Maestría en Ingeniería de Software
Sistemas Distribuidos

RPC (Remote Procedure Call)
Llamada a Procedimiento Remoto

Navarro Guerrero Ma. de los Ángeles
López Martínez María Lina
Hernández González Lizbeth A.

RPC

- (Remote Procedure Call) es un protocolo para solicitar un servicio de un programa localizado en una computadora remota a través de la red, sin importar el tipo de tecnología de red que utilice.

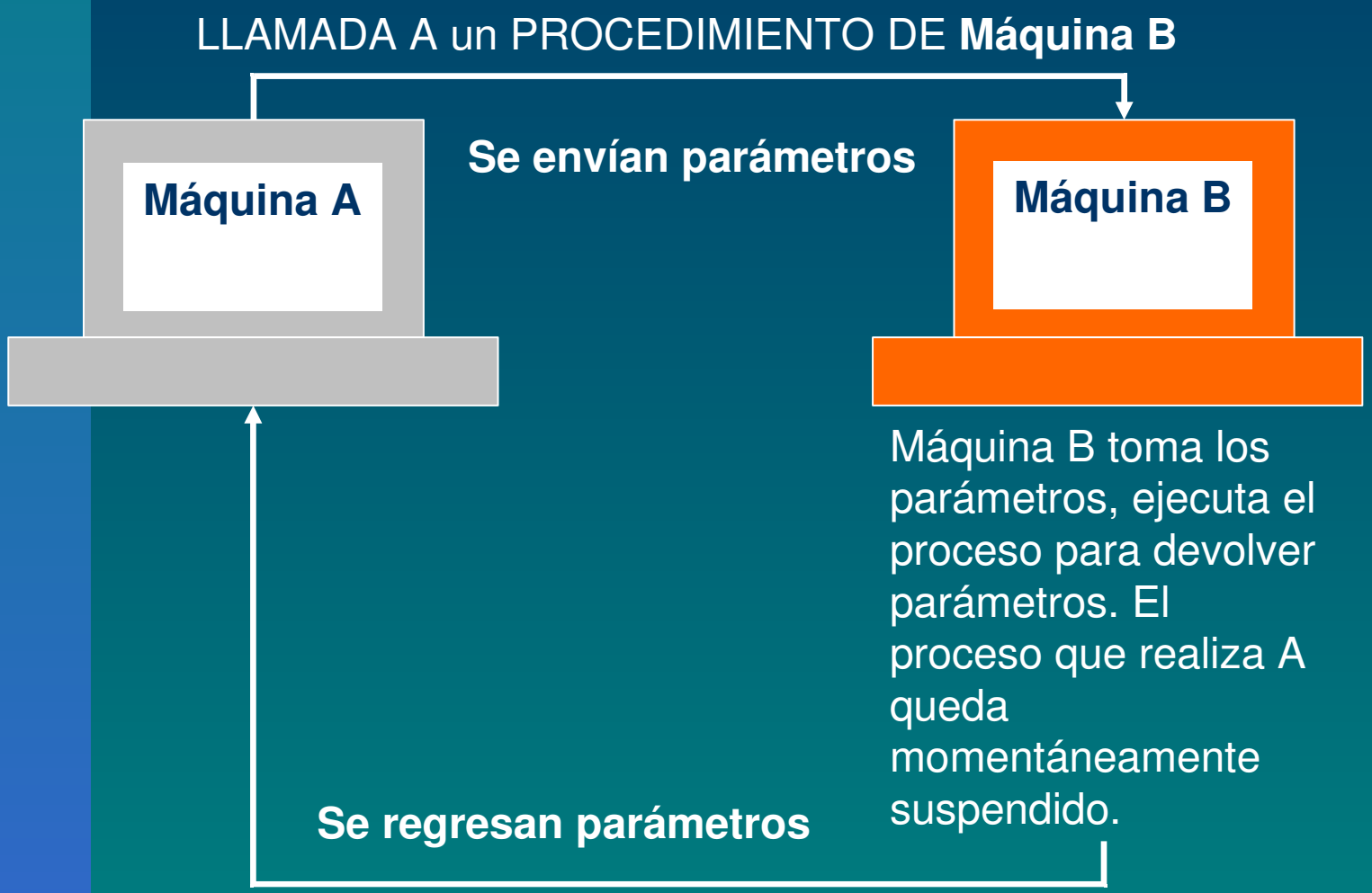
RPC

- Creado por Bireel & Nelson en 1984
- Permiten a los programas llamar procedimientos localizados en otras máquinas
- Un proceso x en una máquina A, puede llamar un procedimiento localizado en una máquina B
- Ningún mensaje u operación de E/S es visible para el programador.
- El protocolo RPC es independiente de los protocolos de transporte
- El protocolo trata solamente con la especificación e interpretación de los mensajes.

RPC

- Usa el modelo cliente/servidor
- Primero el proceso que hace la llamada envía un mensaje que incluye los parámetros del procedimiento al proceso del servidor.
- Después el proceso que llamó espera el mensaje de respuesta.
- Posteriormente del lado del servidor (estaba dormido hasta que llegó la solicitud) extrae los parámetros del procedimiento, realiza los cálculos, obtiene el resultado y envía el mensaje de regreso.
- El cliente recibe el mensaje de respuesta, toma los resultados y prosigue con la ejecución.

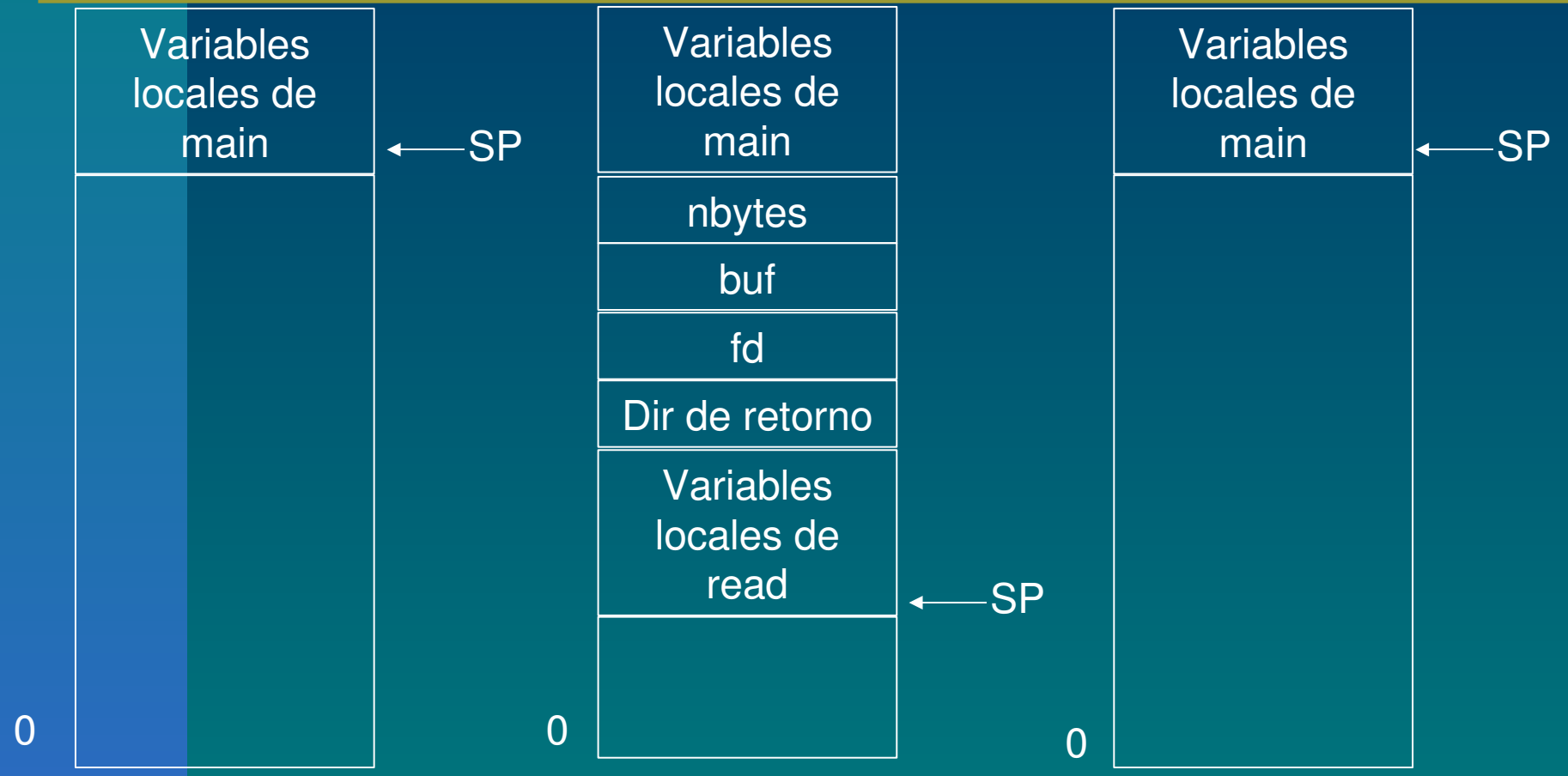
Llamada a procedimientos remotos



Operación básica de RPC

- Comprender el funcionamiento de una llamada convencional.
- La idea detrás de RPC es, que una llamada a un procedimiento remoto parezca lo más posible a una llamada local.
- Ejemplo: count=read(fd, buf, nbytes); donde *fd* es un entero, *buf* es un arreglo de caracteres y *nbytes* es otro entero

Ejemplo:



(a) Pila antes de la llamada de read

(b) Pila mientras el proc. llamado está activo

(c) Pila después del regreso a quien hizo la llamada

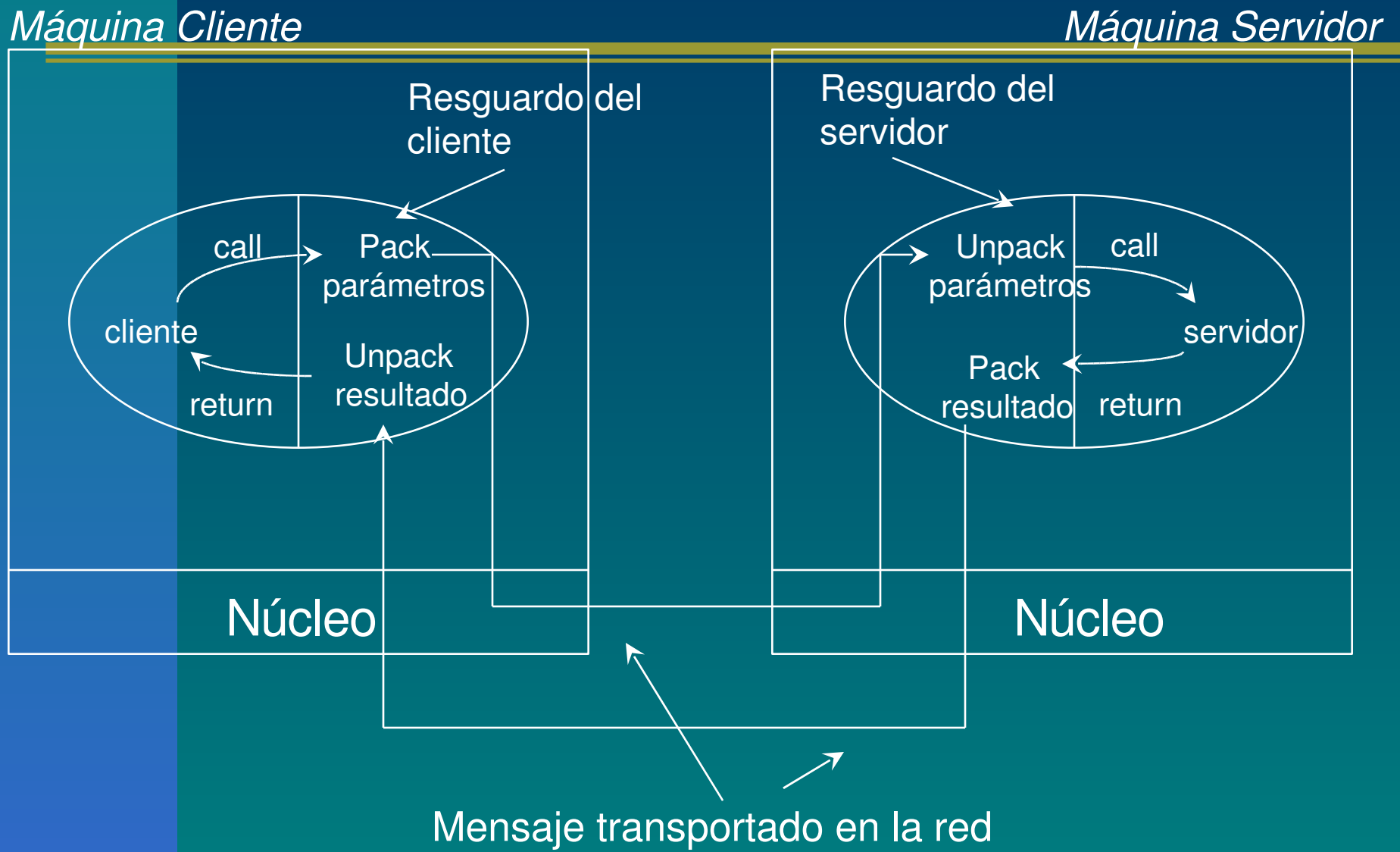
Paso de parámetros

- Por valor
  - en la pila se copia el valor del parámetro
  - valor de salida es igual al valor de entrada
- Por referencia
  - en la pila se almacena la dirección de la variable
  - es posible modificar el valor del parámetro
- Llamada con copia/restauración (No se utiliza en C)
  - Quien recibe la llamada copia la variable en la pila, como en la llamada por valor y entonces la copia después a la llamada, escribiendo sobre el valor original (llamada por referencia).

Resguardos del cliente y del servidor (STUBS)

- Se le llama mediante la secuencia de llamada,
- Se encargan de colocar los parámetros en registros y le pide al núcleo que le envíe el mensaje al servidor llama a send(por el lado del cliente) y llama a receive.
- El resguardo del servidor desempaca el mensaje y llama al procedimiento del servidor, éste lleva a cabo si trabajo y regresa el resultado a quien hizo la llamada quien empaca el resultado en un mensaje y llama a send para regresarlo al cliente.

Llamadas y mensajes en una RPC



Pasos en la ejecución de un RPC

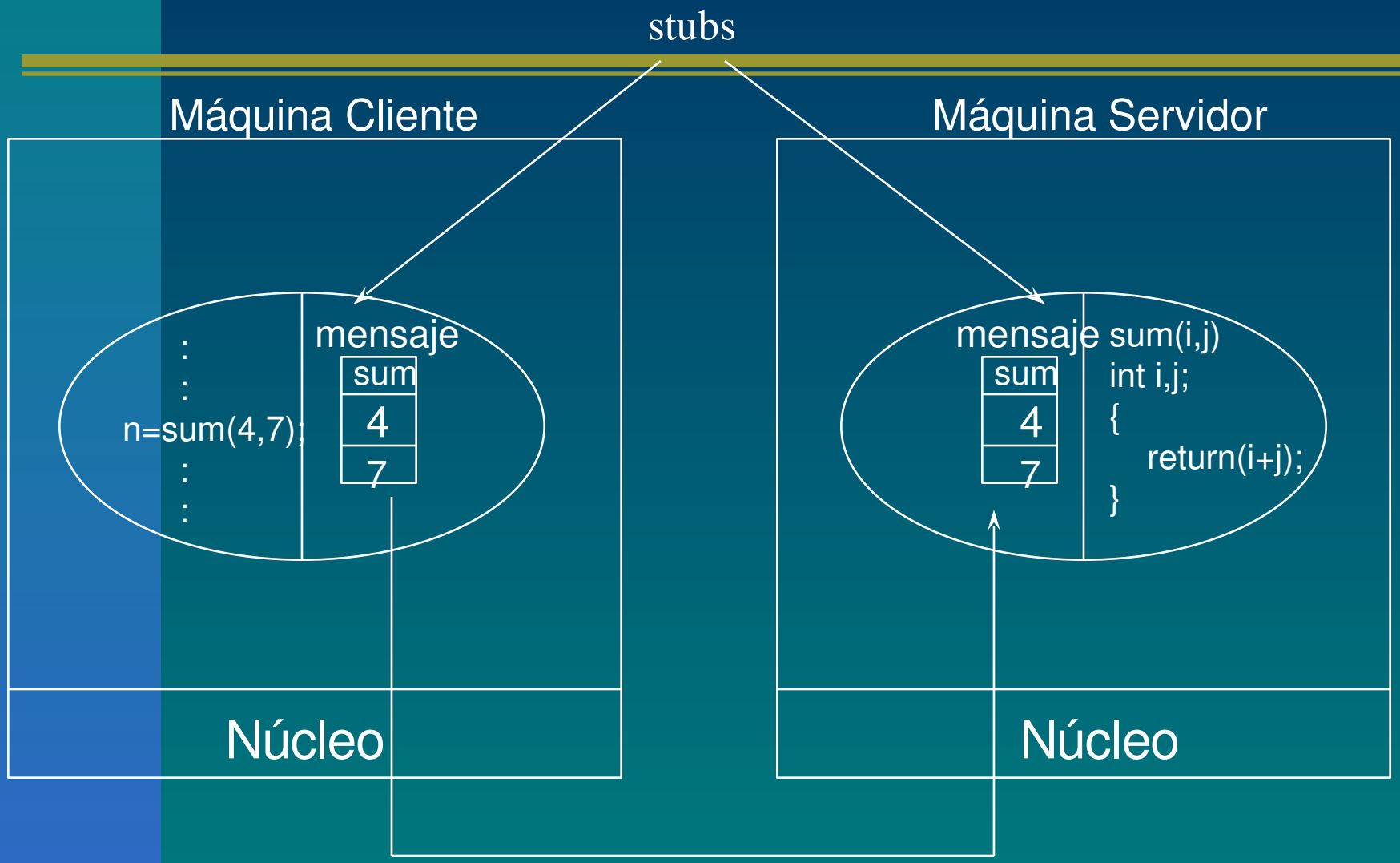
1. El procedimiento del cliente llama al client-stub normalmente.
2. El client-stub construye un mensaje y lo pasa al núcleo.
3. El núcleo envía el mensaje al núcleo remoto.
4. El núcleo remoto pasa el mensaje al *server-stub*.
5. El *server-stub* desempaca los parámetros y llama al servidor.
6. El servidor realiza el trabajo y regresa el resultado al stub.
7. El *server-stub* lo empaqueta en un mensaje y lo pasa al núcleo.
8. El núcleo remoto envía un mensaje al cliente.
9. El núcleo del cliente le da el mensaje al client-stub.
10. El stub desempaca el resultado y lo regresa al cliente.

Transferencia de parámetros

El resguardo del cliente consiste en:

- Tomar sus parámetros, empacarlos en un mensaje y enviarlos al resguardo del servidor, a esto se le conoce como **Ordenamiento de parámetros.**
- El mensaje incluye aparte de los parámetros, el nombre o número del procedimiento por llamar.

El paso de parámetros



Transferencia de parámetros

- Si la máquina cliente y servidor son iguales, los parámetros y los resultados son de tipo escalar (enteros, caracteres, booleanos, etc).
- Es común tener distintos tipos de máquinas con su propia representación de números, caracteres y otros elementos.
- Ejemplo:
  - Mainframes de IBM usan código de caracteres EBCDIC, las personales de IBM utilizan ASCII.
  - Problema peor, máquinas (Intel 486) que numeran sus bytes de derecha a izquierda (little endian) o al revés como Sun SPARC (big endian)

Transferencia de parámetros

- Solución.
  - Los mensajes contienen el Id del procedimiento y los parámetros
  - Así un mensaje con *n* parámetros tendrá *n+1* campos, uno para identificar el procedimiento y uno para cada uno de los *n* parámetros, de tal manera que dada una lista de parámetros sea posible deducir los bytes que pertenecen a cada parámetro, con lo que se resuelve el problema

Transferencia de parámetros

- ¿Cómo se transfieren los apuntadores?
  - Sugerencia, copiar el elemento al que señala el apuntador en el mensaje y enviarlo al servidor.
  - La llamada por referencia se sustituye por la copia/restauración.
  - La optimización para eficientizar este mecanismo consiste en asociar una especificación formal del procedimiento, escrita en cierto lenguaje de especificación que indique la naturaleza de los parámetros, de aquí se generan los resguardos, por medio de un compilador especial de resgtuardos.

Conexión Dinámica (Dynamic Binding)

Conexión dinámica (binding)

- Antecedentes
  - No se ha tomado en cuenta la forma en la que el cliente localiza el servidor
  - Un método consiste en integrar dentro del código del cliente la dirección (en la red) del servidor , solo que este método es muy rígido, si el servidor se desplaza, se duplica o si cambia la interfaz, hay que localizar y volver a compilar programas.

Conexión dinámica (binding)

- Solución a problemas anteriores ya que su punto de partida es la especificación formal del servidor.
- Como especificación debemos entender que se indica el nombre del servidor (file_server), el número de versión (3.1) y una lista de procedimientos que proporciona (read, write, create y delete)

Conexión dinámica (binding)

- Para que concuerden los clientes y los servidores se implementa la **conexión dinámica**.
- Su punto de inicio es la *especificación formal del servidor*, que indica el nombre del servidor, el número de versión y una lista de los procedimientos que proporciona.

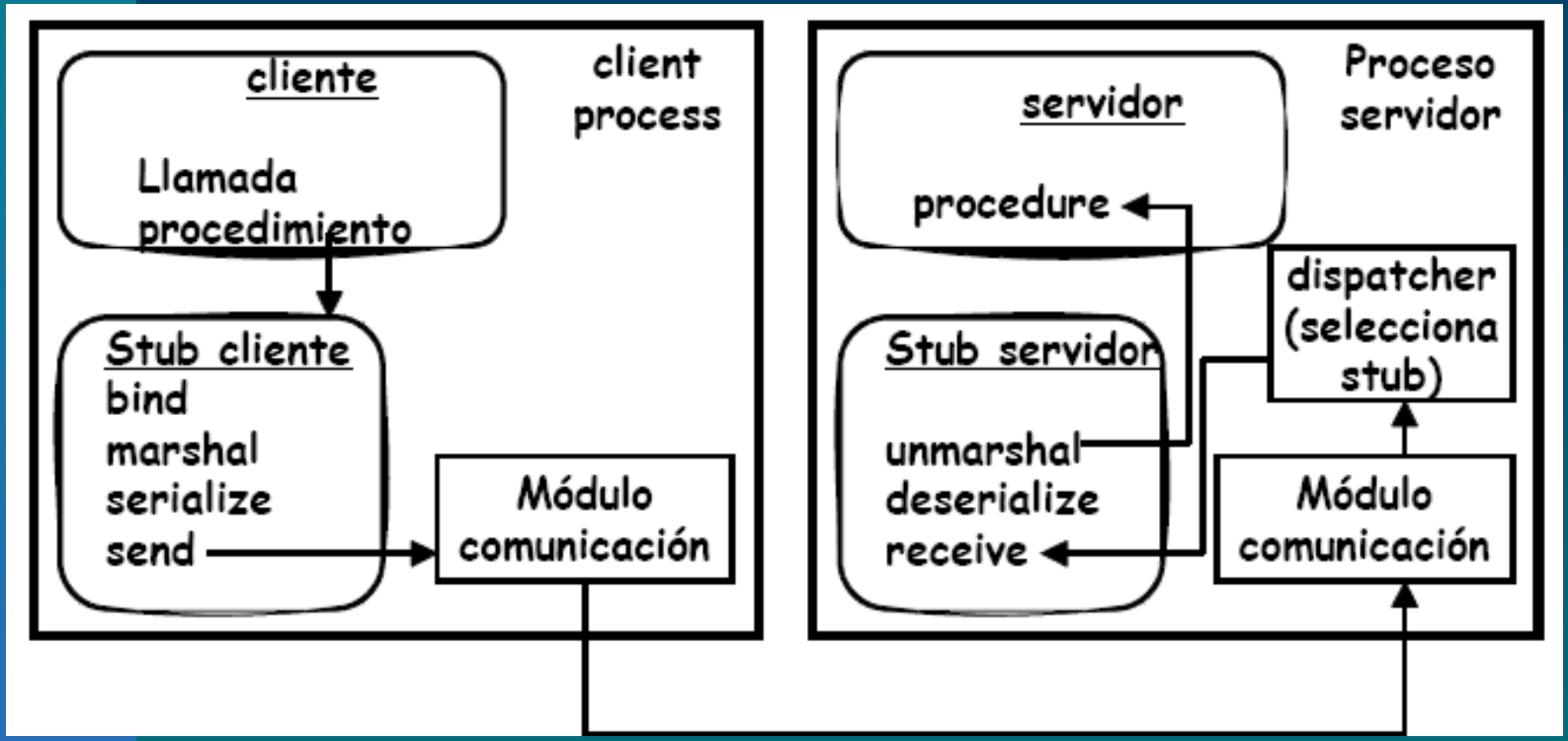
Conexión dinámica (binding)

- Se tienen los *tipos de parámetros* para cada procedimiento y cada parámetro queda determinado como parámetro *in, out* o *in-out.*
- La dirección es relativa al servidor.
- El principal uso de la especificación formal es como entrada del *generador de resguardos:*
- Produce el resguardo del cliente y el del servidor.
- Ambos resguardos se colocan en las bibliotecas respectivas.
- Cuando un *programa (cliente)* llama a cualquiera de los procedimientos definidos mediante esa especificación, el correspondiente *procedimiento resguardo del cliente* se liga con su binario.
- Si se compila un *programa servidor*, los *resguardos del servidor* se le ligan también.

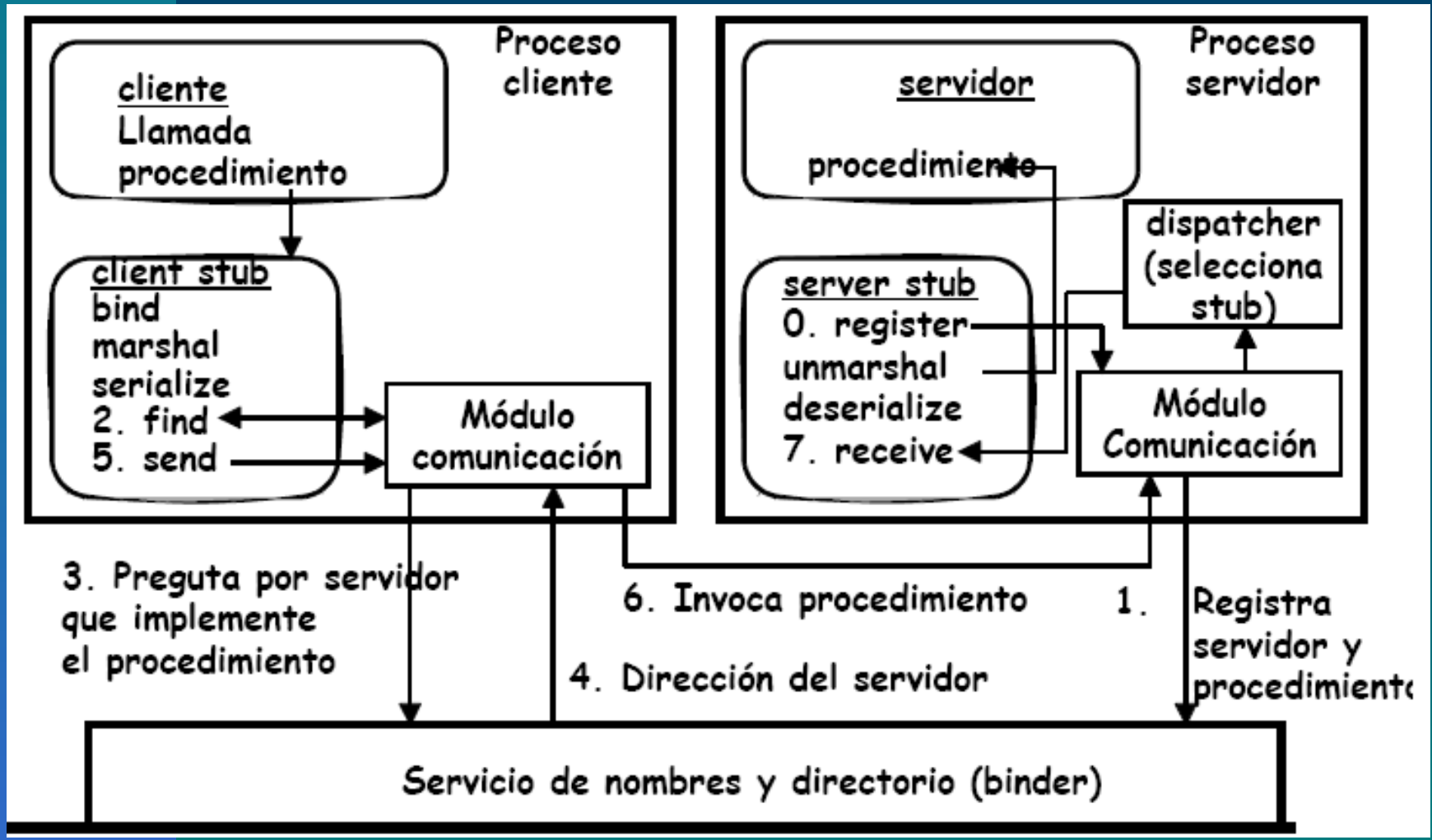
Conexión dinámica (binding)

- Para un servidor del tipo de UNIX, se tendrían otros procedimientos, *open* y *close*, y distintos parámetros de *read* y *write*. El concepto de RPC es en sí mismo neutral y permite a los diseñadores de sistemas construir cualquier tipo deseado de servidores.
- Cuando el servidor inicia su ejecución, la llamada a *initialize* que se encuentre fuera del ciclo principal exporta la interfaz del servidor. Esto quiere decir que envía un mensaje a un programa llamado conector para darle a conocer su existencia. Este proceso se le conoce como registro del servidor.

Binding en RPC



Conexión dinámica (binding)



Conexión dinámica (binding)

- Para registrarse, el servidor proporciona al conector su nombre, número de versión, un identificador, que por lo general tiene una longitud de 32 bits y un asa que se utiliza para localizarlo. El asa depende del sistema y puede ser una dirección ethernet, una dirección IP, una dirección X.500, un identificador ralo de procesos o alguna otra cosa. Ademas, se puede proporcionar otra información, como por ejemplo, la relativa a la autentificación. Un servidor también puede canceclar su registro con el conector, si ya no está preparado para prestar algún servicio. La interfaz del conector se muestra en la sig. Figura.

Conexión Dinámica (Binding dinámico)

Epecificación del servidor sin estado

```
#include <header.h>
specification of file_server, version 3.1
    long read(in char name[MAX_PATH], out char buf[BUF_SIZE],
              in long bytes, in long position);

    long write(in char name[MAX_PATH], in char buf[BUF_SIZE],
               in long bytes, in long position);

    int create(in char[MAX_PATH], in int mode);

   int delete(in char[MAX_PATH]);
end.
```

Conexión dinámica (binding)

Interfaz del conector

Llamada	Parámetros Entrada	Salida
Registro	Nombre, versión, asa, id única	
De registro	Nombre, versión, id única	
Búsqueda	Nombre, versión	Asa, id único

Conexión Binding (Método de importación y exportación)

Ventajas:

- El método de importación y exportación de interfaces es altamente flexible.
- Puede manejar varios servidores que soporten la misma interfaz
- El conector puede difundir los clientes de manera aleatoria entre los servidores.
- Cancela el registro del servidor que no responda.
- Ayuda a la autentificación

Conexión Binding (Método de importación y exportación)

Desventajas:

- Existe un costo adicional en el tiempo generado por la exportación e importación de interfaces..
- El conector se puede convertir en un cuello de botella en sistemas grandes, se necesitarían varios conectores.
- Se necesita un gran número de mensajes para mantener sincronizados y actualizados todos los conectores, mas gasto.
- Ayuda a la autentificación

Semántica RPC en presencia de fallas

Problemas:

- El cliente no puede localizar el servidor.
- Se pierde el mensaje de solicitud del cliente al servidor.
- Se pierde el mensaje de respuesta del servidor al cliente.
- El servidor falla antes de recibir una solicitud.
- El cliente falla después de enviar una solicitud.

Semántica RPC en presencia de fallas

Problemas: El cliente no puede localizar el servidor.

- El servidor podría estar inactivo.
- El servidor podría estar utilizando una nueva versión de la interfaz y nuevos resguardos, que no serían compatibles con la interfaz y los resguardos del cliente.
- En el servidor, cada uno de los procedimientos regresa un valor:
- Generalmente el código -1 indica un fallo.
- También se suele utilizar una variable global (UNIX) *errno* a la que se asigna un valor que indica el tipo de error.
- Un tipo de error sería “no se pudo localizar al servidor”.
- Otra posibilidad para el tratamiento de los errores es mediante una *excepción* provocada por el error:
- Se codifican procedimientos especiales que son llamados ante errores específicos.
- El problema es que se puede destruir la transparencia deseada, ya que se dificulta mantener la similitud entre procedimientos locales y remotos.

Semántica RPC en presencia de fallas

Problemas:

- Pérdida de mensajes de solicitud.
  - Más fácil de tratar, lo que se debe hacer es que el núcleo inicie un cronómetro al enviar la solicitud, si en tiempo termina antes que regrese una respuesta o reconocimiento el núcleo vuelve a enviar el mensaje.

Semántica RPC en presencia de fallas

Problemas:

- Pérdida de mensajes de respuesta.
  - Es mas difícil de enfrentar.
  - Existen muchas preguntas: ¿Se perdió la solicitud? ¿Se perdió la respuesta? ¿Ocurre tan solo que el servidor es lento?
  - Ciertas operaciones se pueden repetir sin que ocurra algún daño.
  - Una solicitud de los primero 1024 bytes se puede ejecutar n veces sin causar daño alguno (solicitud idempotente).

Semántica RPC en presencia de fallas

Problemas:

- Fallas del servidor
  - Se relaciona con la idempotencia
  - Existen 3 escuelas de respuesta a este caso:
    - Semántica al menos una vez. Esperar a que el servidor vuelva a arrancar.
    - Semántica a lo más una vez, la RPC se realiza a lo mas una vez o ninguna.
    - No garantizar nada, fácil de implantar.
  - Ideal: Semántica de exactamente una.

Semántica RPC en presencia de fallas

Problemas:

- Fallas del cliente
  - Si un cliente envía una solicitud a un servidor y falla antes de que éste le responda, queda activa una labor de cómputo y ningún padre espera el resultado, esta labor se llama **Huérfano**.

Aspectos de la implantación

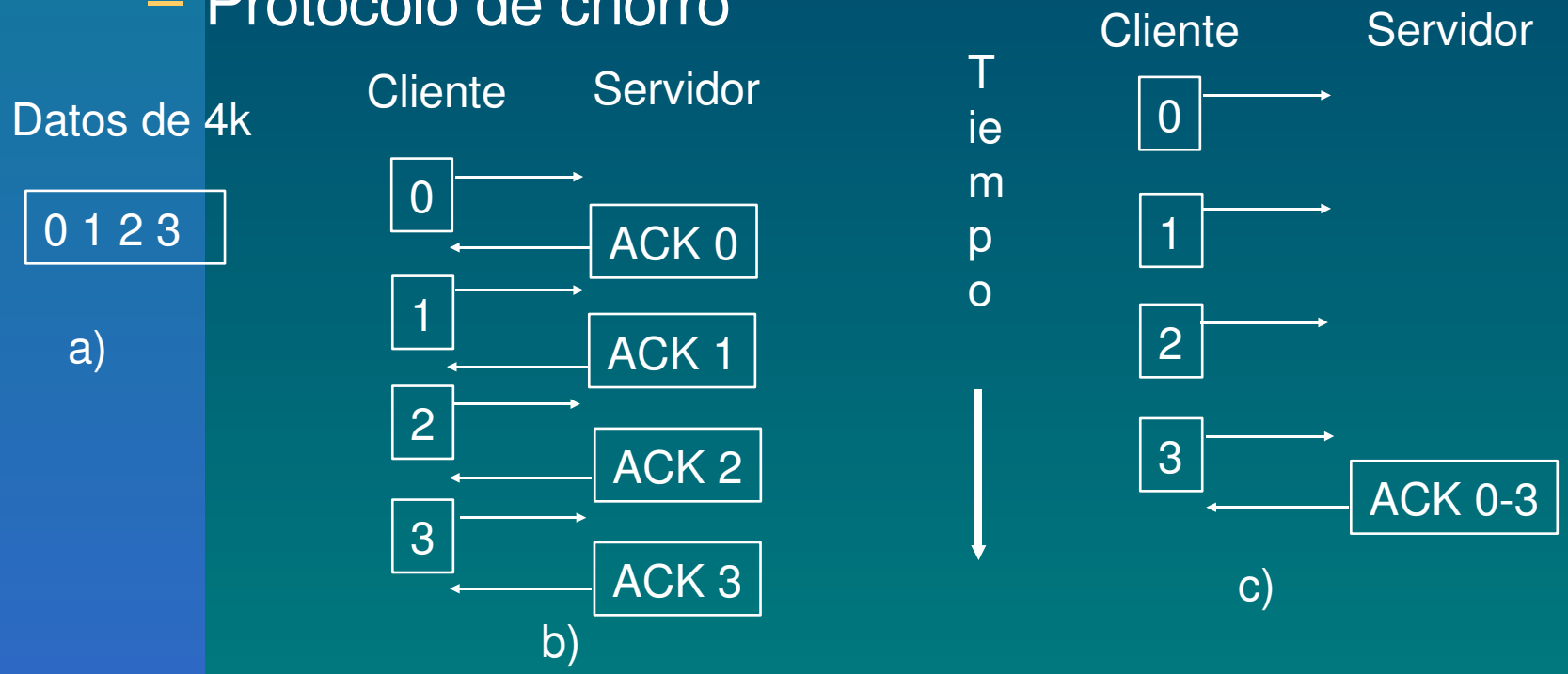
- Protocolos RPC
  - Primer aspecto: elegir el protocolo RPC
  - 2° utilizar un protocolo estándar de propósito general.

Aspectos de la implantación

- Tipo de conexión
  - orientado conexión
  - orientado no conexión
- Tipo de protocolo
  - definido por el usuario
  - uso de uno ya existente
- Manejo de acknowledges
  - protocolo blast
  - protocolo stop-and-wait

Aspectos de la implantación

- Reconocimientos.
  - Protocolo detenerse y esperar
  - Protocolo de chorro



a) Un mensaje de 4k. B) Un protocolo detenerse y esperar. C) Un protocolo de chorro

Ventajas/desventajas uso protocolos no conocidos

- Protocolos adaptados a las verdaderas necesidades del sistema
- Posible mejoramiento del desempeño
- Menor número de mensajes
- Mensajes más pequeños
- Complejidad en el diseño
- Difícil el depuramiento de problemas

Ventajas elección protocolo conocido (p.e. TCP/IP)

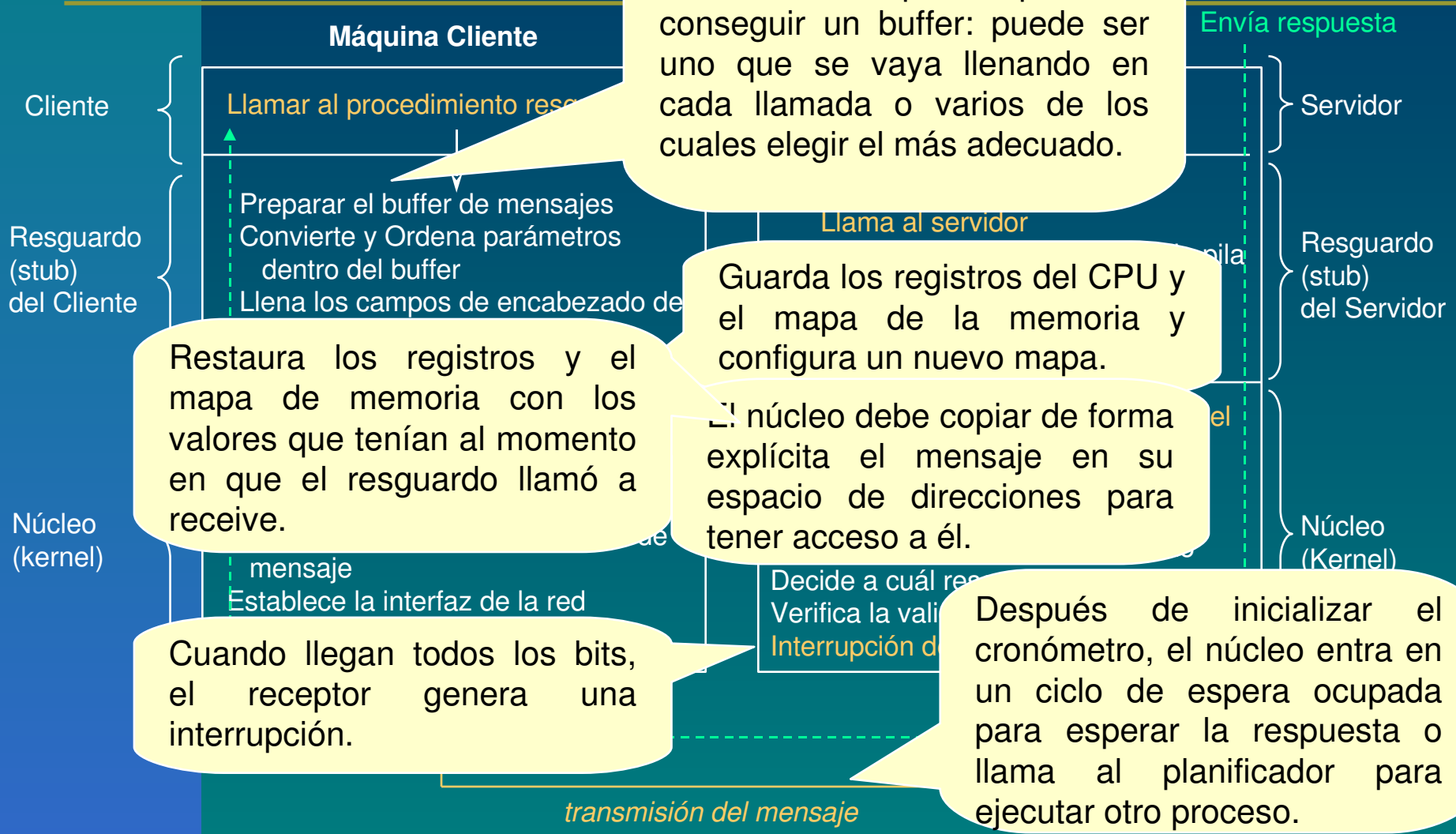
- 1. Protocolo ya fue diseñado, ahorrando trabajo.
- 2. Muchas implementaciones están disponibles.
- 3. Paquetes pueden enviarse y recibirse por casi todos los sistemas Unix.
- 4. Los paquetes IP y UDP son soportados por muchas redes existentes.

Ruta crítica

Ruta crítica

- La serie de instrucciones que se ejecutan con cada RPC se llama la ***ruta crítica.***
- Analizaremos más de cerca lo que ocurre en realidad cuando un cliente lleva a cabo una RPC con un servidor remoto.

Ruta crítica del cliente al servidor



Después de llamar al resguardo del cliente, el primer paso es conseguir un buffer: puede ser uno que se vaya llenando en cada llamada o varios de los cuales elegir el más adecuado.

Guarda los registros del CPU y el mapa de la memoria y configura un nuevo mapa.

Restaura los registros y el mapa de memoria con los valores que tenían al momento en que el resguardo llamó a receive.

El núcleo debe copiar de forma explícita el mensaje en su espacio de direcciones para tener acceso a él.

Cuando llegan todos los bits, el receptor genera una interrupción.

Después de inicializar el cronómetro, el núcleo entra en un ciclo de espera ocupada para esperar la respuesta o llama al planificador para ejecutar otro proceso.

Examinemos los pasos Ruta crítica del cliente

Preparar el buffer de mensajes:

- Después de llamar al resguardo del cliente, el primer paso es conseguir un buffer: puede ser uno que se vaya llenando en cada llamada o varios de los cuales elegir el más adecuado.

Cambio de contexto al núcleo:

- Guarda los registros del CPU y el mapa de la memoria y configura un nuevo mapa.

El núcleo debe copiar de forma explícita el mensaje en su espacio de direcciones para tener acceso a él.

Transmisión del mensaje

- Después de inicializar el cronómetro, el núcleo entra en un ciclo de espera ocupada para esperar la respuesta o llama al planificador para ejecutar otro proceso.
- El primero es más rápido pero no permite la multiprogramación.

Ruta crítica del servidor

- Cuando llegan todos los bits, el receptor genera una interrupción.
- Cuando hace un cambio de contexto, restaura los registros y el mapa de memoria con los valores que tenían al momento en que el resguardo llamó a *receive*.

¿En qué parte de la ruta crítica se ocupa la mayoría del tiempo?

- Una vez que esto se conoce se comienza el trabajo de mejorar la velocidad.

- Caso de estudio de Schroeder y Burrows
  - Resultados de una RPC nula (sin datos).
  - Resultados para un parámetro tipo arreglo de 1440 bytes.

Resultados

- El costo fijo es el mismo.
- Se necesita mucho más tiempo para el ordenamiento de los parámetros y el desplazamiento de un lado a otro de los datos en el segundo caso.
- Costos dominantes en el caso de una RPC nula:
  - Cambio de contexto al resguardo del servidor
  - La rutina de servicio de interrupciones
  - Movimiento del paquete a la interfaz de la red para su transmisión.

Resultados

- Para la RPC de 1440 bytes:
  - Es mayor el tiempo de transmisión en Ethernet (mucho mayor).
  - En segundo lugar se encuentra el tiempo que tarda el desplazamiento del paquete hacia adentro y afuera de la interfaz.

Copiado

- El número de veces que se debe copiar un mensaje varía de uno a ocho, según el hardware, software y tipo de llamada.
- El mejor de los casos:
  - El chip de la red utiliza el DMA (Data Memory Access) para transferir del espacio de direcciones del resguardo del cliente a la red (copia 1).
  - Lo deposita en la memoria del núcleo del servidor.
  - El núcleo inspecciona el paquete y asocia la página que la contiene en el espacio de direcciones del servidor.
  - Si esto no se lleva a cabo, el núcleo copia el paquete al resguardo del servidor (copia 2).

- El peor de los casos:
  - El núcleo copia el mensaje del resguardo del cliente en un buffer del núcleo para su posterior transmisión (por algún inconveniente con la red).
  - El núcleo copia el mensaje, en software, a un buffer de hardware en la tarjeta de interfaz de la red (copia 2).
  - Se inicializa el hardware y el paquete se desplaza a través de la red hacia la tarjeta de la máquina destino (copia 3).
  - Cuando se hace la interrupción, el núcleo lo copia a un buffer del núcleo (copia 4).
  - El mensaje debe ser copiado al resguardo del servidor (copia 5).
- Si se transfiere un parámetro de gran tamaño, se pueden producir otras 3 copias, teniendo un total de 8.

Dispersión-Asociación (scatter-gather)

- Es una característica de hardware que es de gran ayuda para eliminar el copiado innecesario.
- Se puede configurar un chip de la red de tal manera que organice un paquete mediante la concatenación de uno o dos buffers de memoria.
- El hecho de poder asociar un paquete a partir de varias fuentes elimina el copiado.
- También el hecho de dispersar el encabezado y cuerpo de un paquete recibido en distintos buffers es de gran ayuda.

Manejo del cronómetro

- La mayoría de los protocolos inicializan un cronómetro cada vez que se envíe un mensaje y se espere una respuesta (réplica o reconocimiento).
- ***Si la respuesta no llega*** en el tiempo esperado → ***el contador se detiene*** y se vuelve a transmitir el mensaje original.
- Este proceso se repite hasta que el emisor se cansa y se da por vencido.

- El cronómetro requiere de la construcción de una ***estructura de datos*** que especifique el momento en que el cronómetro debe detenerse y la acción a realizar (habrá una **lista de cronómetros pendientes**).
- El valor del tiempo de expiración es una aproximación burda que puede afectar el desempeño del cronómetro.

Método sugerido

- Usar la tabla de procesos, y agregar un campo de tiempo de expiración, en lugar de almacenar los tiempos de expiración en una lista ligada ordenada.
- La activación del cronómetro consta ahora de la suma de la longitud de su tiempo de expiración al tiempo actual y su almacenamiento en la tabla de procesos.
- La desactivación de un cronómetro sería el almacenamiento del valor 0 en el campo.
- Los algoritmos que operan mediante una transferencia secuencial periódica por medio de una tabla se llaman algoritmos de barrido (**sweep algorithms**).

Desventajas modelo cliente/servidor

- Paradigma del tipo Entrada/Salida
  - procedimientos *send(), receive()* están dedicados a realizar E/S
- E/S no es un concepto clave para los sistemas centralizados; pero si para la computación o cálculo distribuido.
- El objetivo es hacer el cálculo distribuido como si fuera cálculo centralizado.
- Cálculo centralizado: llamadas de procedimientos y/o funciones