Programação Paralela e Distribuída

Prof. Cidcley T. de Souza

Introdução

- Características das Aplicações atuais
  - Complexas
  - Multiliguagens

- Características dos Ambientes atuais
  - Distribuídos
  - Heterogêneos

Introdução

- Particionar e distribuir dados e funções é uma tarefa bastante difícil
  - O Paradigma de Orientação a Objetos trata desse problema;
- Também é bastante difícil produzir aplicações complexas em ambiente de rede
  - Os Sistemas Distribuídos abstraem as características da rede;

Orientação à Objetos + Sistemas Distribuídos

Objetos Distribuídos

Remote Method Invocation

- Java e Programação Distribuída
  - Criada para ser uma linguagem simples e de arquitetura neutra;
  - Sun percebe o potencial de Java como uma "Linguagem de Programação para a Internet";
  - Foram incorporadas característica de Networking, Segurança e Multithread;

Remote Method Invocation

- Características de Java relevantes à Programação Distribuída
  - Ambiente Orientado a Objetos
    - Linguagem "pura" orientada a objetos;
    - Ambiente estruturado e bem definido;
    - Distribuição em Java pode ser pensada como a distribuição de objetos e suas conexões através da rede;

Remote Method Invocation

- Características de Java relevantes à Programação Distribuída
  - Suporte a Interfaces Abstratas
    - Descrição das operações e serviços que um objeto fornece;
    - Interfaces implementadas por classes específicas;
    - Ex.: Componentes AWT com implementações para sistemas de janelas nativos (Xwindows, Windows, ...)

Remote Method Invocation

- Características de Java relevantes à Programação Distribuída
  - Independência de Plataforma
    - Bytecodes podem ser interpretados em diversos JVM em diferentes SOs;
    - Torna qualquer máquina potencialmente um servidor ;
    - Classes podem ser "baixadas" pela rede e executadas pela JVM;

Remote Method Invocation

- Características de Java relevantes à Programação Distribuída
  - Tolerância a Falhas através de Exceções
    - Java suporta o "lançamento" e a "captura" de exceções em aplicações;
    - Utilização do comando try/catch/finally;
    - Novos tipos de exceções podem ser criadas para tratar aspectos peculiares à programação distribuída;

Remote Method Invocation

- Características de Java relevantes à Programação Distribuída
  - Suporte à Rede
    - A API Java fornece vários níveis de programação em rede;
    - Sockets, várias classes de Streams para a filtragem e processamento de entrada e saída de dados, ...

Remote Method Invocation

- Características de Java relevantes à Programação Distribuída
  - Segurança
    - Java define um conjunto de restrições de acesso a objetos distribuídos aos recursos da máquina onde este estão sendo executados;
    - Ex: Applets não podem realizar virtualmente nenhum acesso ao sistema de arquivos local;

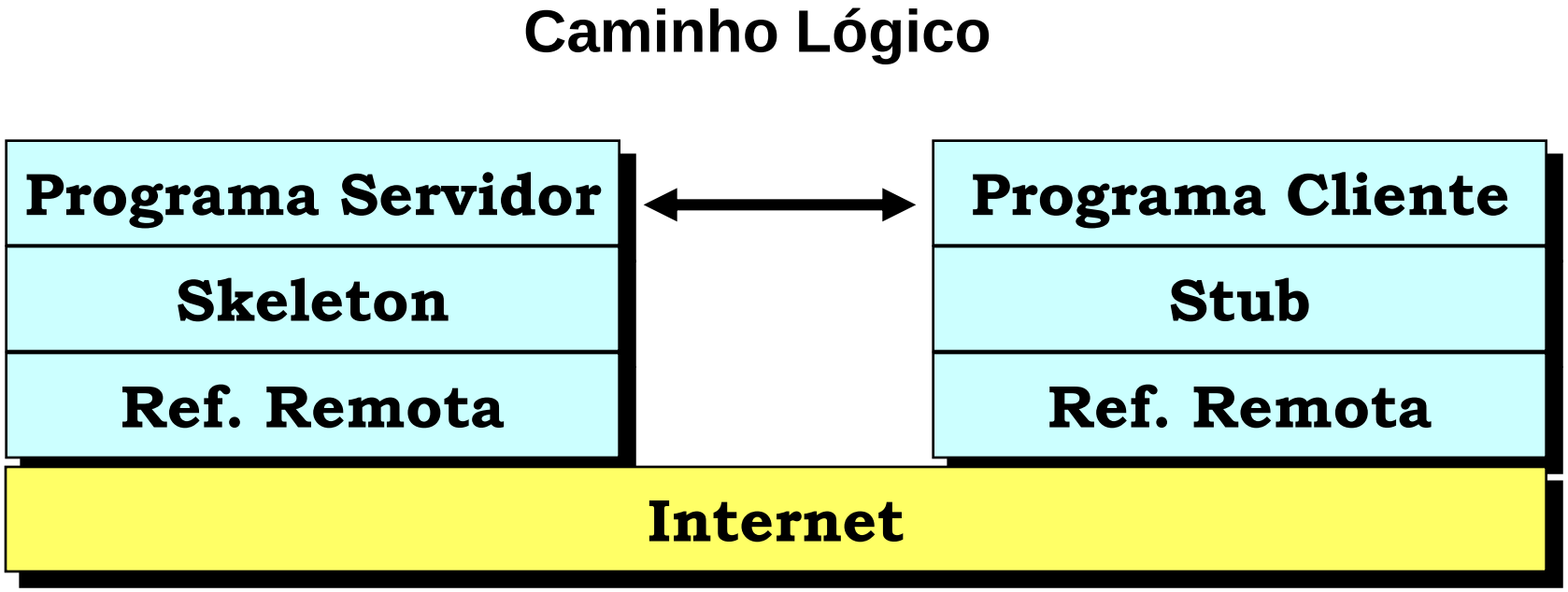
Remote Method Invocation

- Características de Java relevantes à Programação Distribuída
  - Multithread
    - A classe java.lang.Thread fornece as características para a criação de threads em Java;
    - Prioridades podem ser dadas a threads; threads podem criar novos threads sob a demanda de clientes ....

Remote Method Invocation

- A Interface RMI
  - Permite a comunicação entre objetos em diferentes hosts;
  - Um **Objeto Remoto** está localizado no servidor;
  - Cada Objeto Remoto implementa uma **Interface Remota**, que especifica os métodos que os clientes podem invocar;
  - **Clientes** invocam métodos de objetos remotos exatamente como se fossem locais;

Remote Method Invocation



Remote Method Invocation

- Arquitetura
  - Para o desenvolvedor, o cliente se comunica diretamente com o servidor;
  - Mas essa comunicação é realizada através de ***stubs***, passando pela camada de referência remota até a camada de transporte;
  - O Cliente localiza o objeto através de um **Serviço de Nomes**;
  - Os ***stubs*** podem ser "baixados" pela rede;

Remote Method Invocation

- Java Packages
  - A maioria dos métodos para objetos remotos está em quatro pacotes Java:
  - java.rmi
    - Classes, interfaces e exceções para o lado cliente;
  - java.rmi.server
    - Classes, interfaces e exceções para o lado servidor. Cria objetos remotos;

Remote Method Invocation

- Java Packages
  - java.rmi.registry
    - Classes, interfaces e exceções que são usadas para a localização e a nomeação de objetos remotos;
  - java.rmi.dgc
    - Realiza coleta de lixo distribuída;

Remote Method Invocation

- O Lado Servidor
  - Para criar um objeto remoto deve-se definir uma interface que estenda a interface java.rmi.Remote;
  - A interface Remote serve como um tag para identificar objetos remotos;
  - Essa subclasse de Remote deve definir as operações a serem disponibilizadas para clientes;

Remote Method Invocation

- O Lado Servidor
  - Cada método da interface do objeto remoto, deve "lançar" exceções do tipo java.rmi.RemoteException
  - A classe que implementa a interface remota deve estender, direta ou indiretamente, a classe java.rmi.UnicastRemoteObject;
  - Stubs e Skeletons devem ser construídos para que os objetos remotos sejam ativados;

Remote Method Invocation

- O Lado Servidor
  - Um compilador da Sun (rmic) gera os Stubs e Skeletons a partir da classe que implementa a interface remota;
  - Basta o servidor se registrar no serviço de nomes e este poderá ser invocado;

Remote Method Invocation

- O Lado Cliente
  - O cliente deve possuir uma referência do objeto remoto antes de realizar a invocação;
  - Essa referência é conseguida no Registro de Nomes;
  - O método lookup(Obj) do Registro devolve a referência do objeto procurado;

