Computação Distribuída

Odorico Machado Mendizabal

Universidade Federal de Santa Catarina – UFSC
Departamento de Informática e Estatística – INE

RPC e RMI

Motivação

- Paradigmas de programação usuais para programadores
  - Orientação a Objetos e Procedural
    - Invocação de métodos e chamada de Procedimentos

```
int soma(int x, int y){ .. }
r = soma(23,46);
```

```
Calculadora calc;
calc.soma(23,46);
```

    - Mecanismos para tratamento de exceções

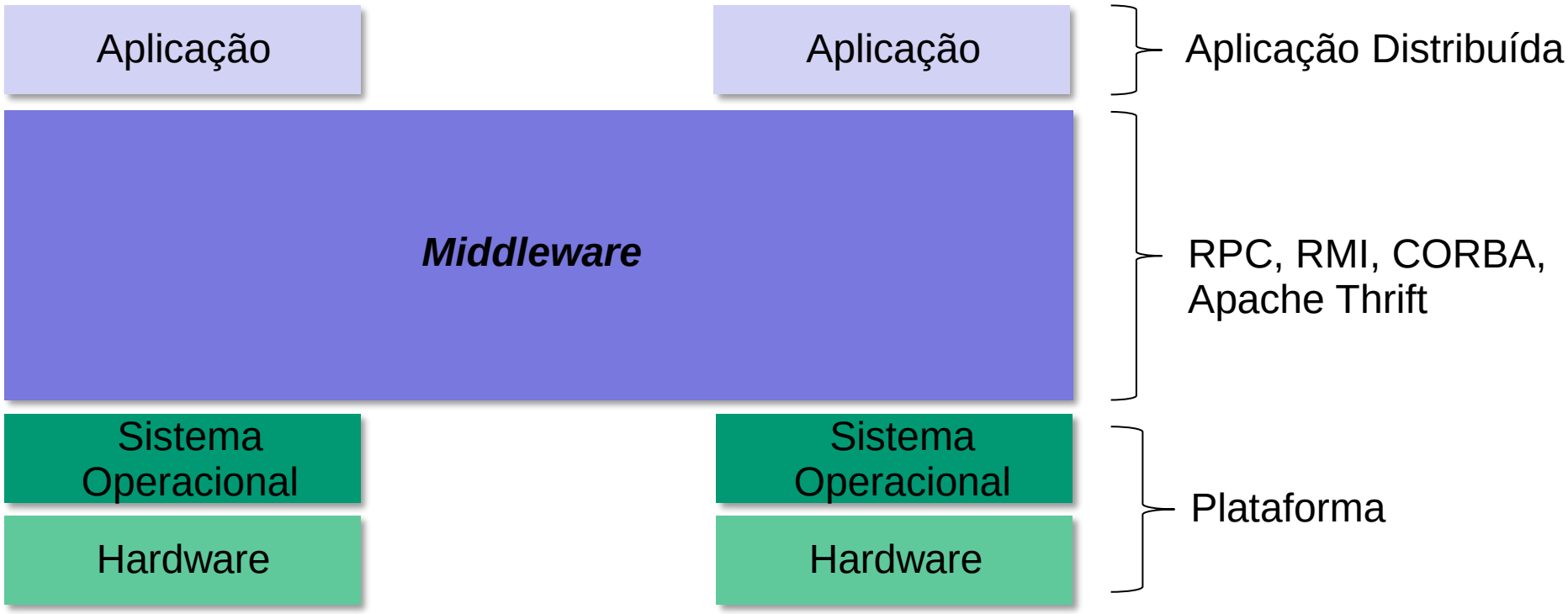
```
try {
  //  trecho de código
}
catch(Exception e) {
  //  tratamento de exceções
}
```

É possível dispor do mesmo grau de abstração para o desenvolvimento de aplicações distribuídas?

