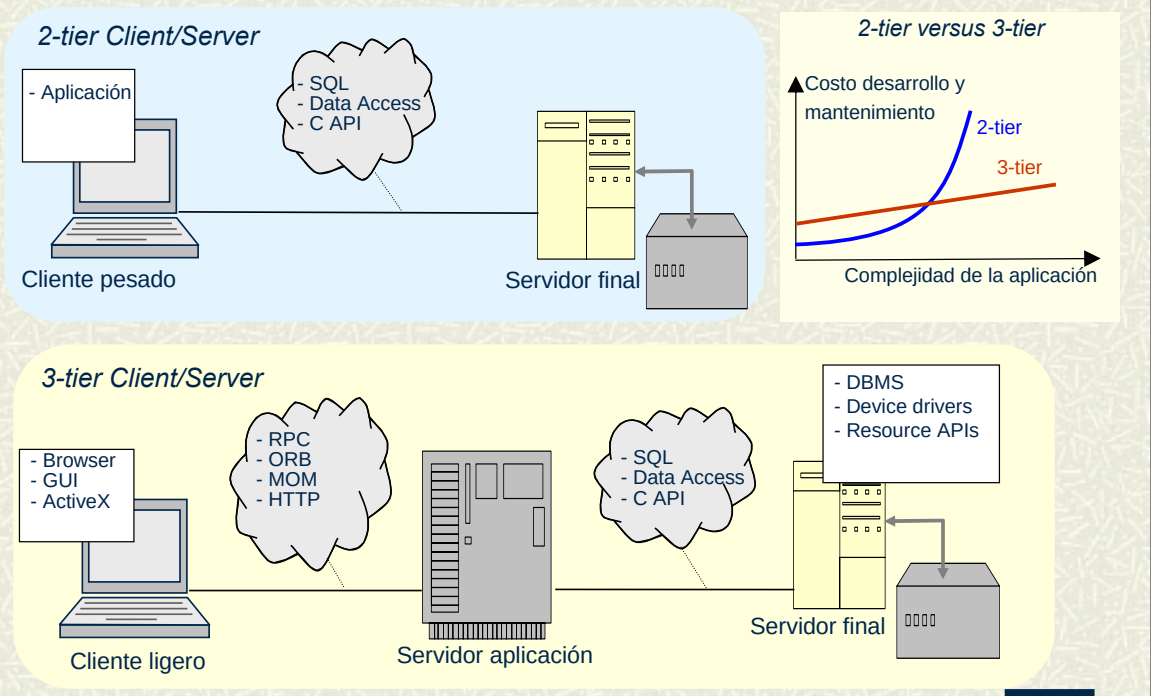
## Estrategias de reparto de la complejidad.

- **Clientes pesados / Servidores ligeros**: La mayor parte de la funcionalidad de la aplicación se implementa en el cliente.
  - Los servidores son mecanismo de acceso a recursos compartidos.
  - Mayor flexibilidad para aplicaciones que implementan nuevas funcionalidades.
  - Ejemplos: Servidores de bases de datos o servidores de ficheros.
- **Clientes ligeros / Servidores pesados**: La mayor parte de la funcionalidad se implementa en los servidores.
  - Incrementar la reusabilidad del código.
  - Son mas fáciles de desplegar y administrar.
  - Se basan en servidores mas abstractos que reducen el flujo por la red.
  - En vez de proporcionar datos, exportan procedimientos.
  - Ejemplos: Servidores de transacciones y servidores web.
- Ambos modelos coexisten y se complementan dentro de una misma aplicación.



Notas:

# Servidor/cliente de 2-niveles y n-niveles






Notas:

# Comparación de sistemas de 2-niveles y 3-niveles

| | 2-Niveles | 3-Niveles |
|---|---|---|
| **Administración del sistema** | Compleja: El cliente se constituye en administrador efectivo | Menos compleja: La aplicación puede gestionarse con las herramientas de gestión de los servidores |
| **Seguridad** | Baja: Seguridad a nivel de la transmisión de la información. | Alta: se puede implementar a nivel del servicio, operación u objeto. |
| **Encapsulado de la información** | Bajo: Las estructuras de datos son públicas | Alta: El cliente invoca servicio y métodos |
| **Carga de la red** | Alta: Se envían por la red comandos de bajo nivel. Deben descargarse datos al cliente para ser analizados. | Baja: Sólo se envían requerimientos de servicios y respuestas elaboradas a éstos |
| **Escalabilidad** | Pobre: No puede gestionarse conjuntamente grupos de clientes. Los servidores son terminales y difíciles de replicar. | Excelente: Permite concentrar los clientes. Posibilita distribuir la carga entre servidores replicados. |
| **Reutilización de aplicaciones** | Pobre: Las aplicaciones son monolíticas y concentradas en el cliente. | Excelente: Los servicios y objetos de aplicación pueden reutilizarse. |



