



**UNIALFA**  
CENTRO UNIVERSITÁRIO  
ALVES FARIA

- Ferramentas de programação para Sistemas Distribuídos
  - Sockets
  - RPC
  - Objetos Distribuídos

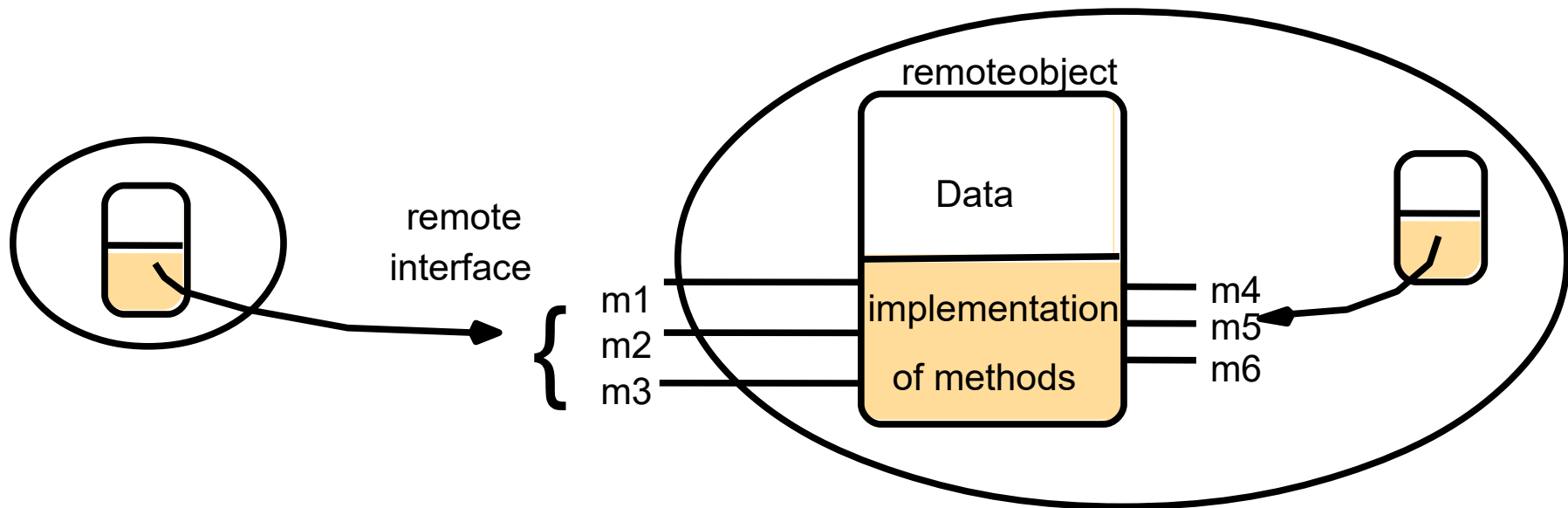


- Evolução do conceito de RPC
- Principais características
  - Encapsulamento de implementação
  - Interfaces bem definidas
  - Baixo grau de acoplamento entre os componentes
  - Unidades computacionais que podem ser reutilizadas para compor várias aplicações
- Exemplos:
  - CORBA, JAVA RMI, DCOM



- 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**





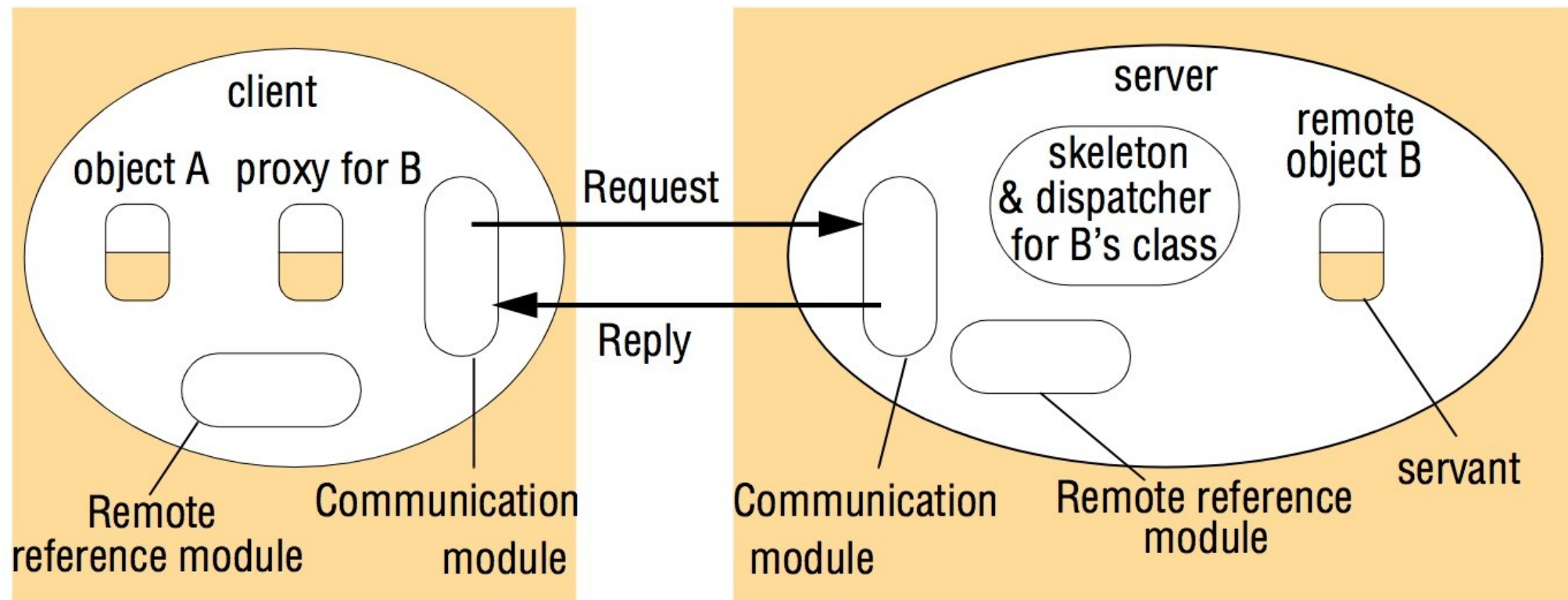
- m1, m2 e m3 podem ser invocados remotamente
- m4, m5 e m6 **não** podem ser invocados remotamente