Remote Procedure Call (RPC) e *Remote Method Invocation* (RMI)

Transparência de Localização

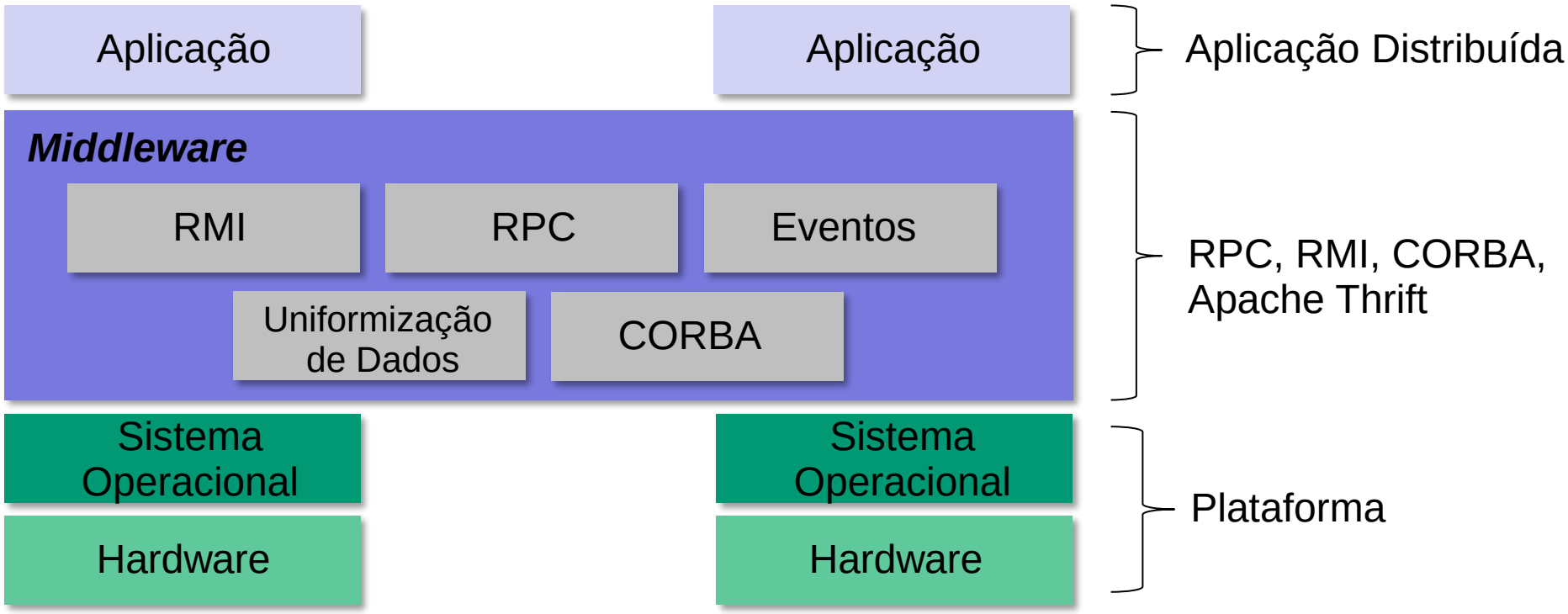
Chamada a procedimento ou invocação à métodos remotos ou locais é transparente



Remote Procedure Call (RPC) e *Remote Method Invocation* (RMI)