Notas:

# Comparación de sistemas de 2-niveles y 3-niveles

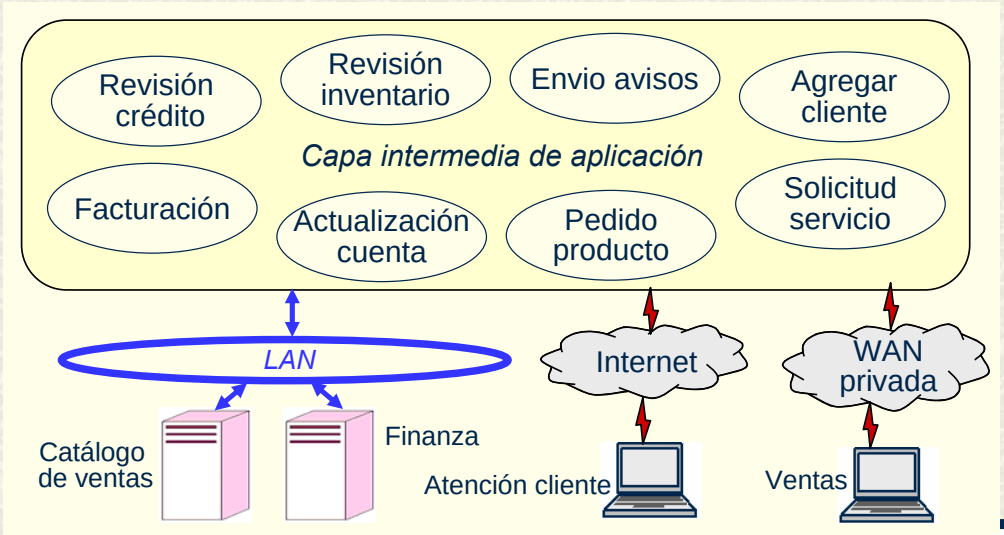
| | 2-Niveles | 3-Niveles |
|---|---|---|
| **Facilidad de desarrollo** | Alta: Solo se requiere conocer la funcionalidad de los servidores terminales. | Requiere nuevas herramientas: Para desarrollar los lados cliente y servidor de las aplicaciones. |
| **Infraestructura entre servidores** | No se requiere | Se requiere middleware para facilitar la interacción entre servidores. |
| **Integración de aplic. legadas** | No | Si: Mediante pasarelas planteadas como servidores u objetos adaptadores. |
| **Soporte de web** | Pobre: La anchura de banda limitada de internet dificulta la descarga de clientes pesados. | Excelente: Los clientes formulados como applets y beams son idóneos para descargarse por la red. |
| **Métodos de comunicación** | Invocaciones síncronas: de tipo RPC. | Invocaciones síncronas y asíncronas: Tipo RPC, mensajes sin conexión, basadas en colas, eventos, etc. |
| **Flexibilidad arquitectural** | Limitada: Solo se pueden establecer las conexiones del cliente con los servidores terminales. | Excelente: Existen múltiples posibilidades de organización y distribución de los servidores por la plataforma. |
| **Robustez a fallos** | Baja | Excelente: Puede reinstalarse los servicios de aplicación en cualquier equipo. |



Notas:

# Arquitectura de sistemas de n-niveles

- Los servidores de aplicación no se organizan como elementos monolíticos 3-nivels, sino como conjuntos de muchos servidores sencillos n-niveles.
- Los clientes combinan servicios de diferentes servidores y cada servidor puede implementar su funcionalidad basándose en otros servidores.






Notas:

## Ventajas de arquitecturas n-niveles: Componentes

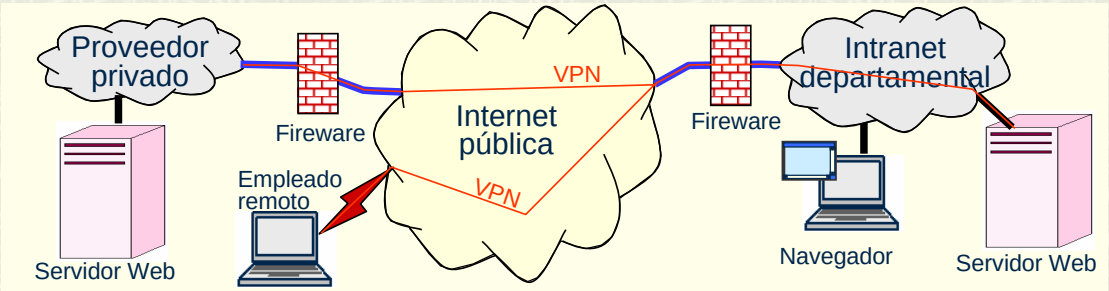
- Los proyectos grandes se pueden desarrollar como conjunto de pequeños proyectos, con mayor posibilidades de éxito.
- Se pueden reutilizar componentes entre aplicaciones. La aplicaciones se construyen agrupando componentes de forma específica a la funcionalidad que requiere.
- Permite elevar el nivel de abstracción. Los clientes pueden requerir los servicios por sus nombres y no necesitan conocer de que base de datos proceden, ni que elementos participan para implementarlos.
- Permiten incorporar fácilmente elementos legados.
- Los entornos de componentes no envejecen sino mejoran:
  - Nuevos clientes pueden generarse añadiendo algún objeto servidor mas.
  - Los objetos servidores pueden modificarse o sustituirse sin que afecten a los clientes.



Notas:

# Tecnología cliente/servidor intergaláctica

- Hay una tendencia hacia arquitecturas cliente/servidor interaláctica basada en WEB.
  - Crecimiento exponencial de la anchura de banda en redes WAN
  - Nueva generación de dispositivos y servicios habilitados para la WEB
  - Irrelevancia de la cercanía.
- Las posibilidades de negocio es de varios ordenes superior que las arquitecturas departamentales.
- En el futuro una red intergaláctica conectada a múltiples intranet departamentales protegidas por firewalls.



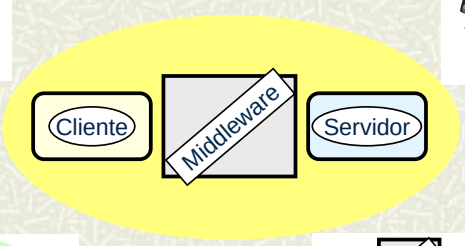


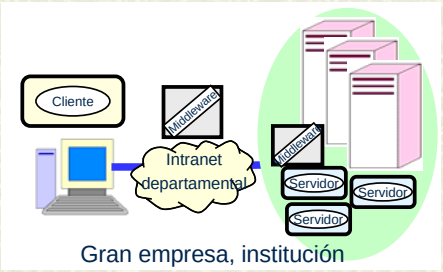

Notas:

# Arquitecturas Cliente/Servidor

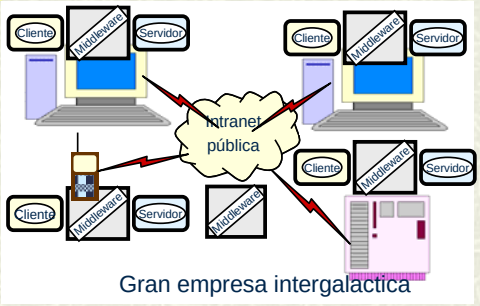

Tienda pequeña, hogar


Departamento



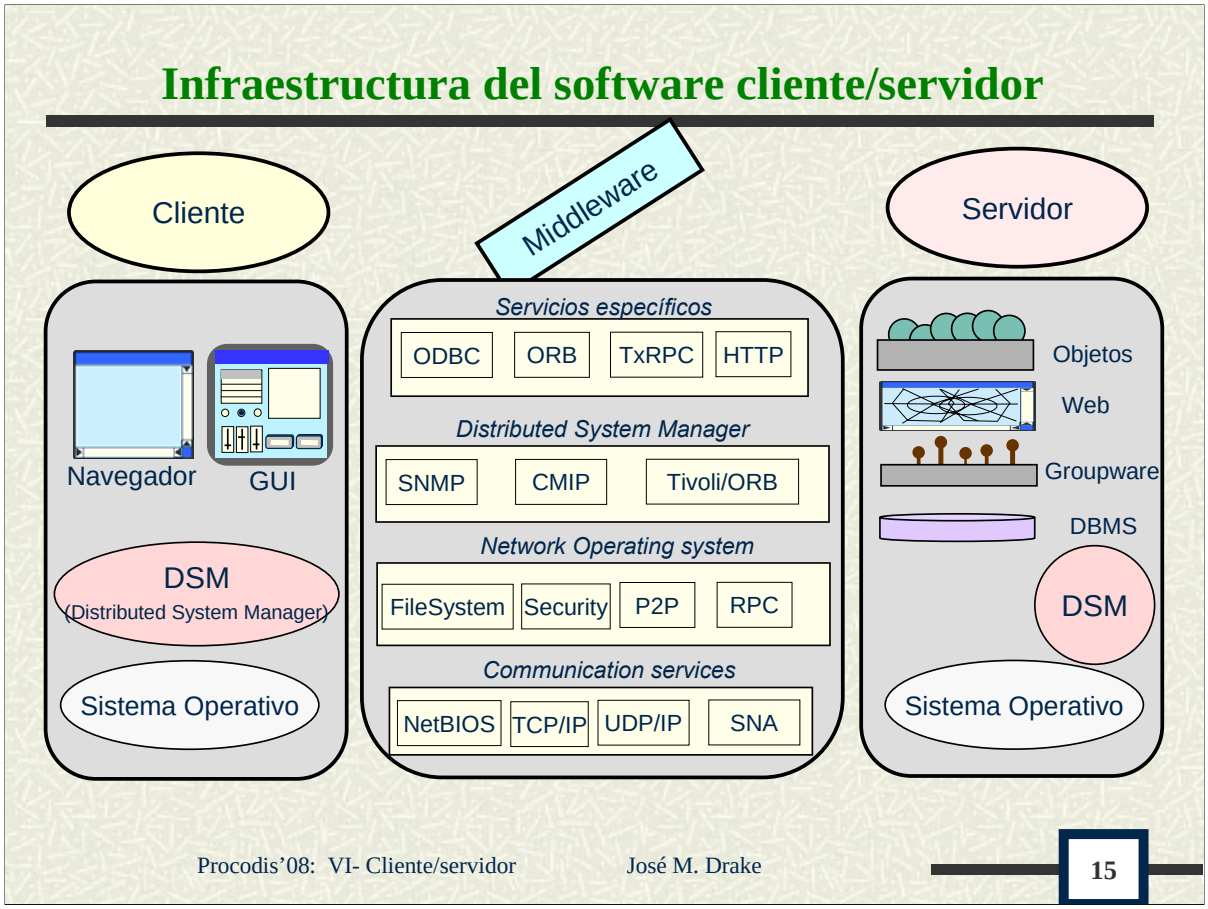

Gran empresa, institución




Gran empresa intergalactica



Notas:


Infraestructura del software cliente/servidor
Cliente
Middleware
Servidor
Servicios específicos
ODBC
ORB
TxRPC
HTTP
Distributed System Manager
SNMP
CMIP
Tivoli/ORB
Network Operating system
FileSystem
Security
P2P
RPC
Communication services
NetBIOS
TCP/IP
UDP/IP
SNA
Navegador
GUI
DSM
(Distributed System Manager)
Sistema Operativo
Objetos
Web
Groupware
DBMS
DSM
Sistema Operativo
Procodis'08: VI- Cliente/servidor
José M. Drake
15

Notas:

## Clientes

- Constituyen el núcleo de una aplicación, aunque puede ser sólo una parte pequeña de ella.
- Los clientes tienen en común que elaboran su funcionalidad solicitando servicio a los servidores. Los clientes se diferencian en con qué desencadenan las invocaciones:
  - Clientes que no necesitan GUI ni concurrencia: Cajeros automáticos, puntos de venta, lectores de código de barras, teléfonos móviles,faxes, etc.
  - Clientes que no necesitan GUI pero si concurrencia: Robots, máquinas herramientas, demonios, etc.
  - Clientes que necesitan GUI: La evolución del programa y los servicios que requiere son resultado de la interacción del operador con la interfaz gráfica que se le ofrece. Utilizan el modelo objeto/acción.
  - Clientes con OOUI (Object Oriented user Interface): La evolución del programa y los servicios que utiliza son resultado de la interacción del operador con iconos mostrados en la consola y que una vez abiertos pueden ser manipulados concurrentemente y en cualquier orden.



Notas:

## Servidor

- Su función es satisfacer los requerimientos de servicio que son realizados por múltiples clientes que quieren hacer uso de la funcionalidad que ofrece o acceder a los recursos protegidos que gestiona.
- Su funcionalidad suele ser compleja y pesada, pero su actividad es monótona y poco creativa:
  - Espera pasivamente los clientes inicien requerimientos de servicios.
  - Ejecuta concurrentemente los requerimientos recibidos de los clientes.
  - Prioriza las respuestas a los clientes de acuerdo con su importancia o urgencia.
  - Arranca y ejecuta demonios y actividades de segundo plno (background)
  - Se mantienen continuamente en ejecución, y si cae por un fallo se recupera.
  - En función de la actividad que en cada momento desarrolla puede acaparar grandes cantidades de capacidad de procesamiento, recursos y anchura de banda de comunicaciones.



