Comunicação - RPC e RMI

Vagner Fonseca, PhD

Motivação

- Dificuldade de comunicação entre processos via sockets

Cliente em Java
Conectar ao Servidor
Lê do teclado dois long
Converter long para String
Envia as Strings
Recebe o resultado

Servidor em C
Recebe via socket vetor de char
Converte char para long
Faz a soma dos dois long
Devolve resultados como vetor de char

Motivação

- Comunicação entre processos cliente e servidores por meio de troca explicita de mensagens
  - Ex: Aplicação JAVA enviando objetos Mensagem

```
saida.writeObject(msg)
Mensagem resp = entrada.readObject();
```

- Não provê a transparência de acesso
  - Esconder as diferenças para representação dos dados
  - Como os dados serão representados em diferentes arquiteturas

RPC – Chamada Remota de Procedimentos

- Muitos sistemas são baseados em troca explicita de mensagem entre processos
- O problema é que as trocas de mensagem não escondem detalhes da comunicação
- A ideia que fundamenta as Chamadas Remotas de Procedimentos é permitir que chamem procedimentos remotos de forma que pareça o o mais possível com uma chamada local

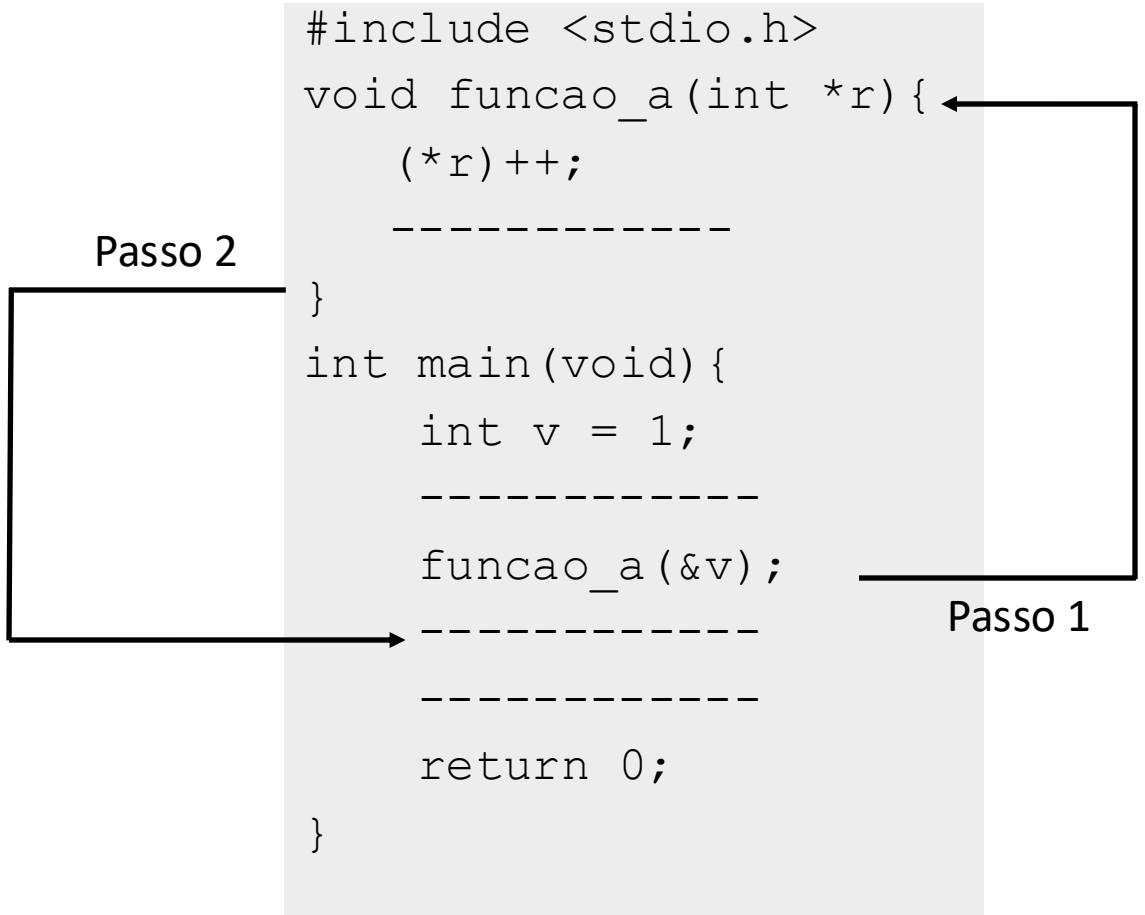
RPC – Chamada Remota de Procedimentos

- O Procedimento de chamada não deve esta ciente de que o procedimento chamado está executando em outra maquina ou vice-versa (transparência)
- Informações podem ser transportadas do chamador para quem foi chamado nos parâmetros e podem voltar no resultado do procedimento
- Nada de troca de mensagens é visível ao programador
- Do ponto de vista do código, a chamada se assemelha a chamadas de procedimentos locais

RPC – Chamada Remota de Procedimentos

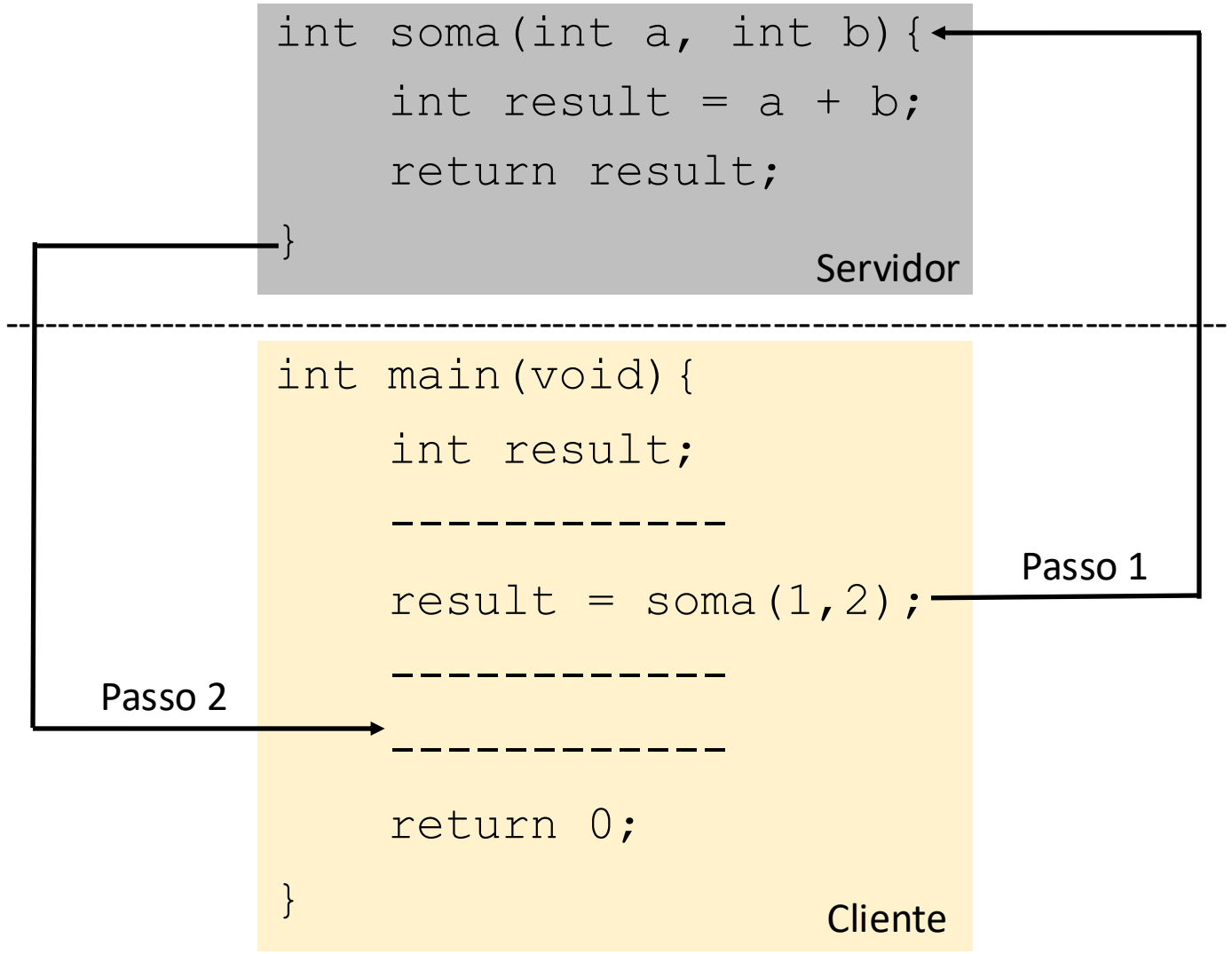
- Embora a ideia parece simples, há alguns pontos delicados que devem ser considerados:
  - Por rodarem em maquinas distintas, procedimentos chamador e chamado executem em espaços de endereços diferentes, o que acarreta em certa complexidade no seu gerenciamento
  - Há também, todo o cuidado nas conversões de dados, principalmente em arquiteturas distintas
  - Por fim, todos os problemas inerentes a sistemas em rede e que executam em mais de um computador
- No entanto, mesmo com todos esses desafios, RPC é uma técnica amplamente utilizada.

Inspiração para RPC: Chamada de Procedimentos em C



- Transferência de controle e dados
  - Lista de parâmetros e tipos de retorno permitem a comunicação entre funções
- Linguagem C: passagem de parâmetros pode ser por
  - Valor
    - Função que fora invocada cria uma copia local da variável
  - Referência
    - Função que fora invocada recebe endereço de memoria da variável

Inspiração para RPC: Chamada de Procedimentos em C



Representação dos dados

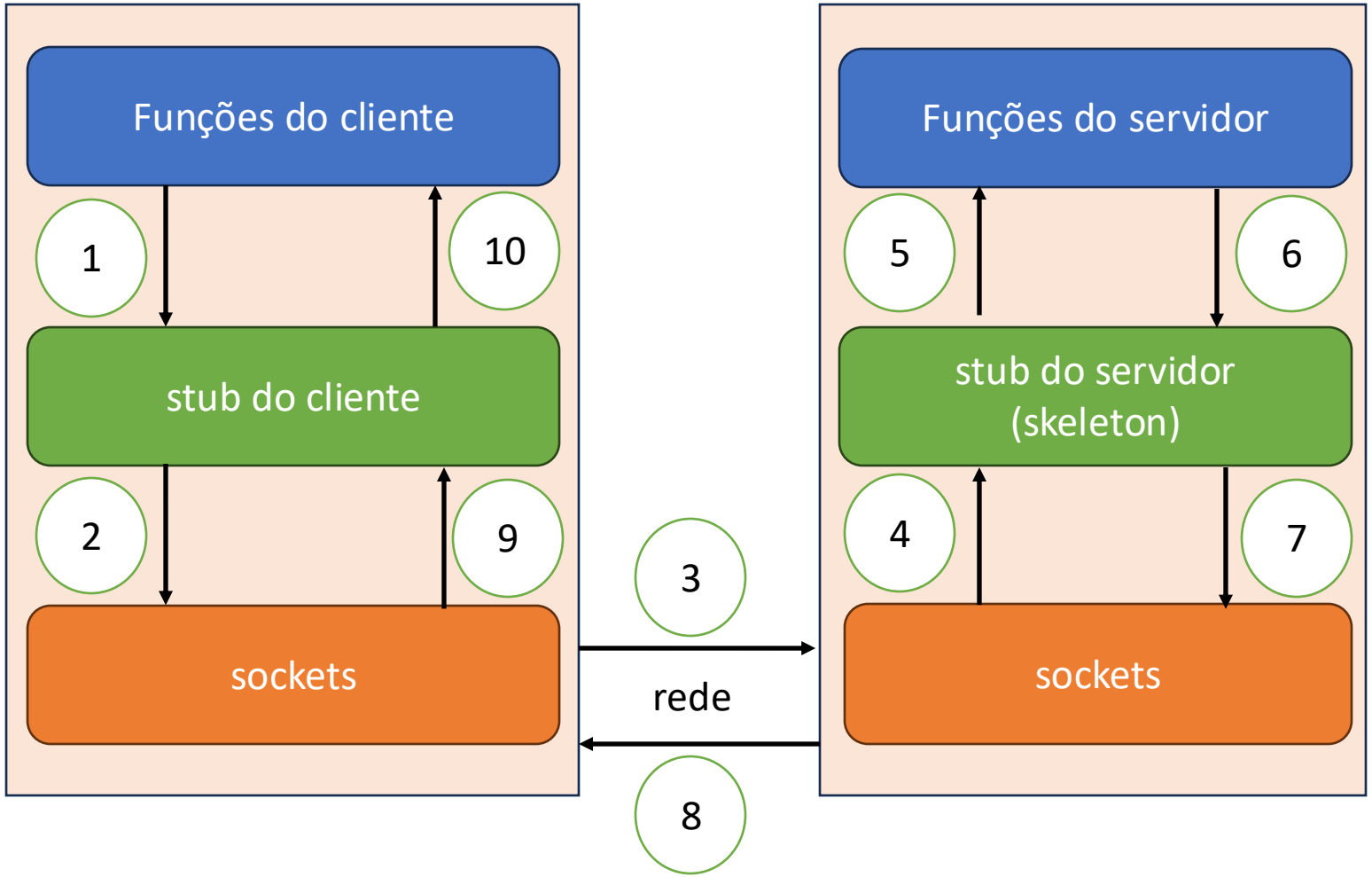
- Necessidade de um padrão de codificação para permitir a comunicação entre sistemas heterogêneos
- Exemplos de formatos
  - XML
  - JSON
  - CORBA CDR (Common Data Representation)
  - XDR (eXternal Data Representation)
  - ASN.1
  - Google Protocol Buffers

Serialização dos dados

- **Serialização** traduz estruturas de dados em um formato que possa ser armazenados em disco ou transportado pela rede
  - Dados são convertidos para um vetor de bytes
- Formato de representação com **tipo de variável implícito**
  - Somente os valores são transferidos
    - XDR: é um padrão IEFT de 1995 de camada de apresentação do modelo OSI, que permite dados serem empacotados em uma arquitetura de maneira independente para que o dado seja transferido entre sistemas de computadores heterógenos
- Formato de representação com **tipo de variável explicito**
  - O tipo de cada valor também é transmitido
  - Ex: JSON, XML, ASN.1, Protocol Buffers.

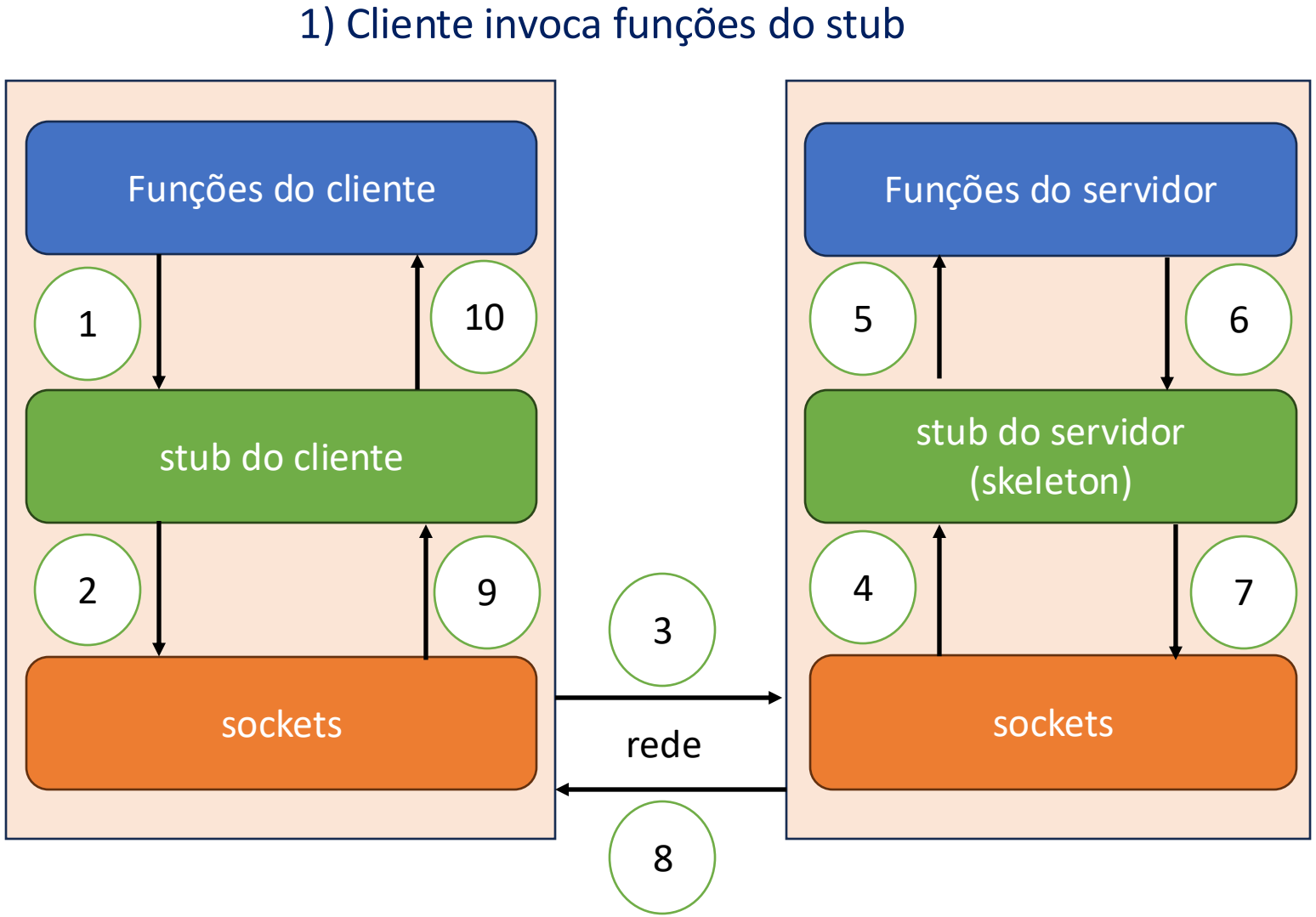
Chamada de procedimentos remotos - RPC

- Processos locais realizam chamadas de **procedimentos** que estão **localizados em outras maquinas**, fazendo o uso de **apêndices** no cliente (*stubs*) e no servidor (*skeletons*)



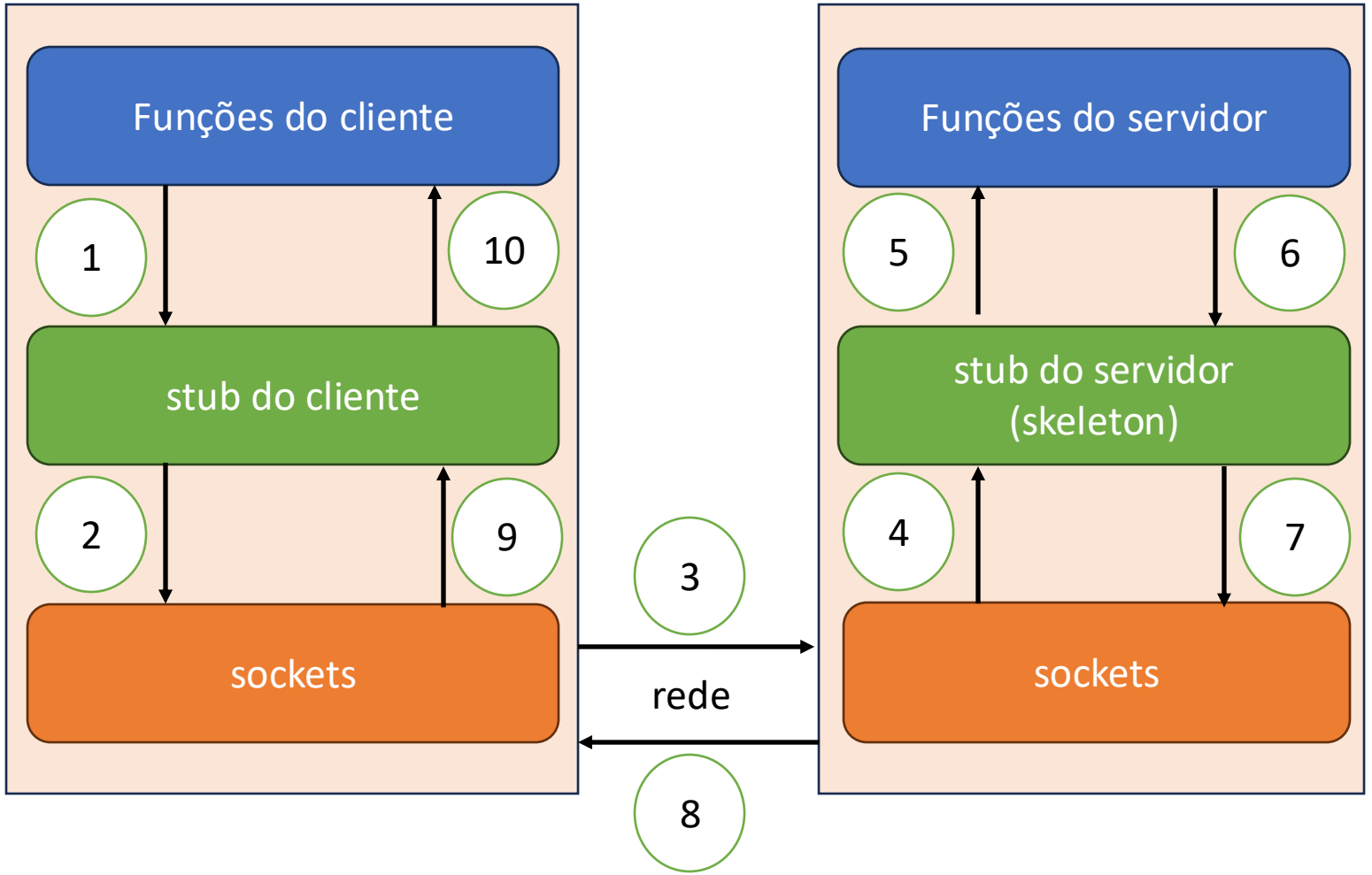
Chamada de procedimentos remotos - RPC

- Processos locais realizam chamadas de **procedimentos** que estão **localizados em outras maquinas**, fazendo o uso de **apêndices** no cliente (*stubs*) e no servidor (*skeletons*)



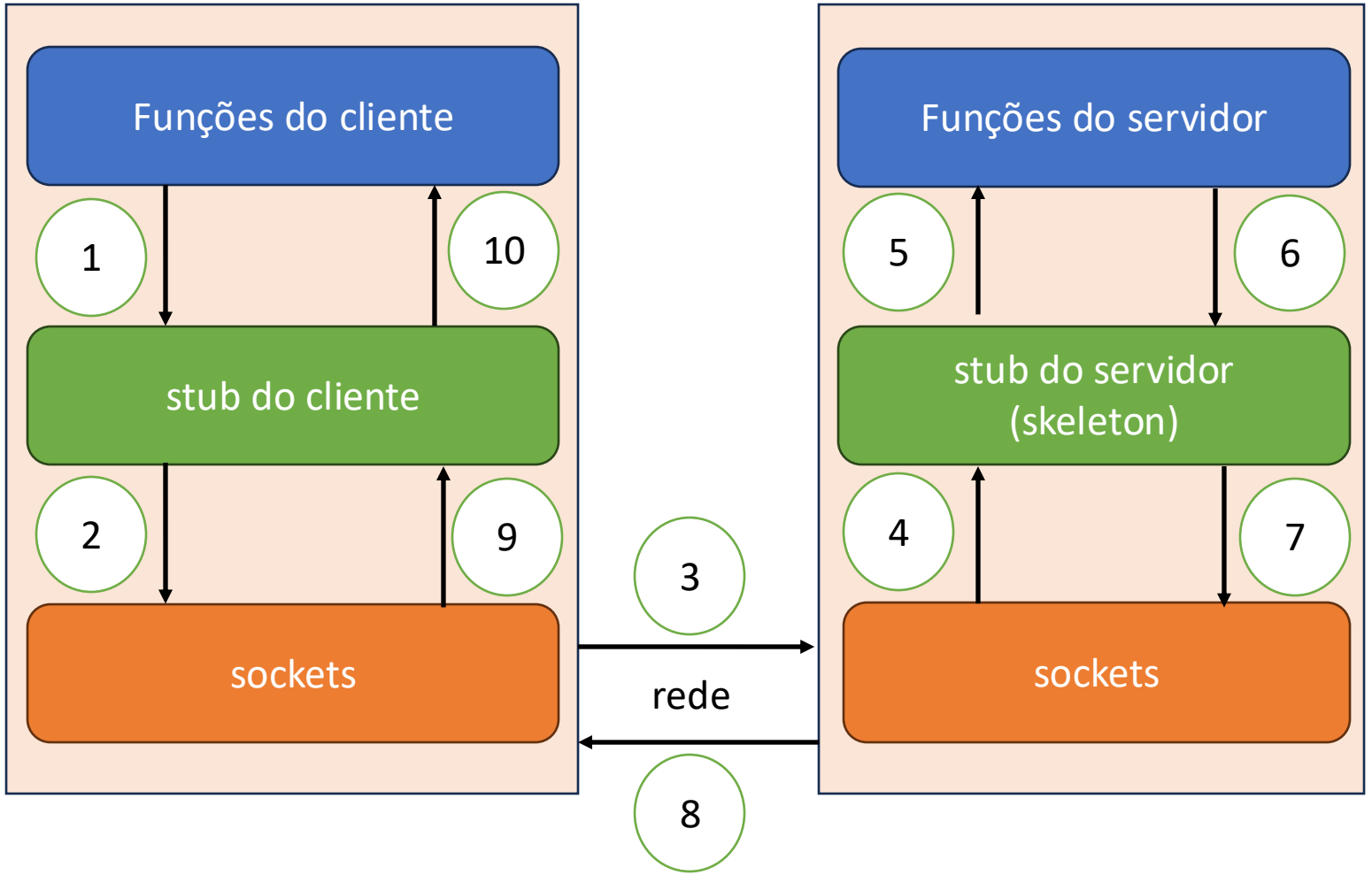
Chamada de procedimentos remotos - RPC

- Processos locais realizam chamadas de **procedimentos** que estão **localizados em outras maquinas**, fazendo o uso de **apêndices** no cliente (*stubs*) e no servidor (*skeletons*)



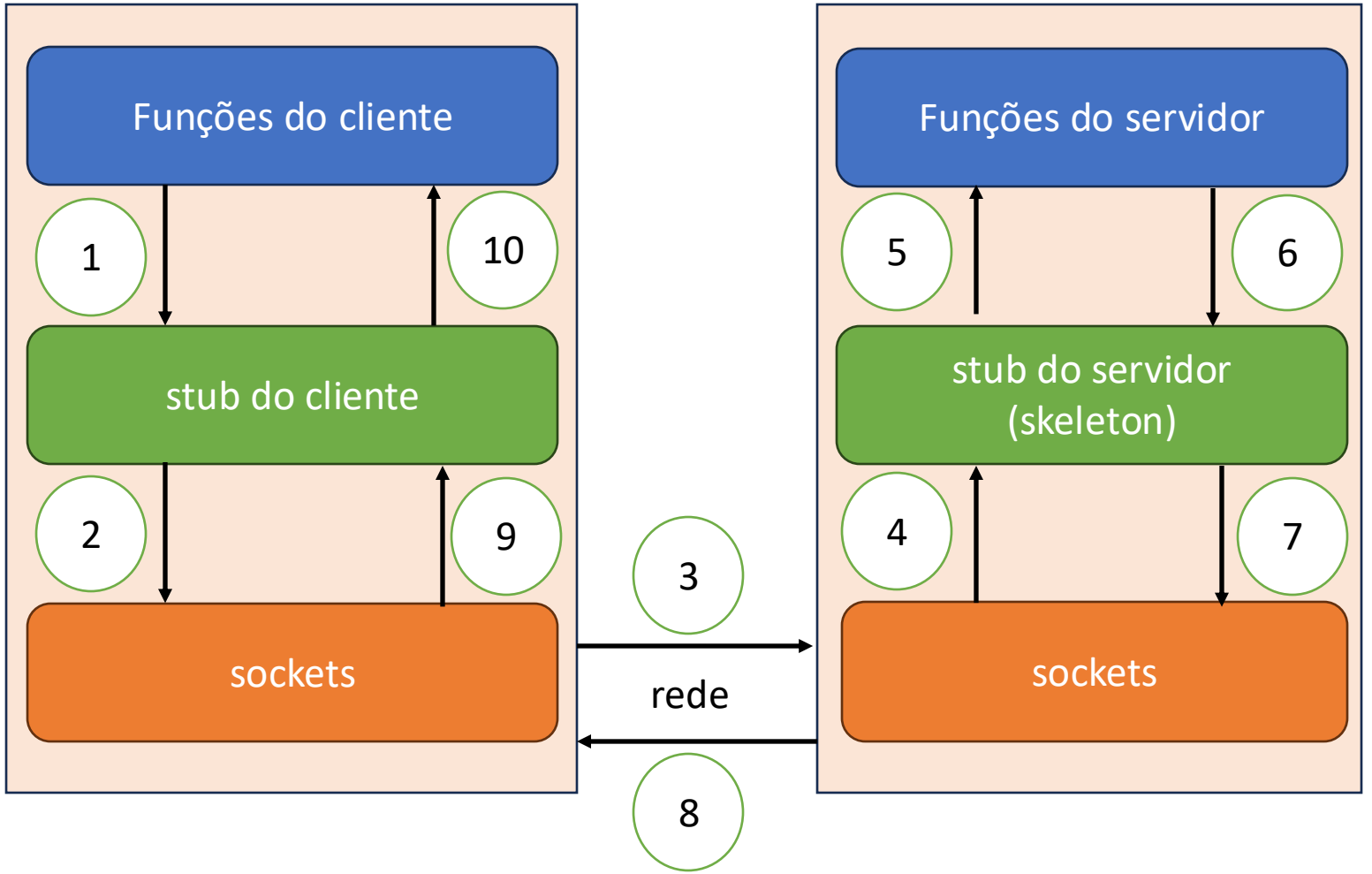
Chamada de procedimentos remotos - RPC

- Processos locais realizam chamadas de **procedimentos** que estão **localizados em outras maquinas**, fazendo o uso de **apêndices** no cliente (*stubs*) e no servidor (*skeletons*)



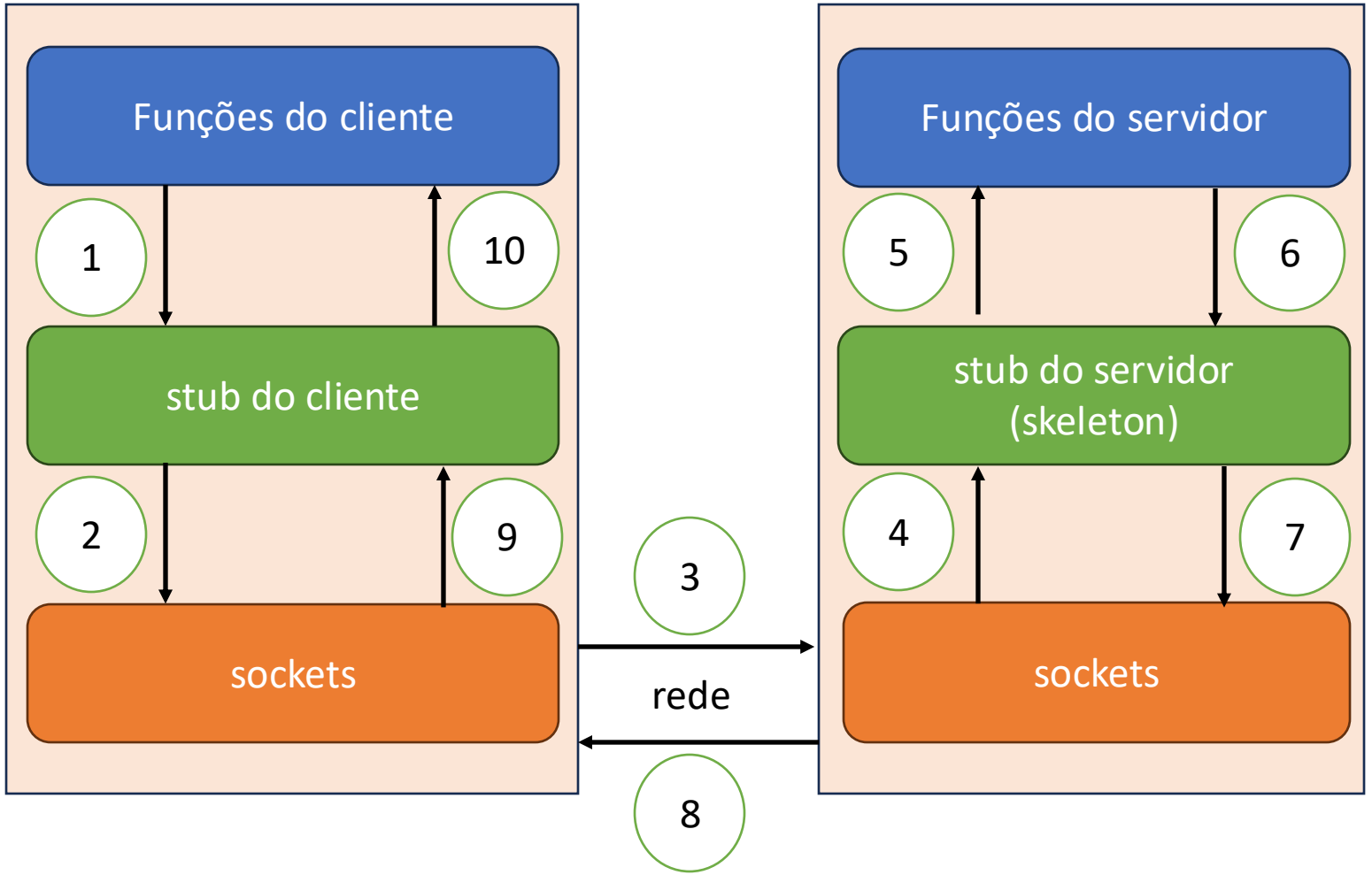
Chamada de procedimentos remotos - RPC

- Processos locais realizam chamadas de **procedimentos** que estão **localizados em outras maquinas**, fazendo o uso de **apêndices** no cliente (*stubs*) e no servidor (*skeletons*)



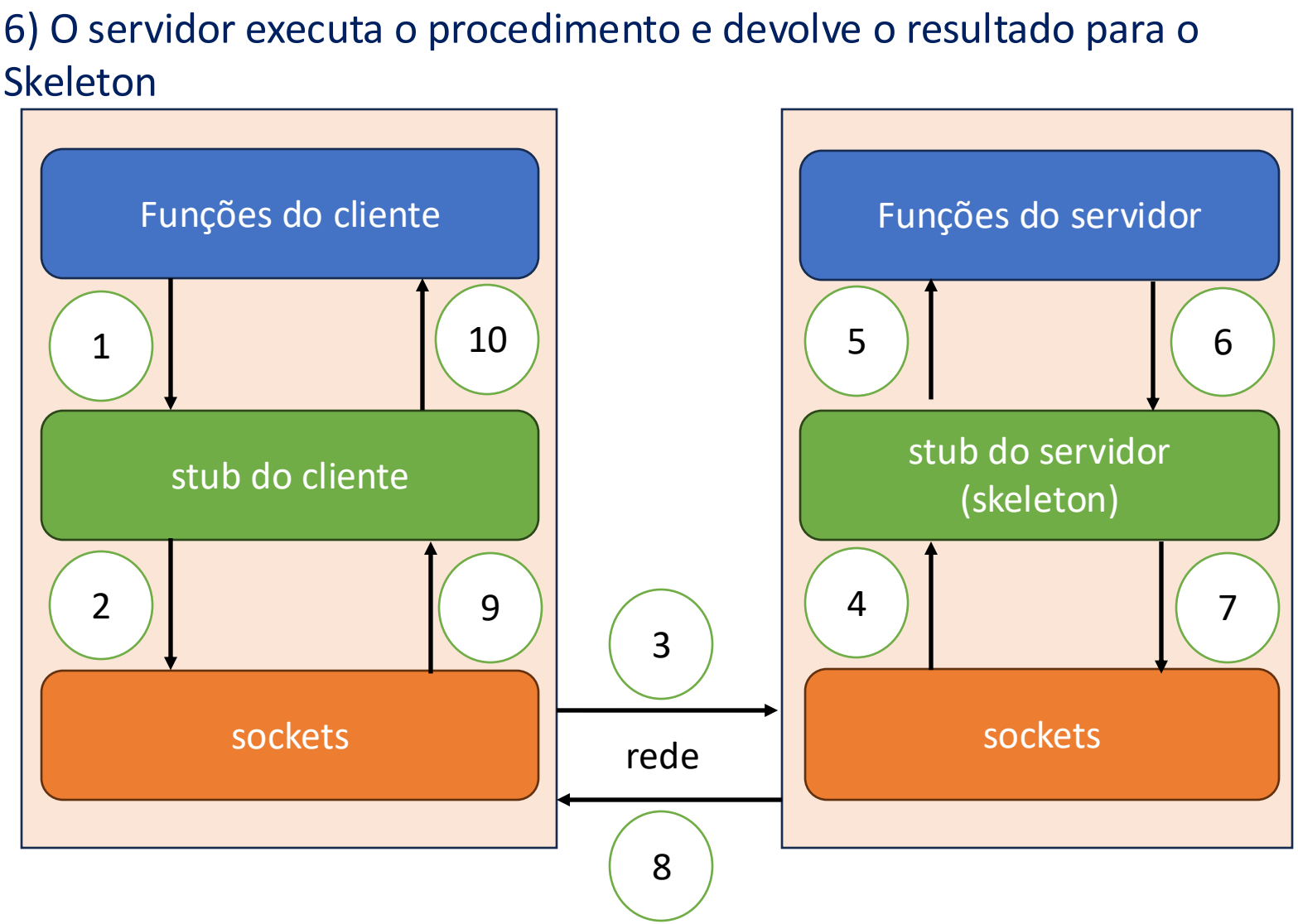
Chamada de procedimentos remotos - RPC

- Processos locais realizam chamadas de **procedimentos** que estão **localizados em outras maquinas**, fazendo o uso de **apêndices** no cliente (*stubs*) e no servidor (*skeletons*)



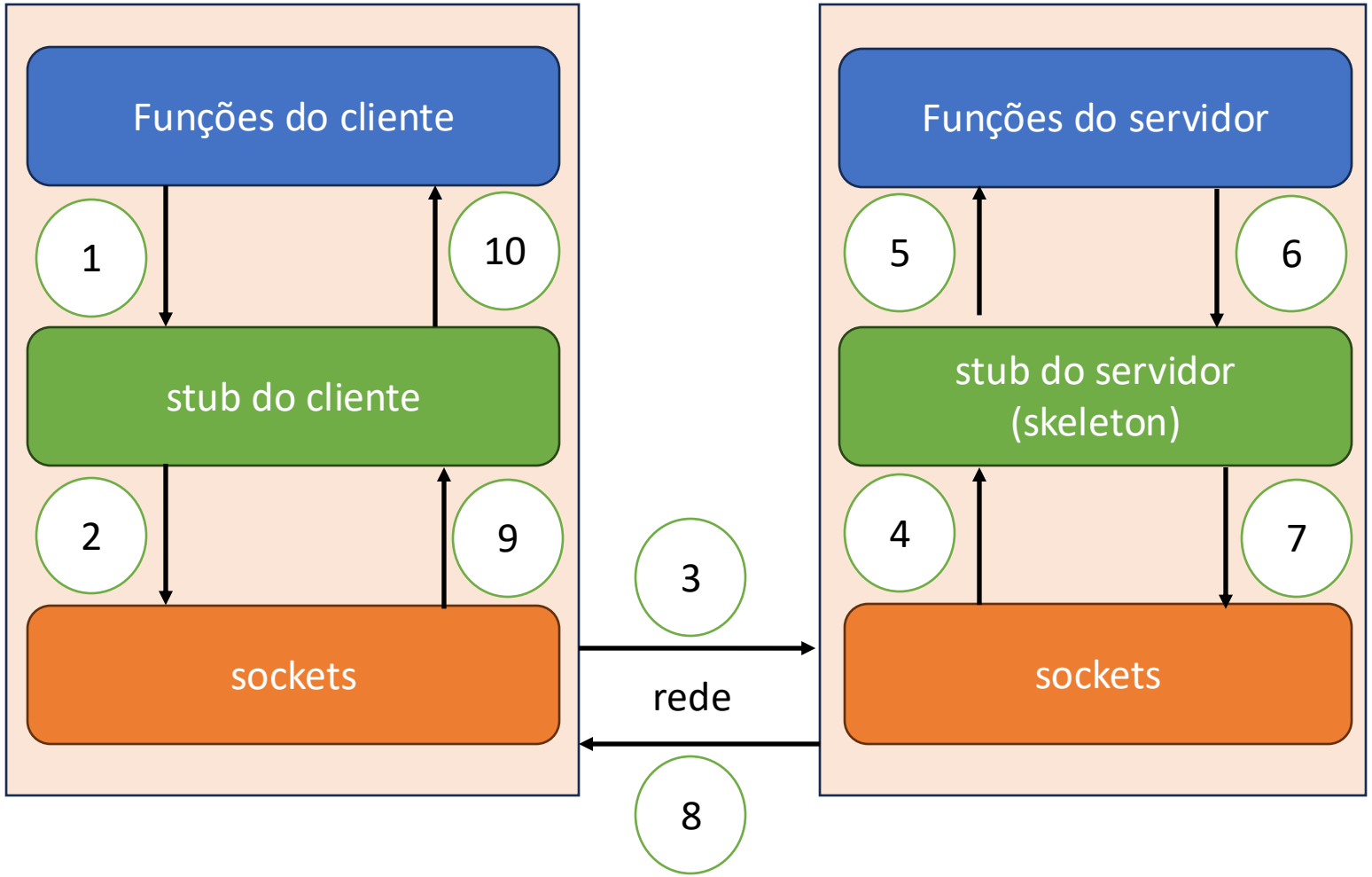
Chamada de procedimentos remotos - RPC

- Processos locais realizam chamadas de **procedimentos** que estão **localizados em outras maquinas**, fazendo o uso de **apêndices** no cliente (*stubs*) e no servidor (*skeletons*)



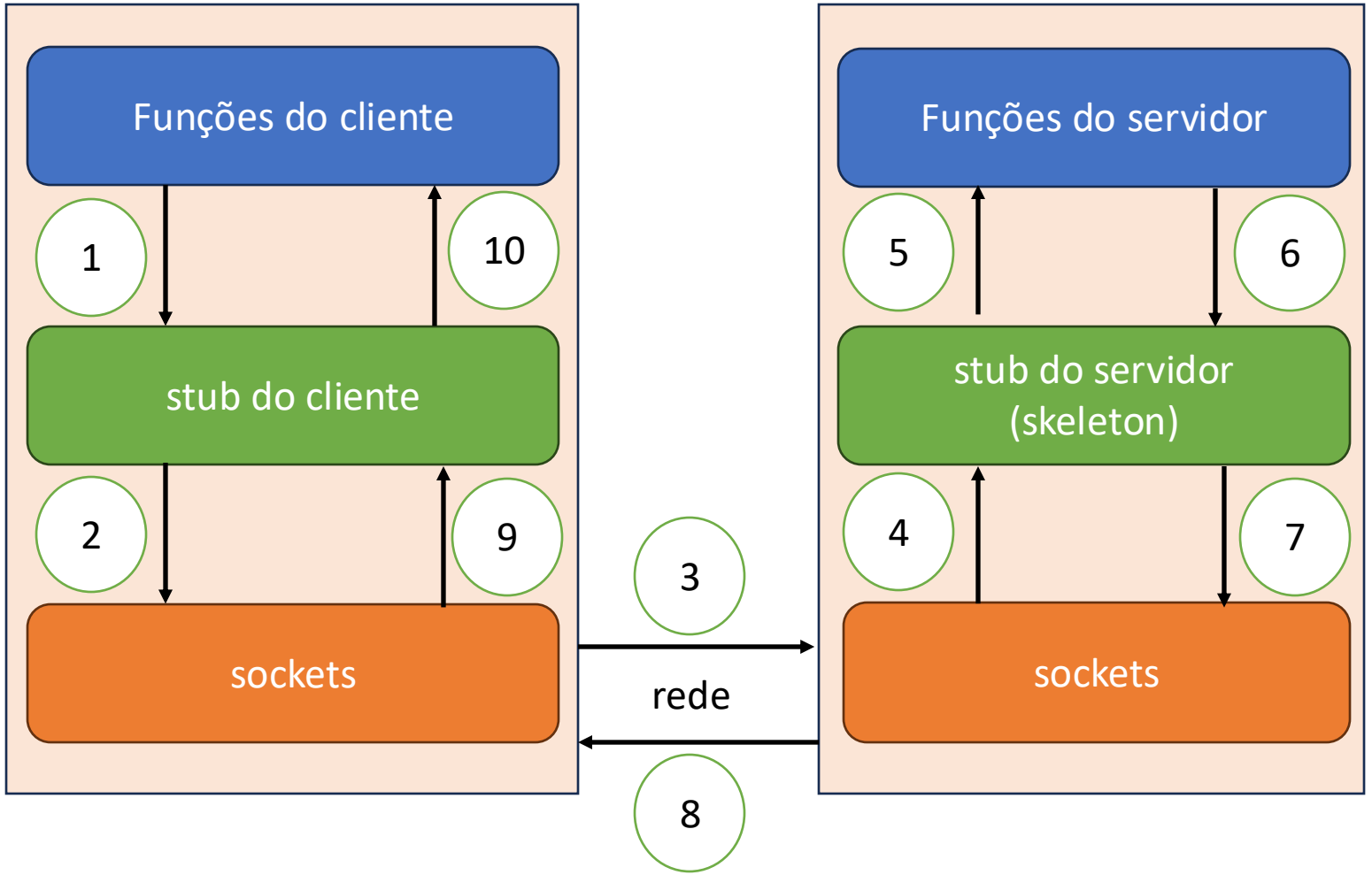
Chamada de procedimentos remotos - RPC

- Processos locais realizam chamadas de **procedimentos** que estão **localizados em outras maquinas**, fazendo o uso de **apêndices** no cliente (*stubs*) e no servidor (*skeletons*)



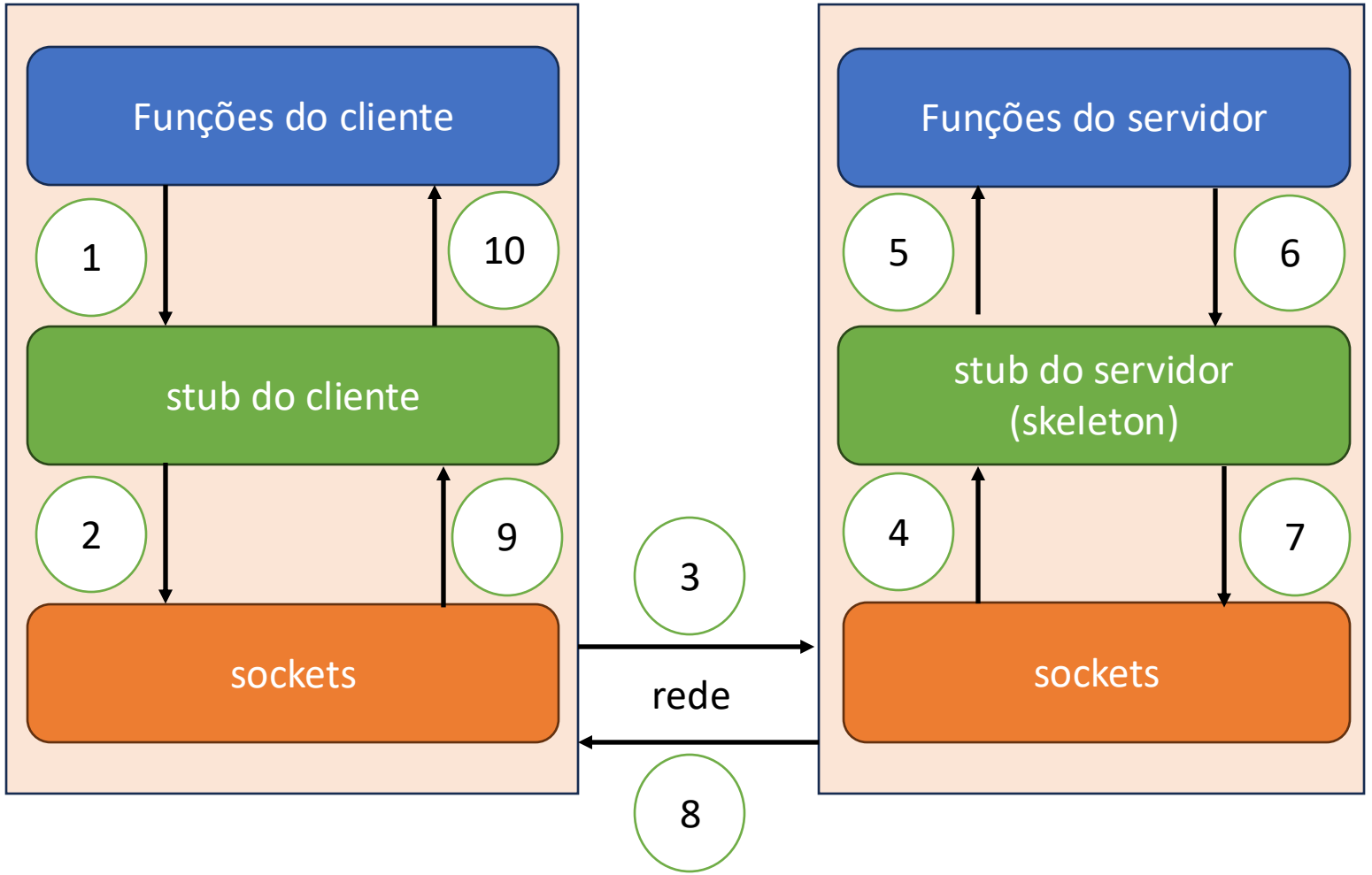
Chamada de procedimentos remotos - RPC

- Processos locais realizam chamadas de **procedimentos** que estão **localizados em outras maquinas**, fazendo o uso de **apêndices** no cliente (*stubs*) e no servidor (*skeletons*)



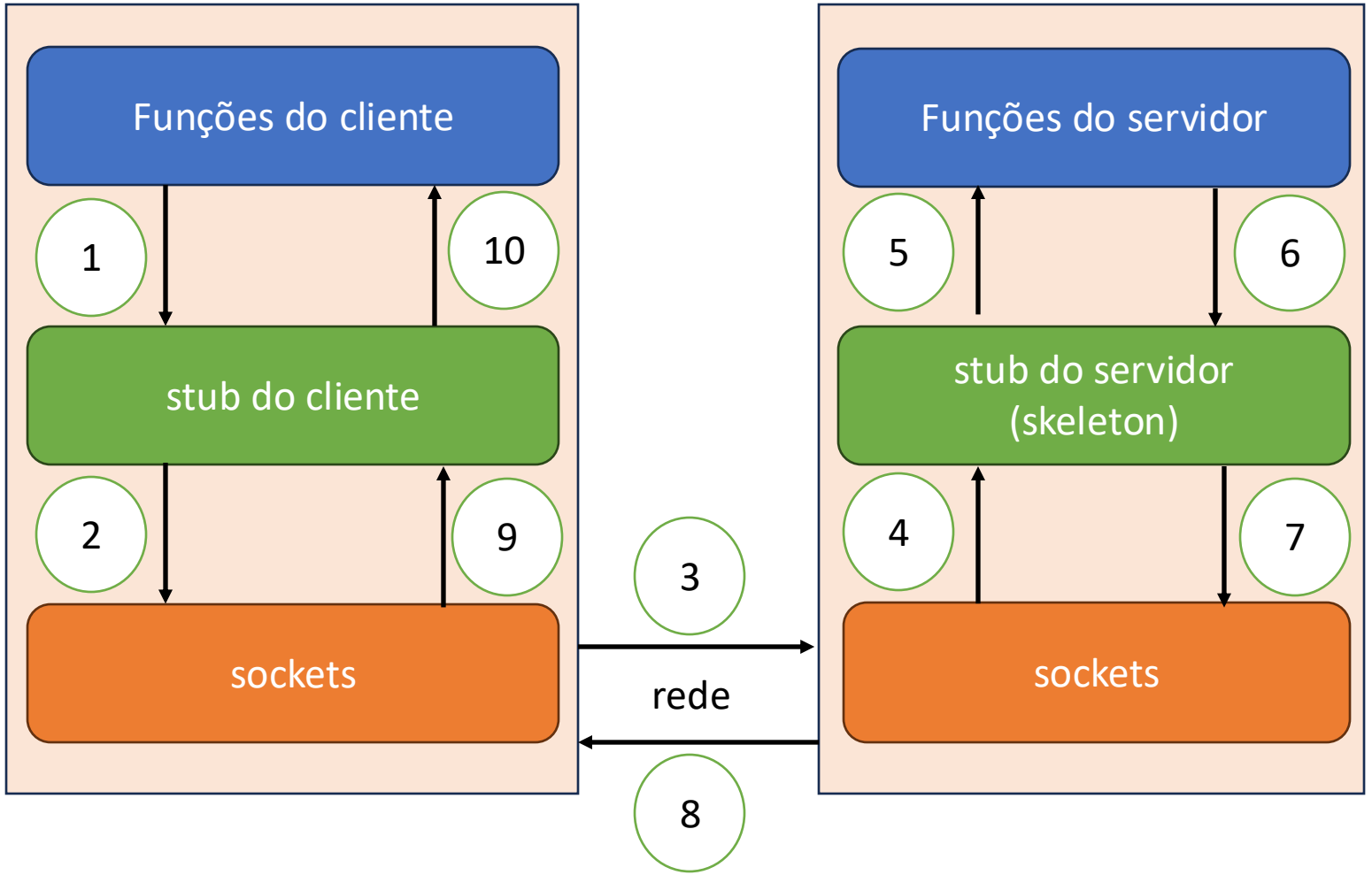
Chamada de procedimentos remotos - RPC

- Processos locais realizam chamadas de **procedimentos** que estão **localizados em outras maquinas**, fazendo o uso de **apêndices** no cliente (*stubs*) e no servidor (*skeletons*)



Chamada de procedimentos remotos - RPC

- Processos locais realizam chamadas de **procedimentos** que estão **localizados em outras maquinas**, fazendo o uso de **apêndices** no cliente (*stubs*) e no servidor (*skeletons*)



Java RMI

- Arquitetura de acesso a objetos distribuídos suportada pela linguagem Java.

- Em termos de complexidade de programação e ambiente, é muito simples construir aplicações RMI, comparando-se com RPC e CORBA.

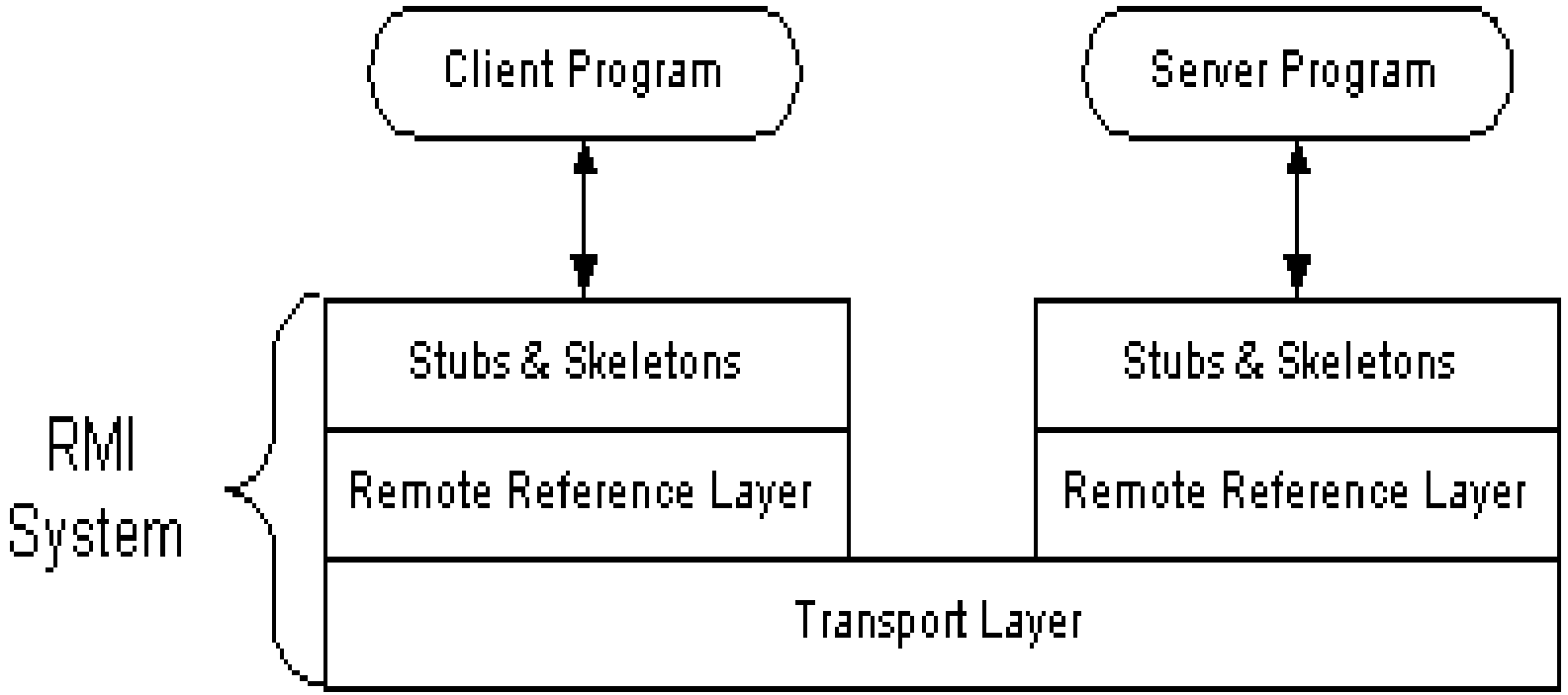
- Em termos de ambiente, exige somente suporte TCP/IP e um serviço de nomes de objetos (rmiregistry), disponilizado gratuitamente com o JDK/SDK.

Arquitetura Java RMI

- Na realidade, o RMI é uma interface que permite a intercomunicação entre objetos Java localizados em diferentes hosts. Cada objeto remoto implementa uma interface remota que especifica quais de seus métodos podem ser invocados remotamente pelos clientes.

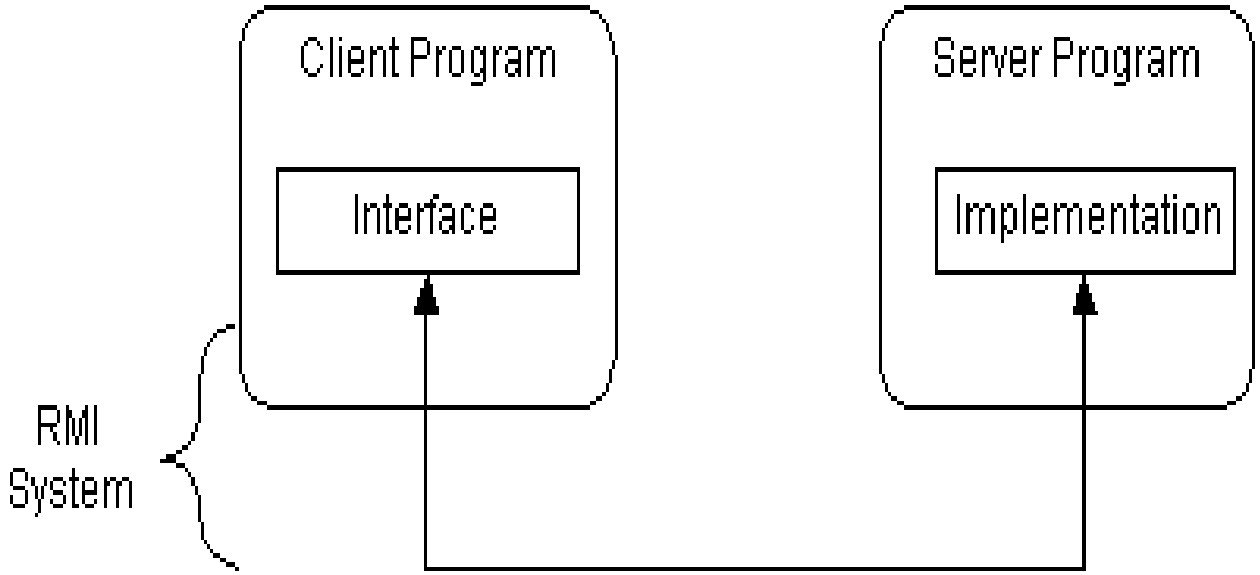
- Os clientes invocam tais métodos exatamente como invocam métodos locais.

Modelo de camadas do RMI



Interfaces para métodos remotos

- A definição do serviço remoto é feita através de uma interface Java.



Interfaces para métodos remotos

- O primeiro passo para disponibilizar métodos que possam ser invocados remotamente consiste na preparação de uma interface remota com tais métodos.

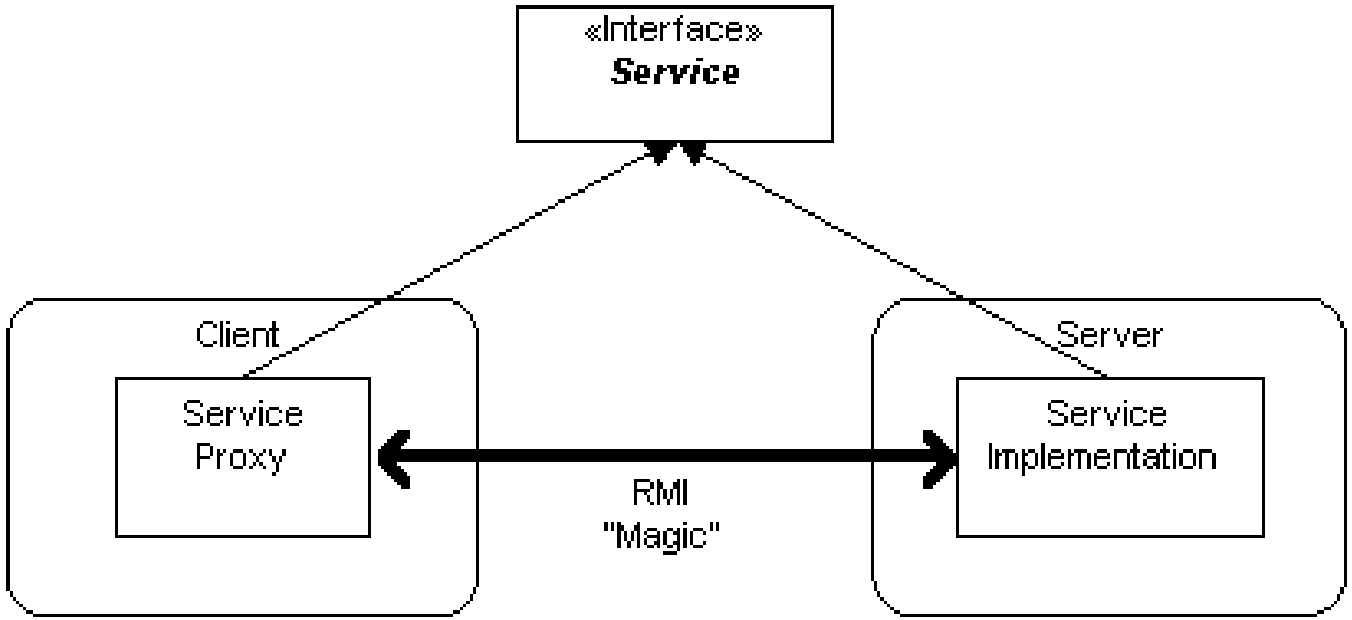
- A construção desta interface pode ser feita com base na extensão da interface Remote do pacote java.rmi.

Interfaces para métodos remotos

- A arquitetura RMI suporta duas classes implementando a mesma interface:
  - Uma, que implementa o serviço e é interpretada no servidor
  - Outra, que age como um mecanismo de proxy e é interpretada no cliente

Interfaces para métodos remotos

- Um cliente faz chamadas de métodos ao objeto proxy, RMI envia a requisição à JVM remota, que executa o método.



Interfaces para métodos remotos

- Valores retornados pelo serviço remoto são enviados, inicialmente, ao objeto proxy, que os repassa para a aplicação cliente.

- Vários servidores podem implementar de maneira diferente a mesma interface de acesso ao serviço.

Exemplo

- Suponha que se queira deixar um método chamado sayHello(), que devolve uma String, disponibilizado para chamada remota.

```
import java.rmi.*;
public interface Hello
     extends Remote{
     public String sayHello()
          throws RemoteException;
     }
```

Implementação do método remoto

- Cada classe que queira disponibilizar tal método remoto precisa implementar a interface especificada anteriormente.
- Além disto, a classe precisa extender a classe UnicastRemoteObject, que é uma especialização de um servidor remoto (classe RemoteServer).

Exemplo

```
import  java.rmi.*;
import java.rmi.server.*;
import java.net.*;
public class servidor extends UnicastRemoteObject implements Hello{
    public servidor() throws RemoteException{//Construtor
        super();
    }
    public String sayHello() throws RemoteException{//Método remoto
        return(“Oi cliente”);
    }
```

Exemplo (continuação)

```
    public static void main(String args[]){
        try{
            servidor serv=new servidor();
            //Registra nome do servidor
            Naming.rebind("ServidorHello",serv);
            System.out.println("Servidor remoto pronto.");
        }catch(RemoteException e){
            System.out.println("Exceção remota:"+e);
        }
        catch(MalformedURLException e){};
    }
}
```

A partir deste ponto, o objeto chamado ServidorHello está apto a aceitar chamadas remotas.

Compilação e execução do servidor

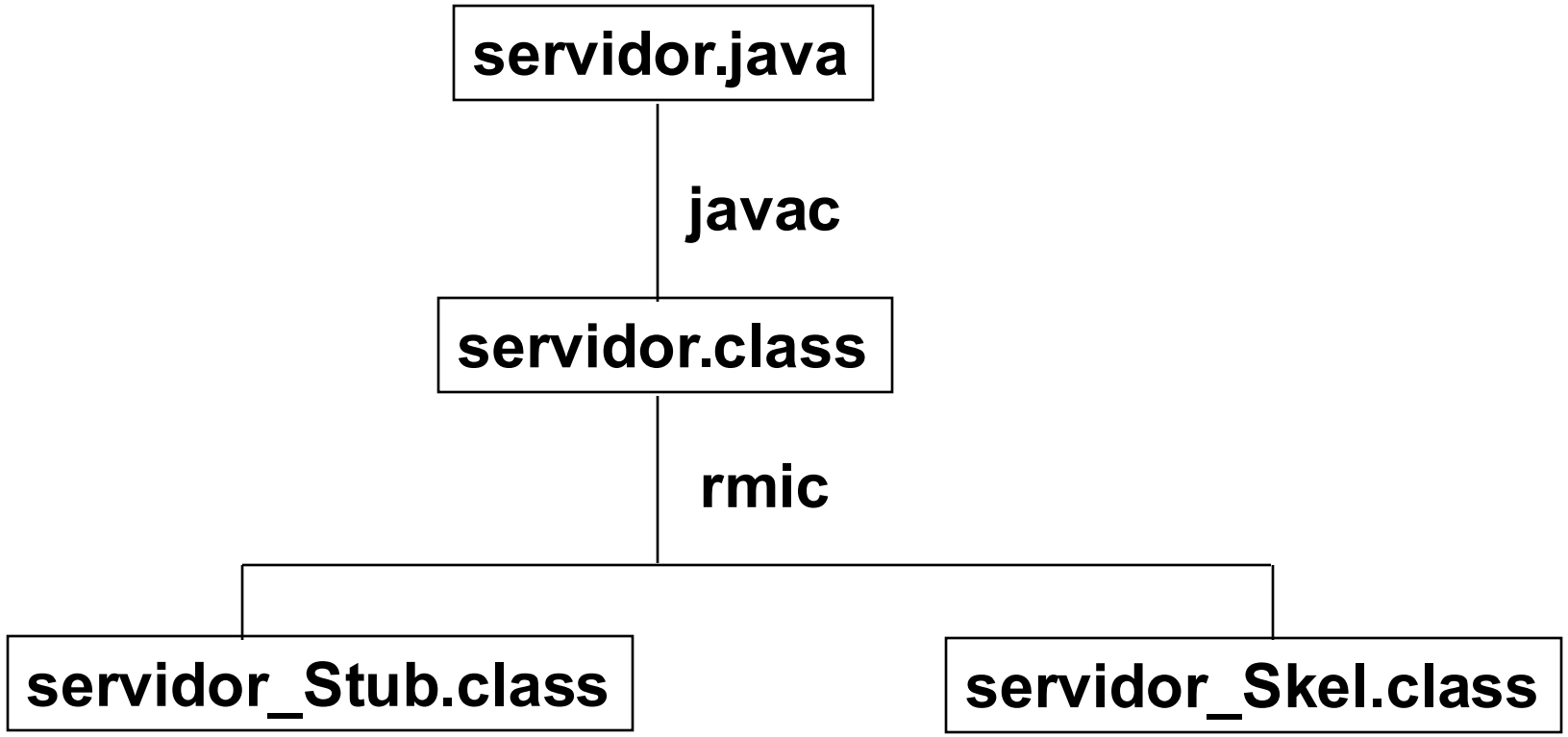
- Não basta apenas compilar e executar o programa anterior.
- Toda a compilação e execução necessita de um ambiente dado pela seguinte sequência:
  - Compilar o arquivo .java
  - Chamar o aplicativo rmic para gerar o Stub e Skel
  - Ativar o controlador de registros (rmiregistry)
  - Chamar o interpretador com o servidor compilado

Stubs e Skels

- O cliente, quando invoca remotamente um método, não conversa diretamente com o objeto remoto, mas com uma implementação da interface remota chamada stub, que é enviada ao cliente. O stub, por sua vez, passa a invocação para a camada de referência remota.

- Esta invocação é passada para um skel (esqueleto) , que se comunica com o programa servidor.

Compilação do exemplo anterior



Execução do servidor

- O primeiro passo antes de executar o servidor é ativar uma espécie de servidor de nomes de servidores que atendem solicitações de métodos remotos. Isto é feito chamando-se o programa rmiregistry. Este programa pode estar ouvindo portas específicas, como por exemplo:

  % rmiregistry 2048 &

- Uma vez que este programa está executando, pode-se chamar o interpretador java para o arquivo servidor.class.

Programação do cliente

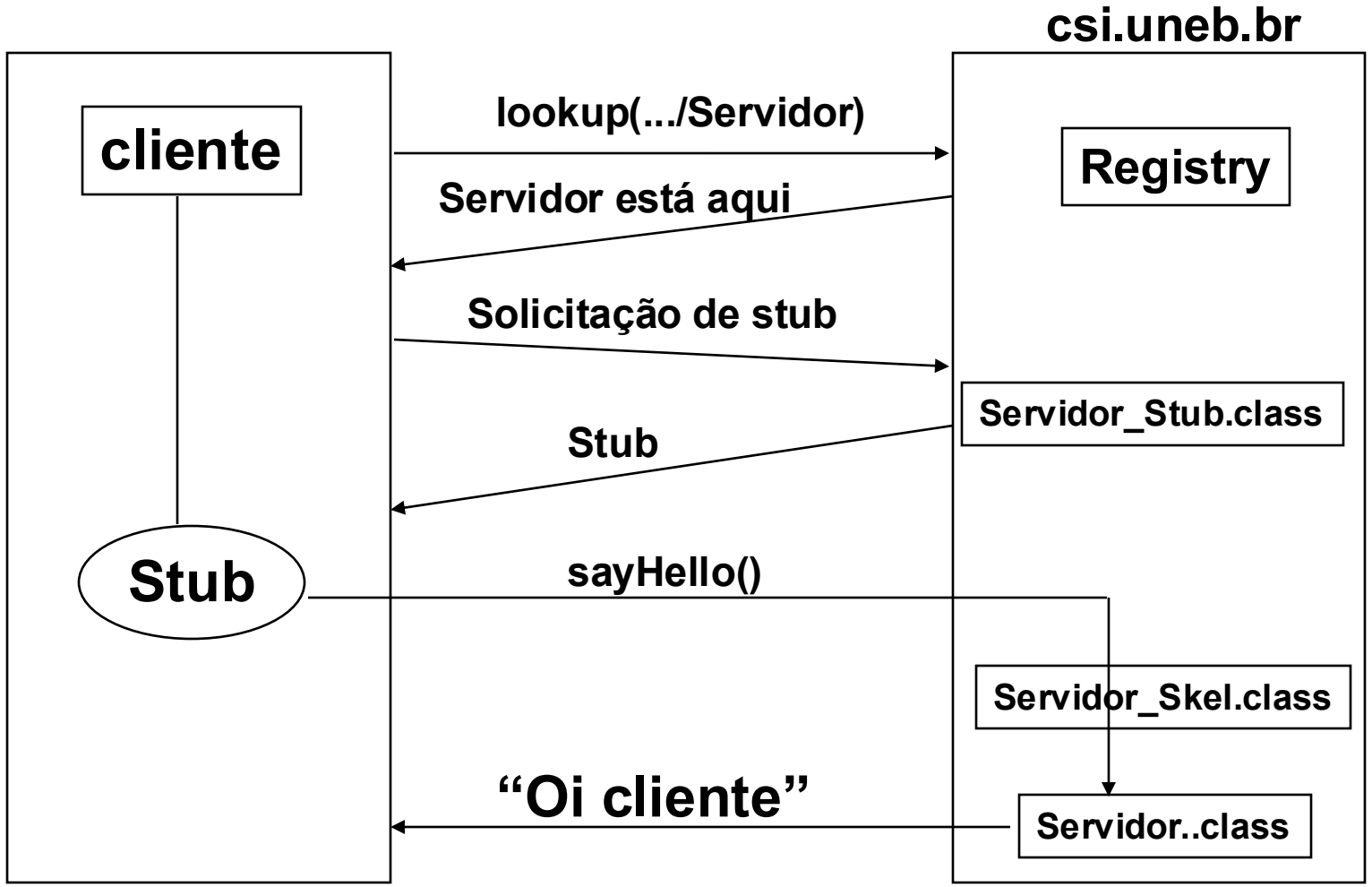
- O primeiro passo de implementação de um cliente que quer invocar remotamente método é obter o stub do servidor remoto. A localização deste stub é feita com o método lookup(endereço).

- Este método devolve uma referência remota do objeto, através do envio do stub.

Exemplo

```
import  java.rmi.*;
class cliente{
    public static void main(String args[]){
        try{
            Servidor serv = (Servidor)
                            Naming.lookup(“rmi://csi.uneb.br:2048
                            /ServidorHello”);
            String retorno=serv.sayHello();
        }
        catch(Exception e);
    }
}
```

Esquema da chamada



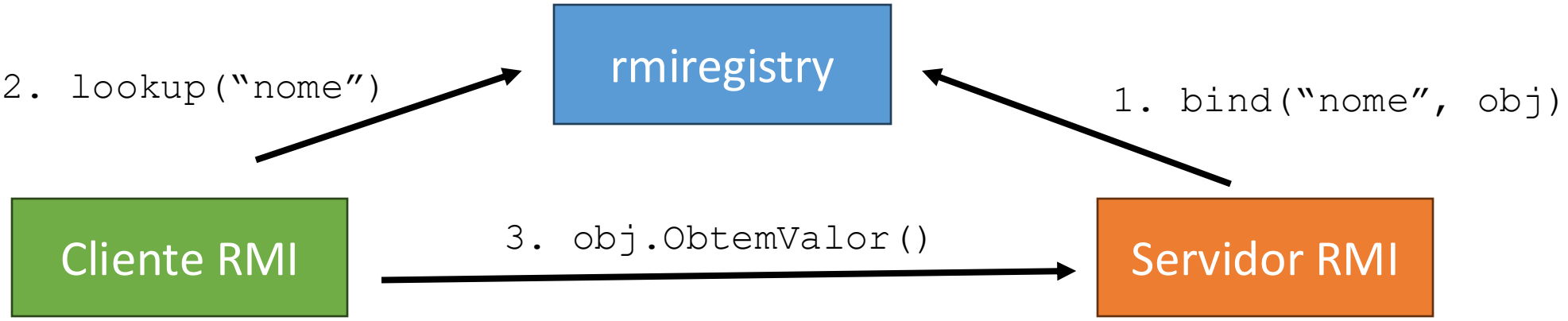
JAVA RMI

- Serviço de registro pode ser inicializado na linha de comando:

```
rmiregistry 12345 &
```

- Ou dentro do código em Java:

```
int PORTA = 12345
Registry registro = LocateRegistry.createRegistry(PORTA);
```



Classe Hello

```
public interface Hello extends Remote{
    String sayHello() throws RemoteException;
}
```

Classe Server

```
public class Server extends UnicastRemoteObject implements Hello{
   public Server() throws Exception{
   }
   public String sayHello() throws RemoteException{
      return "Hello, world";
   }
   public static void main(String args[]) {
      try {
         LocateRegistry.createRegistry(1099);
         Server obj = new Server();
         Registry registry = LocateRegistry.getRegistry();
         registry.rebind("Hello", obj);
         System.out.println("Server ready");
      }catch(Exception e) {
         System.out.println("Server Exception" + e.toString());
         e.printStackTrace();
      }
   }
}
```

Cliente

```
public class Cliente {
   private Cliente() {}
   public static void main(String args[]) {
      try {
         Registry registry = LocateRegistry.getRegistry("localhost");
         Hello stub = (Hello) registry.lookup("Hello");
         String response = stub.sayHello();
         System.out.println("Response: " + response);
      }catch(Exception e) {
         System.out.println("Cliente exception" + e.toString());
         e.printStackTrace();
      }
   }
}
```

Exercicio

- Elabore um sistema RMI de Calculadora com as quatros operações basicas (soma, subtração, multiplicação e divisão) usando dois números.

- Elabore um sistema RMI de votação eletrônica distribuída, onde os eleitores podem votar remotamente em candidatos e os votos são contabilizados de forma segura e precisa pelo servidor. Garanta autenticação e autorização adequadas para garantir a integridade do processo eleitoral.