- 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





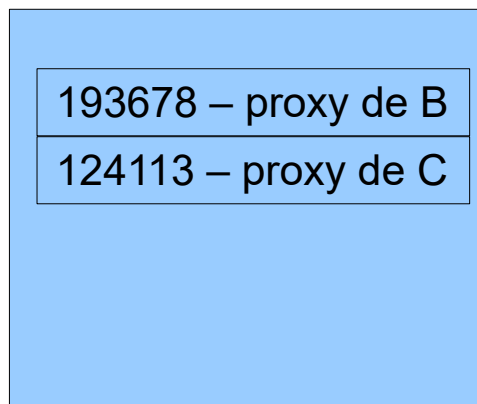
- **Módulo de Comunicação:**
  - Cooperam para executar o protocolo requisição-resposta entre o cliente e o servidor.
  - São responsáveis por fornecer uma semântica de invocação específica (talvez, pelo menos uma vez ou no máximo uma vez)
  - No Servidor, seleciona o despachante para a classe do objeto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



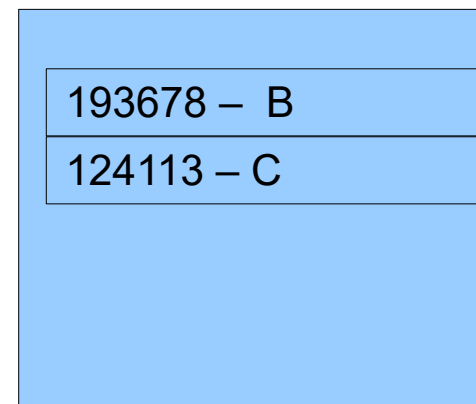


- **Módulo de referência remota**
  - Responsável pela criação das referências de objetos remotas
  - Responsável também pela conversão de referências de objetos remotas em referências de objetos locais

Cliente



Servidor



# Implementação de Objetos Distribuídos

- **Servant**
  - 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

- CORBA

- Programador escreve a IDL, o programa cliente e o programa servidor
- O compilador IDL gera as classes para o proxy, despachante e esqueleto em uma linguagem específica (C++, Java, Lua)

- RMI Java

- Compilador RMI Java gera as classes de proxy, despachante e esqueleto a partir da classe do objeto remoto



# Java RMI



**UNIALFA**  
CENTRO UNIVERSITÁRIO  
ALVES FARIA

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



- 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 “clients” 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.

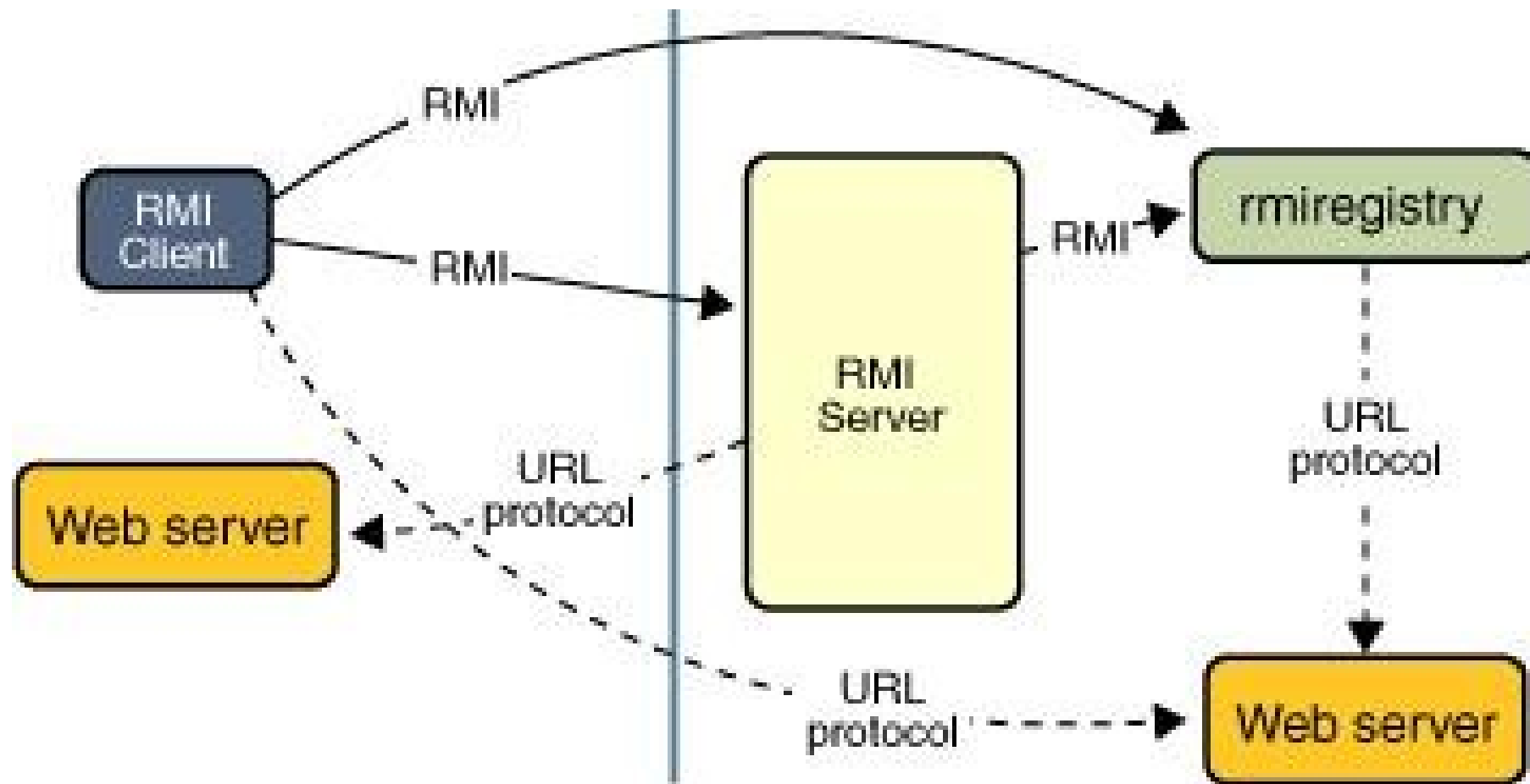




- O servidor chama o “registry” para associar (bind) um nome com um objeto remoto.
- O cliente olha o objeto remoto por seu nome no “registry” do servidor, e invoca então um método nele.



# Funcionamento do RMI



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

1. Projetando e implementando os componentes de sua aplicação distribuída.
2. Compilando o código fonte.
3. Fazendo com que as classes sejam acessíveis via rede.
4. Iniciando a aplicação.



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 tanto pelo server quando pelo client)
- OlaImpl.java (implementação do servidor)
- Cliente.java (client que fará uso de métodos do OlaImpl - server)



# Ola.java

```
package rmi.server;  
  
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
public interface Ola extends Remote {  
    String showMsg(String msg) throws RemoteException;  
}
```



# OlaImpl.java

```
package rmi.server;

import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

public class OlaImplementacao extends UnicastRemoteObject implements Ola {
    public OlaImplementacao() throws RemoteException {
        super();
    }

    public String showMsg(String msg) {
        System.out.println("msg: " + msg);
        return ("msg enviada");
    }

    public static void main(String args[]) {
        try {
            OlaImplementacao obj = new OlaImplementacao();
            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

```
package rmi.client;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import rmi.server.Ola;

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());
        }
    }
}
```





Se não criarmos o registry na aplicação, devemos fazê-lo manualmente:

- start rmiregistry – no windows
- rmiregistry – no linux e MacOS

```
[diego:rmi diego$ rmiregistry
```



**UNIALFA**  
CENTRO UNIVERSITÁRIO  
ALVES FARIA

- Para passar objetos, tem alguma mudança?



# Parâmetro Objeto

- Para passar objetos, tem alguma mudança?
  - Sim. No RMI, como os dados serão transmitidos pela rede, o objeto passado como parâmetro deve ter a capacidade de ser serializado
  - Assim, deve implementar (implements) a interface Serializable.
  - É uma boa prática (as vezes não funciona) se a classe não ter um serialVersionUID



# 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.



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.