Notas:

## ¿Que necesita los servidores del sistema operativo?

- Servicios básicos:
  - Gestión de concurrencia basada en threads
  - Políticas flexibles de planificación de los threads.
  - Mecanismos de sincronización que garantice el acceso seguro a los recursos compartidos.
  - Mecanismos eficiente de comunicación entre procesos.
  - Administración flexible de la memoria.
  - Sistema de ficheros multiusuarios.



Notas:

## ¿Que necesita los servidores del sistema operativo?

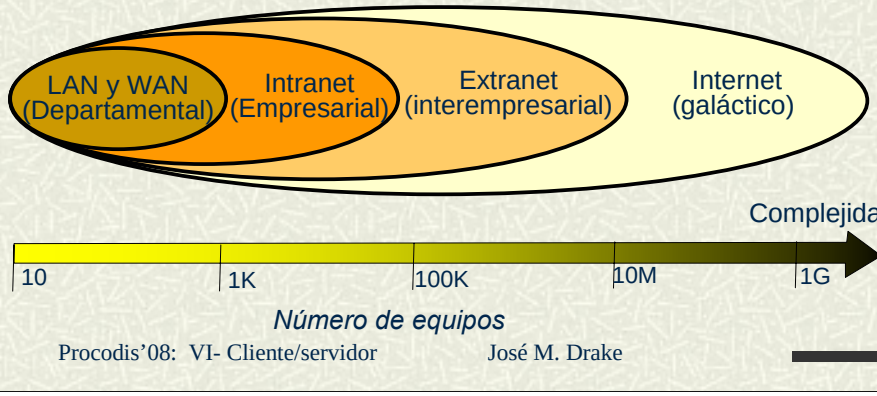
- Servicios extendidos:
  - Mayor facilidades para acceder a través de la red a sistemas con diferentes protocolos.
  - Mayores facilidades a información compartida.
  - Facilidades para que vendedores independientes extiendan por si el sistema operativo.
  - Acceso a diferentes middelwares que en interacciones distribuidas se confunden con el propio S.O.
  - Streaming y mecanismos de transferencia de grandes bloques de información.
  - Directorios globales y servicios de páginas amarillas, que permitan localizar los servidores y los servicios que ofrecen.
  - Servicios de identificación, autentificación y encriptación.
  - Servicio de tiempos y hora global.
  - Servicios de bases de datos
  - Servicios de transacciones.
  - Brocker para la gestión de objetos y componentes.



Notas:

## Middelware: Sistema operativo de red (NOS)

- El sistema operativo de red tiene una doble misión:
  - Proporcionar la sensación de un único sistema de lo que en realidad es una plataforma distribuida heterogénea.
  - Ocultar los detalles y particularidades de los mecanismos de comunicación subyacentes.
- Los sistemas operativos de red evolucionan en función de como se amplía su ámbito de operación:






Notas:

## Transparencia

- Transparencia significa crear la ilusión a los usuarios, y a los programadores de aplicaciones de que los sistemas distribuidos son homogéneos:
  - Transparencia de ubicación: Se opera sin necesidad de conocer la dirección física de los equipos.
  - Transparencia de espacio de nombres: Se utilizan las mismas reglas de identificación para designar recursos de todo el dominio.
  - Transparencia de conexión: Con una única clave se accede a cualquier elemento, con independencia de cuantos equipos y de que naturaleza se atraviesen.
  - Transparencia de replicación: No se conocen cuantas copias del servidor existen ni con cual de ellas se está interactuando.
  - Transparencia de local/remoto: Se puede operar con cualquier recurso remoto como si fuese local.
  - Transparencia de reloj: Existe una hora global única a nivel de sistema distribuido. La datación de los eventos es comparable con independencia de en que nudo se ha establecido.
  - Transparencia de fallos: Debe gestionar los fallos de comunicación y disponer de mecanismos de recuperación automática.
  - Transparencia de administración: La interfaz de administración de recursos es uniforme, y la coordinación con los sistemas operativos locales resultan ocultos.



Notas:

## Transacciones

- Conjunto de actividades que se ejecutan en diferentes nudos de una plataforma distribuida para ejecutar una tarea de negocio.
- Una transacción finaliza cuando todas las parte implicadas clientes y múltiples servidores confirman que sus correspondientes actividades han concluido con éxito.
- Si una actividad falla, todas deben recuperar el estado previo al inicio de la transacción.
- Las transacciones pueden ser simples o compuestas (sagas, encadenadas, anidadas, etc.)
- Un sistema distribuido sin gestor de transacciones es como una sociedad sin ley contractual.



