CHAMADA REMOTA DE PROCEDIMENTO

(RPC - Remote Procedure Call)

Problema do Modelo Cliente-Servidor - O paradigma básico que envolve toda comunicação é construído como Entrada e Saída --> perda da transparência.

Idéia Básica da *RPC* -- Programas (ou processos) podem chamar procedimentos localizados em outras máquinas.

RPC - Remote Procedure Call

HISTÓRICO :

- HOARE, C.A.R. - ***Communicating Sequential Processes***. Com. ACM, 21(8), Agosto 1978
- BRINCH HANSEN, P. - ***Distributed Processes: A Concurrent Programming Concept.*** Com. ACM, 21(11), Nov. 1978.

Pouco foi feito no assunto até que:

- BIRRELL, A. D., e NELSON, B.J. - ***Implementing Remote Procedure Calls***. ACM Trans. on Computer Systems, vol. 2, pp. 39-59, Fev. 1984.

introduziram um modo completamente diferente de abordar o problema.

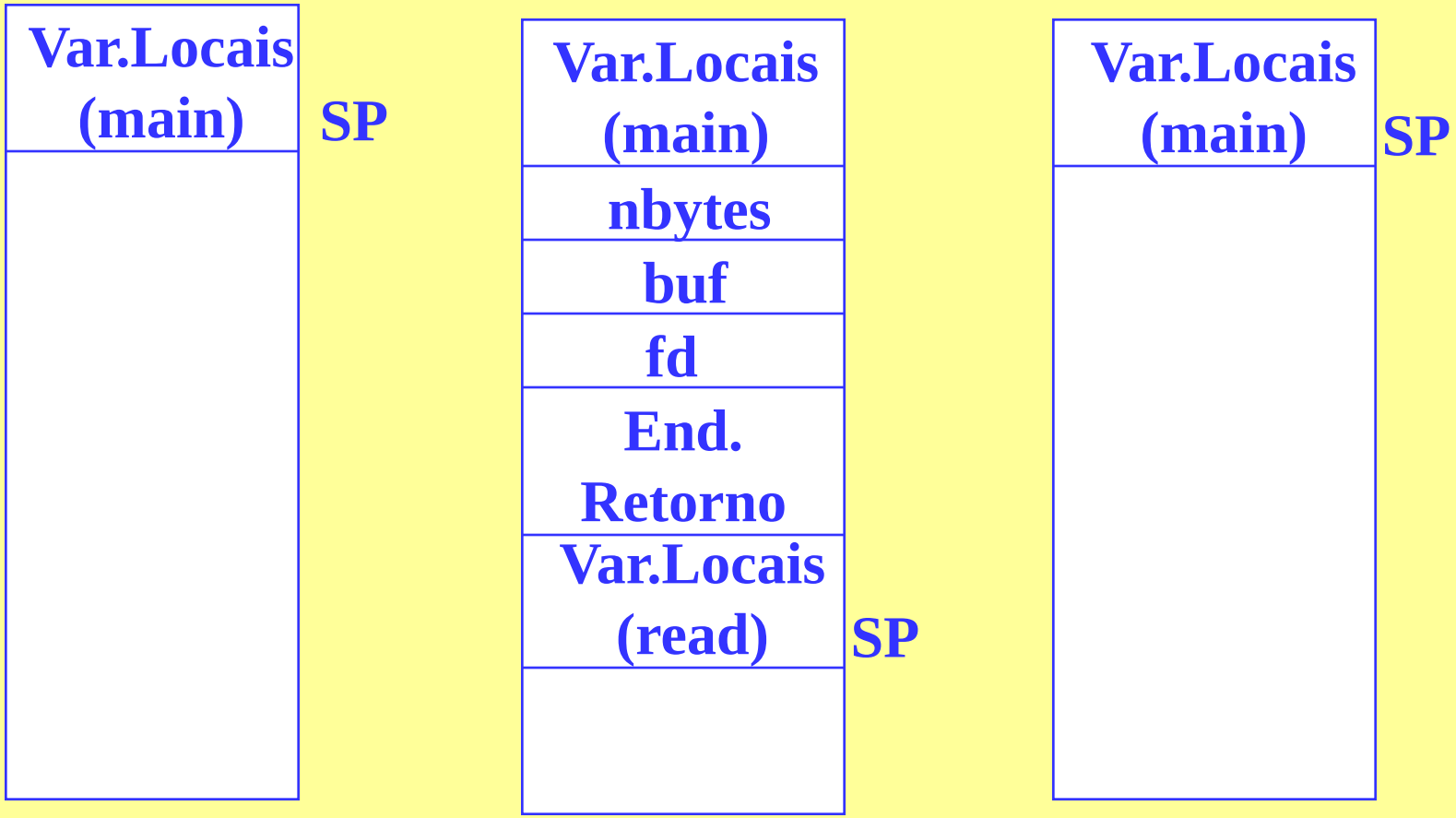
RPC - Remote Procedure Call

CARACTERÍSTICAS:

- Idéia simples e elegante, fazer a chamada remota se parecer o máximo possível com a chamada local.
- Problemas: quando as máquinas são diferentes (espaço de endereçamento, parâmetros) e quando há falha em uma delas.

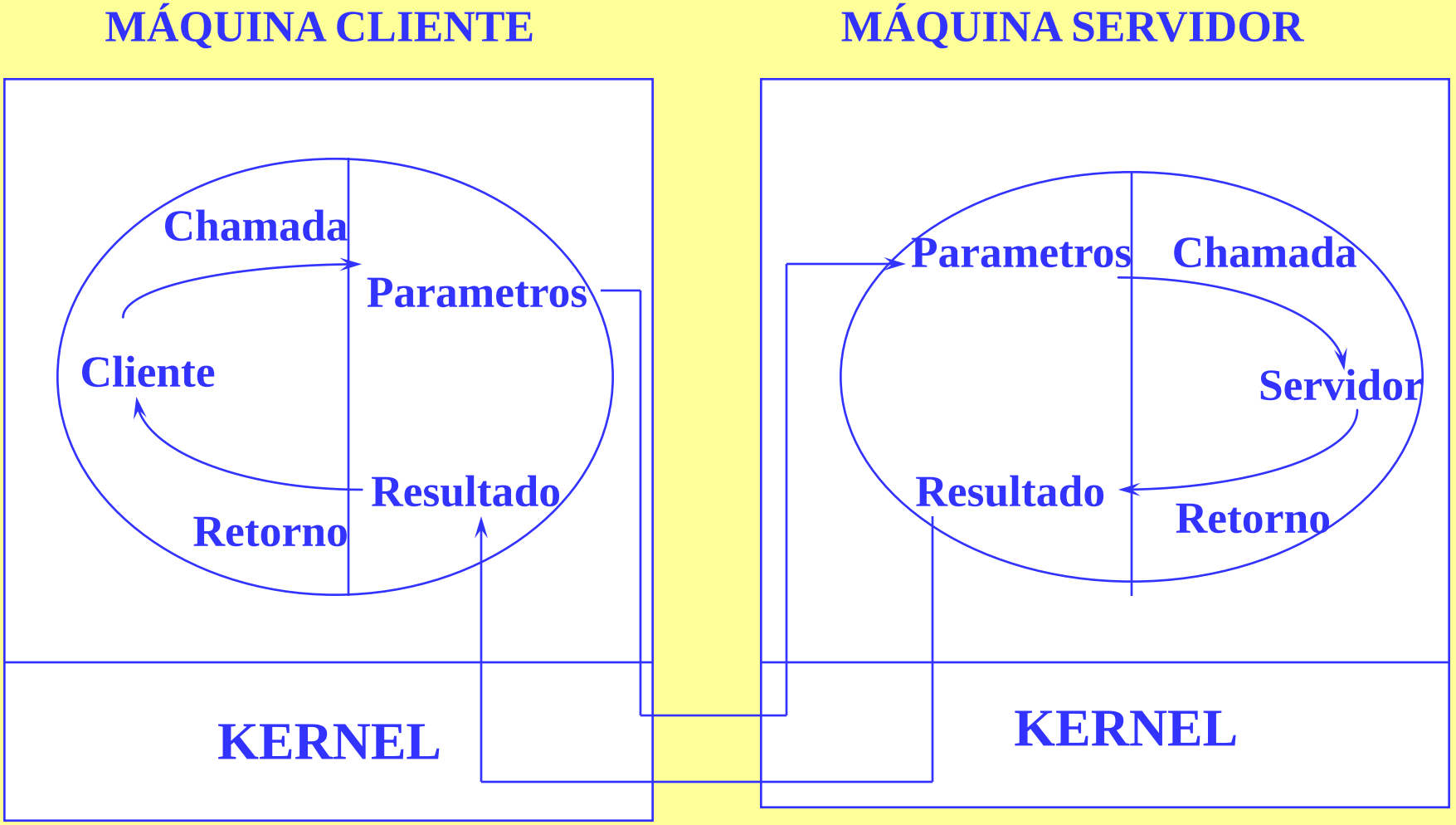
CHAMADA LOCAL

Exemplo: ***count = read (fd, buf, nbytes);***



IDÉIA BÁSICA DE CHAMADA REMOTA

IMPLEMENTAÇÃO DA CHAMADA REMOTA



- Quando a chamada é remota (em vez de local) uma versão diferente do procedimento chamado é usada (client stub). Em vez de colocar os parâmetros nos registros como na chamada local, a chamada remota pede ao kernel que envie uma mensagem com os parâmetros para o servidor. O processo cliente fica bloqueado até receber a resposta com o resultado da chamada remota.
- Do lado do servidor, quando a mensagem chega com o pedido de execução remota, transfere o pedido para uma versão diferente do servidor (server stub). Os parâmetros são disponibilizados e o procedimento servidor é chamado no modo usual (chamada local). Quando o servidor remoto assume o controle novamente depois da chamada ter completado, os resultados são enviados para o cliente.

- Quando a mensagem de resposta chega ao cliente, o kernel identifica o endereço como sendo o do processo cliente. A mensagem é copiada no buffer e o cliente desbloqueado. A versão remota do cliente copia o resultado para o cliente local e retorna ao funcionamento usual. Quando o cliente local ganha o controle ele sabe que o dado está disponível, mas não tem idéia de onde ele foi processado.

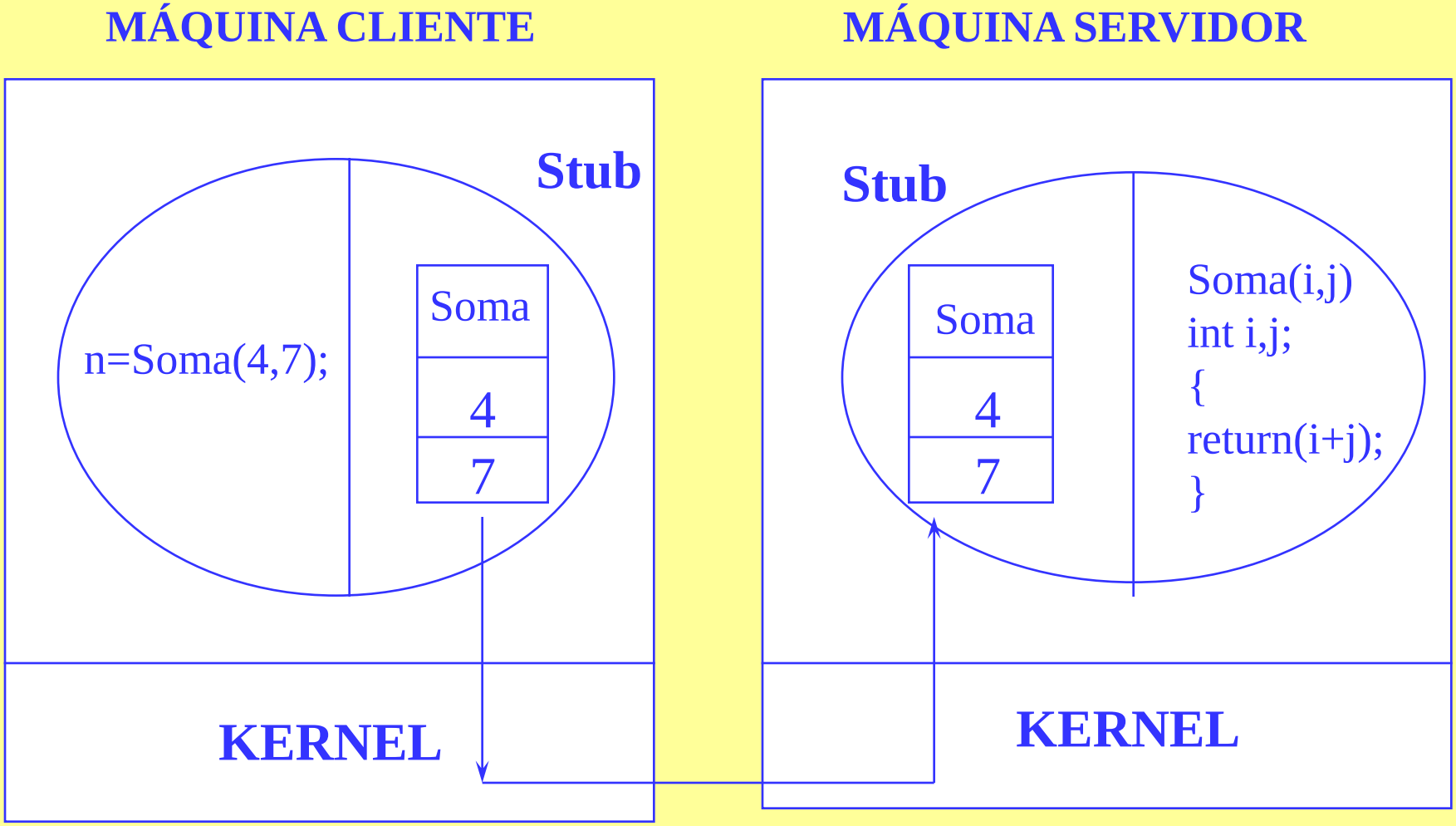
Passagem de Parâmetros

Tem três modos de se passar parâmetros em um procedimento:

- Passagem por Valor;
- Passagem por Referência;
- Passagem por Cópia/Restaura.

Função do Cliente “stub” -- Pegar os parâmetros, colocá-los na mensagem e enviar ao Servidor “stub”.

Exemplo: Soma (i, j); i e j inteiros



Passagem de Parâmetros (...Cont.)

Problema: Em um grande Sistema Distribuído há normalmente múltiplos tipos de máquinas.

Exemplo: IBM mainframes - EBCDIC

IBM PCs - ASCII

Intel 486 - numera os bytes de um inteiro da direita para a esquerda

Sun SPARC - na ordem inversa

0 3	0 2	0 1	5 0
L 7	L 6	I 5	J 4

(a)

5 0	0 1	0 2	0 3
J 4	I 5	L 6	L 7

(b)

0 0	0 1	0 2	5 3
L 4	L 5	I 6	J 7

(c)

Passagem de Parâmetros (...Cont.)

A simples inversão dos bytes de cada palavra depois de recebido não produz o resultado correto (strings não são colocadas na forma reversa).

Solução: Tanto o Cliente como o Servidor conhecem o tipo dos parâmetros, e pelo tipo é possível saber os que devem ser revertidos.

Passagem de Parâmetros (...Cont.)

Outra Solução: Criar um padrão de rede definindo o formato dos inteiros, caracteres, ponto-flutuante, etc. Requerer que os dados sejam convertidos para este padrão antes de serem enviados.

Problema: Ineficiência - máquinas com a mesma representação farão duas conversões quando na realidade não é preciso fazer nenhuma.

Outra Solução: O primeiro byte da mensagem indica qual é o formato usado (a conversão é realizada somente quando os formatos forem diferentes)

Passagem de Parâmetros (...Cont.)

Geração dos "Stubs" -- geralmente os procedimentos "stubs" são gerados automaticamente, tendo os dois "stubs" gerados de uma única especificação formal do servidor facilita a vida do programador, reduz a possibilidade de erros e torna o sistema transparente.

Ponteiros:

Solução 1 -- Proibir o uso de ponteiros e passagem de parâmetros por referência.

Passagem de Parâmetros (...Cont.)

Solução 2 -- Copiar o arranjo apontado pelo ponteiro na mensagem e enviá-lo para o servidor. O servidor "stub" pode então chamar o servidor com um ponteiro para este arranjo. Mudanças que o servidor faz nas posições apontadas pelo ponteiro afetam o buffer de mensagem do servidor "stub". Quando o servidor termina, o arranjo é enviado de volta para o Cliente "stub", que o copia de volta para o Cliente.

Efeito: A chamada por referência foi substituida pela chamada copia/restaura.

Otimização: Se os "stubs" sabem se o buffer é um parâmetro de entrada ou um parâmetro de saída, uma das cópias pode ser eliminada.

”Binding” Dinâmico

Alguns SDs usam o “Binding” dinâmico para o Cliente localizar o Servidor.

O “Binding” dinâmico utiliza uma especificação formal dos servidores. A especificação é composta pelo nome do servidor, versão e lista de procedimentos oferecidos pelo servidor.

A especificação formal é utilizada como entrada do gerador de “stubs”. Eles são colocados dentro das bibliotecas apropriadas. Quando um programa do usuário (Cliente) chama algum destes procedimentos, o cliente “stub” correspondente é linkado com o seu binário.

Especificação de um Servidor

```
#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;
```

”Binding” Dinâmico (...Cont.)

Quando o servidor começa executar, ele chama “inicializa” que exporta a interface do servidor. Isto é, envia uma mensagem para o programa “binder”, para se tornar conhecido. Este processo é chamado de registro do servidor.

O servidor pode ser “desregistrado” com o “binder” quando não pode mais oferecer o serviço.

Interface do “Binder”

Chamada	**Entrada**	**Saída**
Registro	Nome, versão, “handle”, id	
Desregistro	Nome, versão, id	
Consulta	Nome, versão	“handle”, id

”Binding” Dinâmico (...Cont.)

Como o Cliente localiza o Servidor?

Quando o cliente faz uma chamada remota pela primeira vez, o Cliente “stub” vê que o serviço não está ligado a um servidor. Assim sendo, ele envia uma mensagem para o “binder” pedindo para importar a versão 3.1 do file_server. O “binder” verifica se algum servidor exportou uma interface com este nome e versão. Em caso negativo, a chamada remota irá falhar.

”Binding” Dinâmico (...Cont.)

Método de importar e exportar interfaces é altamente flexível:

- Múltiplos servidores com a mesma interface;
- Servidores que falham em responder são “desregistrados”;
- Desvantagem de “overhead” extra;
- Grandes SDs necessitam de múltiplos “binders”.

Presença de Falhas na RPC

1- **Cliente não pode localizar o Servidor**

Causas -- Servidor desligado, versão desatualizada

Soluções -- Retornar código indicando erro, ativar excessão (escrever um procedimento de tratamento de excessão destrói a transparência do sistema)

2- **Mensagem de Requisição Perdida**

Inicializar um timer quando a requisição é enviada, se o timer expirar a requisição é enviada novamente.

Presença de Falhas na RPC (...Cont.)

3- **Mensagem de Resposta Perdida**

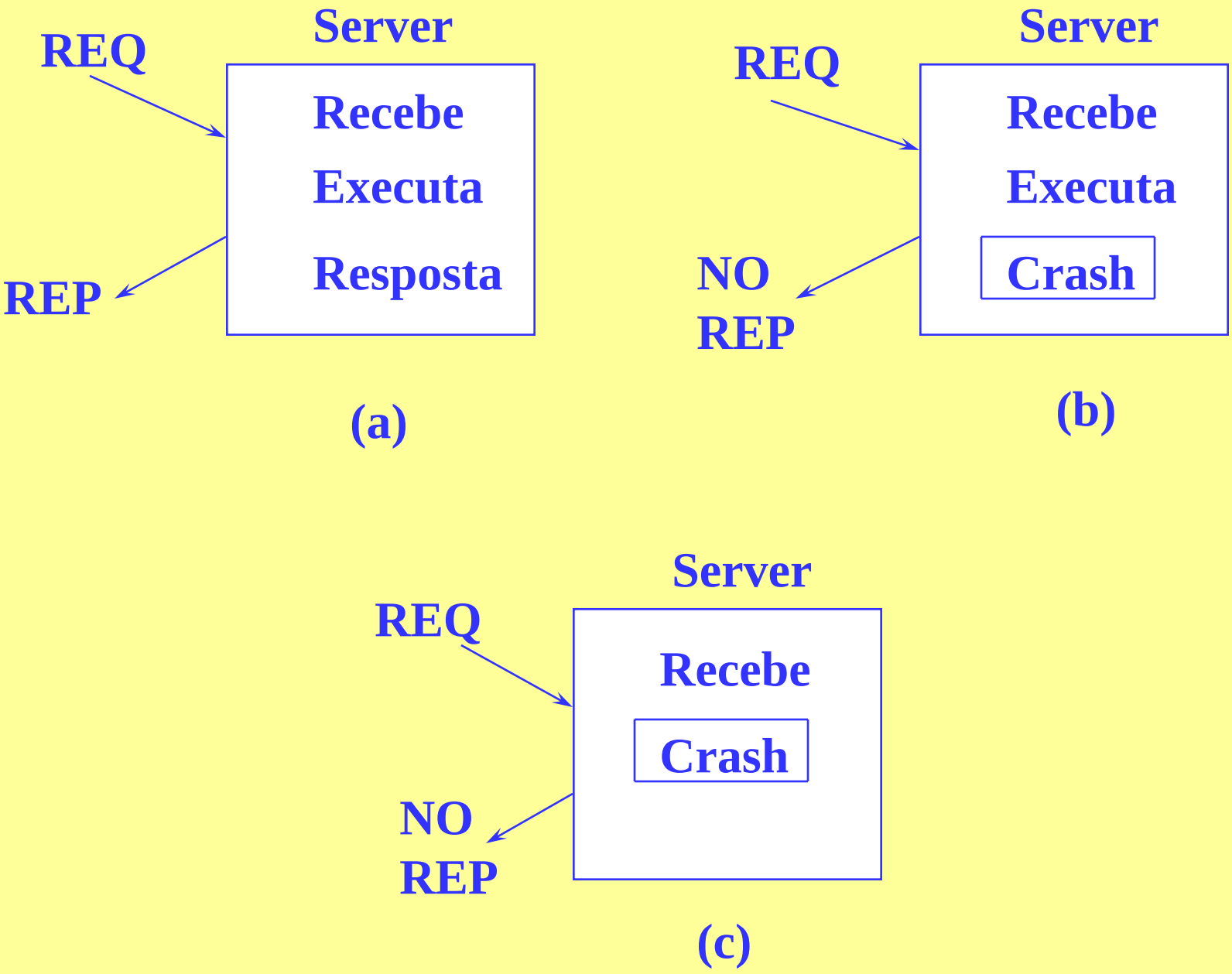
Usar o timer novamente.

Problema - o kernel do Cliente não sabe porque não obteve resposta. A requisição ou a resposta foi perdida ou o servidor está lento.

Algumas operações podem ser repetidas sem problemas (leitura de um arquivo), outros não (transferências bancárias).

Solução: o kernel do Cliente atribui a cada requisição um número de sequência.

Falha do Servidor



4- **Falha no Servidor**

O tratamento de (b) e (c) diferem:

(b) Sistema tem que comunicar a falha para o Cliente

(c) Somente retransmitir a requisição

Abordagem em três modos:

1- Esperar que o servidor seja reinicializado e tentar novamente;

2- Desistir imediatamente e comunicar a falha;

3- Não garantir nada (fácil de implementar).

Resumo: Possibilidade do servidor falhar muda a natureza da RPC e a distingue do sistema uniprocessador.

5- **Falha do Cliente**

O que acontece se o Cliente envia uma requisição para o servidor e falha antes da resposta?

Teremos uma computação ativa sem um processo "pai" esperando pelo resultado. Este tipo de computação não desejada é chamada de ÓRFÃO.

Problemas com órfãos:

- Desperdício de tempo de CPU;
- Podem segurar recursos (lock);
- Quando o cliente é reinicializado e a RPC é executada novamente, pode haver confusão.

Solução 1 -- Cliente “stub” mantem um registro sobre todas as mensagens RPC. O registro é mantido em disco ou outro meio que sobreviva a uma falha. Depois de reinicializado, o registro é verificado e os órfãos são eliminados. Esta solução é chamada ***exterminação***.

Problemas --

- overhead escrevendo o registro no disco;
- pode não funcionar se o órfão fez uma chamada remota;
- se houver uma falha na rede pode ser impossível eliminar o órfão mesmo sabendo onde ele se encontra.

Solução 2 -- *Reencarnação* -- O tempo é dividido em "épocas" numeradas seqüencialmente. Quando o Cliente é reinicializado ele envia uma mensagem para todas as máquinas declarando o começo de uma nova "época". Quando a mensagem é recebida todas as chamadas remotas da "época" passada são eliminadas.

Problema -- se houver uma falha na rede pode ser impossível eliminar o órfão mesmo sabendo onde ele se encontra, mas neste caso o órfão poderá ser, posteriormente, detectado facilmente e eliminado.

Solução 3 -- *Reencarnação Suave* -- Quando a mensagem de uma nova "época" é recebida, cada máquina verifica se tem computações remotas; em caso positivo tenta localizar seu Cliente. Se ele não for encontrado a computação é eliminada.

Solução 4 -- *Expiração* -- Cada RPC recebe uma quantidade de tempo padrão T para executar seu trabalho. Depois de uma falha a reinicialização é feita após um tempo T, quando todos os órfãos terão terminado.

Problema - Escolher um valor para T (RPC tem diferentes requerimentos)