Sistemas Distribuídos

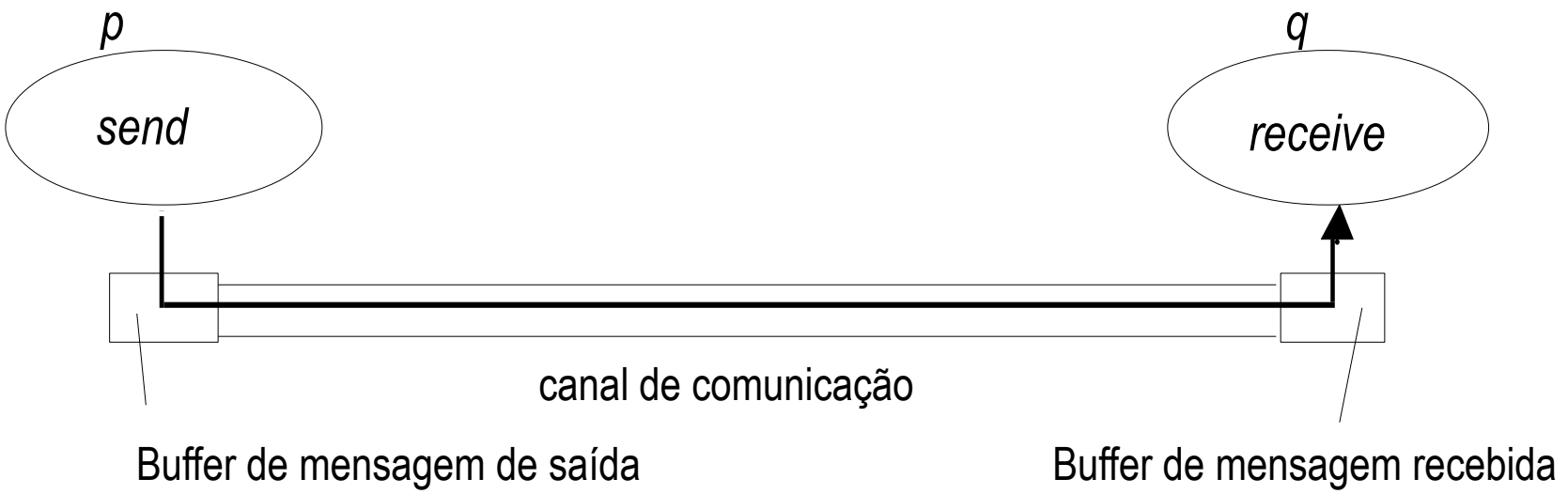
Prof.: Diego Américo Guedes

Instituto de Informática
Universidade Federal de Goiás

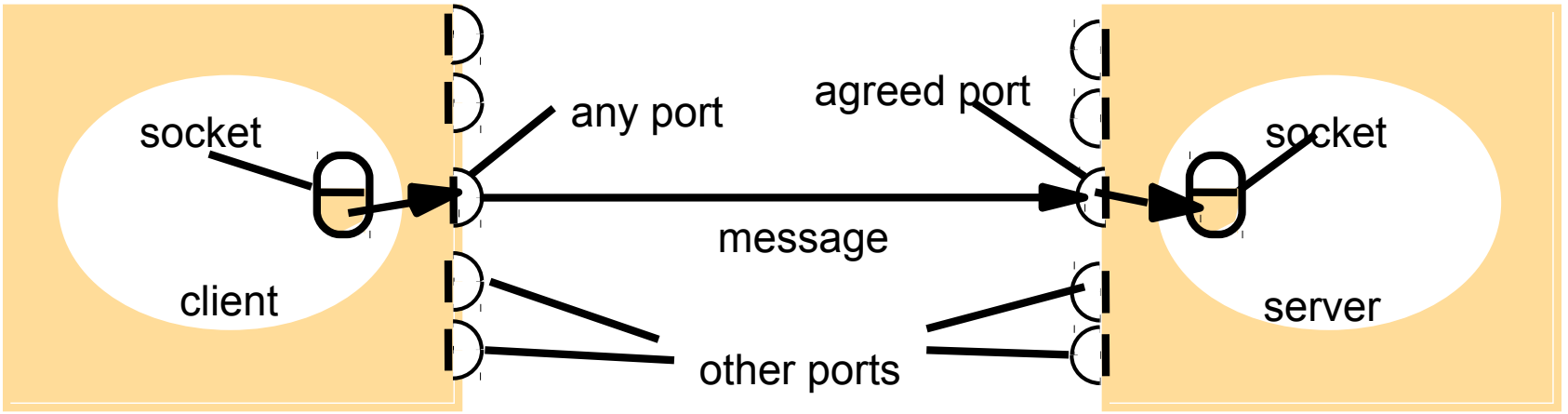
Sumário

Sumário

Aula anterior
Primitivas de Comunicação



Aula Anterior
Socket



- **Todo socket está vinculado a uma porta e a um endereço IP**
- **Há 2^{16} números de portas disponíveis**
- **Os processos podem usar o mesmo socket para ler e enviar mensagens**
- **Cada socket é associado a um protocolo de transporte (UDP ou TCP)**

Aula Anterior
Aula de laboratório

- **Aplicação Servidor:**
  - Recebe como argumento a porta em que vai atender os clientes
  - Recebe uma requisição de um cliente via TCP
  - Imprime a mensagem recebida, IP e porta do cliente na tela
  - Envia uma resposta com o seguinte formato:
    - “Mensagem recebida com sucesso!”
- **Aplicação Cliente:**
  - Abre conexão TCP com servidor na porta informada como argumento
  - Envia a mensagem: “SD é muito divertido!”
  - Recebe resposta e a imprime na tela
  - Fecha conexão