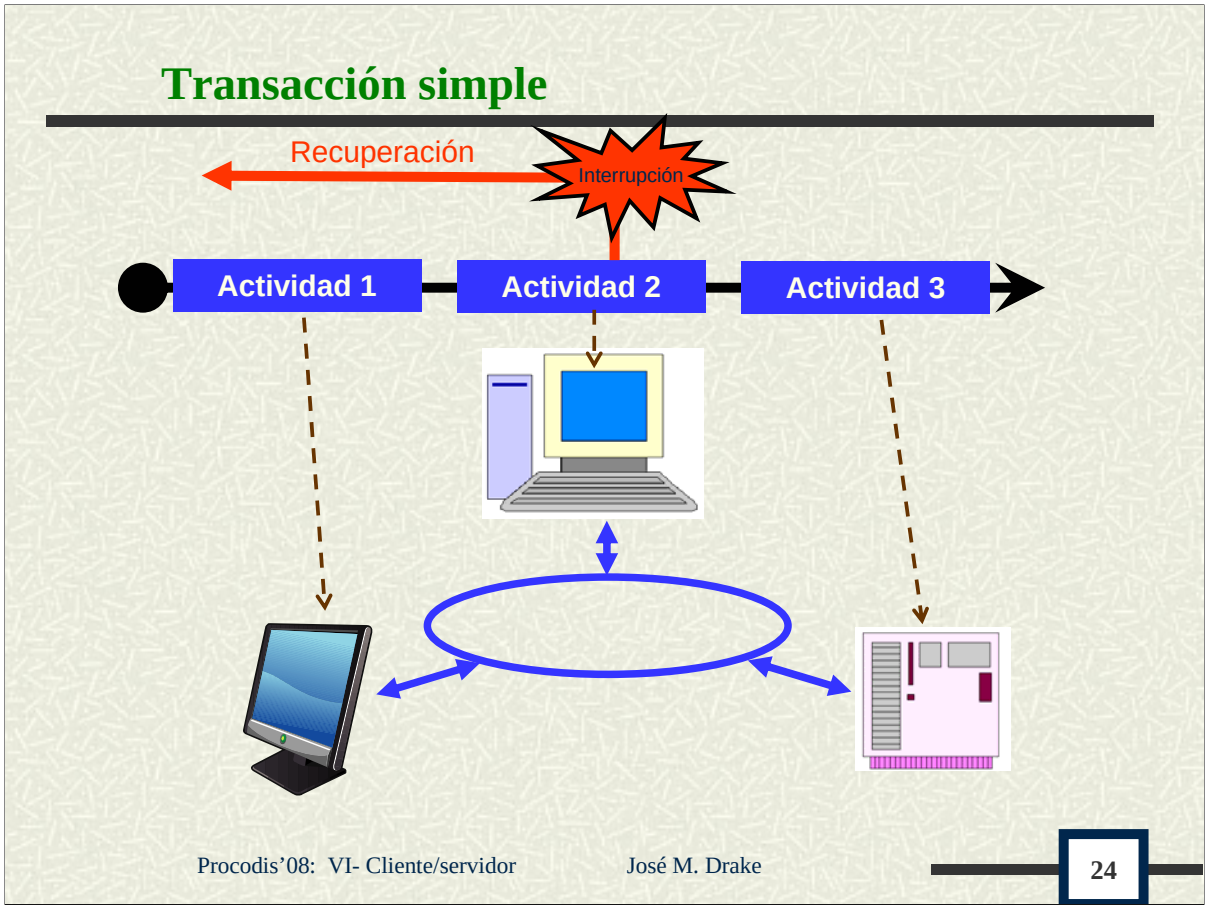
Notas:

## Transacciones ACID (Atomicity, Consistency, Isolation, Durability)

- **Atomicidad**: Una transacción es una unidad indivisible de trabajo, Todas las actividades que comprende deben ser ejecutadas con éxito.
- **Congruencia**: Después de que una transacción ha sido ejecutada, la transacción debe dejar el sistema en estado correcto. Si la transacción no puede realizarse con éxito, debe restaurar el sistema al estado original previo al inicio de la transacción.
- **Aislamiento**; La transacciones que se ejecutan de forma concurrente no deben tener interferencias entre ellas. La transacción debe sincronizar el acceso a todos los recursos compartidos y garantizar que las actualizaciones concurrentes sean compatibles entre si.
- **Durabilidad**: Los efectos de una transacción son permanente una vez que la transacción ha finalizado con éxito.