Remote Method Invocation

Classes e Interfaces de Registro e Segurança

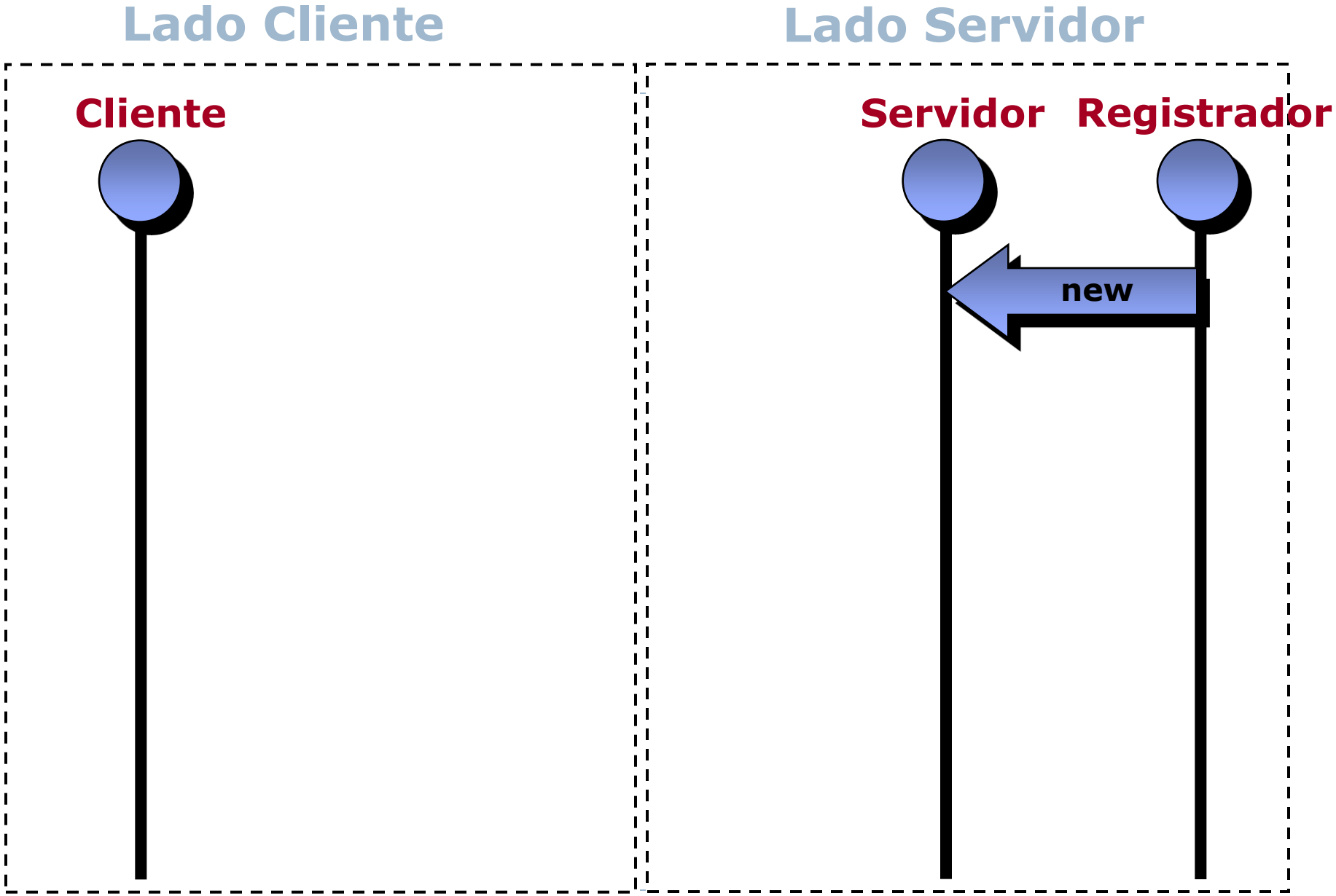
Remote Method Invocation

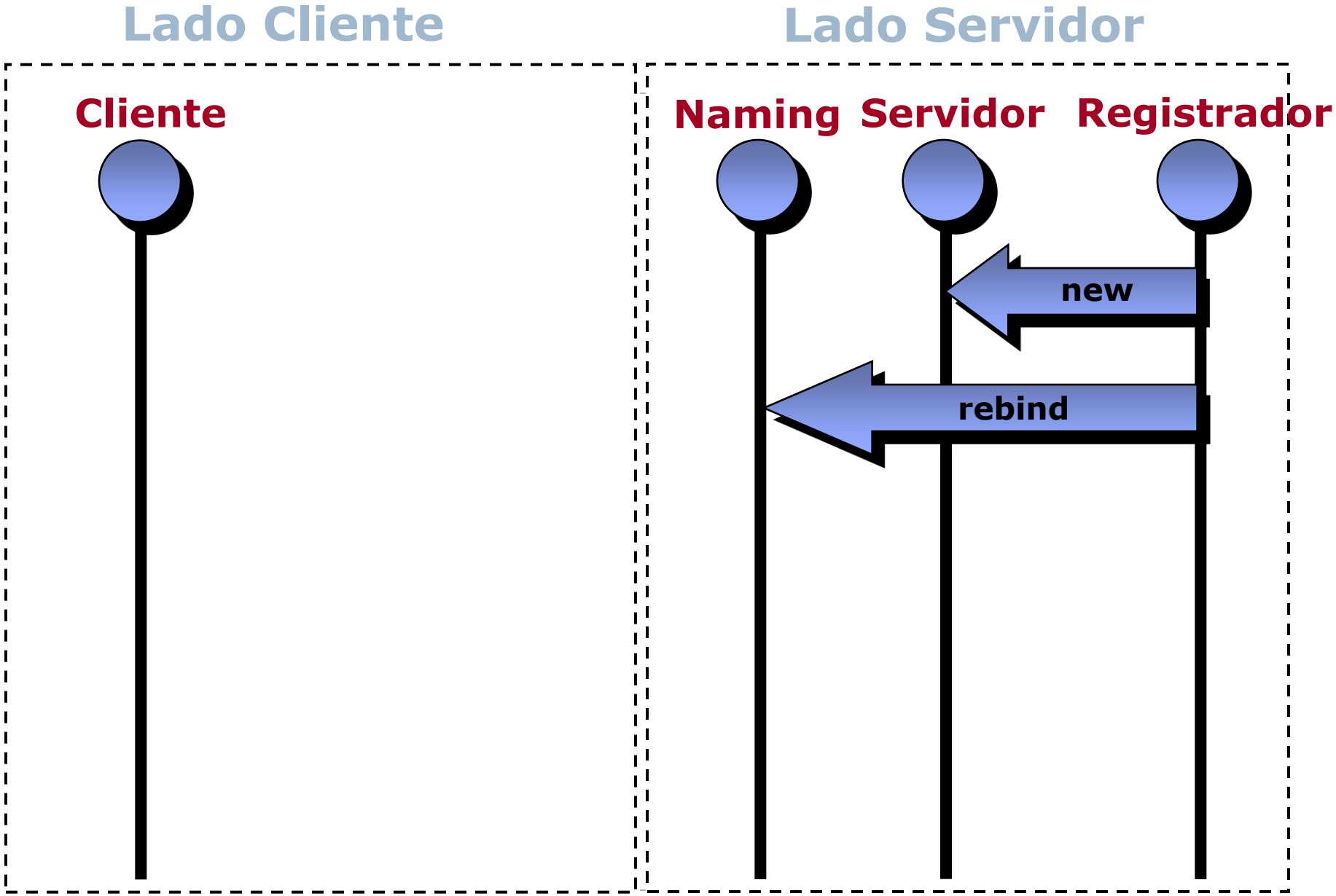
- Serviço de Nomes
  - Registry: Interface dos métodos para utilizar o serviço de nomes;
    - bind - mapear um objeto em um nome;
    - lookup - obter uma referencia de um determinado objeto;
  - Naming: Essa classe utiliza o serviço de nomes para permitir a manipulação de objetos remotos através de sintaxe URL (rmi://host:port/name). A porta default do serviço de nomes é 1099;

