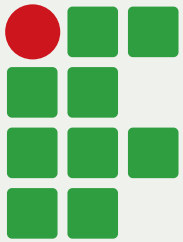
Chamada de Procedimento Remoto

STD29006 – Engenharia de Telecomunicações

Prof. Emerson Ribeiro de Mello

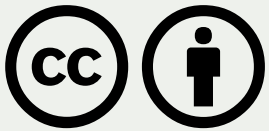
mello@ifsc.edu.br



INSTITUTO FEDERAL
Santa Catarina

Câmpus
São José

Licenciamento



Motivação para RPC

Dificuldades na comunicação entre processos via sockets

Cliente em Java

1. Conecta no servidor
2. Lê do teclado dois long
3. Converte long para String
4. Envia as Strings
5. Recebe o resultado

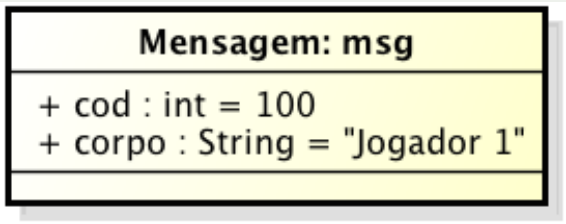
Servidor em C

1. Recebe via sockets vetor de char
2. Converte char para long
3. Faz a soma dos dois long
4. Devolve resultado como vetor de char

Motivação para RPC

Dificuldades na comunicação entre processos via sockets

- **Comunicação** entre processos clientes e servidores por meio de **troca explícita de mensagens**
  - Ex: Aplicação Java enviando objetos `Mensagem`

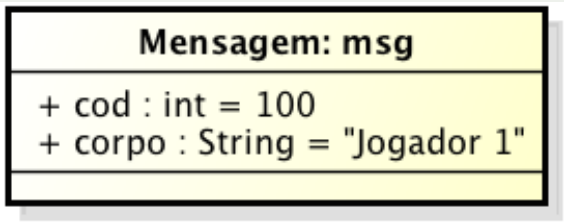



```
1 // send
2 saida.writeObject(msg);
3 //receive
4 Mensagem resp = entrada.readObject();
```

Motivação para RPC

Dificuldades na comunicação entre processos via sockets

- **Comunicação** entre processos clientes e servidores por meio de **troca explícita de mensagens**
  - Ex: Aplicação Java enviando objetos `Mensagem`




```
1 // send
2 saida.writeObject(msg);
3 //receive
4 Mensagem resp = entrada.readObject();
```

Não provê a transparência de acesso, requisito importante para SD

- *"...Esconder as diferenças para representação dos dados..."*
- *"...Como os dados serão representados em diferentes arquiteturas..."*

Representação dos dados

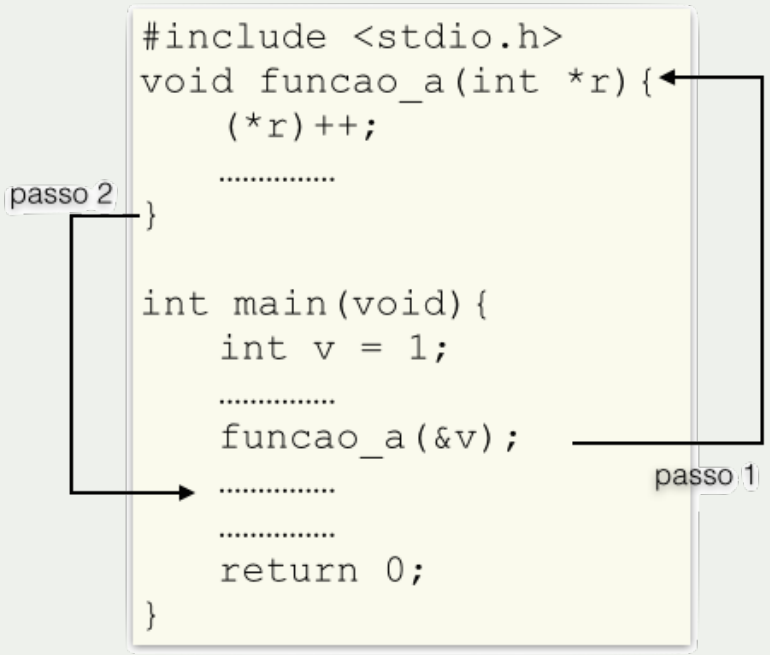
Necessidade de um padrão de codificação para permitir a comunicação entre sistemas heterogêneos

- Formato de representação com **tipo de variável implícito**, sendo que somente os valores são transmitidos
  - Exemplo: XDR (*eXternal Data Representation*)
- Formato de representação com **tipo de variável explícito**, sendo que o tipo de cada valor também é transmitido
  - Exemplos: ASN.1, XML, JSON e Google Protocol Buffers

Serialização dos dados

- **Serialização** traduz estruturas de dados em um formato que possa ser armazenado em disco ou transportado pela rede
  - dados são convertidos para um vetor de bytes
- **Marshalling** semelhante a serialização, porém consiste na transformação da representação de um objeto em memória em formato que possa ser armazenado em disco ou transportado pela rede

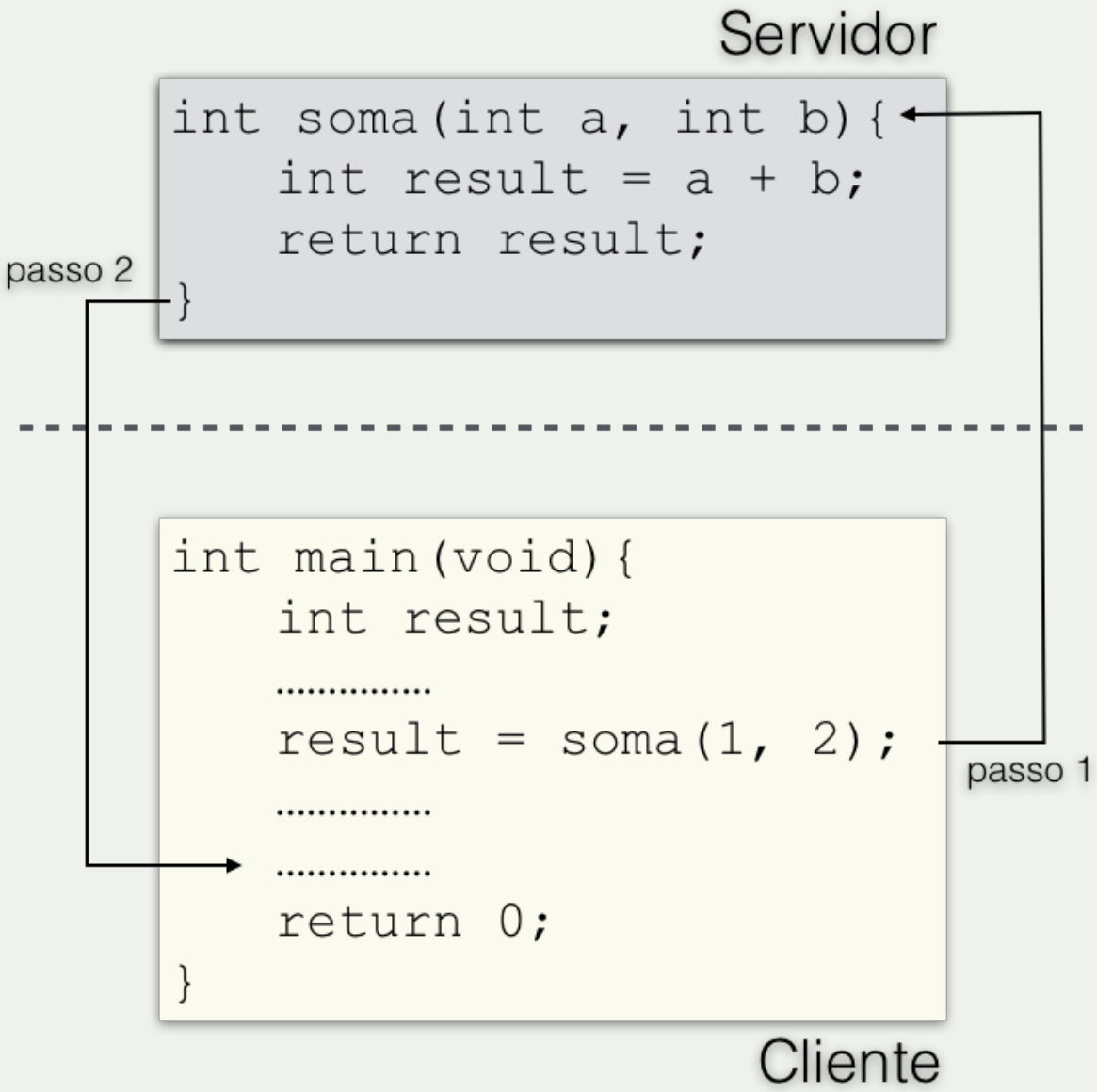
Inspiração para RPC: Chamada de procedimentos em C



http://docente.ifsc.edu.br/mello/livros/linguagem-c

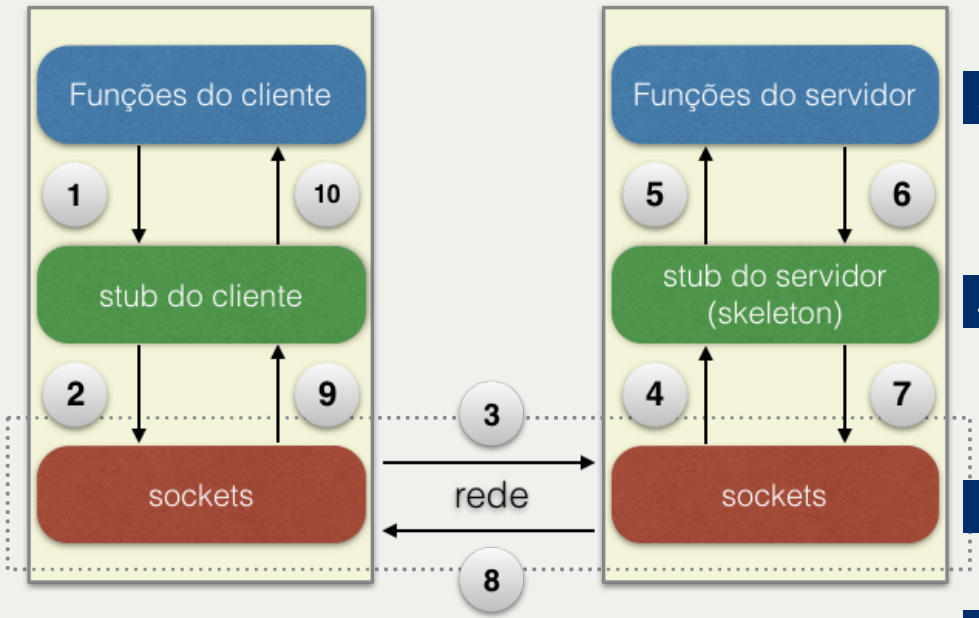
- **Transferência de controle e dados**
  - Lista de parâmetros e tipos de retorno permitem a comunicação entre funções
- **Linguagem C: passagem de parâmetros pode ser por**
  - Valor
    - Função que fora invocada cria uma cópia local da variável
  - Referência
    - Função que fora invocada recebe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

Inspiração para RPC: Chamada de procedimentos em C



Chamada de procedimento remoto – RPC

- Processos locais realizam chamadas de **procedimentos** que estão **localizados em outras máquinas**



1. Cliente invoca funções do stub
2. Stub serializa dados
3. Dados são transmitidos via sockets
4. Dados recebidos são entregues para o stub
5. Dados são deserializados
6. Resposta segue caminho inverso

Stub e Skeleton

- O ***stub* do cliente** atua como um proxy para as funções remotas do servidor
  - Faz a serialização dos parâmetros
  - Envia a mensagem pela rede
  - Aguarda pela resposta do servidor
  - Faz a deserialização e retorna a resposta para o cliente
  - Diante de problemas, dispara exceções
- O ***skeleton* do servidor** fica esperando pedidos dos clientes e ao receber, invoca a função do servidor
  - Faz a deserialização, etc.

Tudo isso fica transparente para o programador (do cliente e do servidor)

Implementação ONC RPC

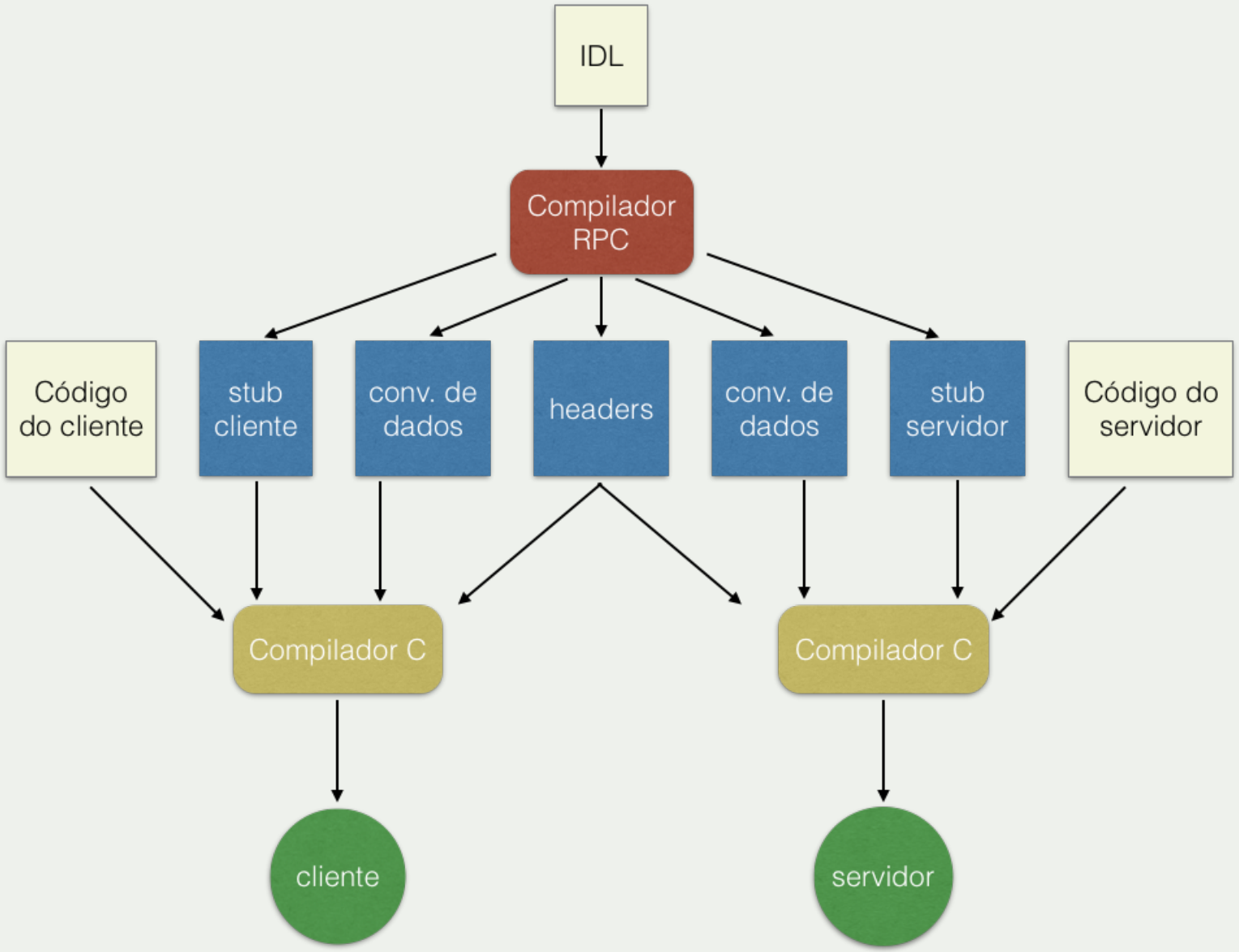
- Em 1980 a **SUN RPC** foi a primeira implementação
  - Usada no SUN Network File System (NFS)
- Padronizada pela Open Network Computing (**ONC**)
  - Presente no SunOS, BSDs, Linux e macOS
- Microsoft faz uso do seu próprio RPC (DCOM + Object RPC)
  - Surgiu em 1996 com o Windows NT 4.0

Linguagem de Descrição de Interfaces

Interface Description Language (IDL)

- Linguagem para descrever interface de componentes de software
- Descrição de interface independente de qualquer linguagem de programação e arquitetura de máquina
- Possibilita a comunicação entre componentes escritos em diferentes linguagens
- **RPC faz uso de IDL** para descrever os procedimentos remotos expostos por um servidor

Passos para escrever cliente e servidor com ONC RPC



Como escrever programas com ONC RPC

- Descrever a interface dos procedimentos (IDL RPC)

```
1  // Dois procedimentos que recebem uma struct, faz a computacao e
       retorna o resultado (int)
2
3  struct operandos{
4      int a; int b;
5  };
6  program CALCULADORA_PROG{
7      version CALCULADORA_VERSAO{
8      int SOMA(operandos) = 1;
9      int SUB(operandos) = 2;
10     } = 1; // versao do procedimento
11 } = 0x20000001; // versao do programa
```

- Compilar IDL para gerar stubs para cliente e servidor

```
1 rpcgen calculadora.x
```

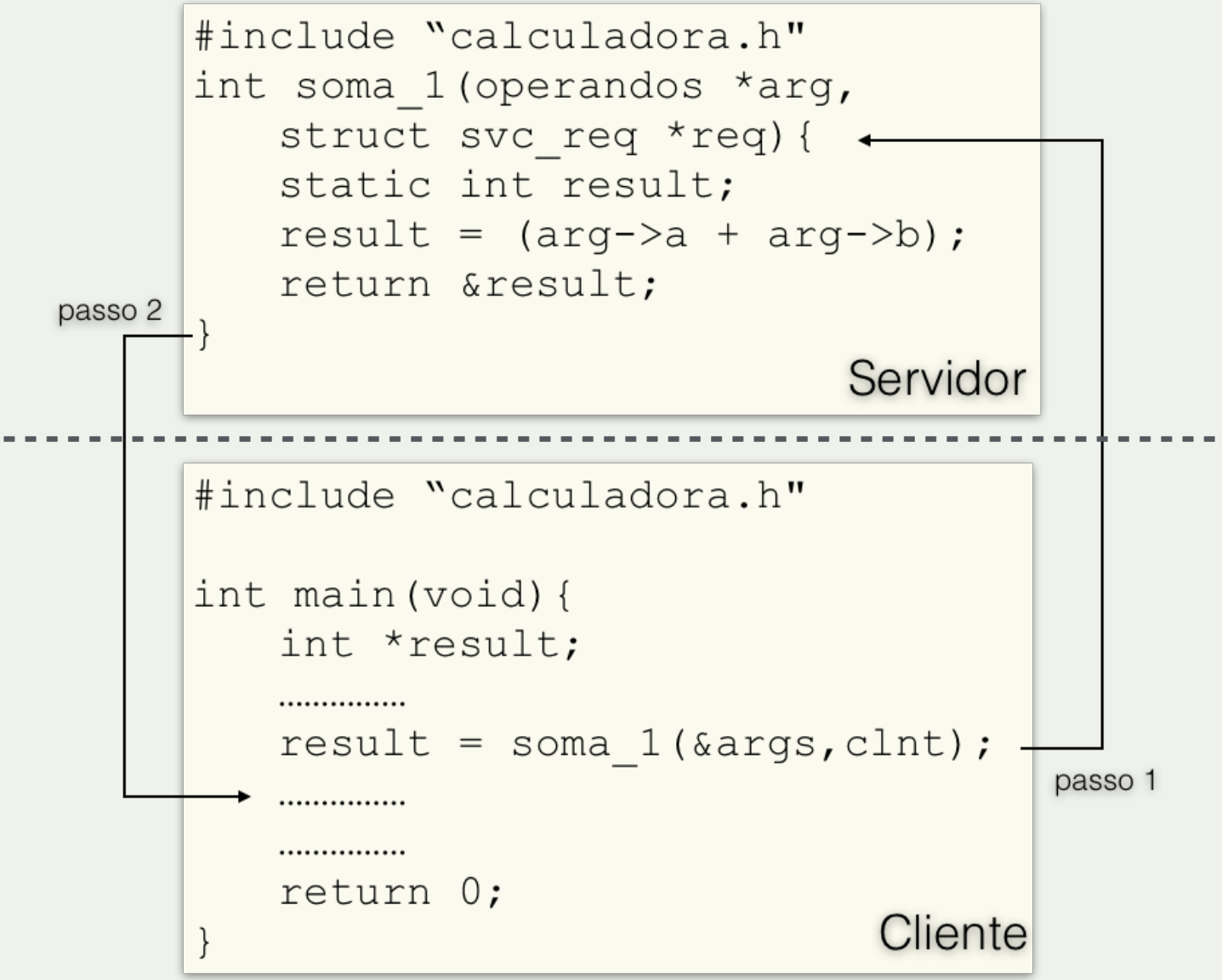

Servidor

```c
#include "calculadora.h"
int * soma_1(operandos *argp, struct svc_req *rqstp){
   static int result;
   result = argp->a + argp->b;
   return &result;
}
```

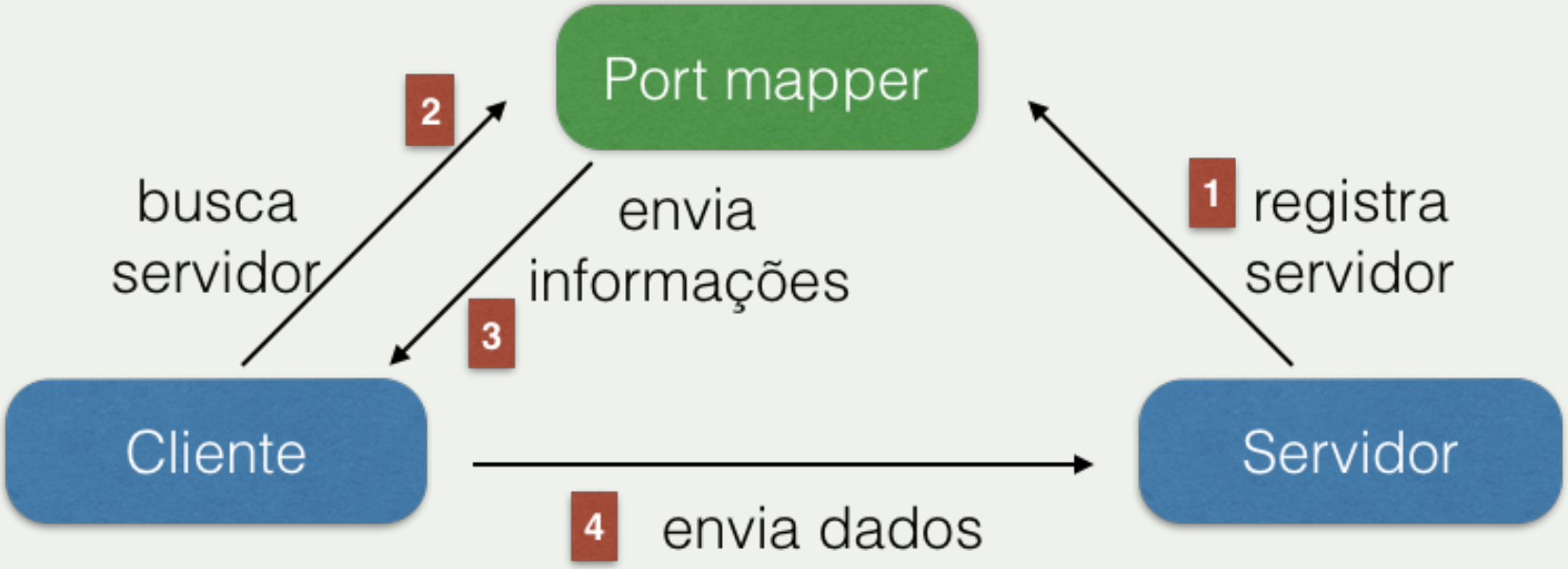
Cliente

```c
#include "calculadora.h"
int main(void){
   char *host = "127.0.0.1";
   CLIENT *clnt;
   operandos argumentos;
   int *resultado;

   clnt = clnt_create(host, CALCULADORA_PROG, CALCULADORA_VERS, "udp");

   argumentos.a = 1;
   argumentos.b = 2;

   resultado    = soma_1(&argumentos,clnt); // invocando procedimento remoto
   printf("Resultado da soma: %d", *resultado);
}
```


Chamada de procedimento remoto – RPC



Em qual porta o servidor está ouvindo?



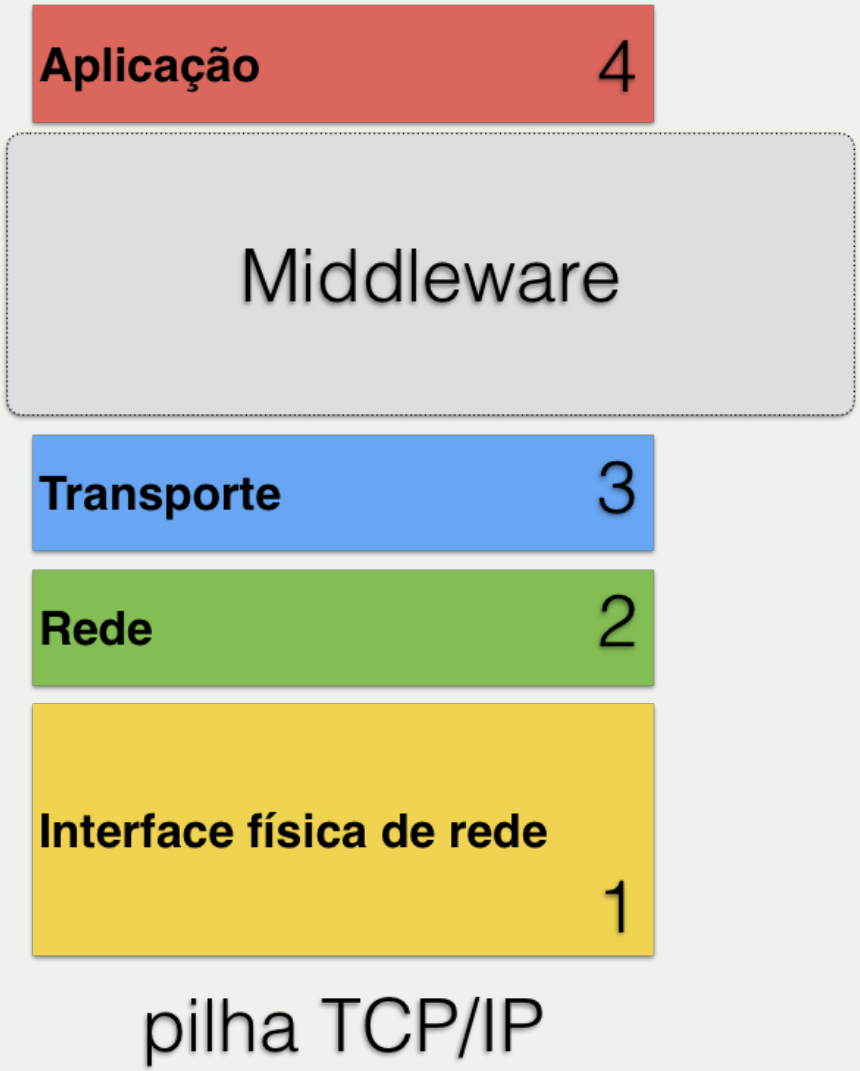
- **Port mapper** é um serviço de descoberta que permite aos clientes descobrirem em qual porta TCP/UDP o servidor está ouvindo
  - Executa por padrão na porta 111 (TCP ou UDP)
  - Versões 3 e 4 são chamadas de `rpcbind protocol`
- Faça o teste no teu S.O. e veja quais são os serviços registrados

```
1 rpcinfo -p
```

Resumo sobre RPC

- **Camada 6** – Representação e serialização dos dados
- **Camada 5** – Gerenciamento de conexão

Resumo sobre RPC



- **Camada 6** – Representação e serialização dos dados
- **Camada 5** – Gerenciamento de conexão

Resumo sobre RPC

- **SOAP** – https://www.w3.org/TR/soap12-part1/
  - Protocolo baseado em XML e usado com *Web Services*
  - Representação dos dados em XML
  - Geralmente usa o protocolo HTTP como transporte
- **gRPC** – https://grpc.io
  - Representação dos dados, descrição do serviço e protocolo com o *Protocol Buffers*
  - Faz uso do protocolo HTTP/2 como transporte
  - Geralmente empregado na arquitetura de microserviços

Remote Method Invocation (Java RMI)

Objetos distribuídos e invocação de métodos remotos

- **Chamada Remota de Procedimento – RPC**
  - Permite que processos locais realizem chamadas de procedimentos que estão localizados em outras máquinas

Objetos distribuídos e invocação de métodos remotos

- **Chamada Remota de Procedimento – RPC**
  - Permite que processos locais realizem chamadas de procedimentos que estão localizados em outras máquinas
- **Invocação de Métodos Remotos – Java RMI**
  - **Semelhante ao RPC**, porém voltado para **objetos distribuídos**
  - Estende o conceito de referência de objetos para um ambiente global distribuído
    - Objeto residente em um processo pode invocar métodos de um objeto residente em um outro processo
  - Oferece *distributed garbage-collection*

RPC *vs* RMI

- **Semelhanças entre RPC e RMI**

- **Diferenças entre RCP e RMI**

RPC *vs* RMI

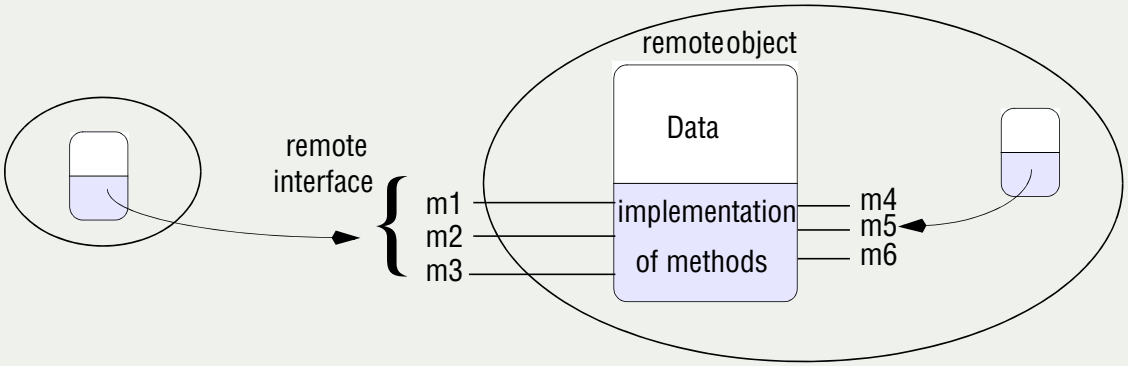
- **Semelhanças entre RPC e RMI**
  - Fazem uso de linguagem de descrição de interfaces (IDL)
  - São construídos sobre protocolos pedido e resposta
  - Oferecem o mesmo nível de transparência ao desenvolvedor
- **Diferenças entre RCP e RMI**

RPC *vs* RMI

- **Semelhanças entre RPC e RMI**
  - Fazem uso de linguagem de descrição de interfaces (IDL)
  - São construídos sobre protocolos pedido e resposta
  - Oferecem o mesmo nível de transparência ao desenvolvedor
- **Diferenças entre RCP e RMI**
  - Com o RMI o desenvolvedor poderá usufruir das facilidades da POO
    - objetos, classes, herança, . . .
  - Objetos possuem uma referência única em todo o sistema
    - Objetos podem ser passados por referência (e não somente por valor)

Modelo de objetos distribuídos

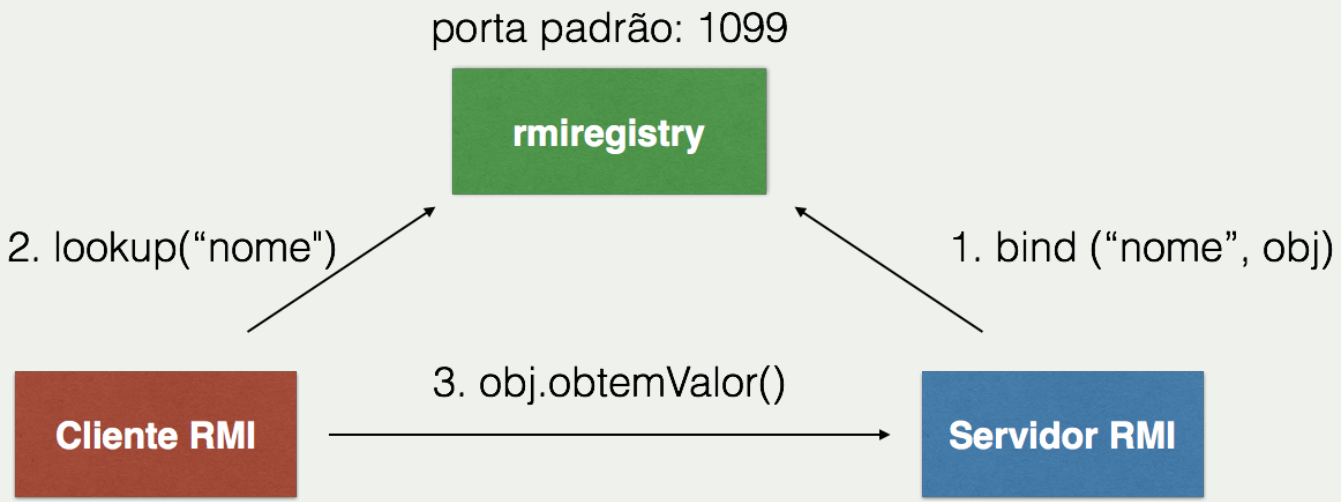
- **Referência de objeto remoto**
  - um objeto só poderá invocar métodos de outros objetos que este possuir a referência
- **Interface remota**
  - cada objeto remoto possui uma interface que especifica quais de seus métodos poderão ser invocados remotamente
  - objetos dentro de um mesmo processo poderão invocar os métodos remotos e também os locais



Entidades participantes

- **Servidor**
  - Processo que oferta objeto remoto
- **Cliente**
  - Processo que invoca um método de um objeto remoto
- **Serviço de registro**
  - Serviço de nomes que associa nomes a objetos
  - Ex: `rmi://host:port/pathName`

Java RMI



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

```
1 rmiregistry 12345 &
```

- ou dentro do código Java do processo servidor:

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

Remote Method Invocation (Java RMI)

Exemplo: Um Contador

Descrição da interface

```
1 package std;
2
3 public interface ContadorDistribuido extends Remote{
4
5     public void incrementa() throws RemoteException;
6
7     public int obtemValor()  throws RemoteException;
8
9 }
```

Essa interface Java deve ser compartilhada entre servidor e clientes. Deve-se manter o mesmo nome de pacote Java.

Implementação da Interface no Servidor

```
1  package std;
2
3  public class Contador implements ContadorDistribuido{
4      public int valor = 0;
5
6      public void incrementa() throws RemoteException {
7          this.valor++;
8      }
9
10     public int obtemValor() throws RemoteException {
11         return this.valor;
12     }
13 }
```


Implementação do Servidor

```java
package std;

public class Servidor{

  public static void main(String[] args) throws Exception{
    Contador c = new Contador(); // criando objeto

    ContadorDistribuido stub =
        (ContadorDistribuido) UnicastRemoteObject.exportObject(c, 0);

    // Criando registry
    System.setProperty("java.rmi.server.hostname","127.0.0.1");
    Registry registro = LocateRegistry.createRegistry(12345);

    // Registrando objeto com o nome MeuContador
    registro.bind("MeuContador", stub);

    System.out.println("Servidor pronto!");
  }
}
```


Implementação do Cliente

```
1  package std;
2
3  public class Cliente{
4
5    public static void main(String[] args) throws Exception{
6
7      // Obtendo referência para o registro (tem que conhecer IP e porta)
8      Registry registro = LocateRegistry.getRegistry("127.0.01", 12345);
9
10     // Obtendo referência para o objeto instanciado pelo Servidor
11     ContadorDistribuido stub =
12           (ContadorDistribuido) registro.lookup("MeuContador");
13
14     // invocando métodos do objeto remoto
15     System.out.println("valor: " + stub.obtemValor());
16     stub.incrementa();
17     System.out.println("valor: " + stub.obtemValor());
18   }
19 }
```


Aulas baseadas em

TANENBAUM, ANDREW S.; STEEN, MAARTEN VAN

SISTEMAS DISTRIBUIDOS: PRINCÍPIOS E PARADIGMAS

COULOURIS, GEORGE; KINDBERG, TIM; DOLLIMORE, JEAN

SISTEMAS DISTRIBUÍDOS: CONCEITOS E PROJETO

PAUL KRZYZANOWSKI

DISTRIBUTED SYSTEMS – RUTGERS UNIVERSITY