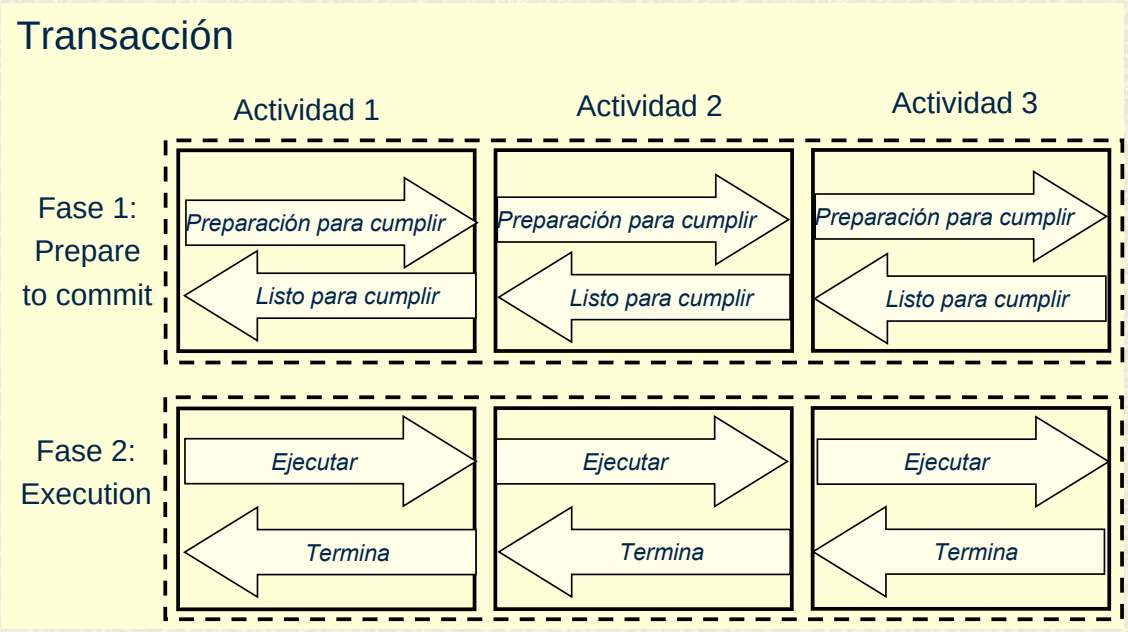


Notas:


Transacción simple
Recuperación
Interrupción
Actividad 1
Actividad 2
Actividad 3
Procodis'08: VI- Cliente/servidor
José M. Drake
24

Notas:

# Protocolo de ejecución en dos fases






Notas:

## Situaciones en las que las transacciones simples fallan

- Transacciones de negocio que deben volver parcialmente a un estado anterior.
- Transacciones de negocio en las que interviene un operador humano.
- Transacciones de negocio que se extiende por largos periodos de tiempo.
- Transacciones de negocios masivas.
- Transacciones de negocio que abarcan varias empresas.



Notas:

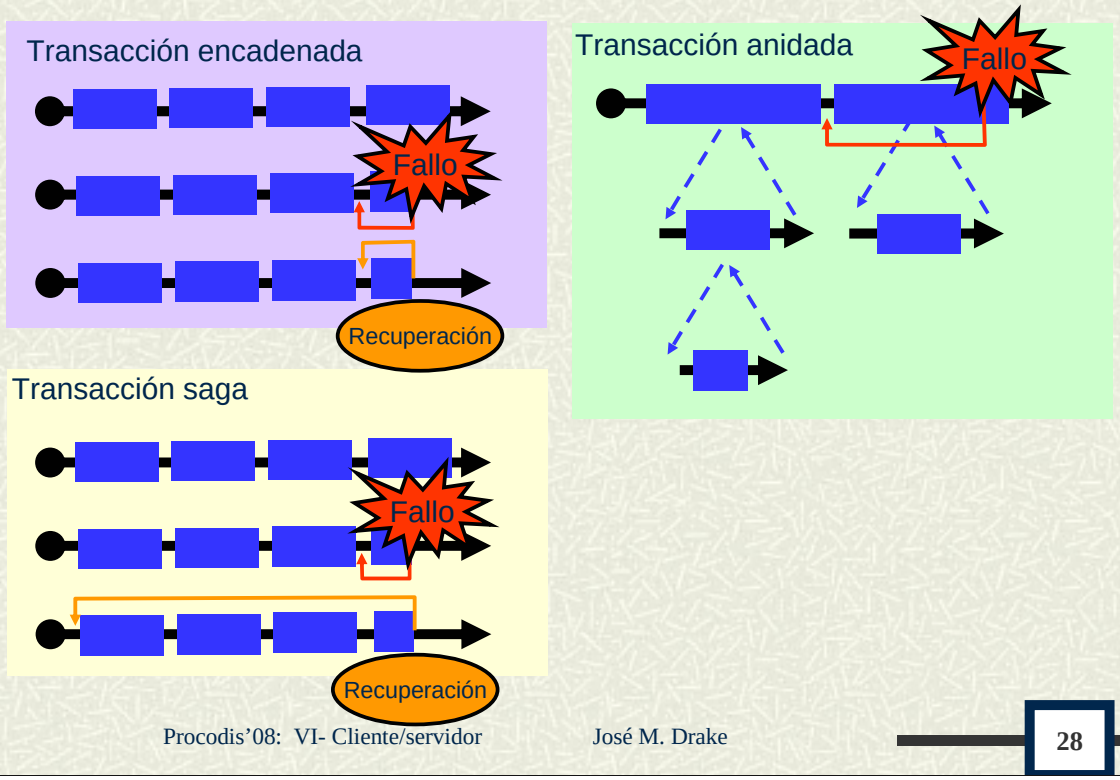
## Principios para superar las transacciones lineales