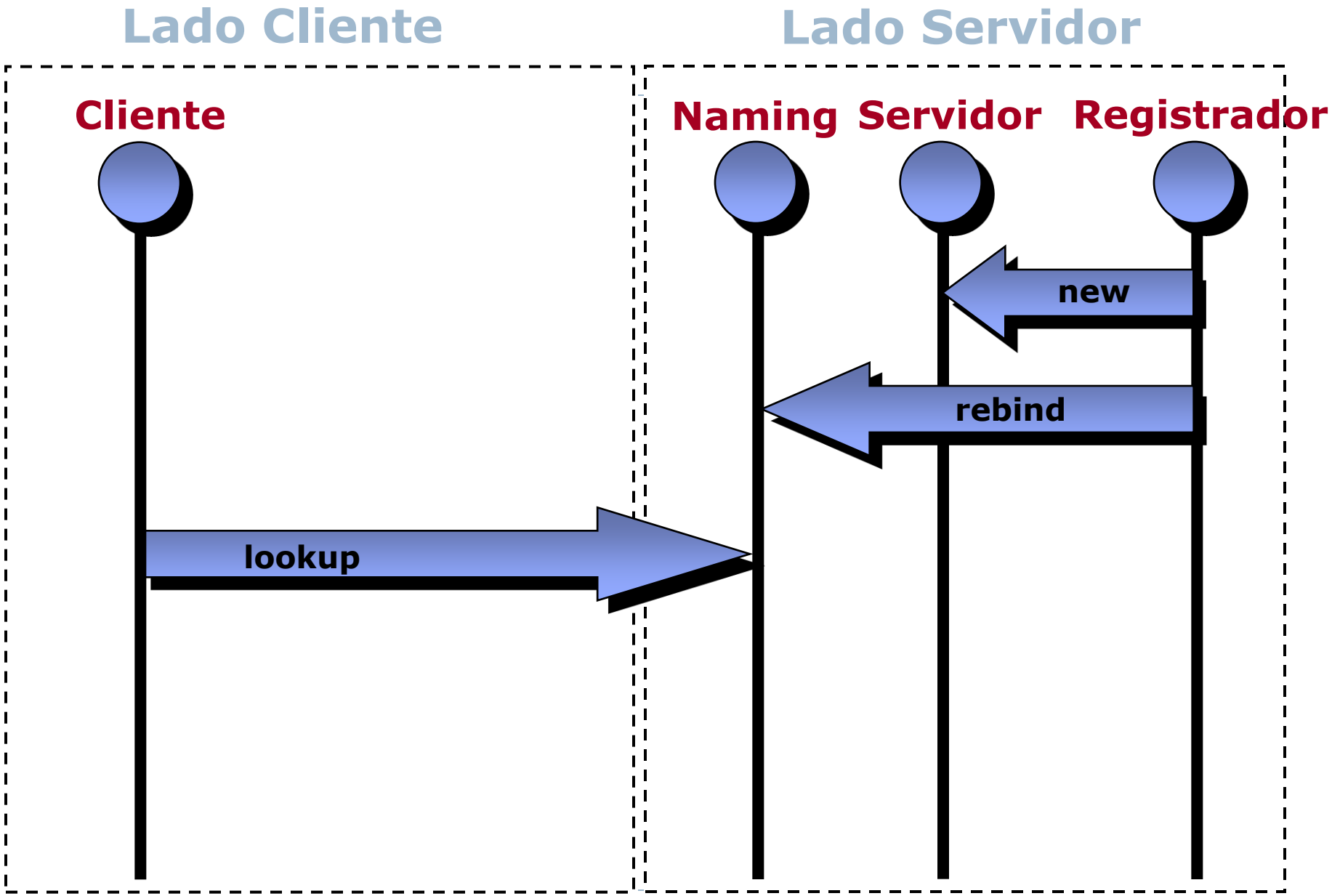
Remote Method Invocation

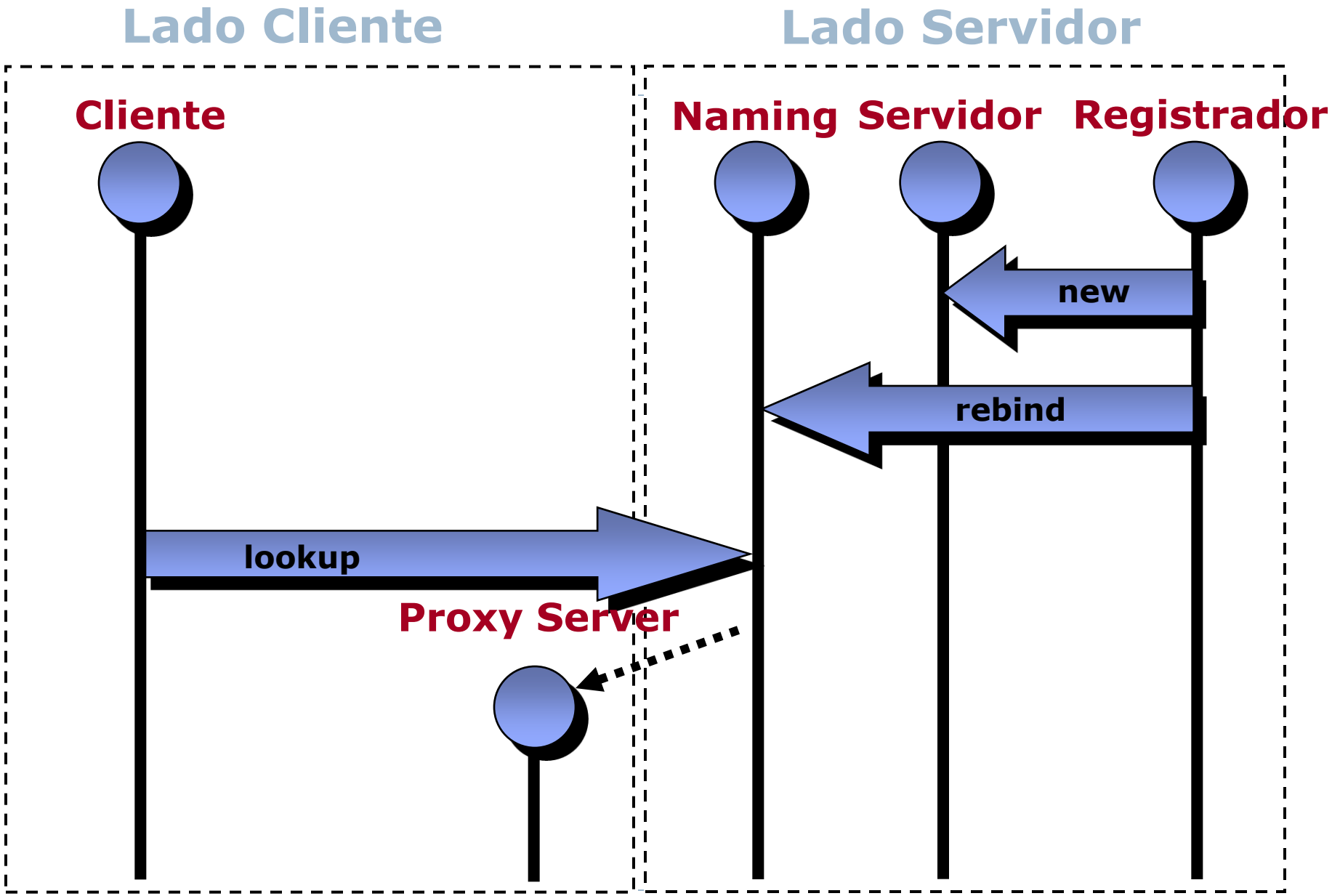
Desenvolvimento de Aplicações

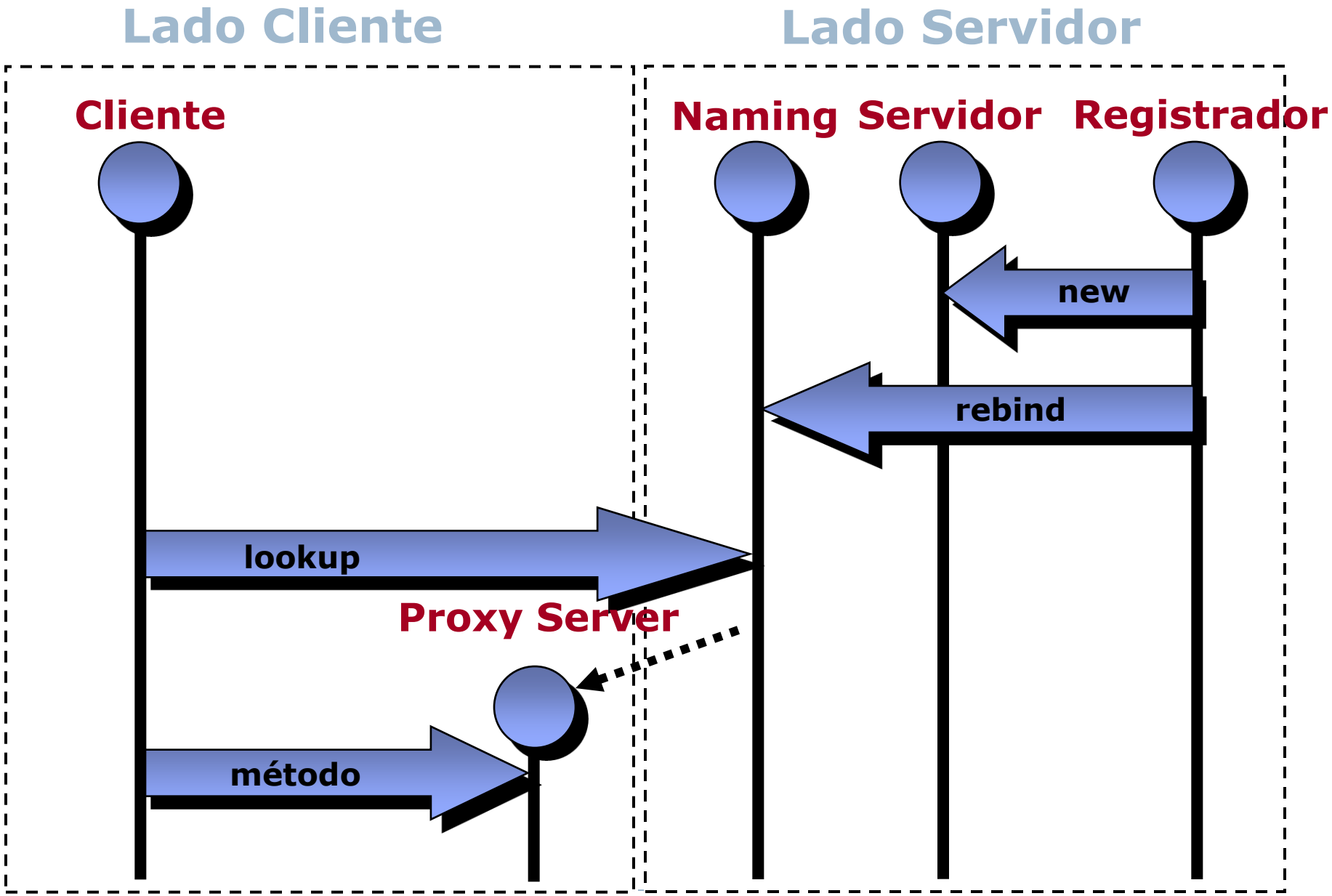

Lado Cliente
Lado Servidor
Cliente
Registrador

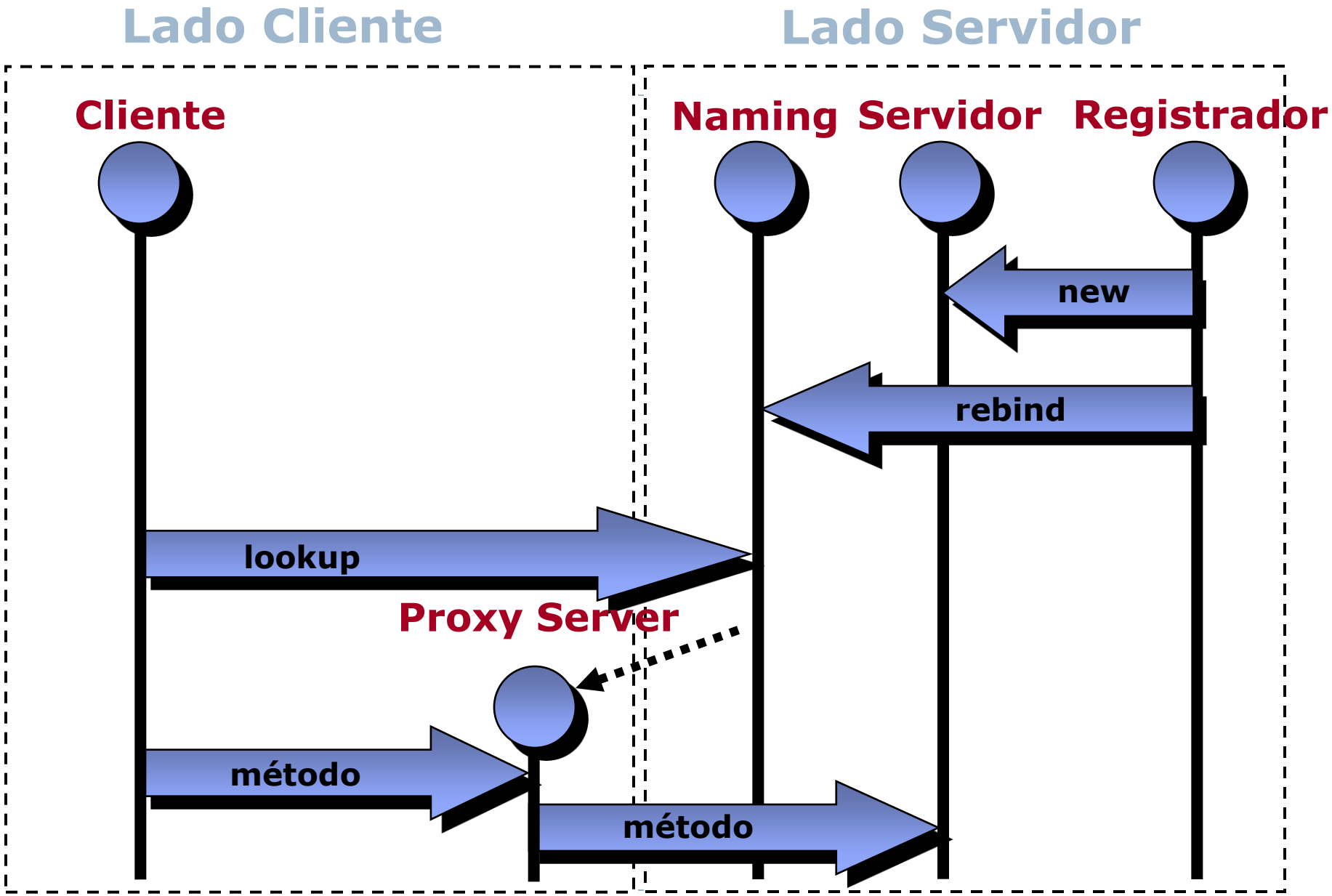

Lado Cliente
Lado Servidor
Cliente
Servidor
Registrador
new


Lado Cliente
Lado Servidor
Cliente
Naming
Servidor
Registrador
new
rebind


Lado Cliente
Lado Servidor
Cliente
Naming Servidor
Registrador
new
rebind
lookup


Lado Cliente
Lado Servidor
Cliente
Naming Servidor
Registrador
new
rebind
lookup
Proxy Server


Lado Cliente
Lado Servidor
Cliente
Naming Servidor
Registrador
new
rebind
lookup
Proxy Server
método