Transparência de Localização

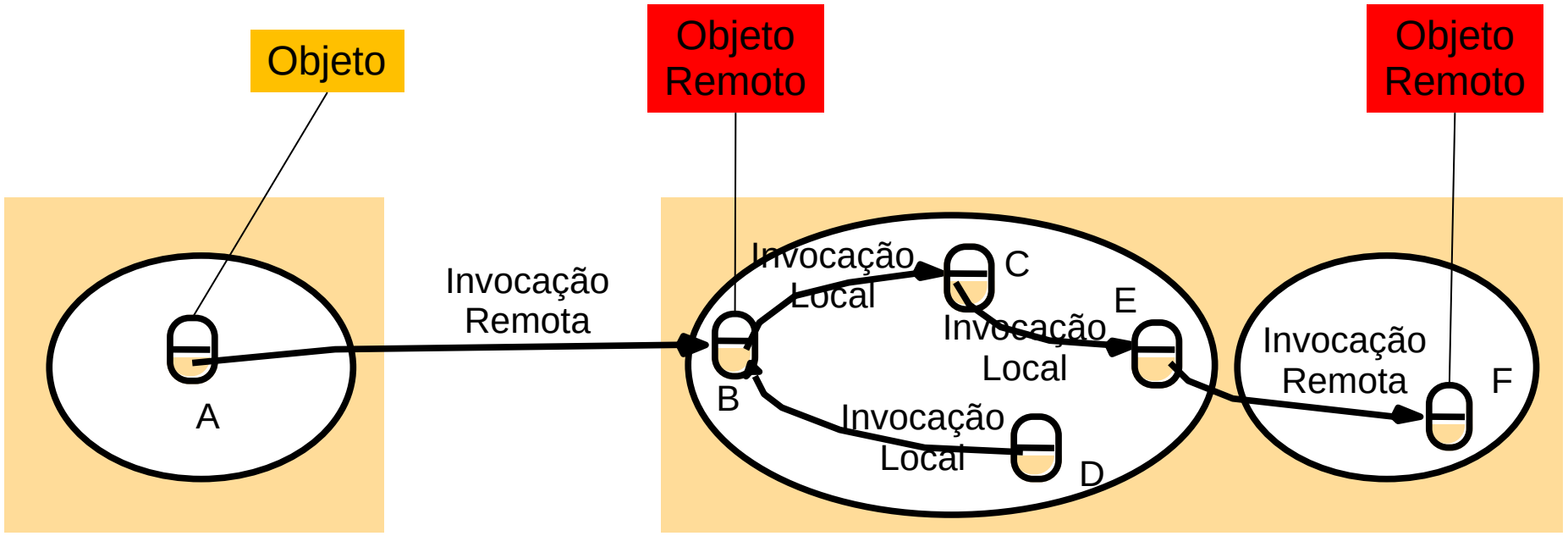
Chamada a procedimento ou invocação à métodos remotos ou locais é transparente



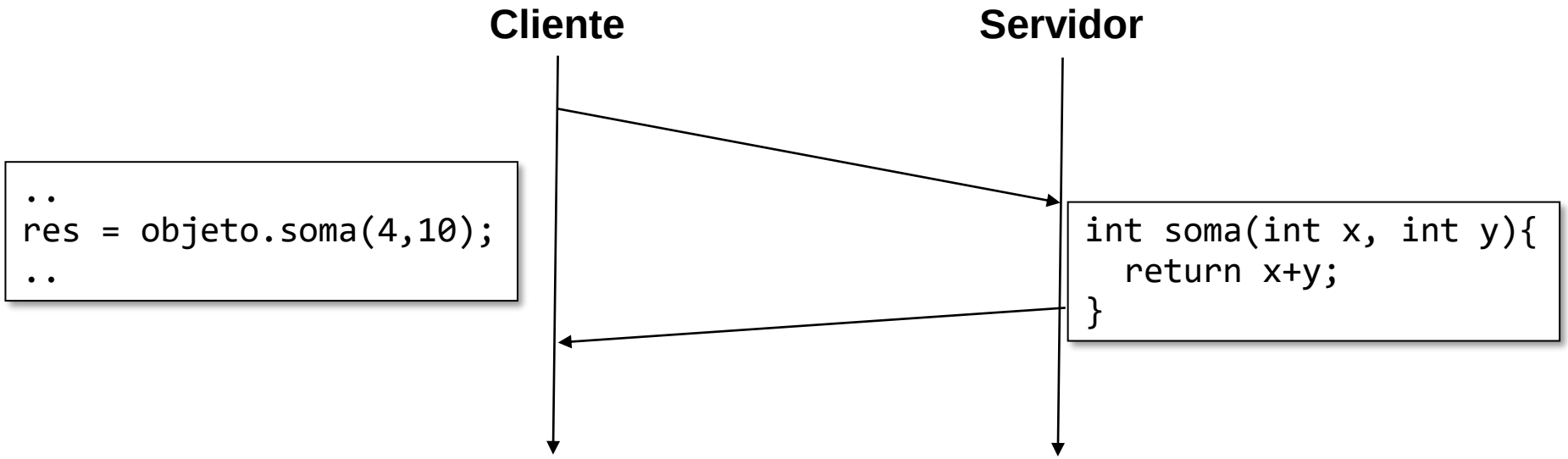
Transparência até certo ponto:

Semântica de Invocação Remota é diferente da Semântica de Invocação Local

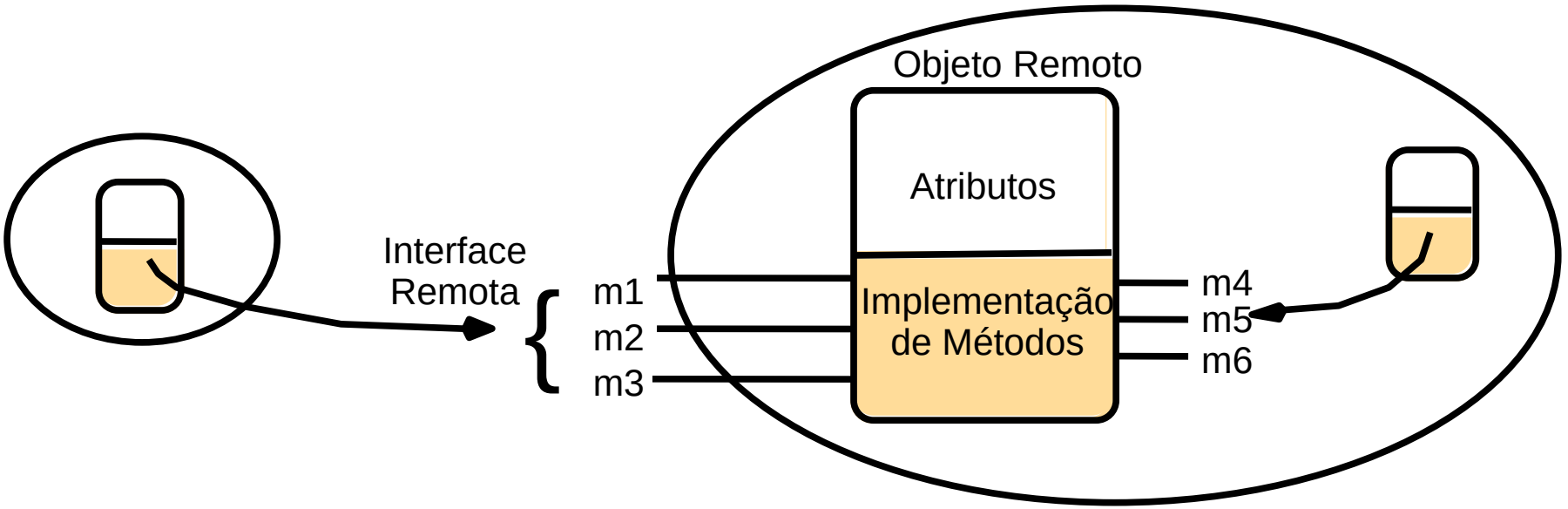
Invocação à Método Remoto



(Coulouris et al., 2005)



Um objeto Remoto e sua Interface



(Coulouris et al., 2005)

- A interface dos objetos de favorecer a Interoperabilidade entre:
  - Diferentes plataformas de execução
  - Diferentes linguagens de programação

➢ Uso de uma linguagem de descrição de interface independente de tecnologia

Exemplo de declaração de IDL – *Interface Description Language*

Uma linguagem é utilizada para definição de interfaces remotas (IDL)

EM CORBA os objetos remotos podem ser implementados em várias linguagens, como Java, C++, Cobol ou Python desde que haja um mapeamento para IDL e compiladores para estas linguagens

Java RMI permite a definição de interfaces remotas da mesma forma que interfaces convencionais, estendendo-se a interface *Remote*

Exemplo de IDL CORBA

```
// In file Person.idl
struct Person {
    string name;
    string place;
    long year;
} ;
interface PersonList {
    readonly attribute string listname;
    void addPerson(in Person p) ;
    void getPerson(in string name, out Person p);
    long number();
};
```

Limitações de uma Interface Remota

- Não permite acessar diretamente variáveis de outros módulos

- Não é possível fazer passagem de argumentos por referência para modificar os argumentos na origem
  - Ponteiros não podem ser utilizados como argumentos de chamadas

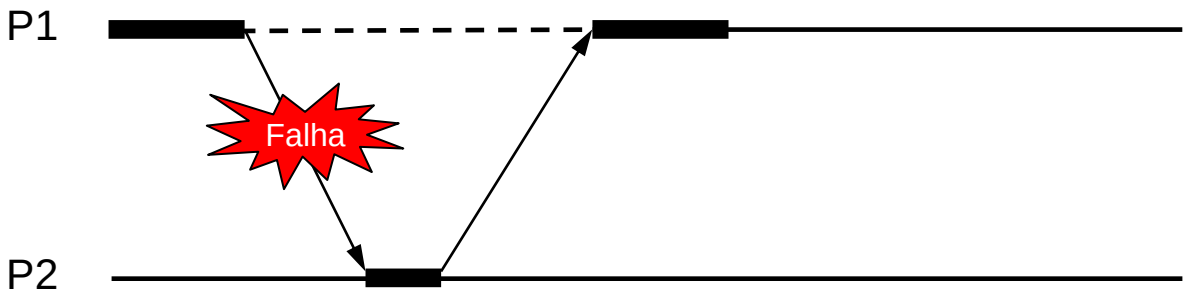
Transparência em Implementações RMI

- As implementações de RMI preocupam-se com:
  - Limitações das interfaces
    - Tipos de dados e tipos de passagens de argumentos
  - Transparência das invocações
    - Similares às invocações locais: sintaticamente e semanticamente
    - A latência é maior, devido à comunicação
  - Semântica das invocações
    - Invocação local: exatamente uma vez
    - Invocação remota: várias possibilidades
  - Exceções
    - A gama de exceções é maior: falhas de comunicação, falhas no servidor e problemas de concorrência!

Transparência em Implementações RMI

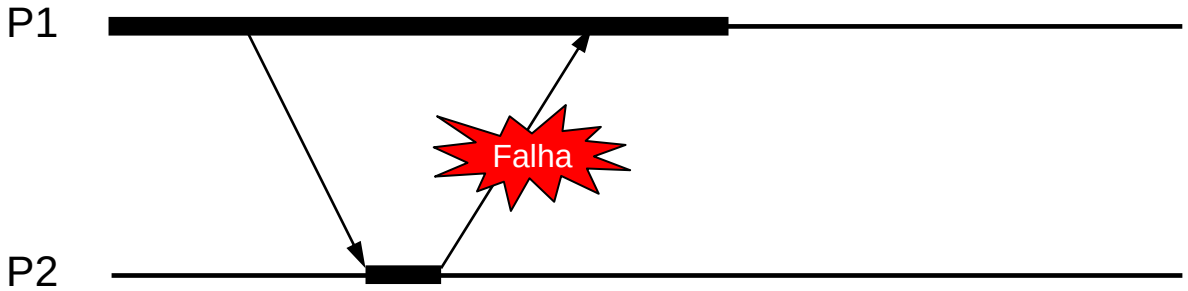
- Comunicação em RMI:

Comunicação Síncrona



Suscetíveis a falhas

Comunicação Assíncrona



Semântica de Invocação Remota – TALVEZ (0 ou 1)

• Ao receber uma resposta, o invocador tem certeza que o método remoto foi executado somente uma vez

• Ao receber exceção (resposta não chegou, *timeout*), o invocador não sabe se o método foi executado ou não, pois se:

- Falha do servidor
  - Requisição chega, mas servidor cai antes ou durante a execução do método (0)
  - Requisição chega, servidor executa o método e falha em seguida (1)
- Falha no canal de comunicação
  - Requisição não chega no servidor (0)
  - Resposta não chega: servidor executa o método, envia resposta, mas resposta não chega no cliente ou chega depois do timeout (1)

Semântica de Invocação Remota – Ao Menos Uma (1 ou +)

- Ao receber uma resposta, o invocador sabe que o método remoto foi executado ao menos uma vez, pois:
  - Falha do canal de comunicação
    - Requisição chega, servidor executa o método e envia resposta
    - Resposta não chega, requisição é retransmitida
    - Servidor executa o método e envia resposta, ... até que a resposta chegue! (1 ou +)
- Ao receber exceção, o invocador não saberá se o método foi executado ou não, pois se:
  - Falha do servidor
    - Requisição chega, mas servidor cai antes ou durante a execução do método, retransmissões do requisição, ..., exceção! (0)
    - Requisição chega, servidor executa o método e falha em seguida, retransmissões da requisição, ..., exceção! (1)
  - Falha no canal de comunicação
    - Requisição não chega no servidor, retransmissões da requisição, ..., exceção! (0)
    - Resposta não chega: servidor executa o método, envia resposta, mas resposta não chega no cliente, retransmissão da requisição, servidor executa método, responde, mas resposta não chega ... exceção! (1 ou +)

Semântica de Invocação Remota – No Máximo Uma (0 ou 1)

- Ao receber uma resposta, o invocador sabe que o método foi executado uma só vez, pois se:
  - Falha do canal de comunicação
    - Requisição chega no servidor
    - Servidor executa o método
    - Servidor envia resposta
    - Resposta não chega...
    - Retransmissão da requisição
    - Servidor detecta que a requisição já foi executada e retorna a resposta com o resultado anterior
    - Se a resposta chegar no cliente
      - Método terá sido executado uma só vez
    - Se não
      - Após um certo número de retransmissões, uma exceção será gerada
- Ao receber exceção, o invocador sabe que o método não foi executado ou foi executado somente uma vez

Programação RMI

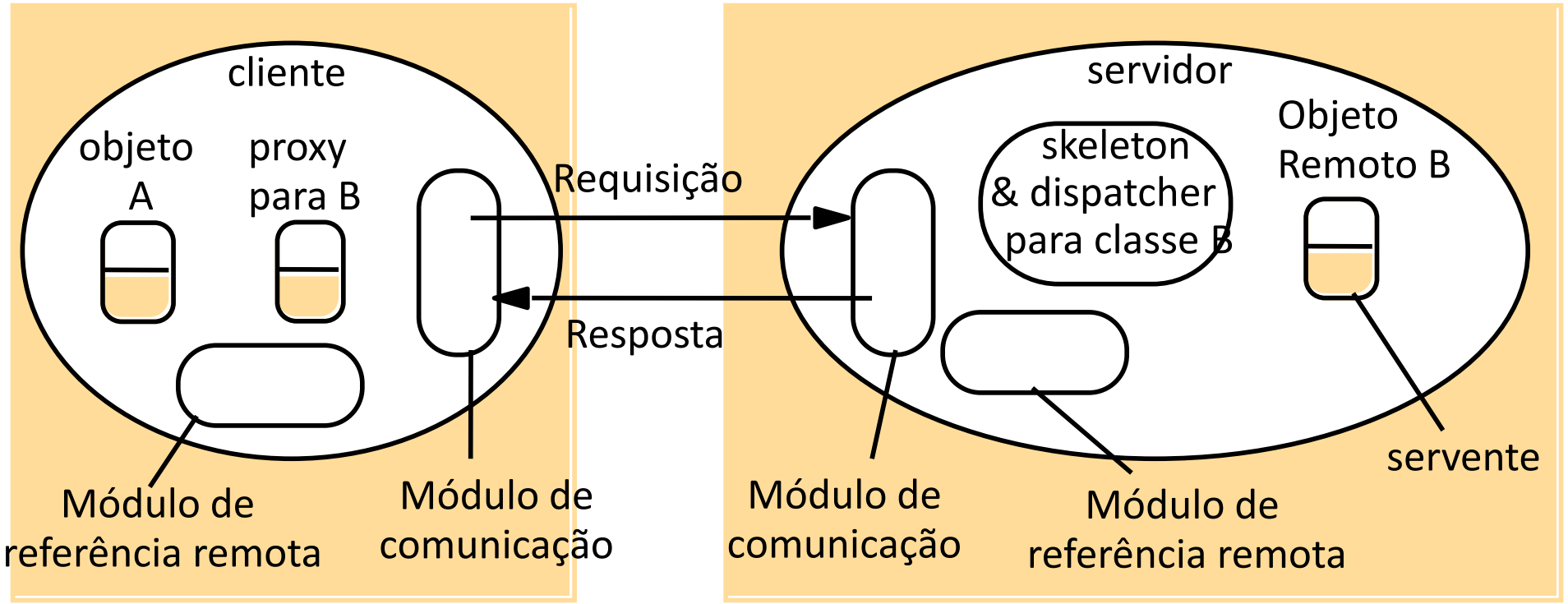
- *Software* RMI
  - Código auxiliar gerado pelo compilador de IDL ou pelo ambiente de execução da linguagem (exemplo JVM)

