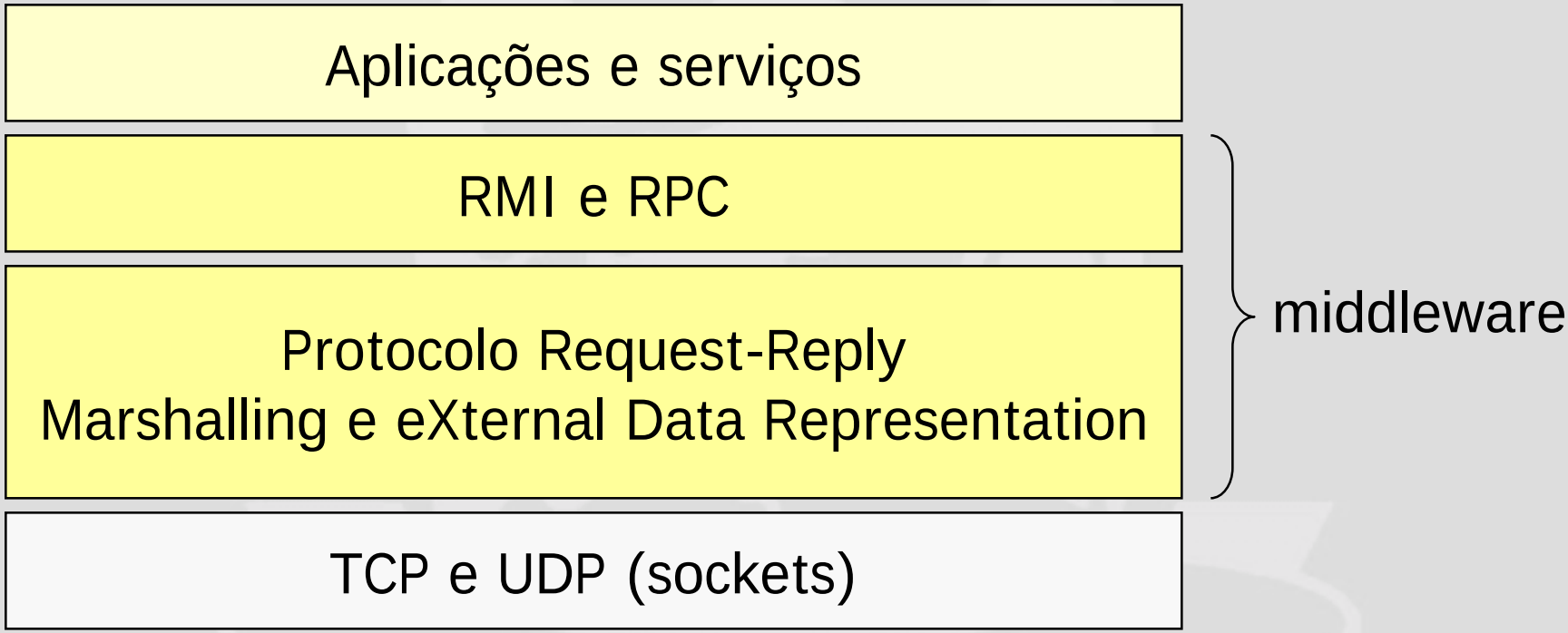
Comunicação

Carlos Ferraz
cagf@cin.ufpe.br

Tópicos

- Elementos básicos de comunicação
  - Transmissão de dados
  - Endereçamento
  - Sincronismo
  - Enfileiramento (*Bufferização*)
  - Confiabilidade
- *Sockets*
- RPC: Remote Procedure Call
- Comunicação de objetos distribuídos

Camadas



Transmissão de dados

- Dados em programas são estruturados enquanto que mensagens carregam informação sequencial:
  - » Linearização/Restauração de dados
- Heterogeneidade na representação de dados em computadores:
  - » Uso de um formato externo comum
  - » Inclusão de uma identificação de arquitetura na mensagem

Marshalling / Unmarshalling

- Marshalling:
  - Linearização de uma coleção de itens de dados estruturados
  - Tradução dos dados em formato externo (ex: XDR)
- Unmarshalling:
  - Tradução do formato externo para o local
  - Restauração dos itens de dados de acordo com sua estrutura

Endereçamento

- Esquemas:
  - Endereçamento máquina.processo
  - Descoberta de endereço via broadcasting (difusão)
  - Descoberta de endereço via um servidor de nomes
- Problemas potenciais: transparência de localização, sobrecarga, escalabilidade

Comunicação síncrona

- Primitiva send é bloqueante: processo cliente aguarda enquanto o núcleo envia a mensagem para o processo servidor
- Primitiva receive é bloqueante: processo servidor aguarda até que o núcleo receba uma mensagem endereçada para aquele processo

Comunicação assíncrona

- Primitiva send não é bloqueante: o processo cliente aguarda somente enquanto a mensagem é copiada para o buffer do núcleo
- Primitiva receive pode ser:
  - bloqueante: o processo servidor aguarda por uma mensagem
  - não bloqueante: o processo servidor simplesmente comunica ao núcleo que espera receber uma mensagem

Enfileiramento

- Situação:
  - Um cliente faz um send enquanto o servidor ainda atende a outro cliente
- Solução trivial: clientes devem insistir ...
- Solução pragmática: mailbox (uma fila de mensagens controlada pelo núcleo):
  - mailbox criado a pedido do servidor
  - mensagens endereçadas ao mailbox

Confiabilidade

- Mensagens se perdem, atrasam, duplicam...
- Abordagens:
  - Send tem semântica não confiável: as aplicações devem garantir entrega de mensagens (ex: timeout)
  - Mensagem de acknowledgement enviada pelo servidor (no nível núcleo)
  - Mensagem de acknowledgement implícita na resposta do servidor (carona)

Sockets

- Fornece “pontos finais” (*endpoints*) para a comunicação entre processos (*end-to-end*)
- Deve ser ligado a uma porta local
- *Socket pair*: (`endereço_IP_local:porta_local, endereço_IP_remoto:porta_remota`) identifica unicamente uma comunicação

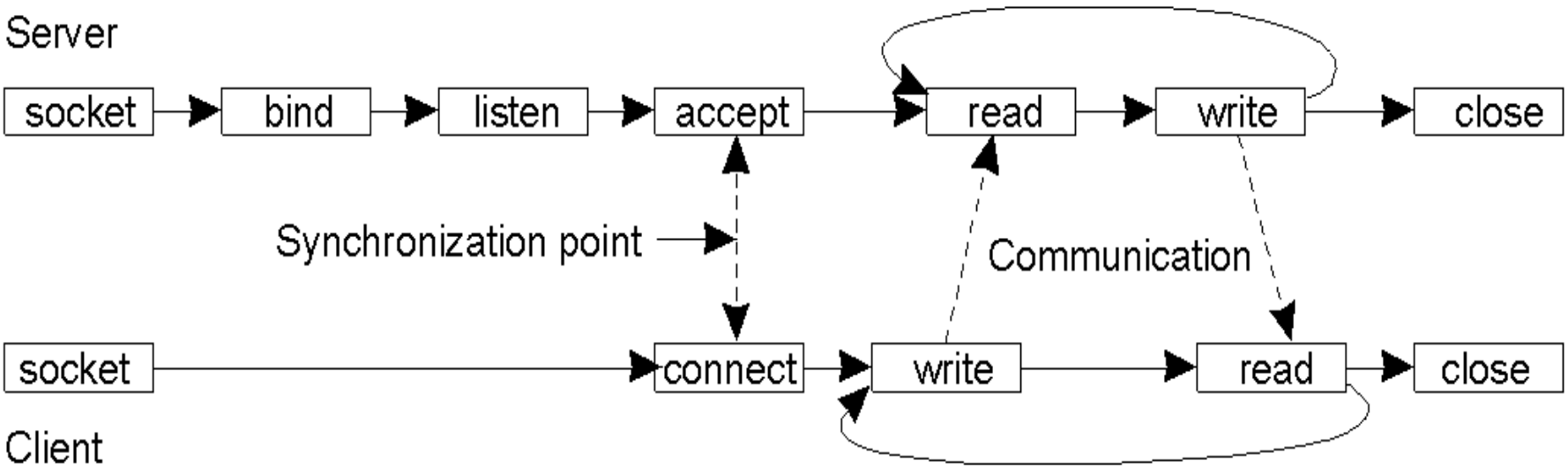
Berkeley Sockets (1)

- Primitivas de *sockets* para TCP/IP

Primitiva	**Significado**
Socket	Cria um novo *communication endpoint*
Bind	Liga um endereço local a um *socket*
Listen	Anuncia vontade de aceitar conexões
Accept	Bloqueia até que um pedido de conexão chegue
Connect	Ativamente tenta estabelecer uma conexão
Send	Envia dados sobre uma conexão
Receive	Recebe dados em uma conexão
Close	Libera a conexão

Berkeley Sockets (2)

- Padrão de comunicação orientada a conexão usando *sockets*



Protocolo Pedido-Resposta

- Request-Reply (RR)
- Típico na interação cliente-servidor
- Primitivas:
  - DoOperation - clientes invocam operações remotas
  - GetRequest - servidor adquire os pedidos de serviços
  - SendReply

Chamada de Procedimentos Remotos (RPC)

- Ideal: programar um sistema distribuído como se fosse centralizado
- RPC objetiva permitir chamada de procedimento remoto como se fosse local, ocultando entrada/saída de mensagens
- Integração relativamente transparente com linguagens de programação, que inclui uma notação para definir interfaces (“IDL”)

Chamadas de procedimento

- O procedimento “chamador”, que já tem suas variáveis locais empilhadas, empilha os parâmetros da chamada e o endereço de retorno
- O procedimento chamado aloca suas variáveis locais
- No retorno do procedimento chamado, os parâmetros e o endereço de retorno são desempilhados

RPC: considerações em função da distribuição

- Mantida a semântica de chamadas de procedimentos convencionais, mas...
  - evitar passagem de endereço e variáveis globais
  - novos tipos de erros
    - Incl.: tratamento de exceções (ex.: atraso de comunicação ➔ timeout)

RPC: Implementação

- Suporte a RPC:
  - Processamento de interface
  - Manipulação da comunicação
  - Ligação

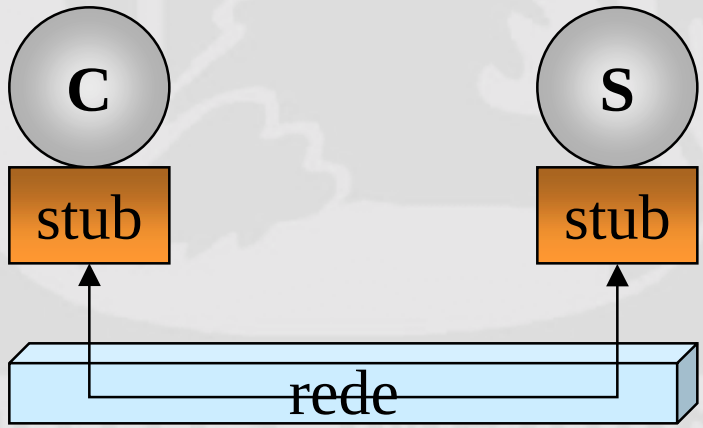
segue....

RPC: processamento de interface

- Integração dos mecanismos de RPC com os programas cliente e servidor escritos em uma linguagem de programação convencional
- Cliente e servidor assinalam o mesmo identificador de procedimento para cada procedimento na interface

RPC: processamento de interface

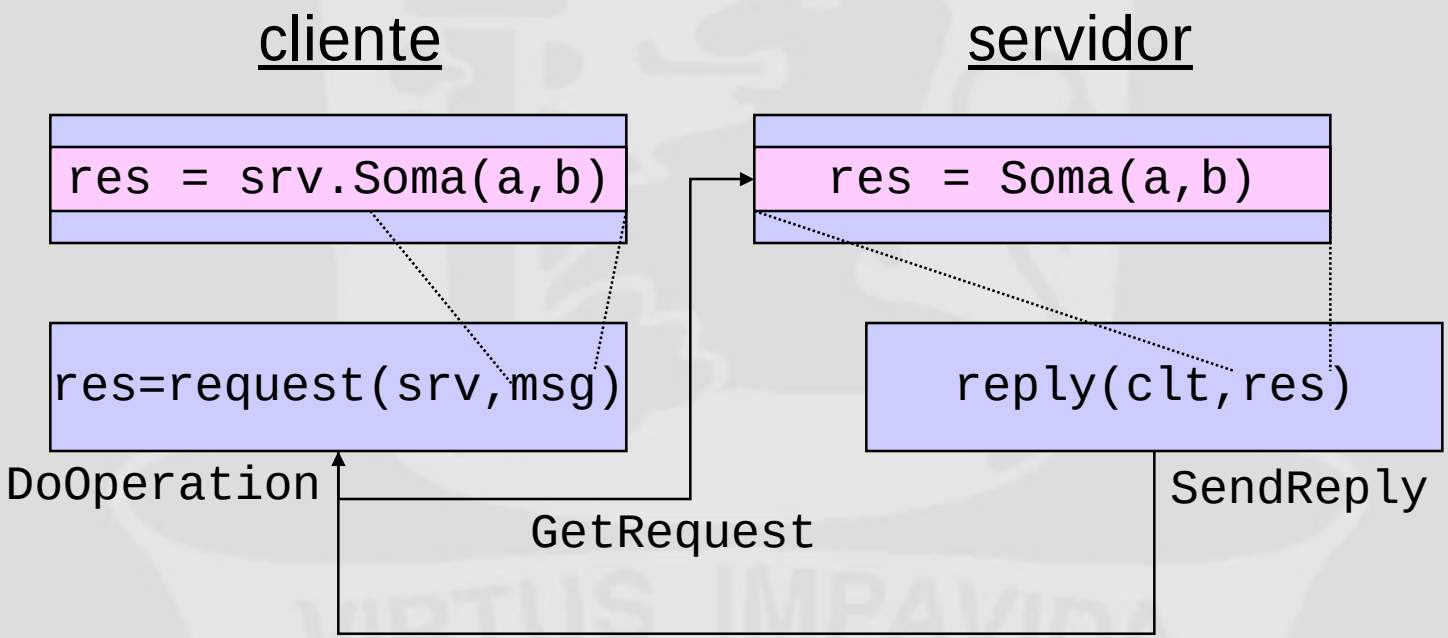
- *Stubs* (no cliente e no servidor): transparência de acesso
  - tratamento de algumas exceções no local
  - marshalling
  - unmarshalling



RPC: manipulação da comunicação

- Módulo de comunicação usa protocolo *pedido-resposta* para troca de mensagens entre cliente e servidor

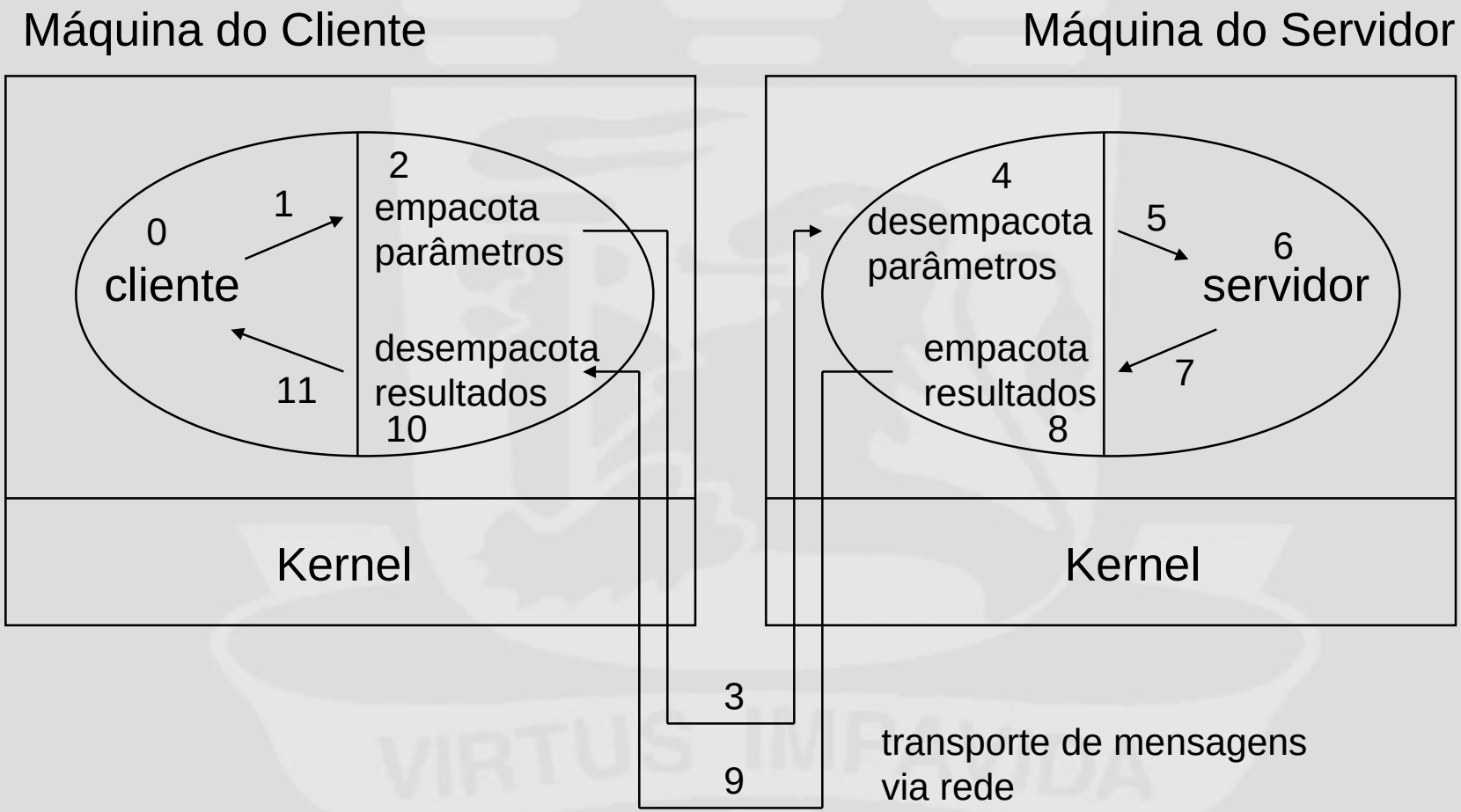
Mensagem de aplicação encapsulada em um request/reply



RPC: ligação

- O mecanismo possui um *binder* para resolução de nomes, permitindo
  - Ligação dinâmica
  - Transparência de localização

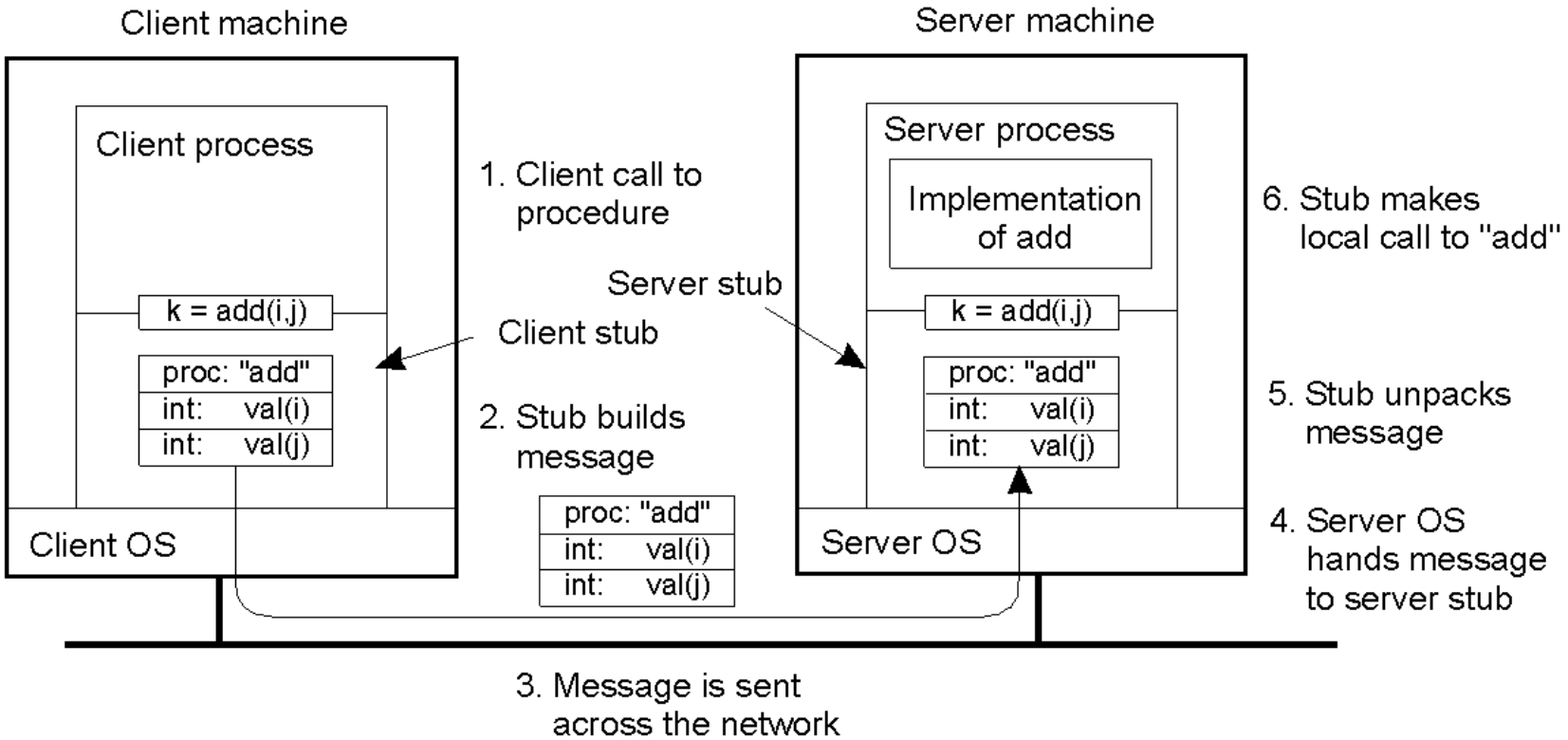
Chamadas e mensagens em RPC



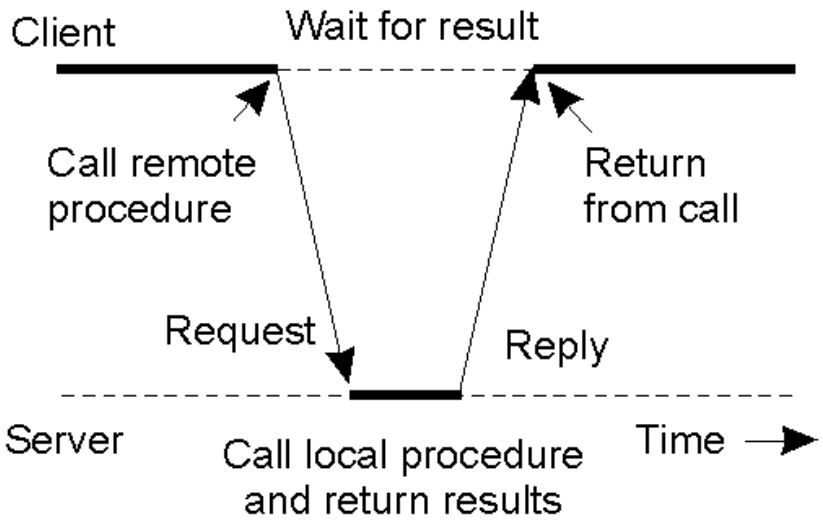
Funções dos Stubs

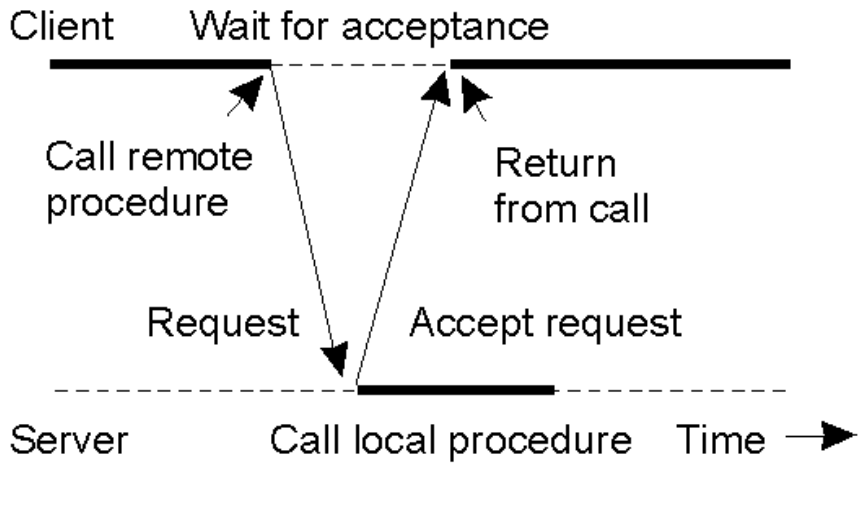
- Client stub
  1. intercepta a chamada
  2. empacota os parâmetros (marshalling)
  3. envia mensagem de request ao servidor (através do núcleo)
- Server stub
  4. recebe a mensagem de request (através do núcleo)
  5. desempacota os parâmetros (unmarshalling)
  6. chama o procedimento, passando os parâmetros
  7. empacota o resultado
  8. envia mensagem de reply ao cliente (através do núcleo)
- Client stub
  9. recebe a mensagem de reply (através do núcleo)
  10. desempacota o resultado
  11. passa o resultado para o cliente

RPC: Passagem de Parâmetros



RPC Assíncrono

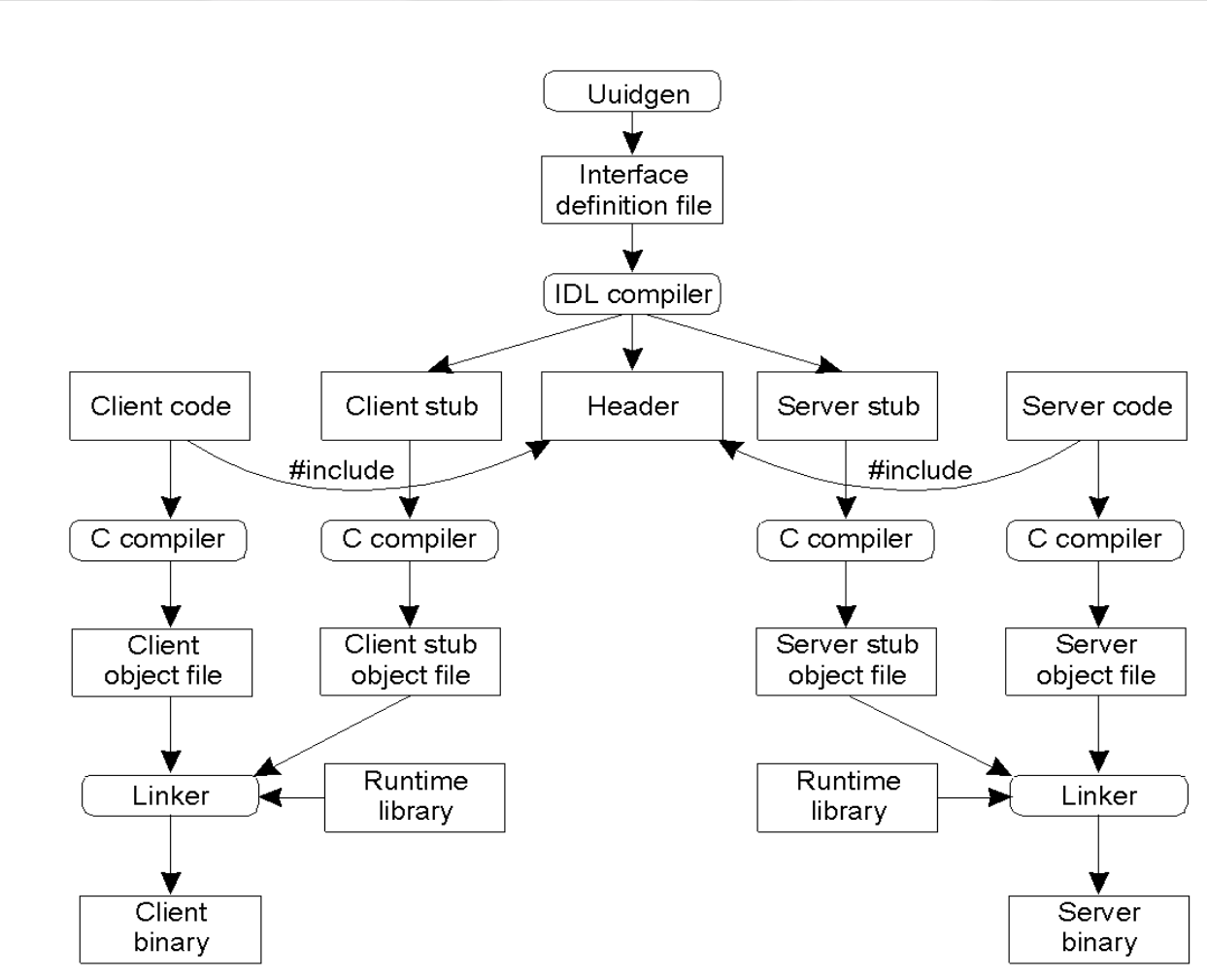




a) Interação cliente-servidor em um RPC tradicional

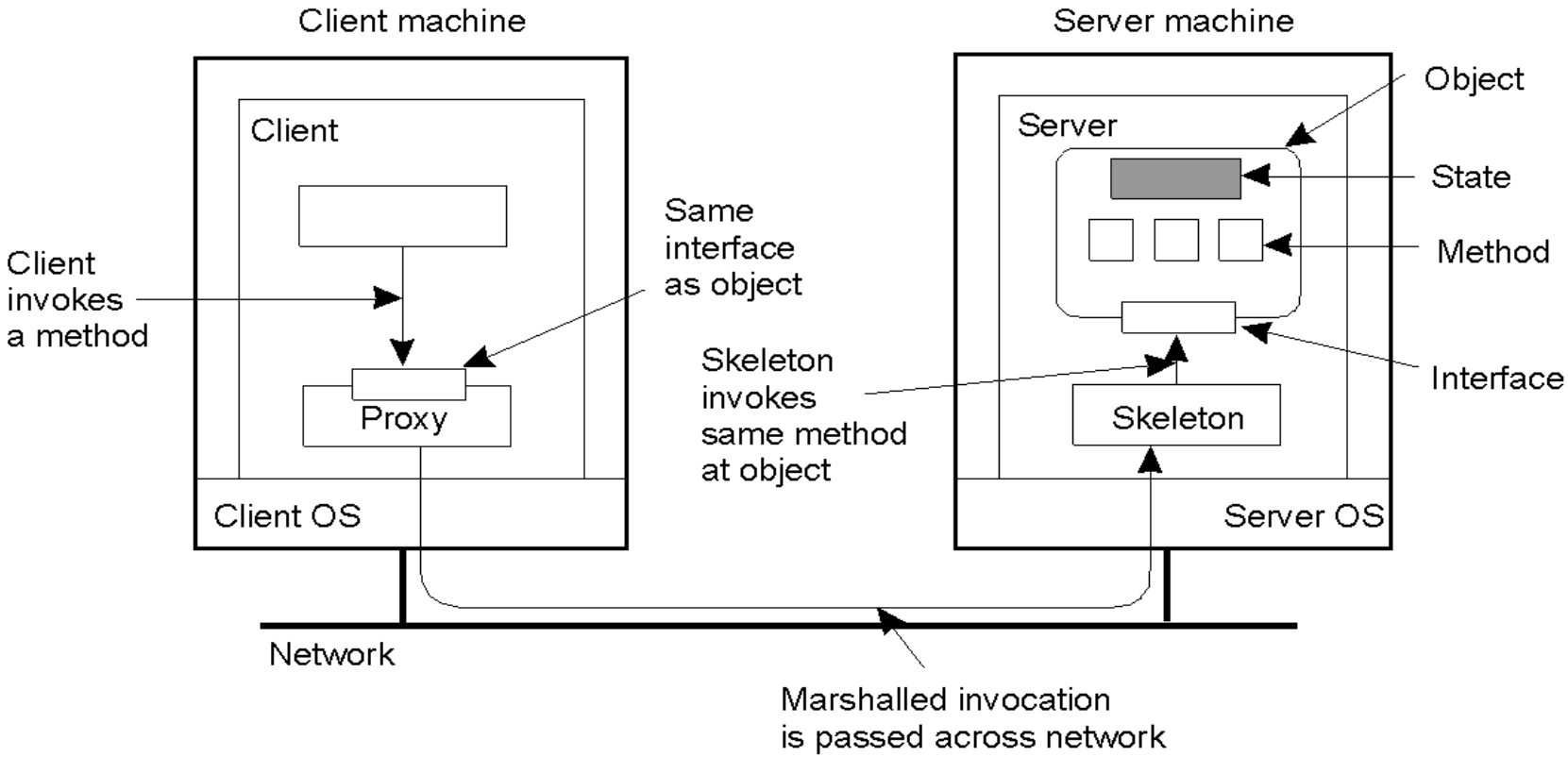
b) Interação usando RPC assíncrono

Construção de um Cliente e de um Servidor



Objetos Distribuídos: RMI

- Semelhante a RPC



Distributed Systems: Principles and Paradigms, Chapter 02
© Tanenbaum and van Steen 2002

Referências de Objetos

- Geralmente valem em todo o sistema
  - Passadas livremente entre processos em máquinas diferentes (inclusive como parâmetros na invocação de métodos)
- Implementação de referências é escondida
  - Transparência de distribuição melhorada em relação a RPC
- A ligação entre um cliente e um objeto resulta em colocar um *proxy* do objeto no espaço de endereçamento do cliente, contendo uma interface com os métodos a serem chamados (figura anterior)

Implementação de Refs de Objetos

- Um referência deve conter informações o bastante para que um cliente se ligue a um objeto
  - Mínimo: endereço de rede, porta, id. do objeto
  - Mais: protocolo de ligação, protocolos de comunicação suportados pelo servidor – ex. o cliente pode carregar o proxy mais apropriado, como um que implemente TCP ou UDP

Invocações dinâmicas x estáticas

- Invocações dinâmicas: o método **invoke**
  - **invoke** (objeto, método, parâm_ent, parâm_saída)
- **Ex:**
  - Estático

    ```
    res = objref.soma(a, b);
    ```

  - Dinâmico

    ```
    invoke (objref,id(soma),in_struc,out_struct);
    ```

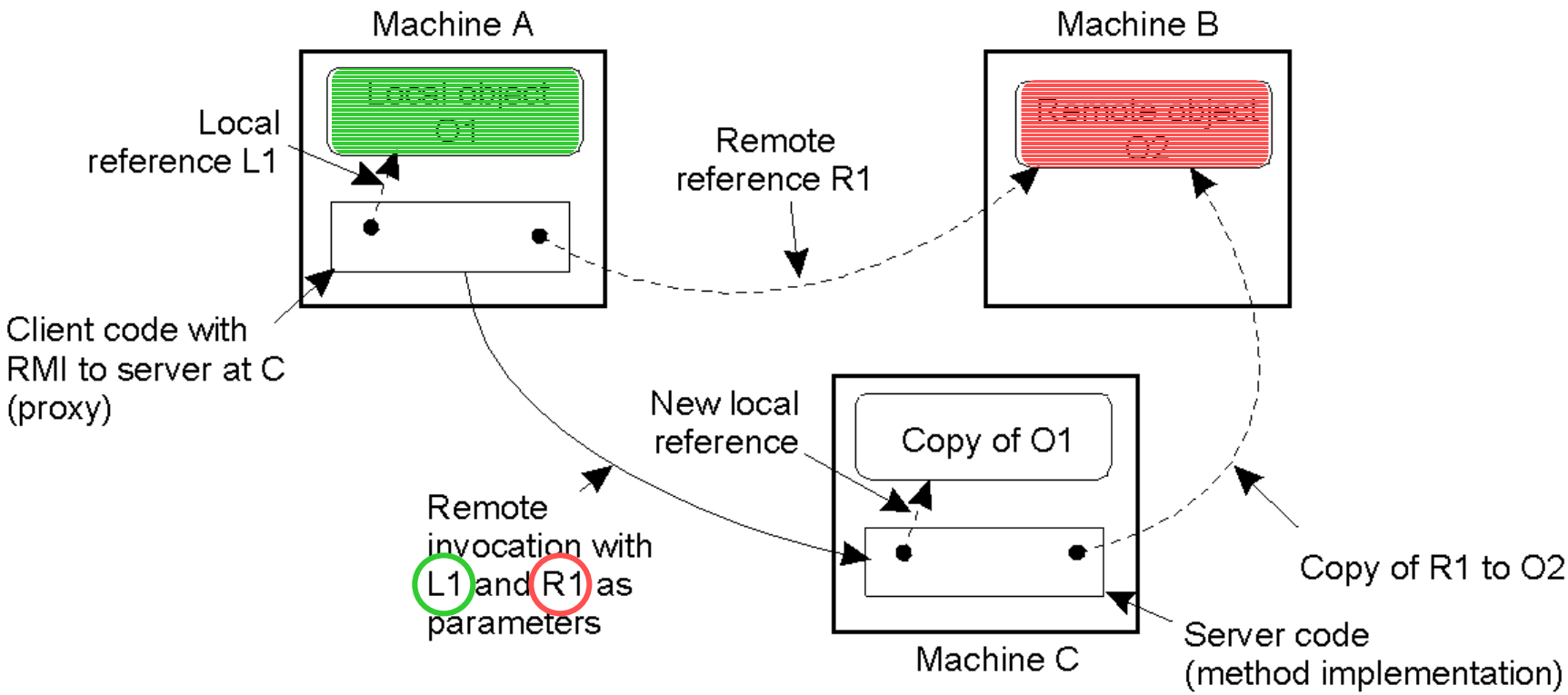
    onde:
    - `id(soma)` retorna um identificador para o método soma
    - `in_struct` é uma estrutura de dados que contém os parâmetros a e b
    - `out_struct` é uma estrutura de dados para o parâmetro de saída (res)

Passagem de objetos como parâmetro (I)

- Quando o parâmetro é uma referência de objeto (remoto), cópia da referência é passada por valor → o objeto é "*literalmente*" passado por referência
- Quando o parâmetro é um objeto (local), ou seja, um objeto no mesmo espaço de endereçamento do cliente, cópia do objeto é passada por valor → o objeto é passado por valor

Passagem de objetos como parâmetro (II)

- objeto passado por referência e por valor



Distributed Systems: Principles and Paradigms
© Tanenbaum and van Steen 2002