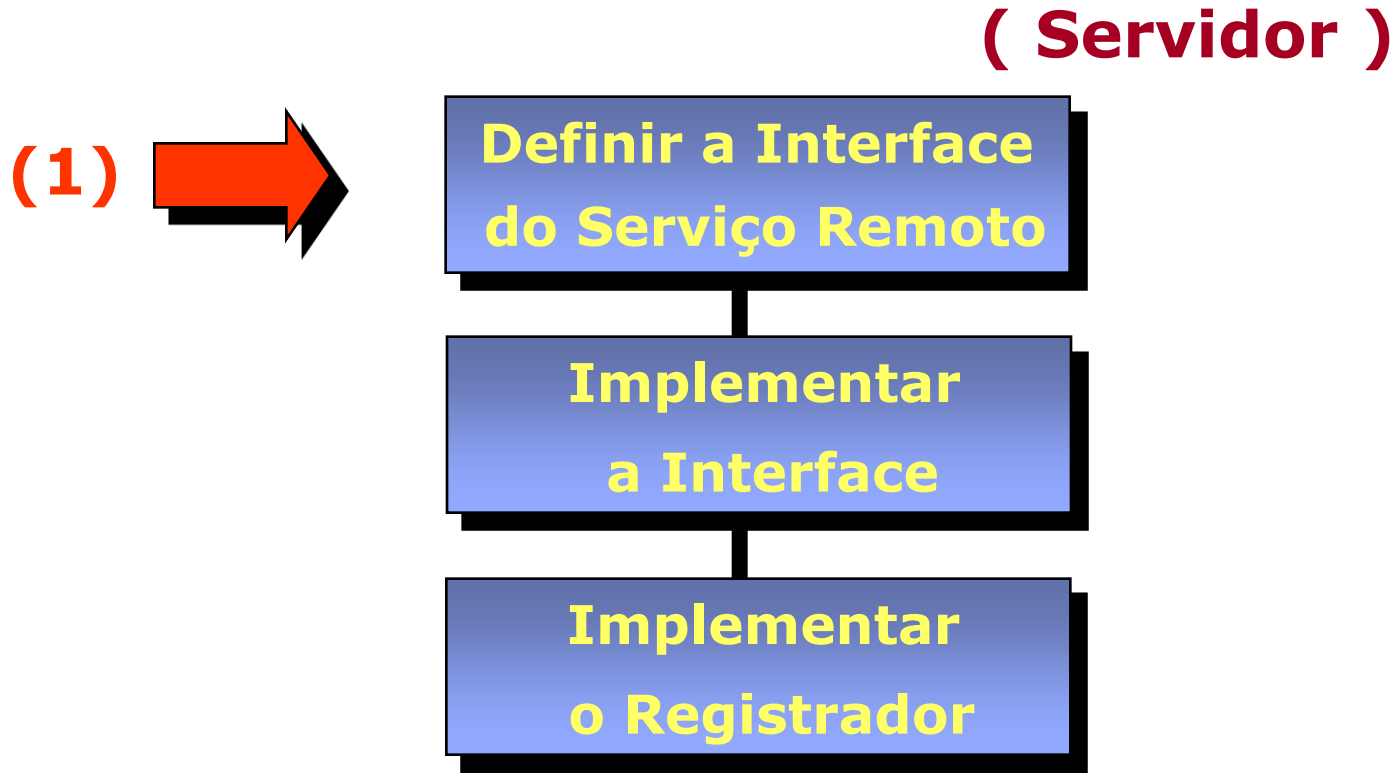

Lado Cliente
Lado Servidor
Cliente
Naming Servidor
Registrador
new
rebind
lookup
Proxy Server
método
método

Desenvolvimento de Aplicações

- Aplicação Exemplo
  - Um objeto que recebe uma string de um cliente e retorna a mesma string de forma invertida;
  - Um cliente que recebe uma string do teclado, invoca a operação Inverter de um objeto servidor remoto e recebe como resultado a mesma string de forma invertida.

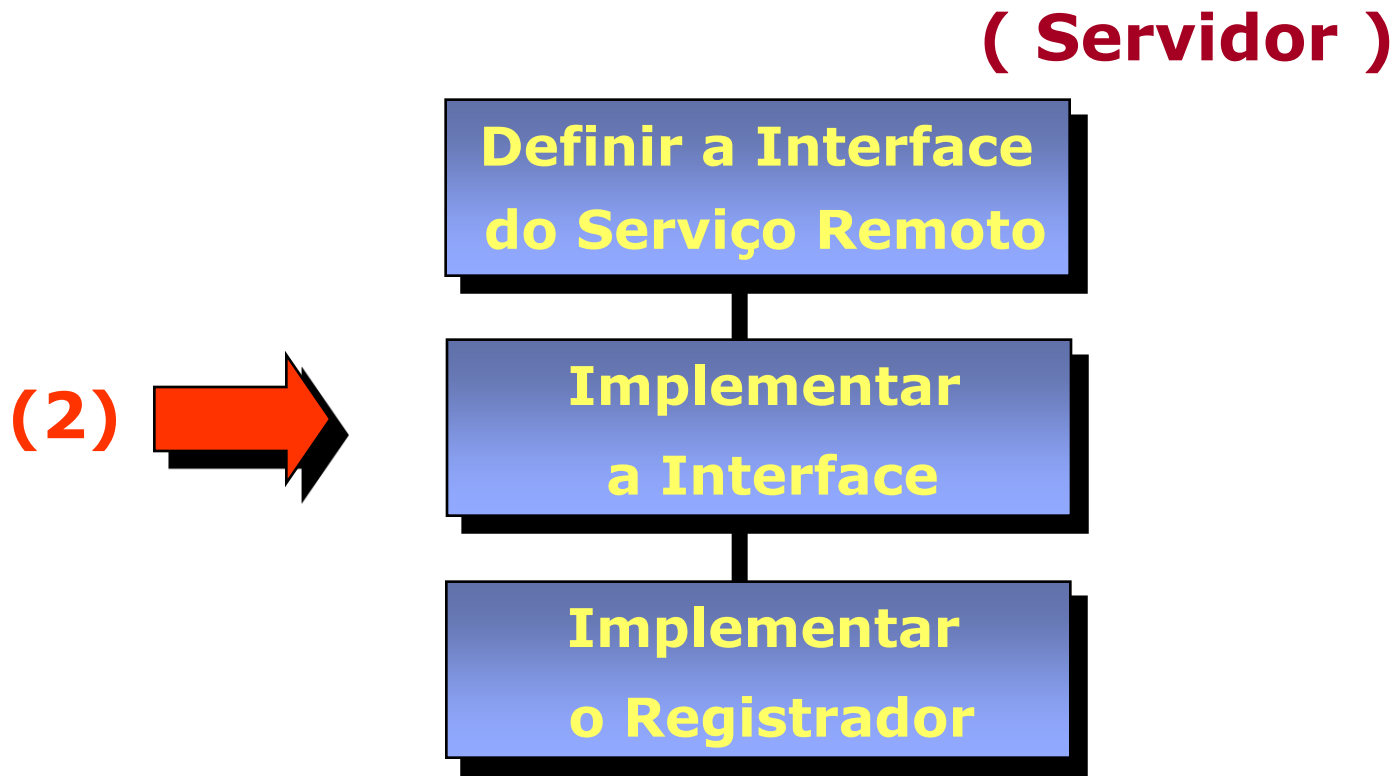
Desenvolvimento de Aplicações

- Etapas do desenvolvimento



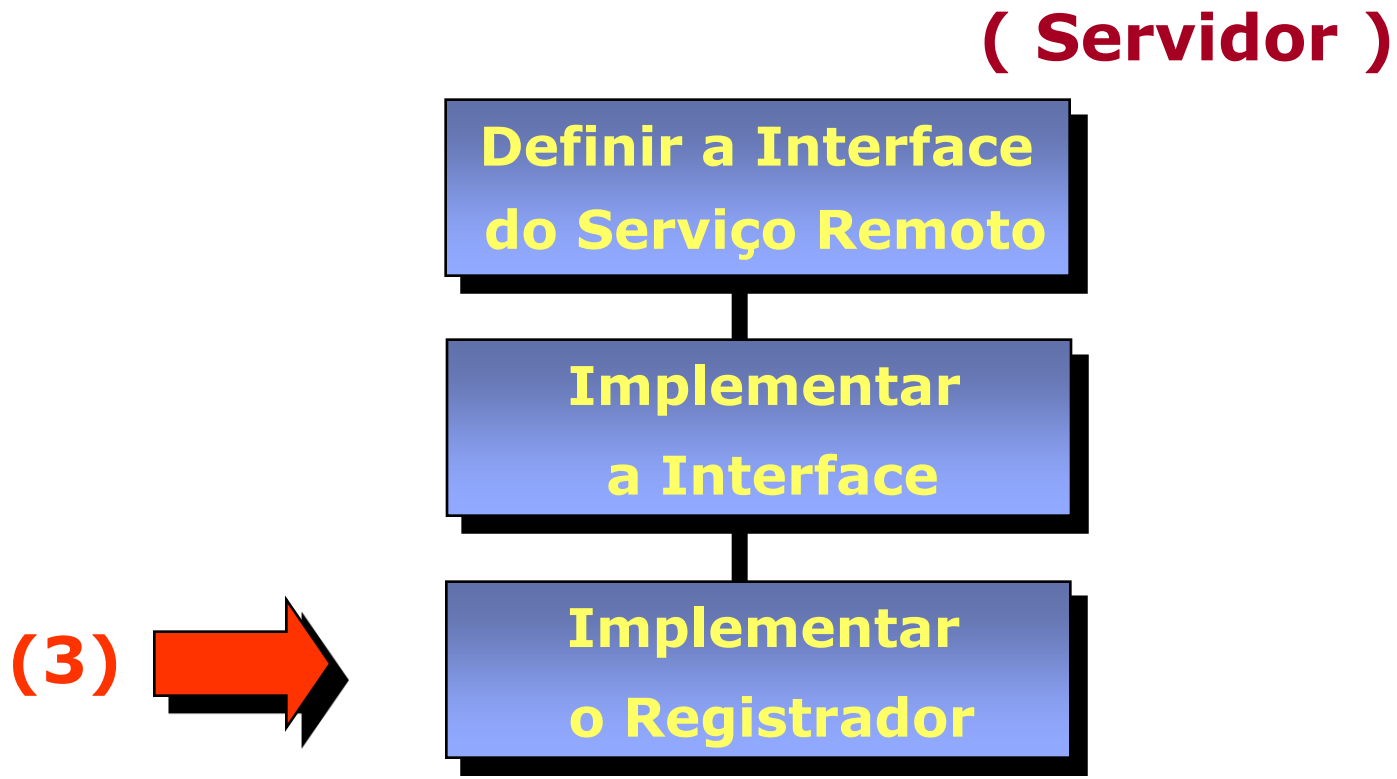
Desenvolvimento de Aplicações

- Etapas do desenvolvimento



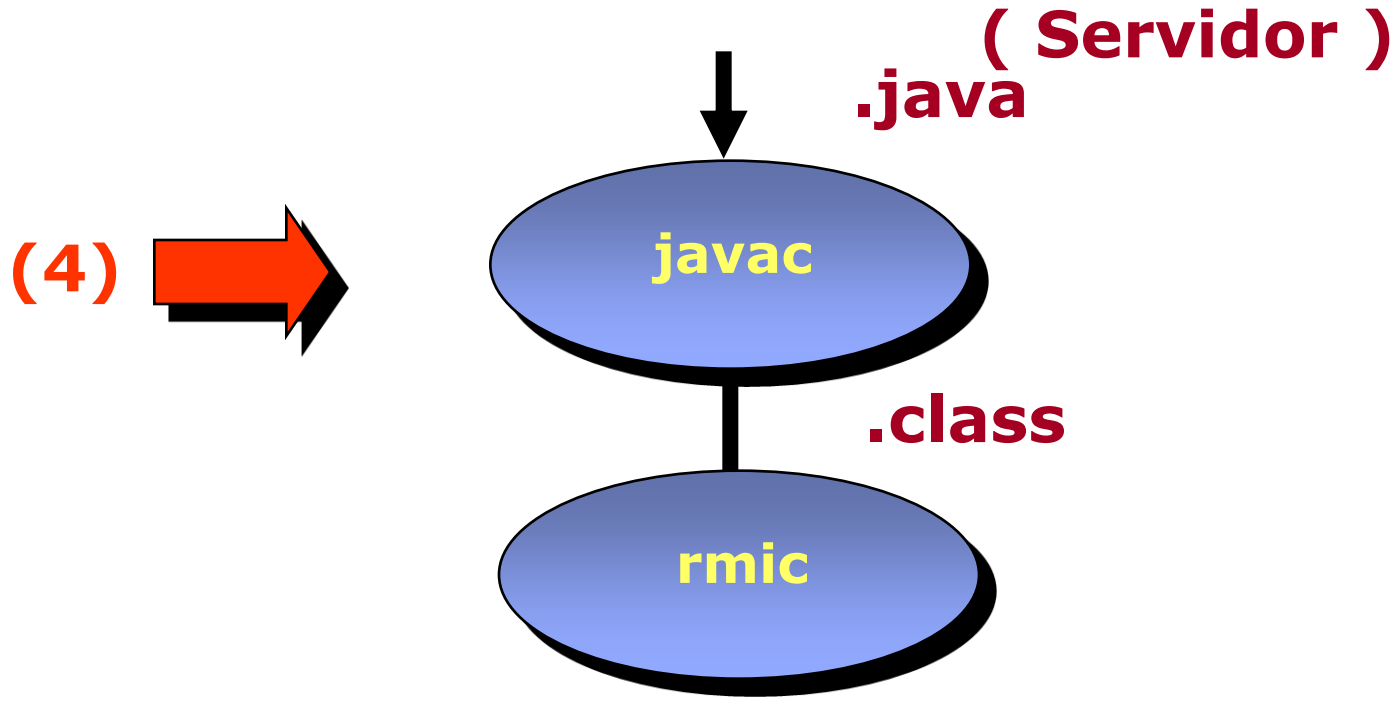
Desenvolvimento de Aplicações

- Etapas do desenvolvimento



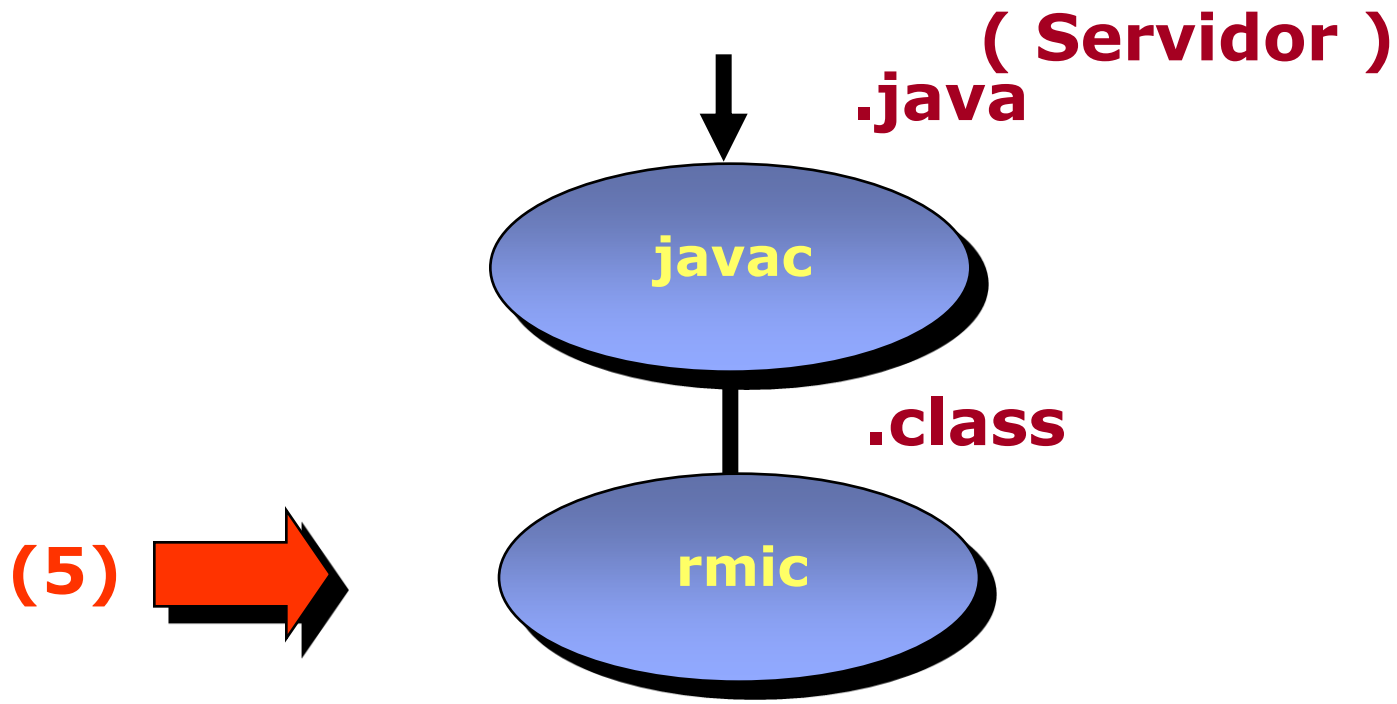
Desenvolvimento de Aplicações

- Etapas do desenvolvimento



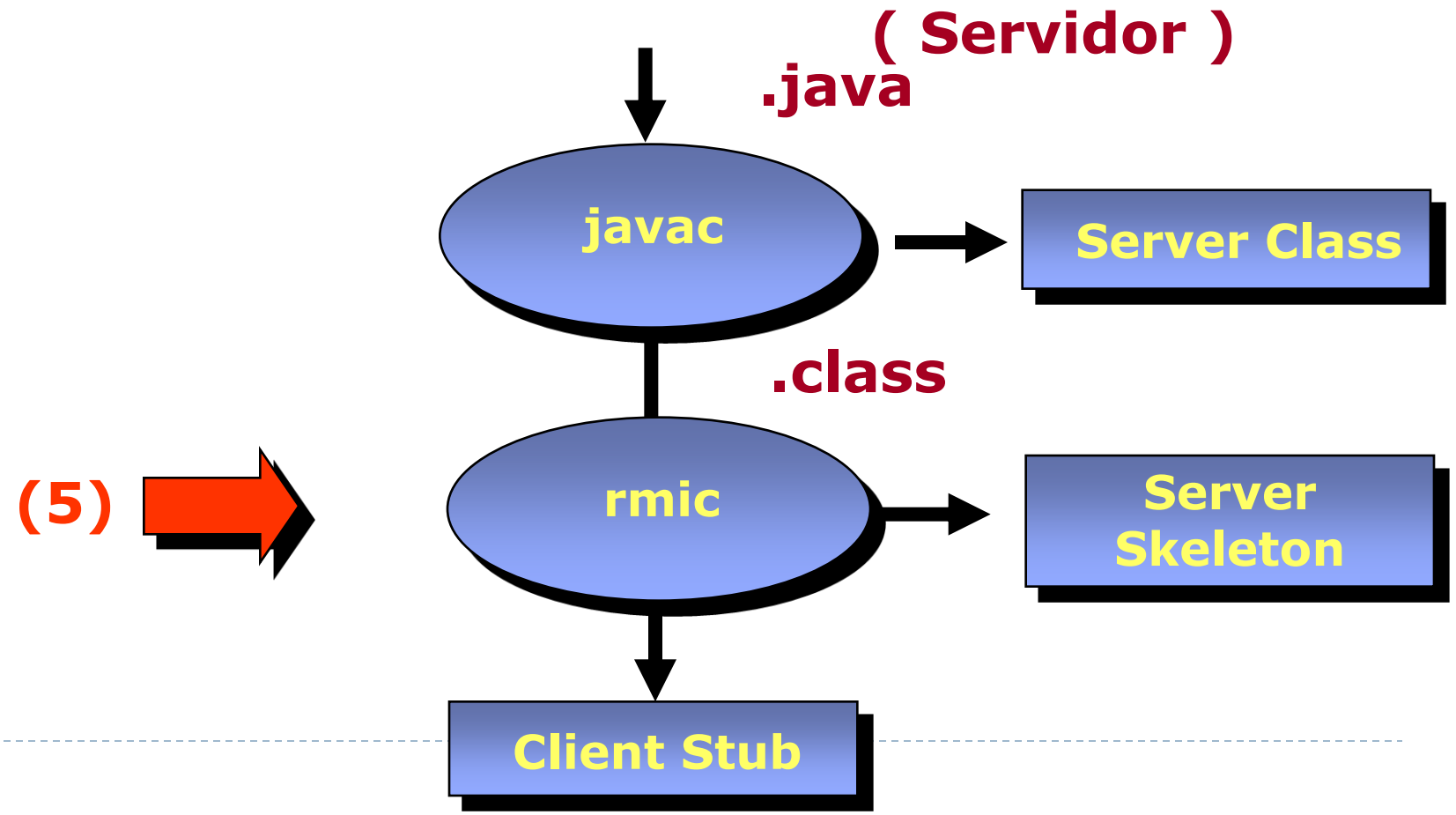
Desenvolvimento de Aplicações

- Etapas do desenvolvimento



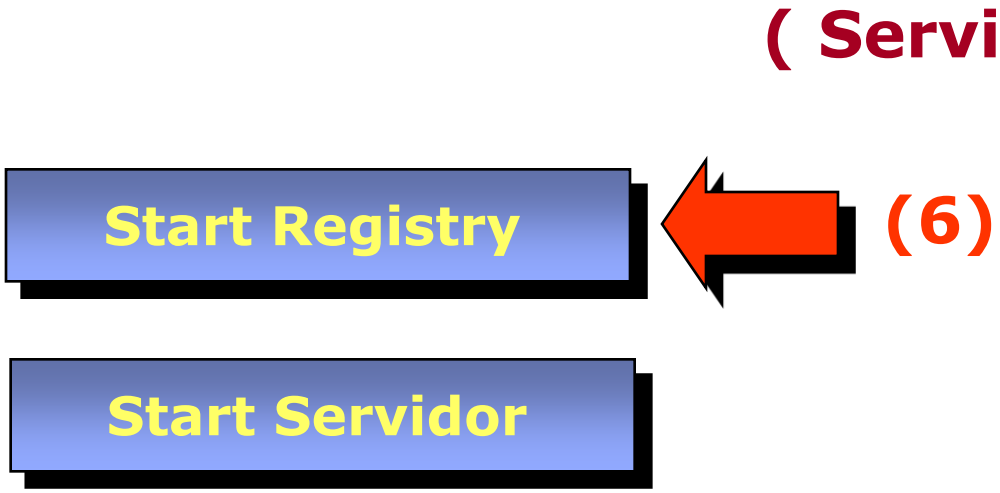
Desenvolvimento de Aplicações

- Etapas do desenvolvimento



Desenvolvimento de Aplicações

- Etapas do desenvolvimento



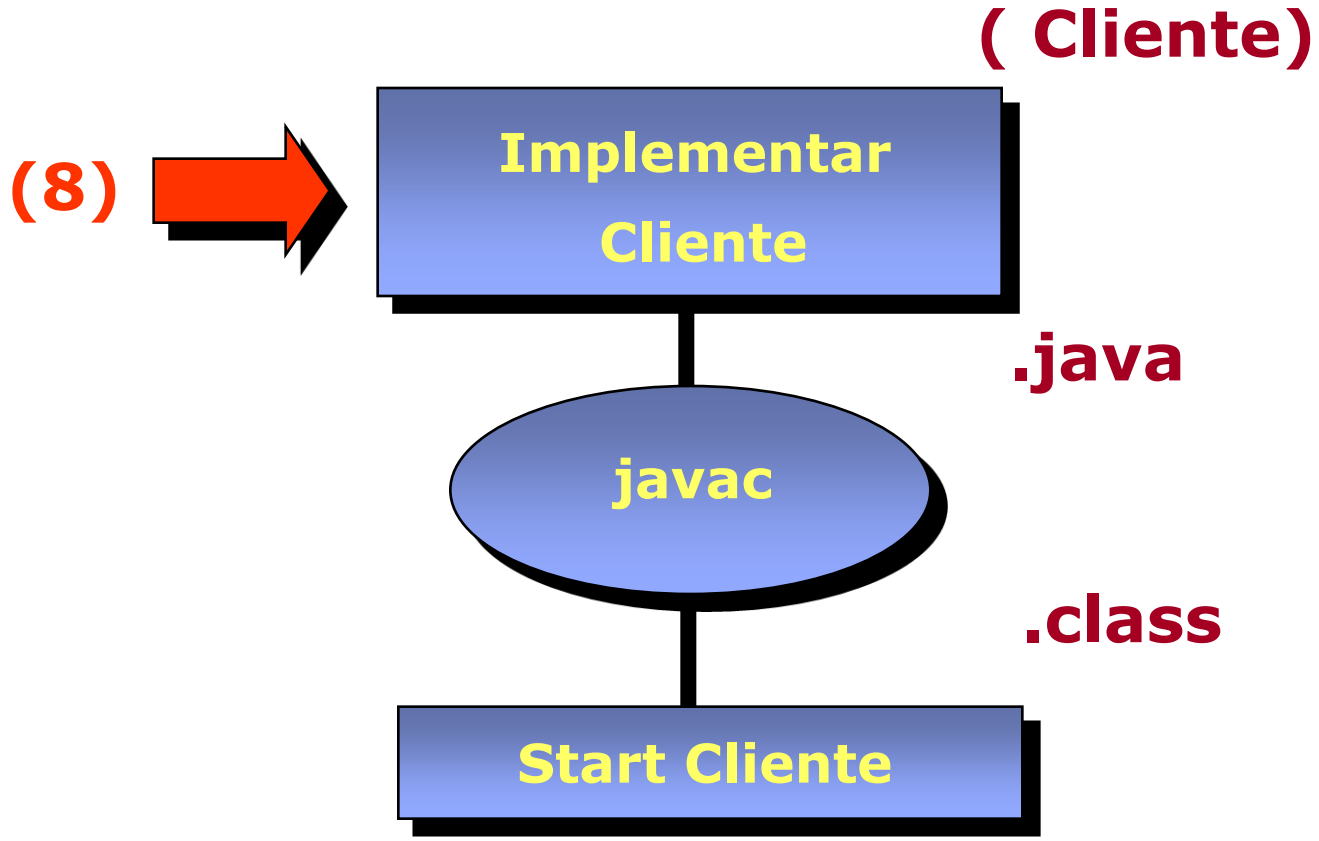
Desenvolvimento de Aplicações

- Etapas do desenvolvimento

(Servidor)

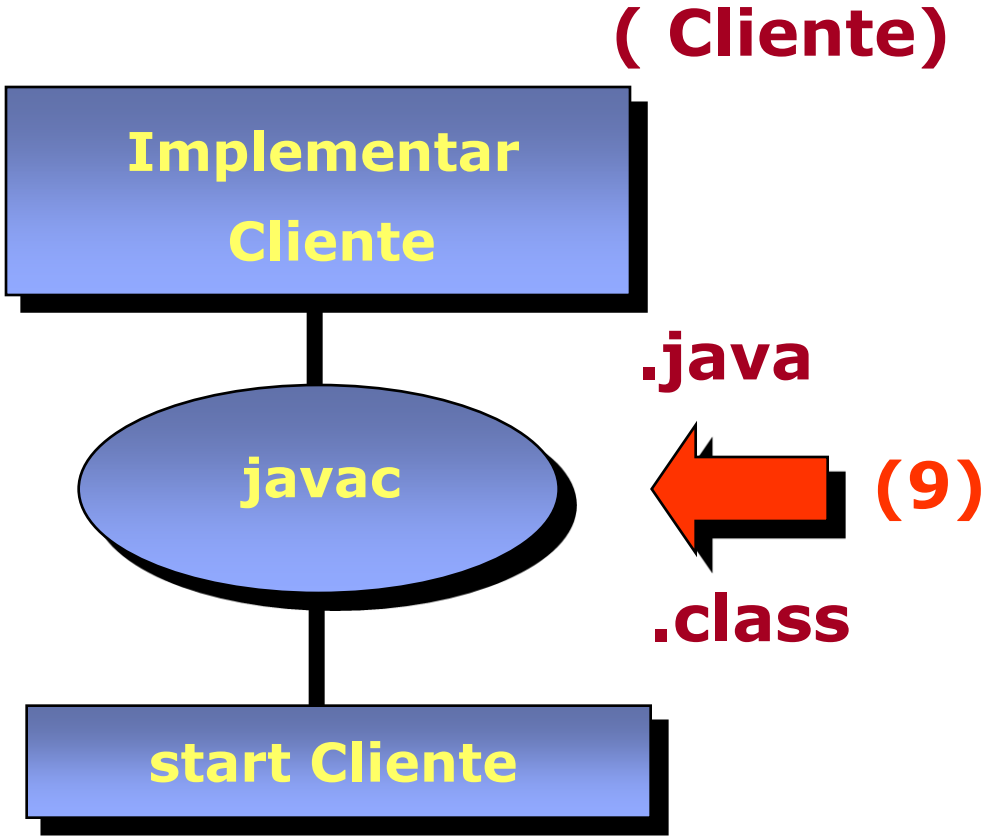
Desenvolvimento de Aplicações

- Etapas do desenvolvimento



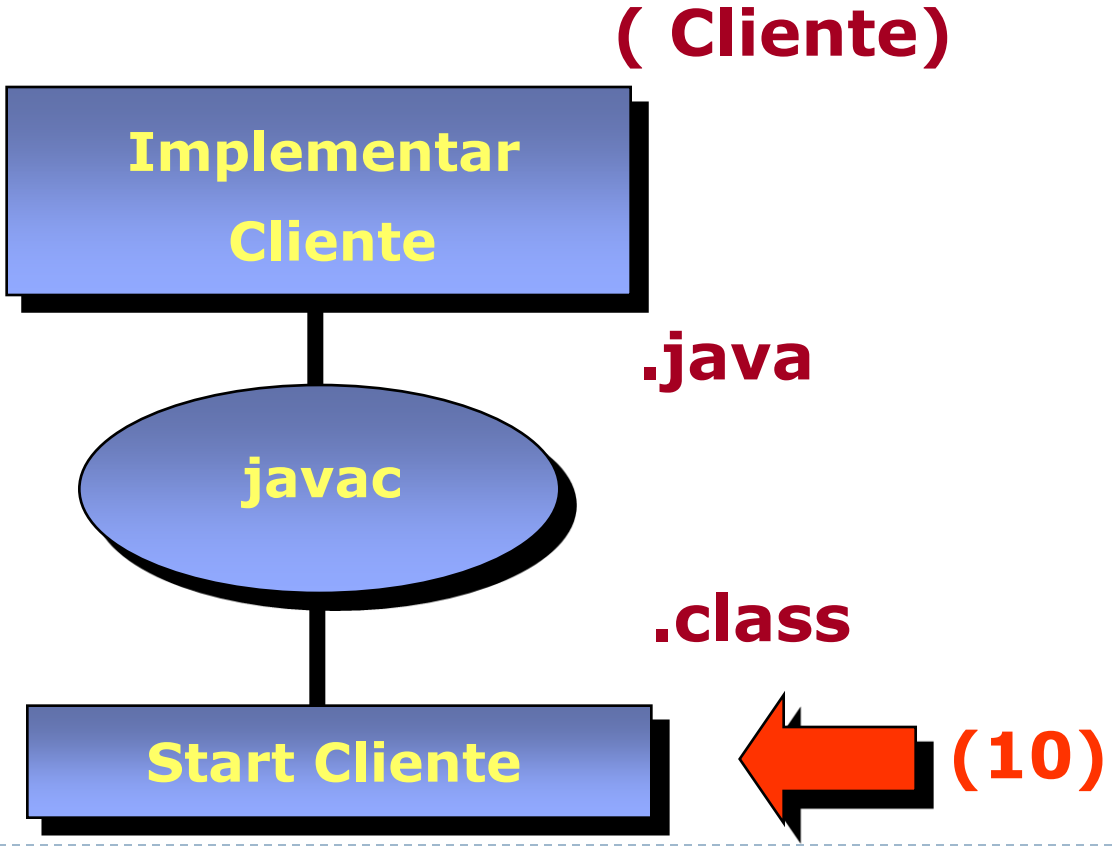
Desenvolvimento de Aplicações

- Etapas do desenvolvimento



Desenvolvimento de Aplicações

- Etapas do desenvolvimento



Etapa 1 - Definir a Interface do Serviço Remoto

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface InverterItf extends Remote {

   String inverter(String msg) throws
                  RemoteException;

}
```

Desenvolvimento de Aplicações

Etapa 2 - Implementar a Interface

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class Servidor extends
UnicastRemoteObject implements InverterItf {

  public Servidor() throws RemoteException{
      super();
      System.out.println("Servidor criado!");
   }
```

Desenvolvimento de Aplicações

Etapa 2 - Implementar a Interface

```
public String inverter(String msg) {

   StringBuffer strbuf = new StringBuffer(msg);

   System.out.println("Recebido: "+msg);

   String retorno = (strbuf.reverse()).toString();

   return retorno;
}

}
```

Desenvolvimento de Aplicações

Etapa 3 - Implementar o Registrador

```
import java.rmi.*;
import java.rmi.server.*;

public class Registrador {

 public static void main(String args[]) {
```

Desenvolvimento de Aplicações

Etapa 3 - Implementar o Registrador

```
try{

   Servidor obj = new Servidor ();
   Naming.rebind("InverterRef",obj);

   System.out.println("Servidor Registrado!");

  } catch (Exception e){
          System.out.println("Erro");}

 }
}
```

Etapa 4 - Compilar o Registrador

```
> javac Registrador
```

Desenvolvimento de Aplicações

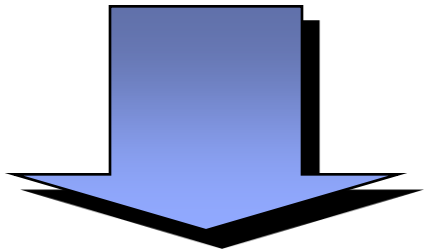
Etapa 5 - Gerar os Stubs

> rmic Servidor

Desenvolvimento de Aplicações

Etapa 5 - Gerar os Stubs

> rmic Servidor



Servidor_Stub.class

Servidor_Skel.class

Etapa 6 - Inicializar o Servidor de Nomes

```
> rmiregistry
```

Desenvolvimento de Aplicações

Etapa 7 - Inicializar o Registrador

```
> java Registrador
```

Desenvolvimento de Aplicações

Etapa 8 - Implementar Objeto Cliente

```
import java.io.*;
import java.rmi.*;

public class Cliente{
 public static void main(String args[])  {
  try {
   InverterItf Inv =
 (InverterItf)Naming.lookup("//localhost/InverterRef");

   System.out.println("Objeto Localizado!");
```

Desenvolvimento de Aplicações

Etapa 8 - Implementar Objeto Cliente

```
for(;;) {
   System.out.print("Digite a Frase:");
   BufferedReader r = new BufferedReader(
                      new InputStreamReader(System.in));
   String line = r.readLine();
   String retorno = Inv.inverter(line);
   System.out.println("Frase Invertida = " + retorno);
   }
  } catch(Exception e){System.err.println("Erro");}
      System.exit(0);
 }
}
```

Etapa 9 – Compilar Código do Cliente

```
>javac Cliente
```

Desenvolvimento de Aplicações

Etapa 10 - Inicializar o Cliente

```
>java Cliente
```

Fim