Objetivos

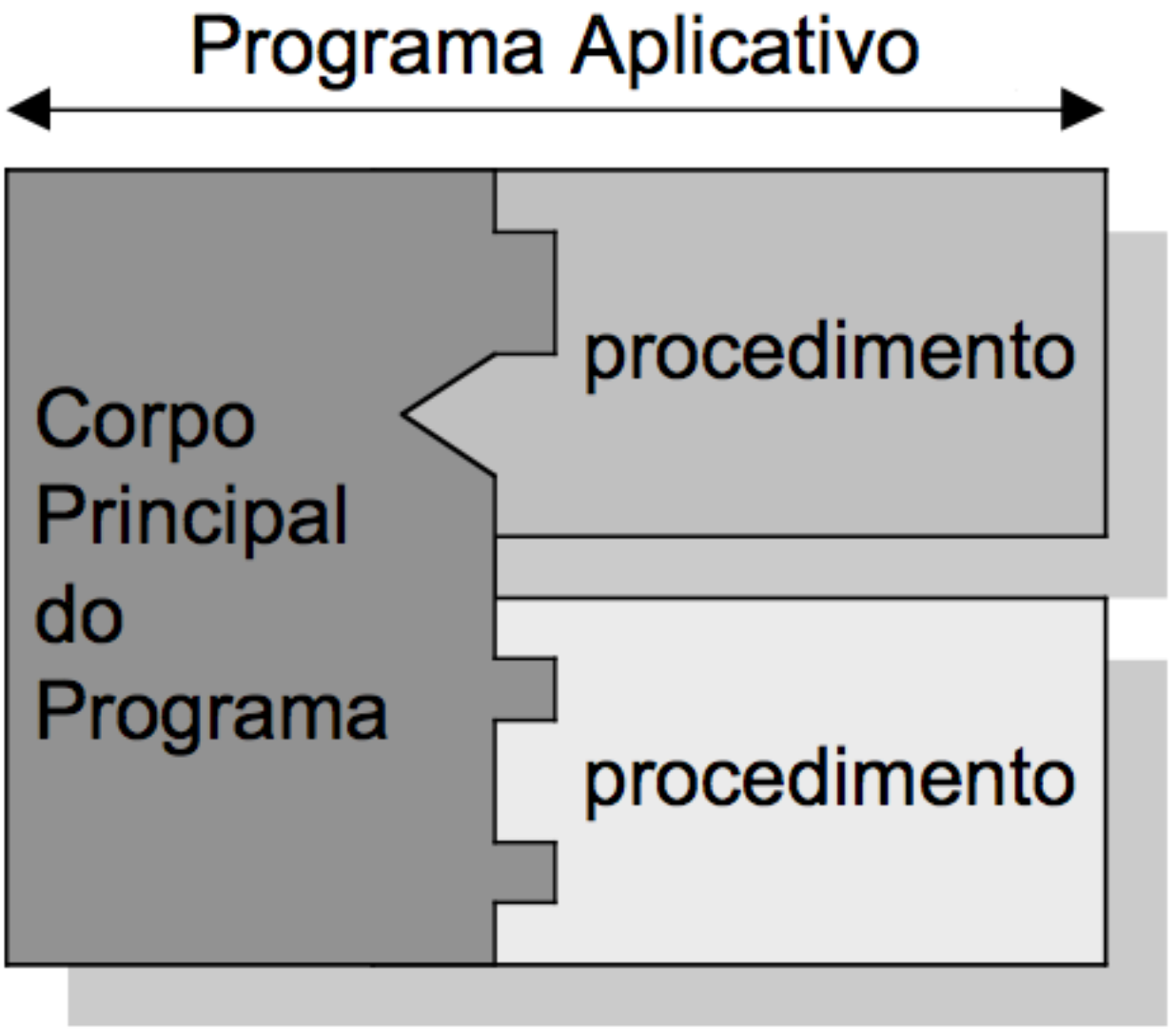
- Ferramentas de programação para Sistemas Distribuídos
  - *Remote Procedure Call* - RPC
  - Objetos Distribuídos