- Introducir los puntos de sincronización como puntos en los que se salva el trabajo realizado y a los que se puede retroceder si existen fallos o cambios de criterio.
- Diferencias entre punto de sincronización y compromiso:
  - A un punto de sincronización se puede recular, y sin embargo mantener viva la transacción.
  - Los puntos de sincronización ofrecen mayor capacidad granular sobre lo que se guarda y se deshace.
  - La gran diferencia es la persistencia, el compromiso es durable mientras que el punto de sincronización es efímero y volátil.
- Si una transacción se cae la información almacenada en todos los puntos de sincronización se pierde.
- Existen diversos modelos: Contratos (*Contracts*), transacciones migrantes (*Migrating Transactions*), carrito de compras.(*ShopingCart Transaction*)



Notas:

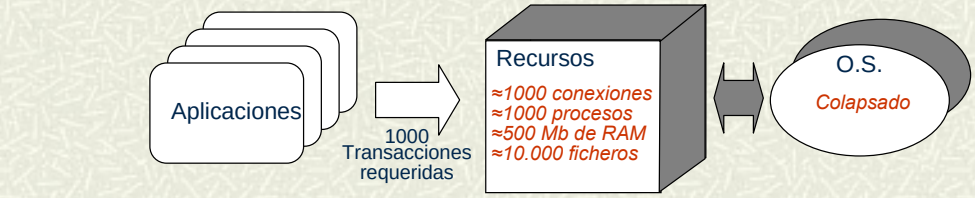
# Clasificación de la transacciones



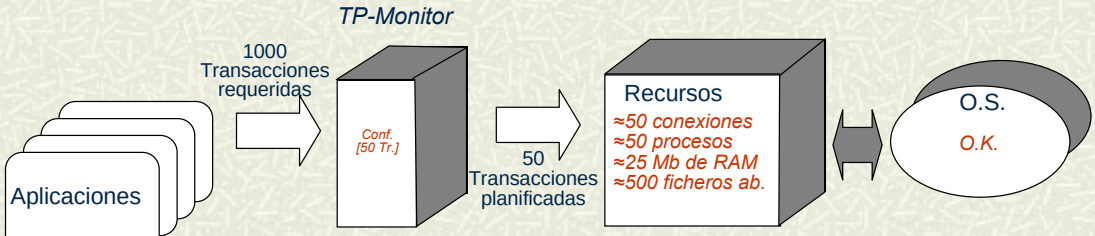



Notas:

# Necesidad de un gestor de transacciones




***Recursos requeridos sin TP Monitor***




***Recursos requeridos con TP Monitor***



Notas:

## Gestor de transacciones (*Monitor TP*)

- Un gestor de transacciones se encarga en la administración de la transacción desde:
  - El punto de inicio por un cliente.
  - La ejecución de los servicios en los diferentes servidores.
  - Y en el retorno de resultados y la aceptación en el cliente inicial.
  - Bajo situación de fallo recupera la situación inicial de cada elemento afectado.
- Son gestores genéricos, capaces de supervisar miles de transacciones de diferentes clientes, sin necesitar conocer la tarea que corresponde a cada transacción.
- Es el que garantiza que en la transacción se satisfagan las propiedades ACID.



Notas:

## ¿Que gestiona?

- Administración de procesos:
  - Inicio de los procesos de servicio.
  - Supervisión de la ejecución.
  - Redistribución de las cargas de trabajo.
- Administración de las transacciones:
  - Garantizar las propiedades ACID en cada una de ellas.
  - Restaurar el sistema y reiniciar la transacción en caso de fallo.
- Administración de las comunicaciones:
  - Posibilita los diferentes tipos de invocación
  - Publicitar y suscribir los servicios.



Notas: