CC5303 – Sistemas Distribuidos

1.- Introducción a Sistemas Distribuidos

Parte 2

Sebastián Blasco V.

Resumen Clase Anterior

- SSDD son computadoras autónomas que trabajan juntas para dar la apariencia de un solo sistema coherente.
- Facilitan la **integración** de diferentes aplicaciones que se ejecutan en distintas computadora.
- Optan a la virtud del “fácil” **escalamiento**.
- **Transparencia**: Ocultar las dificultades inherentes (distribución de datos y control), dentro de lo posible.
- Lograr transparencia => diseño de SSDD
- 8 pecados

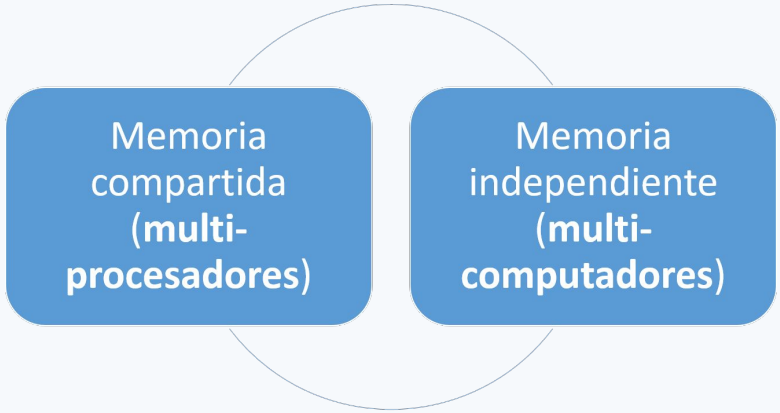
Contenidos

- Tipos de SSDD
  - Distinción HW y Software

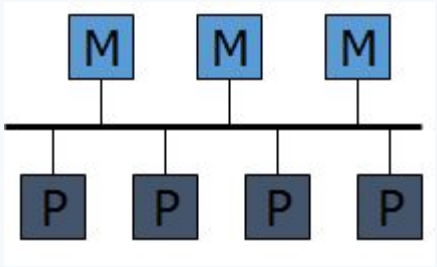
Arquitectura de Hardware

- Los sistemas de cómputo distribuidos (generalmente) se utilizan para aplicaciones de alto rendimiento que se originan de modo habitual en el campo del **cómputo en paralelo**.

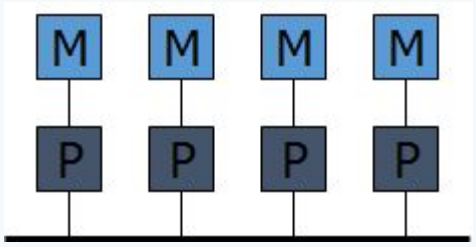
- En forma general todo sistema distribuido consiste en múltiple CPUs. Existen distintas maneras de lograr ese escenario, definiendo la **Arquitectura de Hardware** del SSDD



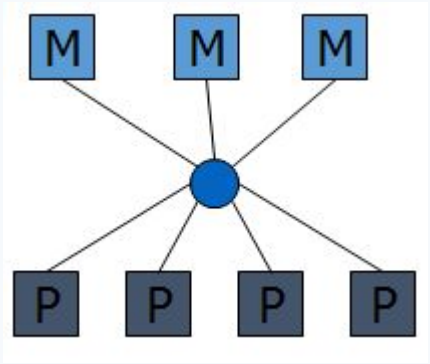
Arquitectura de Hardware



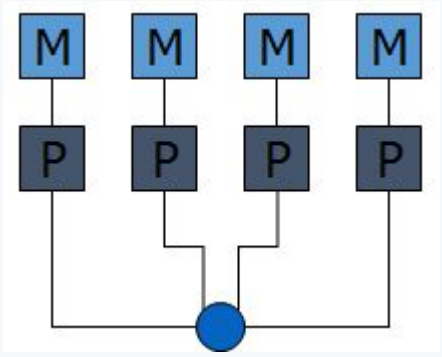
Memoria compartida - BUS



Memoria independiente - BUS



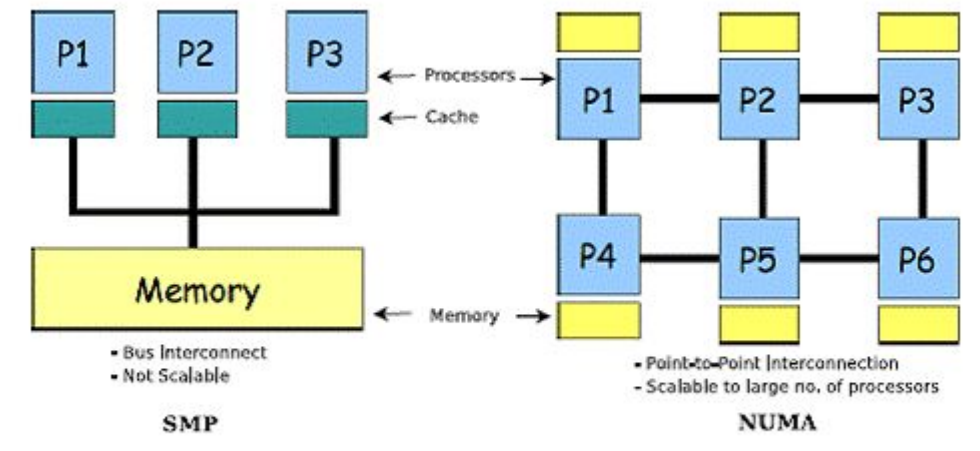
Memoria compartida - Switch



Memoria independiente - Switch

Arquitectura de Hardware

- Memoria compartida (**multi-procesadores**)
  - Todos los CPU tienen acceso a una memoria compartida
  - Symmetric Multiprocessor System (**SMP**).
  - Exige coherencia en el estado de la memoria.
  - En la medida que aumentan los CPUs mantener la coherencia afecta fuertemente el rendimiento



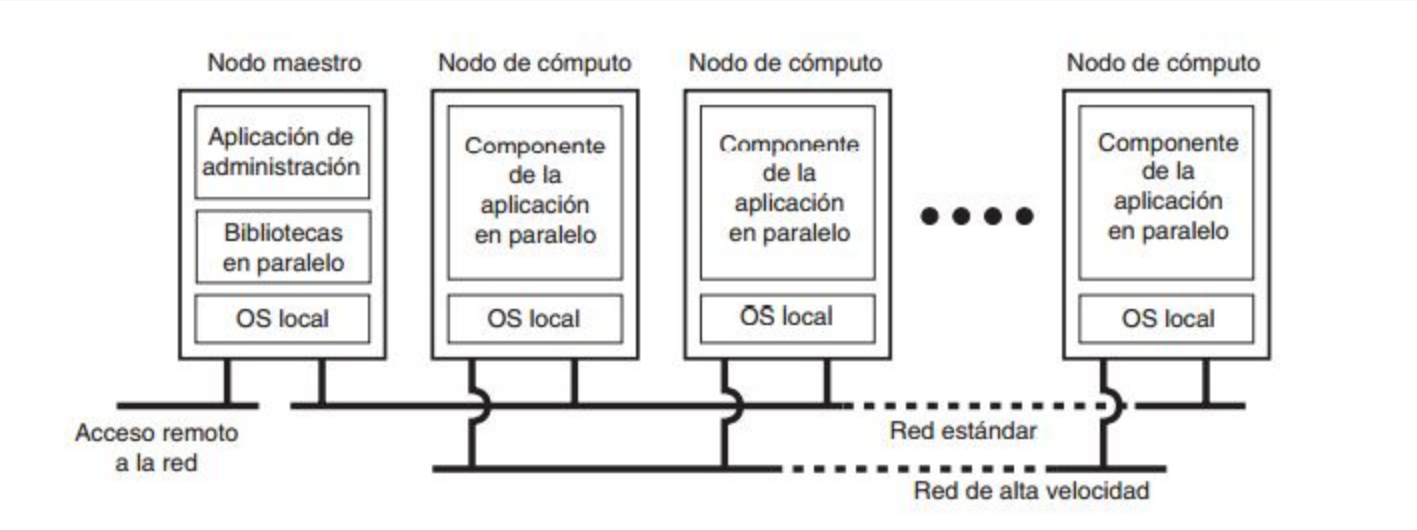
Es poco escalable y su escalabilidad es cara

Arquitectura de Hardware

- Memoria independiente (**multi-computadores**)
  - Se subdividen en dos categorías:
    - Homogéneos
      - La arquitectura y memoria es igual o similar en todos los nodos.
      - Generalmente conectados a través de una única -Usualmente de alto rendimiento- interfaz de red.
      - Orientada a programación paralela de un sólo programa de cálculo intensivo, sobre distintas máquinas
      - Ejemplo típico: clusters.
    - Heterogéneos
      - Múltiples arquitecturas
      - Diferentes comunicaciones
      - Diferentes formas de comunicación
      - Ejemplo típico: GRID

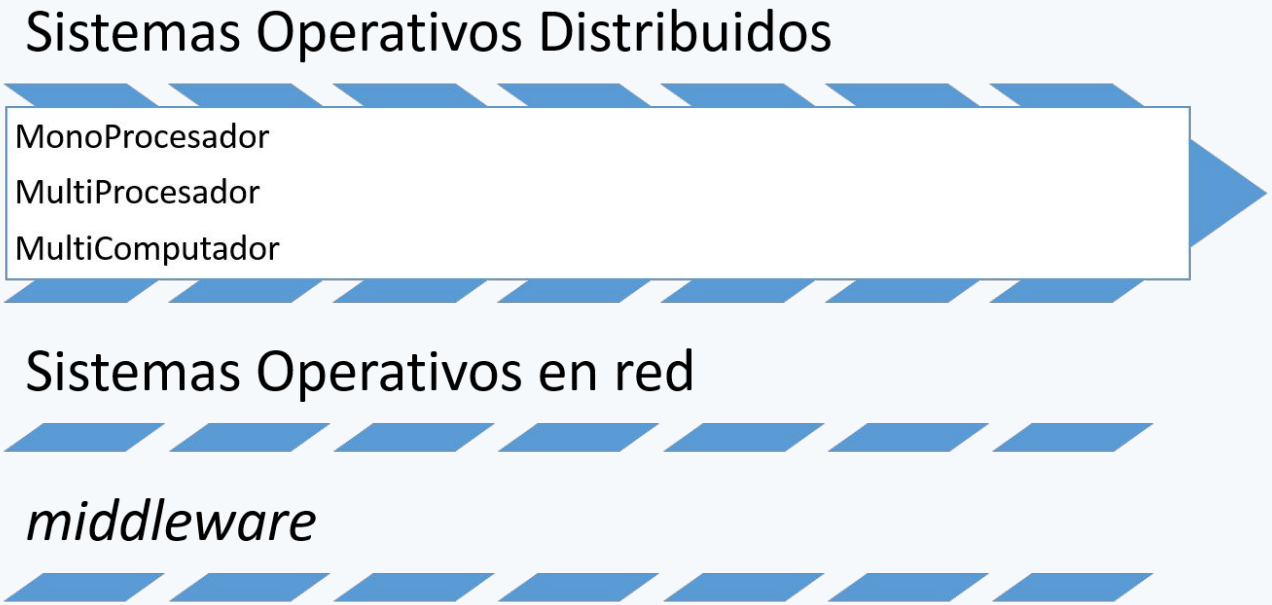
Arquitectura de Hardware

- Organización de control de cluster basada en nodos master/slave
  - Usada en *Linux Beowulf*



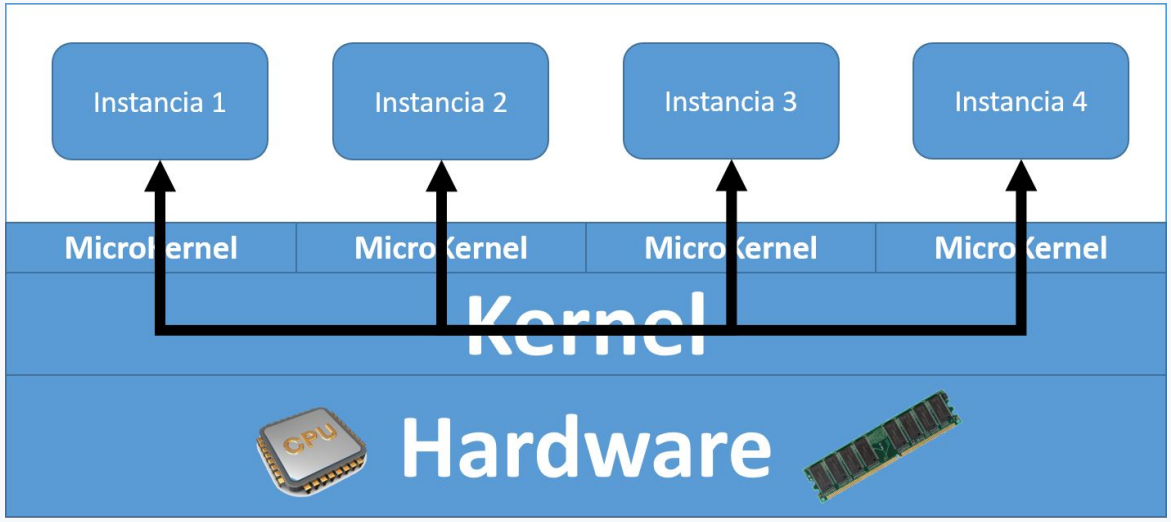
Arquitectura de Software

- La organización de software es clave también en el diseño de un SSDD. Dividiremos en 3 las plataformas de software posibles a atener el diseño de un SSDD:



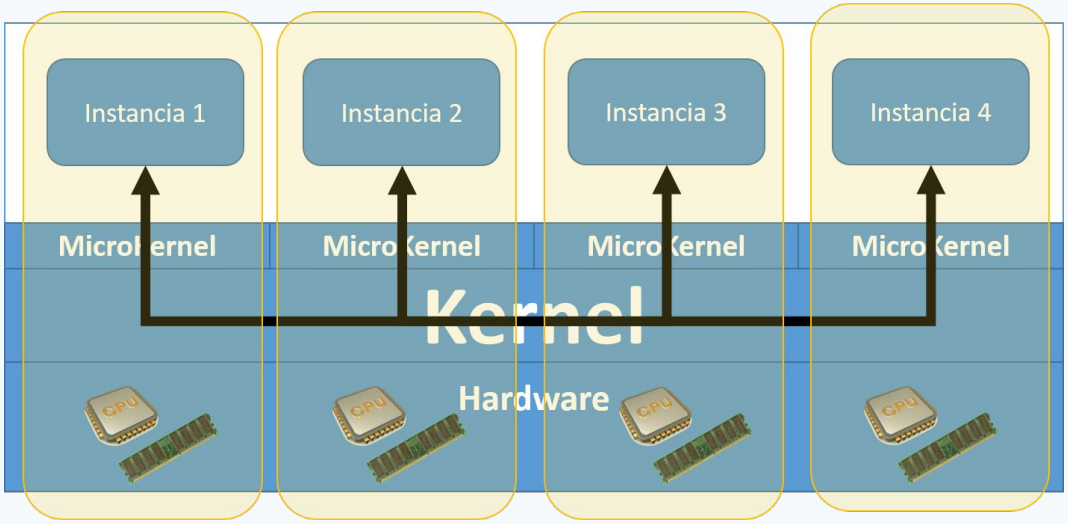
Arquitectura de Software

- Sistemas Operativos Distribuidos
  - S.O.D. Monoprocesador
    - Múltiples procesos compartiendo recursos sobre una máquina, gracias a un microkernel que implemente una máquina virtual.



Arquitectura de Software

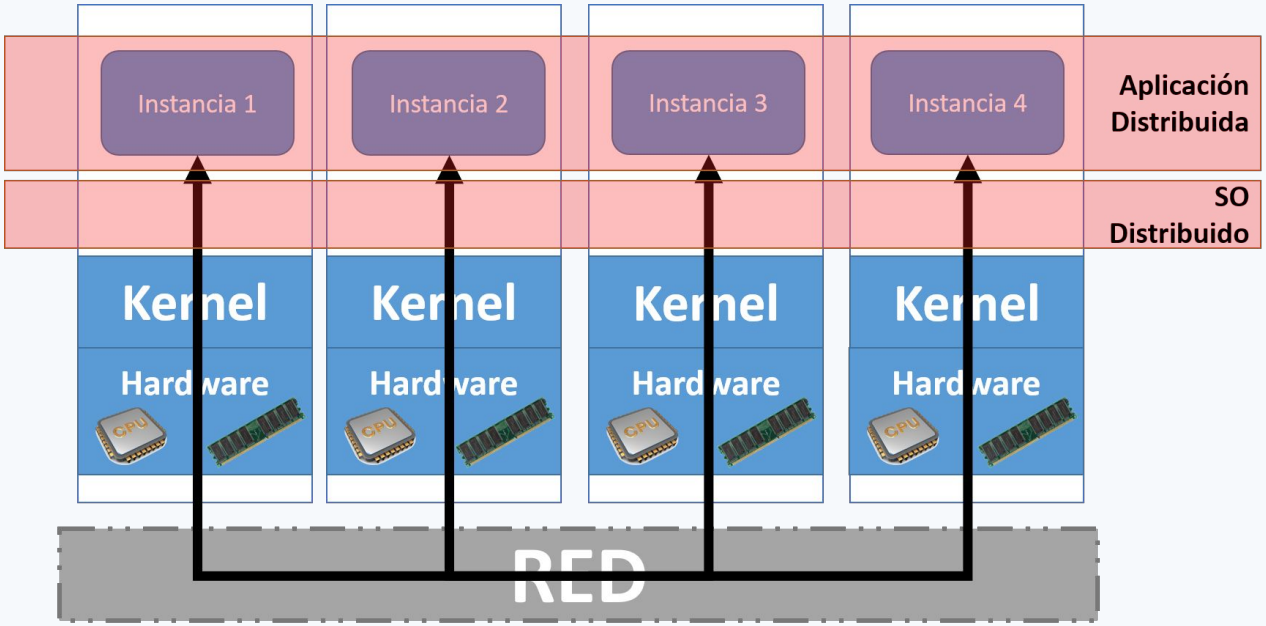
- Sistemas Operativos Distribuidos
  - S.O.D. Multiprocesador



- Múltiples procesadores compartiendo datos a través de memoria compartida.
- **Datos deben ser protegidos para garantizar consistencia.**
- Múltiples procesadores deben ser transparentes para la aplicación.
- Primitivas de sincronización: semáforos, locks, monitores.

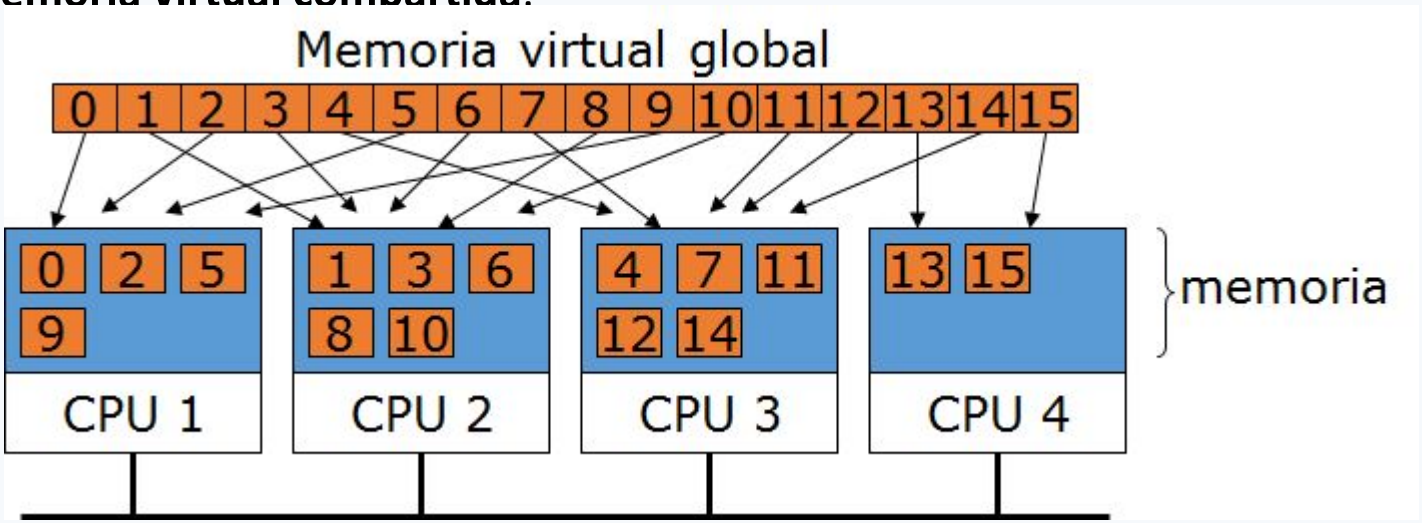
Arquitectura de Software

- Sistemas Operativos Distribuidos
  - S.O.D. Multicomputador
    - Cada procesador tiene su propia memoria.
    - Comunicación únicamente a través de paso de mensajes.
    - Sincronización basada en semánticas particulares de paso de mensajes



Arquitectura de Software

- Arquitecturas altamente distribuidas aprovechan la compartición de recursos, en particular: **memoria**
  - Memoria compartida distribuida
    - Múltiples procesadores -cada uno con su caché- comparten acceso a un banco de memoria de la máquina.
    - Múltiples máquinas -cada una con su banco de memoria- hacen uso de la **memoria virtual compartida**.



Arquitectura de Software

- *False Sharing* (PROBLEMA!)
  - Memoria de dos procesos distintos, alocada en la misma página de memoria
    - Procesos *p1* y *p2* (ejecutándose en distintas CPU's), trabajan con una misma página *z* de memoria.
      - *p1* accede a datos de *z* que nunca siempre se mantienen iguales
      - *p2* accede y modifica datos en z (ninguno que use a)
    - Los protocolos de coherencia de caché detectan el cambio de algo en *z* y pueden solicitar a *p1* que recargue constantemente *z* en su caché.

En la práctica, reduce el rendimiento de los cachés distribuidos, en pos de la consistencia.

Arquitectura de Software

- *False Sharing* (PROBLEMA!)

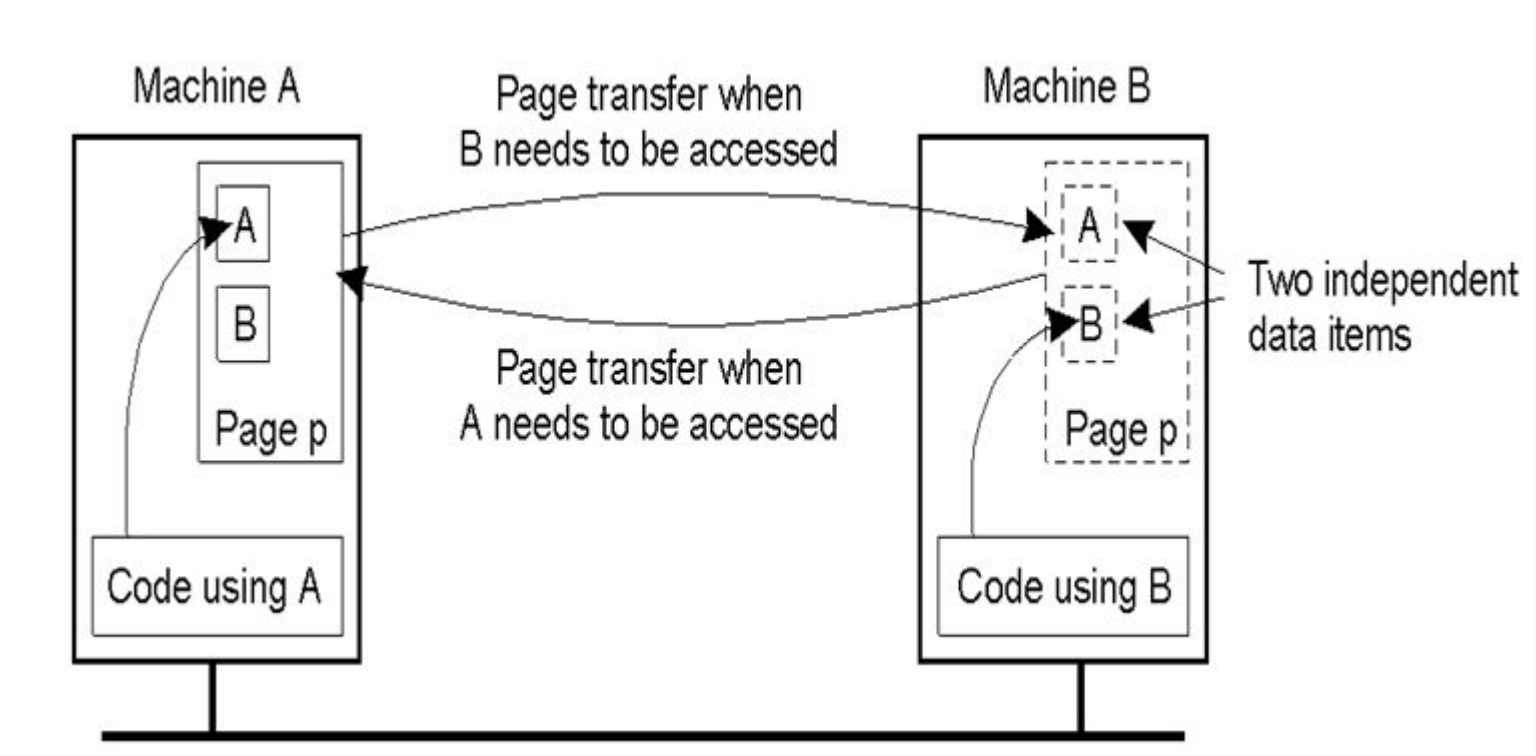
```
struct foo {
    int x;
    int y;
};

static struct foo f;

/* The two following functions are running concurrently: */

int sum_a(void)
{
    int s = 0;
    int i;
    for (i = 0; i < 1000000; ++i)
        s += f.x;
    return s;
}

void inc_b(void)
{
    int i;
    for (i = 0; i < 1000000; ++i)
        ++f.y;
}
```

Considerar una ejecución concurrente de los métodos ***sum_a*** y ***inc_b***

sum_a deberá constantemente re-leer ***x*** desde la memoria virtual compartida (en lugar de su caché) pues ***inc_b*** ha realizado un cambio en f que invalida su caché, aún cuando las modificaciones realizadas por ***inc_b*** son irrelevantes para ***sum_a***

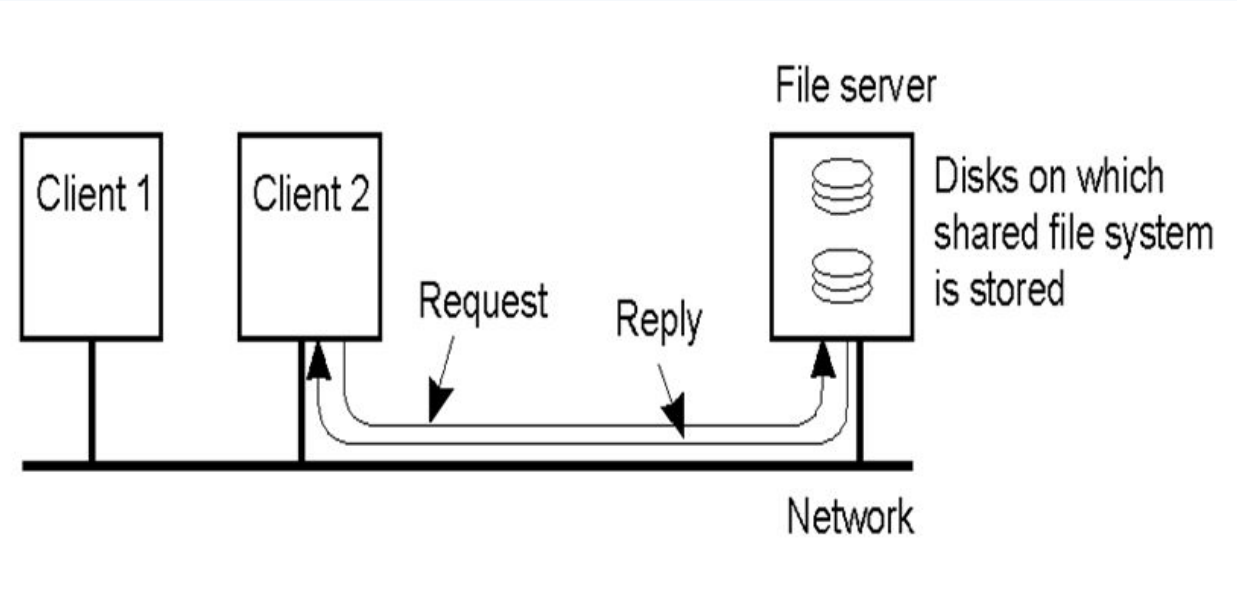
Arquitectura de Software

- *False Sharing* (PROBLEMA!)



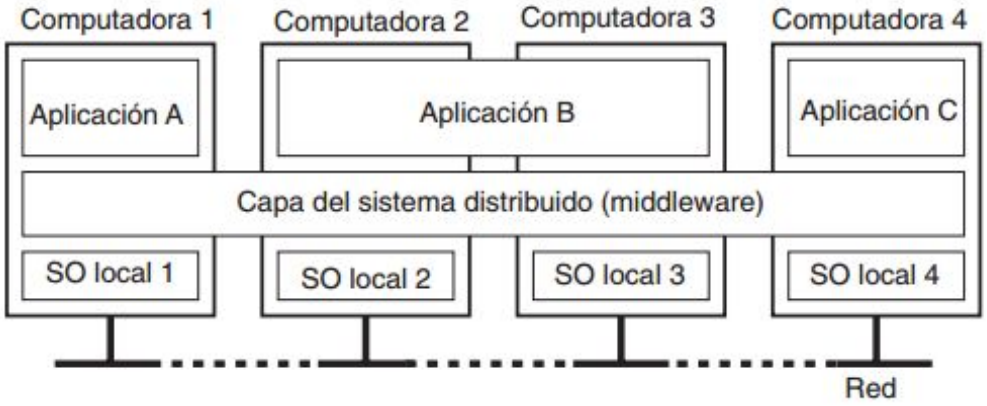
Arquitectura de Software

- Sistemas Operativos en Red
  - Dos o más computadores conectados por red.
  - Comparten los diferentes **recursos y la información** del sistema mediante la red.
  - Novell Netware, Personal Netware,Windows NT Server, UNIX, etc.



Arquitectura de Software

- *middleware*
  - Capa de software que ejecuta sobre el sistema operativo local de cada computador ofreciendo uno servicios distribuidos.
  - Abstrae la complejidad y heterogeneidad de los computadores del sistema
  - Proporciona una API para la programación y manejo de aplicaciones distribuidas
  - Existen diversos tipos de comunicación middleware (RPC, RMI, etc.)

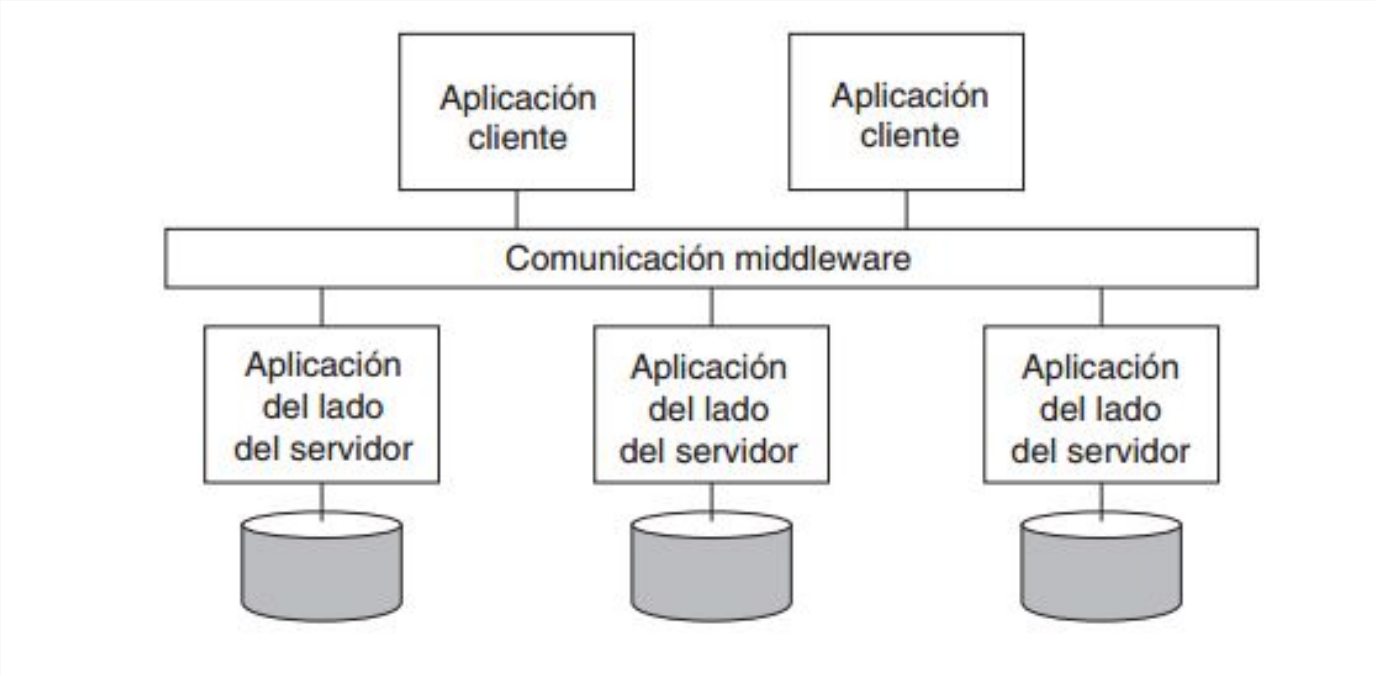


Arquitectura de Software

- *middleware*
  - Llamadas a procedimientos remotos (RPC)
    - Un componente de aplicación puede enviar de manera efectiva una petición a otro componente de aplicación, si realiza una llamada a un procedimiento local, lo cual resulta en una petición que se empaca como un mensaje y se envía al componente invocado (remoto).
    - El resultado se enviará de regreso y será devuelto a la aplicación como resultado de la llamada al procedimiento.
  - Invocaciones a métodos remotos (RMI)
    - Es básicamente lo mismo que una RPC, excepto que la RMI opera sobre objetos en lugar de aplicaciones, y por consiguente, en el llamado de sus métodos.

Arquitectura de Software

- *middleware*



Arquitectura de Hardware

Item	S.O. Distribuido		S.O. de Red	Middle-Ware
	Multi-proc.	Multi-comp.		
Grado de Transparencia	Muy Alto	Alto	Bajo	Alto
Mismo S.O.	Sí	Sí	No	No
Copias del S.O.	1	N	N	N
Comunicación	Memoria compartida	Mensajes	Archivos	Específico al modelo
Manejo de recursos	Global, central	Global, distribuido	Por nodo	Por nodo
Escalabilidad	No	Moderado	Si	Variable
Usabilidad	Cerrada	Cerrada	Abierta	Abierta

Capas de Comunicación

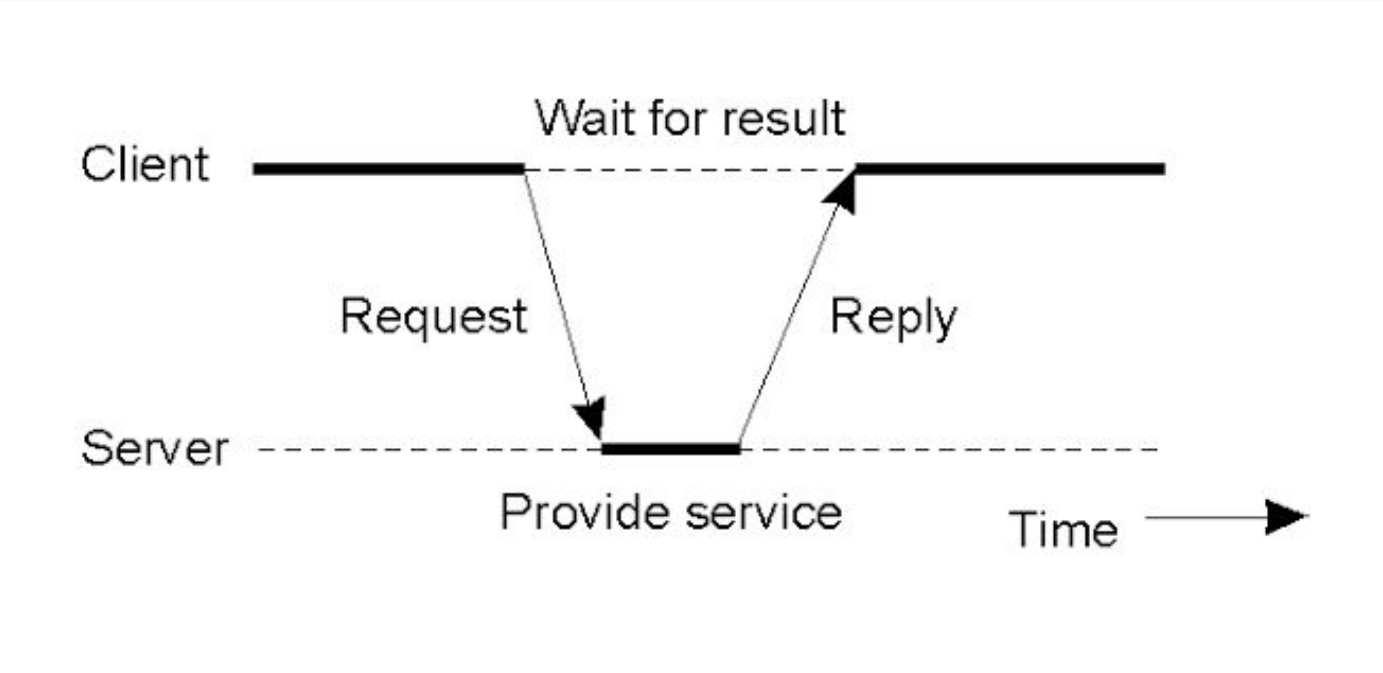
- Un SSDD es posible sólo gracias al aprovechamiento de las **capas superiores** de nuestro modelo de comunicaciones.

Aplicación	Capa Aplicación	Capa Transporte
Email	smtp [RFC 821]	TCP
Remote Terminal Access	telnet [RFC 854]	TCP
WEB	http [RFC 2068]	TCP
Telefonía IP	Propietario (ej. Skype)	TCP o UDP
Streamming multimedia	Propietario (ej. RealMedia)	TCP o UDP

Protocolos de SSDD

Protocolos de red

Esquema Cliente/Servidor



Capas de Comunicación

– Un SSDD es el resultado de aplicar técnicas de coordinación entre máquinas usando sistemas de comunicación básicos provistos por las **capas inferiores**.