Sumário

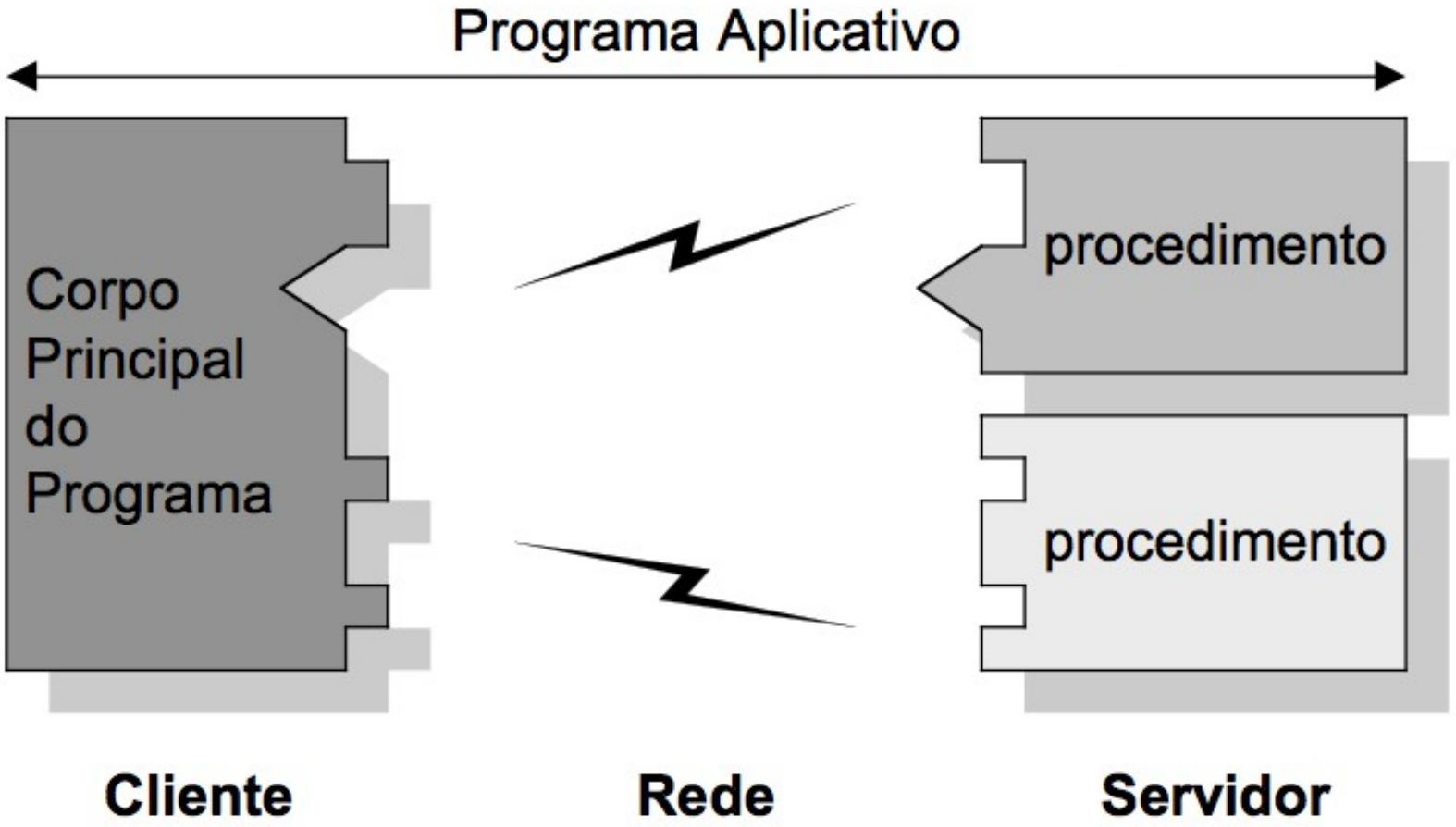
O que há de errado com Socket?

- A troca de mensagem entre os componentes do sistema é feita através de uma interface de entrada e saída
- Essa, no entanto, não é a forma como os componentes de programas não distribuídos interagem.
- A **chamada de procedimento** é o modelo de interface padrão de sistemas não distribuídos

Chamada Local



Chamada Remota



Chamada Remota de Procedimento (RPC)

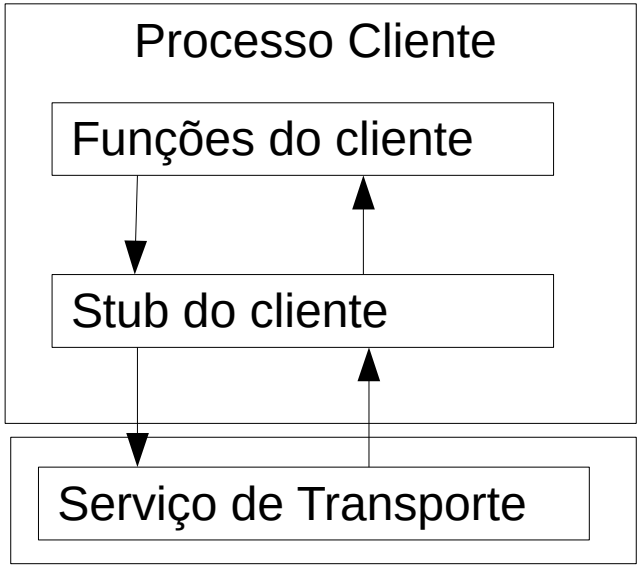
- Criada em 1984 com o objetivo de tornar a computação distribuída parecida com a computação centralizada
- Permite que processos possam invocar procedimentos/funções de outros processos (numa mesma máquina ou não)

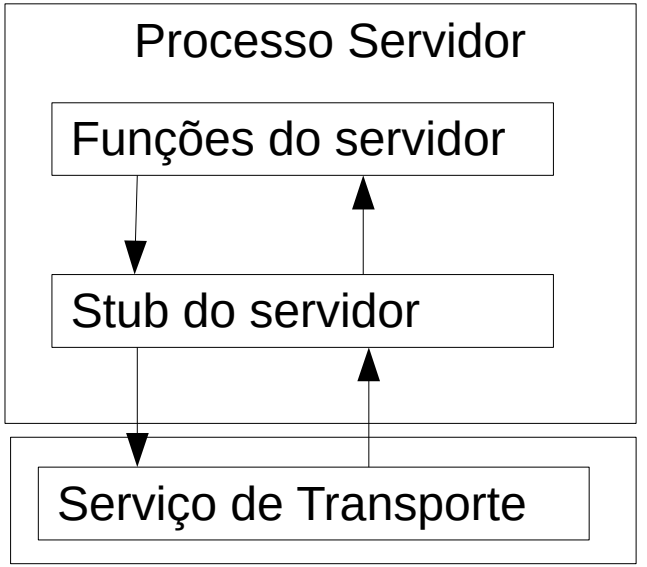
Chamada Remota de Procedimento (RPC)

- Chamada Remota
  - Mesma semântica da chamada local:
    - chamador bloqueia até que o código chamado termine
  - Simula uma chamada local com procedimentos locais e socket
  - É uma construção de linguagem (language level construct) e não de sistema operacional

Chamada Remota de Procedimento (RPC)

- Como é implementada?
  - Funções *stub* são geradas localmente, em cada ponto da comunicação
  - Uma função *stub* se parece com a função que se deseja chamar, mas contem código para enviar e receber mensagens pela rede





Chamada Remota de Procedimento (RPC)

- Etapas
  - O cliente chama um procedimento local, a função *stub*.
  - A função *stub* do cliente empacota os argumentos e constrói a mensagem a ser enviada pela rede (*marshalling*)
  - A função *stub* faz uma chamada ao sistema (socket) para enviar a mensagem para o outro processo
  - A função *stub* do servidor recebe a mensagem e desempacota os parâmetros (*unmarshalling*)
  - A função *stub* do servidor chama o procedimento local
  - A função *stub* do servidor empacota o resultado e faz uma chamada ao sistema para enviar o resultado para o outro processo
  - A função *stub* do cliente recebe a mensagem do servidor, desempacota e retorna para o cliente.
  - O cliente continua sua execução

Questões relacionadas a RPC

- Representação de dados
  - Representação externa de dados: um protocolo ou conjunto de regras para representar tipos de dados

Questões relacionadas a RPC

- Que tipo de protocolo de transporte usar
  - Muitas implementações de RPC permitem o uso apenas de TCP

Questões relacionadas a RPC

- O que acontece se uma falha ocorrer
  - Na chamada remota existe maior oportunidade para falhas (transmissão, cliente, servidor)
  - Esse tipo de falha não ocorre em chamadas locais
  - Portanto, as falhas quebram a transparência da chamada remota.
  - O cliente que faz uma chamada remota deve estar preparado para tratar exceções dessa natureza

Questões relacionadas a RPC

- Qual a semântica de uma chamada remota?
  - A semântica de uma chamada local é simples: o procedimento chamado é executado exatamente uma vez.
  - E na chamada remota?
    - Semântica de invocação talvez: Surge quando não há medidas de tolerância a falhas. Não há retransmissão nem filtragem de duplicata
    - Semântica de invocação pelo menos uma vez: Surge quando há retransmissão mas não há filtragem de duplicata.
    - Semântica no máximo uma vez: Surge quando há retransmissão de requisição, filtragem de duplicata e retransmissão de resposta.

Questões relacionadas a RPC

- RMI Java
  - No máximo uma vez
- CORBA
  - No máximo uma vez
- RPC SUN
  - Pelo menos uma vez

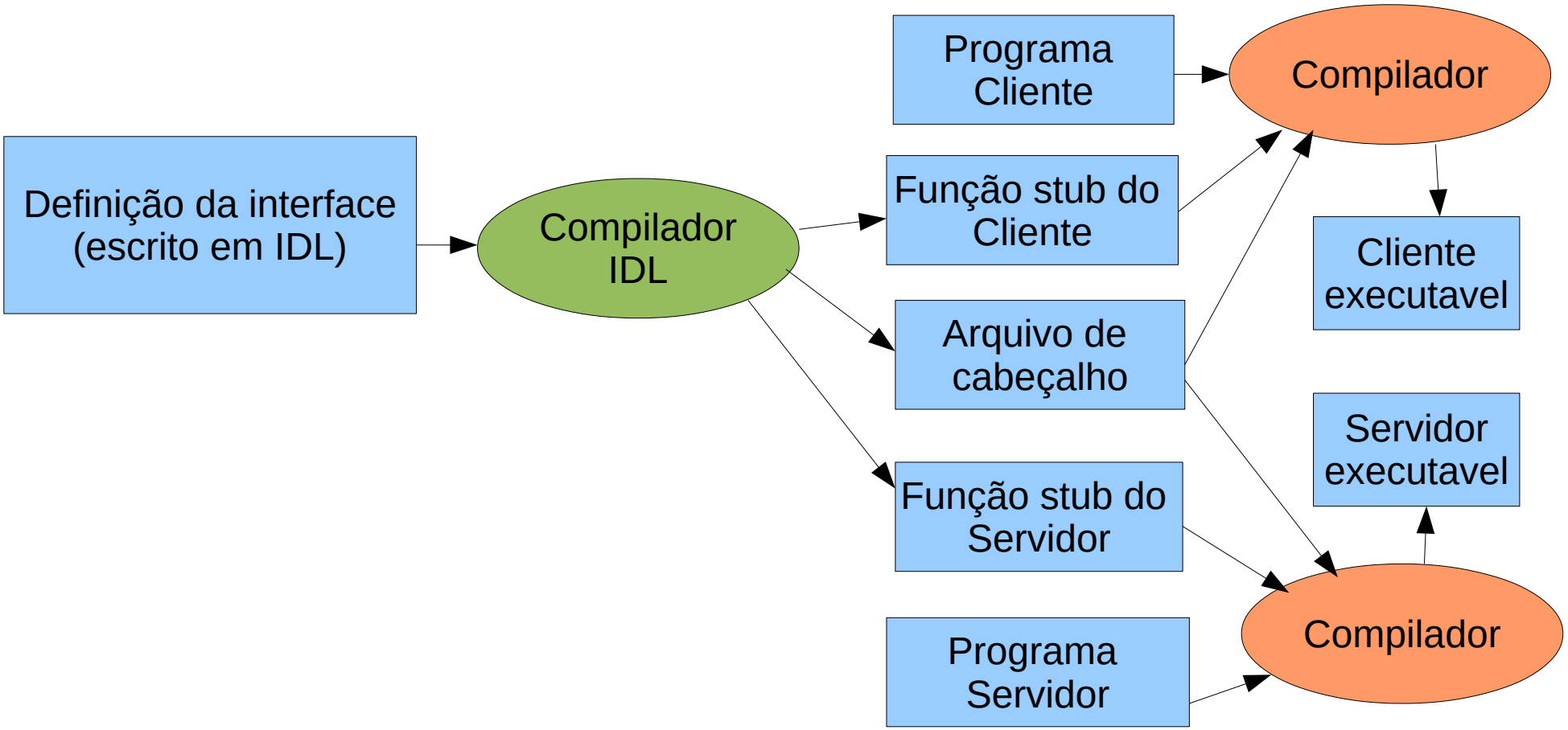
Programando com RPC

- É possível programar com RPC usando linguagens populares, como por exemplo C
- Um compilador separado (a parte do compilador da linguagem) gera as funções *stub* do cliente e do servidor
- Esse compilador toma como entrada a definição do procedimento remoto, escrito em uma linguagem de definição de interface(IDL)

Programando com RPC

- Definição de Interface
  - Declaração conteúdo:
    - O protótipo dos procedimentos que podem ser chamados remotamente
    - Os parâmetros de entrada e retorno do procedimento

Programando com RPC



RPC – Estudo de Caso

- Sun RPC
  - Foi uma das primeiras bibliotecas de RPC
  - Compilador IDL: rpcgen
  - Pode usar UDP e TCP
  - Representação externa de dados: XDR (padrão IETF - Internet Engineering Task Force)
  - Programador
    - Escreve a definição dos procedimentos remotos
    - Implementa o programa cliente
    - Implementa o programa servidor

RPC – Estudo de Caso

- Suponha o seguinte programa idl (add.x)

```
struct intpair {
        int a;
        int b;
};
program ADD_PROG {
        version ADD_VERS {
                int ADD(intpair) = 1; //só aceita um parâmetro. Caso se deseja mais
                                      //parâmetros, deve-se criar uma estrutura
        } = 1; //Número da versão
} = 999; //Número do programa
```

RPC – Estudo de Caso

- Compilando a idl com rpcgen, os seguintes arquivos são gerados
  - add.h : contém a definição do programa, da versão e as declarações dos procedimentos remotos
  - add_svc.c : código C que implementa o stub do servidor
  - add_cln : código C que implementa o stub do cliente
  - add_xdr.c : contém as rotinas de marshalling e unmarshalling

Sumário

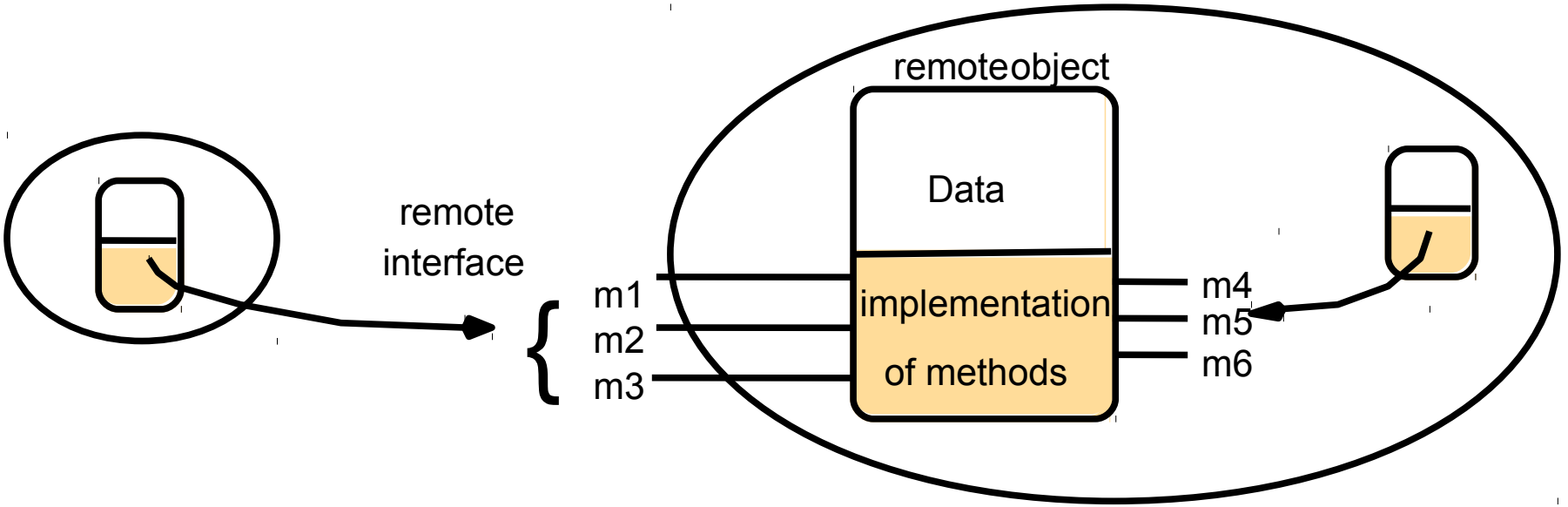
Objetos Distribuídos

- Modelo baseado em objetos para SDs
- Evolução do conceito de RPC
- Principais características
  - Encapsulamento de implementação
  - Interfaces bem definidas
  - Baixo grau de acoplamento entre os componentes
  - Unidades computacionais que podem ser reutilizadas para compor várias aplicações
- Exemplos:
  - CORBA, JAVA RMI, DCOM

Objetos Distribuídos

- Conceitos Fundamentais
  - Objeto remoto: objeto que implementa um ou mais métodos que podem ser invocados remotamente, ou seja, invocados por outros processos
  - Interface remota: **Todo objeto remoto deve implementar uma interface remota.** A interface remota descreve os métodos do objeto que podem ser invocados remotamente
    - CORBA : interfaces remotas são definidas em IDL
    - RMI Java: interfaces remotas são interfaces Java que estendem a interface **Remote**

Objetos Distribuídos

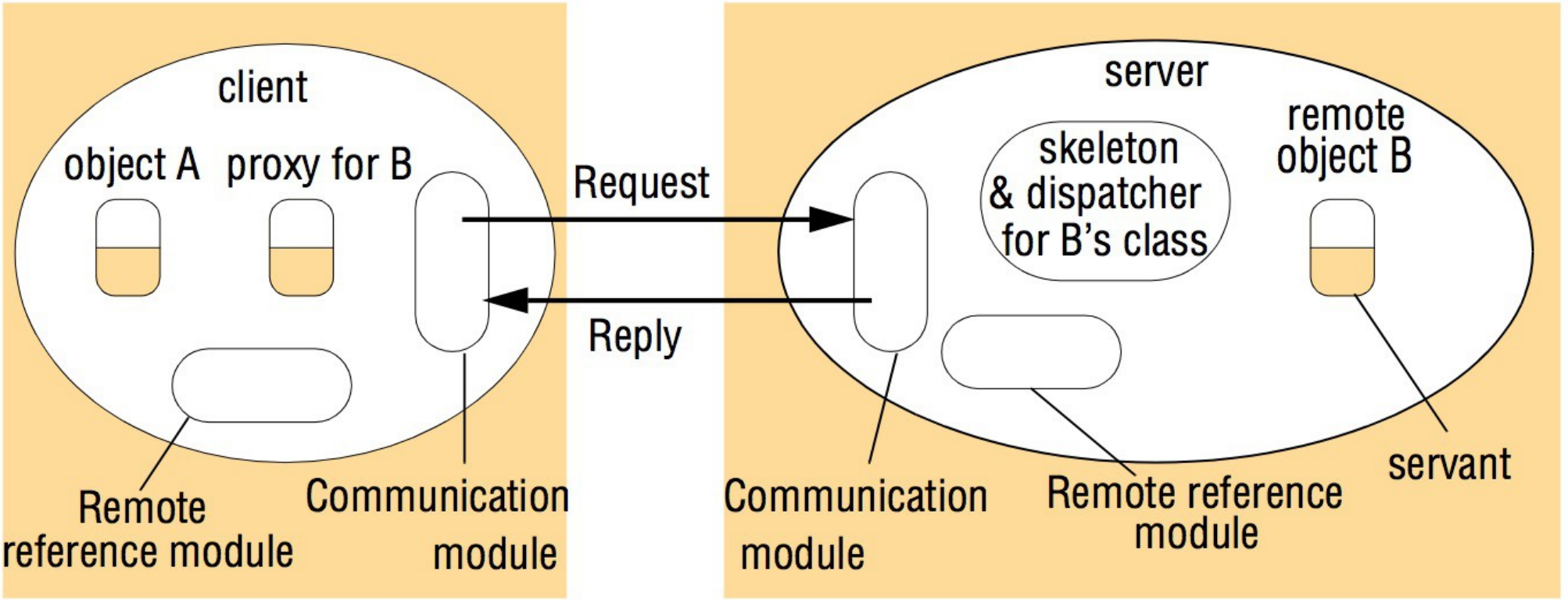


- m1, m2 e m3 podem ser invocados remotamente
- m4, m5 e m6 **não** podem ser invocados remotamente

Objetos Distribuídos

- Conceitos Fundamentais
  - Referência de objeto remoto: é um identificador que pode ser usado por todo o sistema distribuído para se referir a um objeto remoto único em particular.
  - Uma referência de objeto remoto se assemelha a uma referência a objeto local, no sentido que
    - Ela pode ser usada para invocar um método do objeto
    - Ela pode ser passada como argumento ou retorno de métodos

Implementação de Objetos Distribuídos

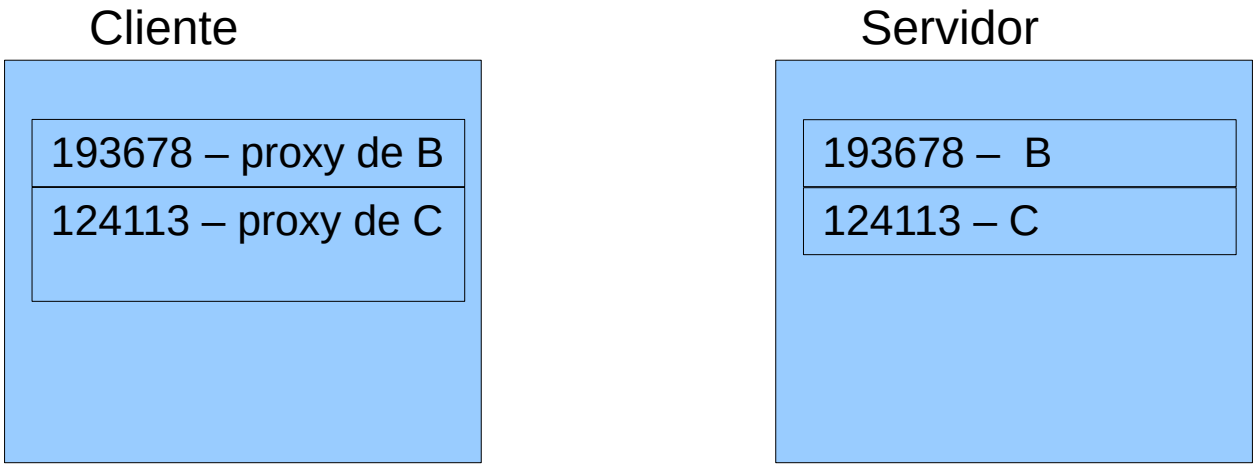


Implementação de Objetos Distribuídos

- Módulo de Comunicação:
  - Cooperam para executar o protocolo requisição-resposta entre o cliente e o servidor.
  - São responsáveis por fornecer uma semântica de invocação específica (talvez, pelo menos uma vez ou no máximo uma vez)
  - No Servidor, seleciona o despachante para a classe do objeto a ser invocado

Implementação de Objetos Distribuídos

- Módulo de referência remota
  - Responsável pela criação das referências de objetos remotas
  - Responsável também pela conversão de referencias de objetos remotas em referencias de objetos locais



Implementação de Objetos Distribuídos

- Servent
  - Objeto de uma classe que implementa uma interface remota
  - Trata as requisições remotas repassadas pelo esqueleto
- Proxy
  - Comporta-se como um objeto local para o invocador
  - Em vez de executar uma invocação local, encaminha uma mensagem para o objeto remoto, que efetivamente implementa o método invocado
  - Responsável pelo marshalling de parâmetros e unmarshalling do resultado
  - Existe um proxy para cada objeto remoto

Implementação de Objetos Distribuídos

- Despachante
  - Recebe a mensagem de requisição do módulo de comunicação e seleciona o método apropriado no esqueleto, repassando a mensagem a esse
- Esqueleto
  - Módulo que simula o objeto remoto no lado do servidor.
  - Implementa o unmarshalling dos argumentos da chamada remota e invoca o método correspondente no Servent.
  - executa o marshalling do resultado em uma mensagem de resposta.
  - Existe um Despachante e um Esqueleto para cada **classe** que implementa uma interface remota

Java RMI

RMI - Invocação de Objetos Remotos

- Java RMI (Remote Method Invocation), é um mecanismo que permite ao usuário, criar aplicações distribuídas utilizando Java.

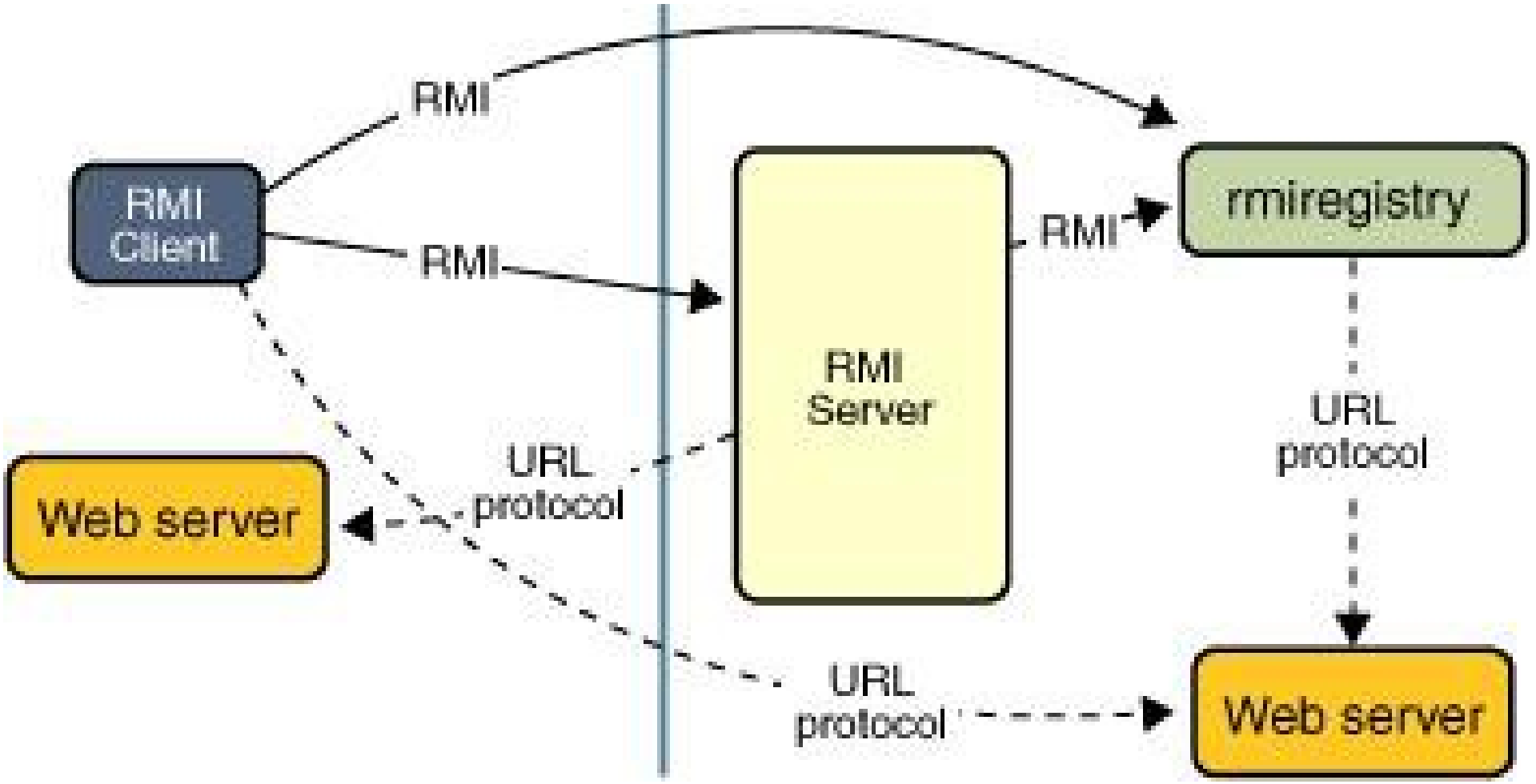
RMI - Invocação de Objetos Remotos

- As aplicações do RMI compreendem frequentemente dois programas separados, um servidor e um cliente.
- Um programa servidor cria alguns objetos remotos, faz referência a esses objetos, e aguarda os clientes invocarem os métodos desses objetos.
- Enquanto que um programa cliente obtém uma referência remota ao objetos criados pelo servidor e invoca os métodos desses objetos.

Funcionamento do RMI

- O servidor chama o “registry” para associar (bind) um nome com um objeto remoto.
- O cliente olha o objeto remoto por seu nome no “registry” do servidor, e invoca então um método nele.

Funcionamento do RMI



Criando aplicações distribuídas usando o RMI

Utilizar RMI para desenvolver uma aplicação distribuída envolve estas etapas gerais:

1. Projetar e implementar os componentes de sua aplicação distribuída.
2. Compilar o código fonte.
3. Fazer com que as classes sejam acessíveis via rede.
4. Iniciando a aplicação.

Projetando e implementando os componentes de sua aplicação distribuída.

Iremos criar uma classe servidora que irá conter um método que será invocado remotamente através da utilização de RMI, para isso iremos precisar de alguns arquivos:

- Ola.java (interface que será implementada pelo server)
- OlaImpl.java (implementação do servidor)
- Cliente.java (client que fará uso de métodos do OlaImpl - server)

Ola.java

```
import java.rmi.Remote;

import java.rmi.RemoteException;


public interface Ola extends Remote {

    String showMsg(String msg) throws
RemoteException;

}
```

OlaImpl.java

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
public class OlaImpl extends UnicastRemoteObject implements Ola {
    public OlaImpl() throws RemoteException {  super();  }
    public String showMsg(String msg) {
        System.out.println("msg: " + msg);
        return("msg enviada"); }
    public static void main(String args[]) {
     try { OlaImpl obj = new OlaImpl();
         Registry registry = LocateRegistry.createRegistry(2001);
    registry.rebind("OlaServidor", obj);
            System.out.println("Servidor carregado no registry");
        } catch (Exception e) {
            System.out.println("OlaImpl erro: " + e.getMessage());
        }}}
```

Client.java

```
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
public class Client {
    public static void main(String[] args) {
        Ola obj = null;
        String msg = "minha mensagem";
        String retorno = null;
        try {
         Registry registry = LocateRegistry.getRegistry("127.0.0.1", 2001);
obj = (Ola) registry.lookup("OlaServidor");
            retorno = obj.showMsg(msg);
            System.out.println(retorno);
          }
        catch (Exception e) {
            System.out.println("Client exception: " + e.getMessage());
          } } }
```

Observações sobre o RMI

O RMI permite que vários “Objetos Clientes” se conectem a um “Objeto Servidor”

- O mesmo “Objeto Servidor” irá atender a todos os clientes;
- Não é multi-thread;
- Os clientes compartilham o mesmo servidor.

Observações sobre o RMI

Não é possível registrar em um determinado registry um objeto que esteja em outra máquina

- Is it possible to run rmiregistry on a different machine than the remote object?
- http://www.jguru.com/faq/view.jsp?EID=417817

Observações sobre o RMI

Quando houver problemas de conexão em relação ao endereço publicado pelo servidor (Connection refused to host: 127.0.1.1) deve-se passar a propriedade java.rmi.server.hostname para a máquina virtual Java:

```
java -Djava.rmi.server.hostname=192.168.0.116 LigadorImpl

java -Djava.rmi.server.hostname=192.168.0.117 ServidorTempoImpl 192.168.0.116
```

- http://stackoverflow.com/questions/2624752/starting-rmi-server-on-ubuntu-laptop
- Se essa propriedade não for especificada no Ubuntu, será publicado o endereço 127.0.1.1 (ver arquivo /etc/hosts). Já no Windows, o IP da máquina para acesso remoto foi publicado corretamente nos testes que fiz, não precisando definir a propriedade java.rmi.server.hostname.

Observações sobre o RMI

Se for usado algum SecurityManager é preciso criar um arquivo que determina a política de segurança e dizer isso para o RMI através da propriedade java.security.policy:

```
java -Djava.rmi.server.hostname=192.168.0.116
-Djava.security.policy=security.policy LigadorImpl
```

Observações sobre o RMI

Um exemplo de arquivo security.policy que garante todas as permissões é dado abaixo:

```
grant {
  permission java.security.AllPermission "", "";
};
```

Entretanto, se for para dar todas as permissões, é melhor não usar nenhum SecurityManager.

Próxima Aula

Eventos e Notificações (Modelo Publish-Subscribe)

Bibliografia

- COULOURIS, G; DOLLIMORE, J; KINDBERG, T; BLAIR, G. Sistemas Distribuídos: Conceitos e Projeto. 5a. Edição, Editora Bookman, 2013

Dúvidas?

Contatos

- E-mail: professordiegoguedes@gmail.com
- Facebook: http://facebook.com/professordiegoguedes
- Github: https://github.com/diegoguedes