  - *Proxy / Stub*
    - Comporta-se como o objeto remoto no lado cliente (*invoker*)
    - Serializa (*marshelling*) os argumentos, encaminha mensagens aos objetos remotos, deserializa (*unmarshelling*) os resultados, retorna os resultados ao cliente

  - Esqueleto (*Skeleton*)
    - *Stub* no lado servidor
    - Deserializa argumentos, invoca o método, serializa os resultados e envia para o método do *proxy* chamador

  - Despachante (*Dispatcher*)
    - Recebe mensagens de requisição do módulo de comunicação, encaminha a mensagem para o esqueleto do método apropriado

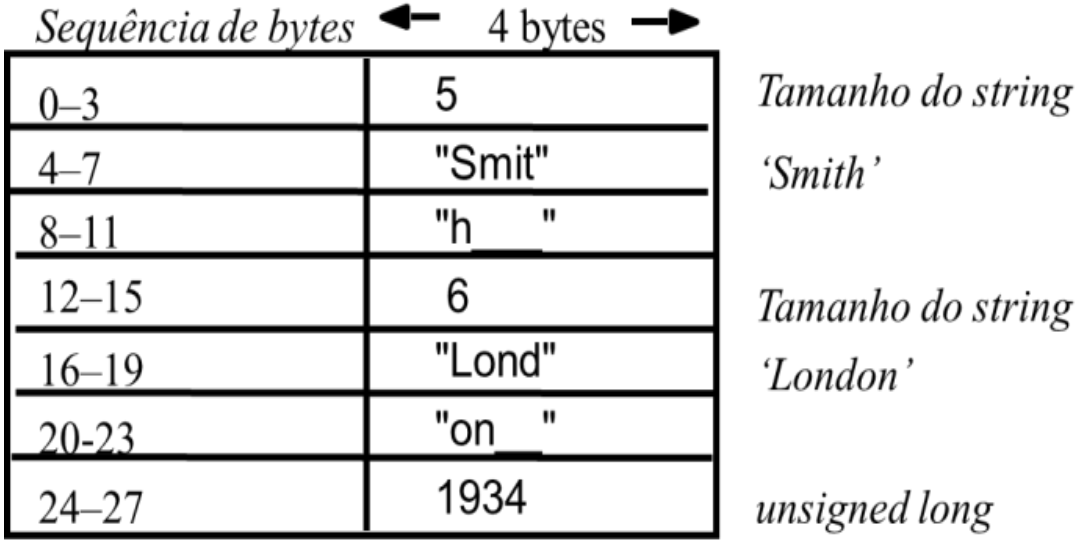
Visão Geral do Mecanismo de RMI



(Coulouris et al., 2005)

Representação Comum de Dados

struct Person:
{‘Smith’, ‘London’, 1934}.



Sequência de bytes ← 4 bytes →		
0–3	5	*Tamanho do string*
4–7	"Smit"	*‘Smith’*
8–11	"h___"	
12–15	6	*Tamanho do string*
16–19	"Lond"	*‘London’*
20-23	"on__"	
24–27	1934	*unsigned long*

Outros padrões para serialização de dados:

- XML, JSON – formatos textuais
- Protocol Buffers (Google), Apache Thrift (Facebook), Ion (Amazon) – formatos binários

C. Dollimore and T.Kindberg. Distributed Systems: Concepts and Design

Algumas Tecnologias para Objetos Distribuídos

OMG CORBA (*Common Object Request Broker Architecture*)

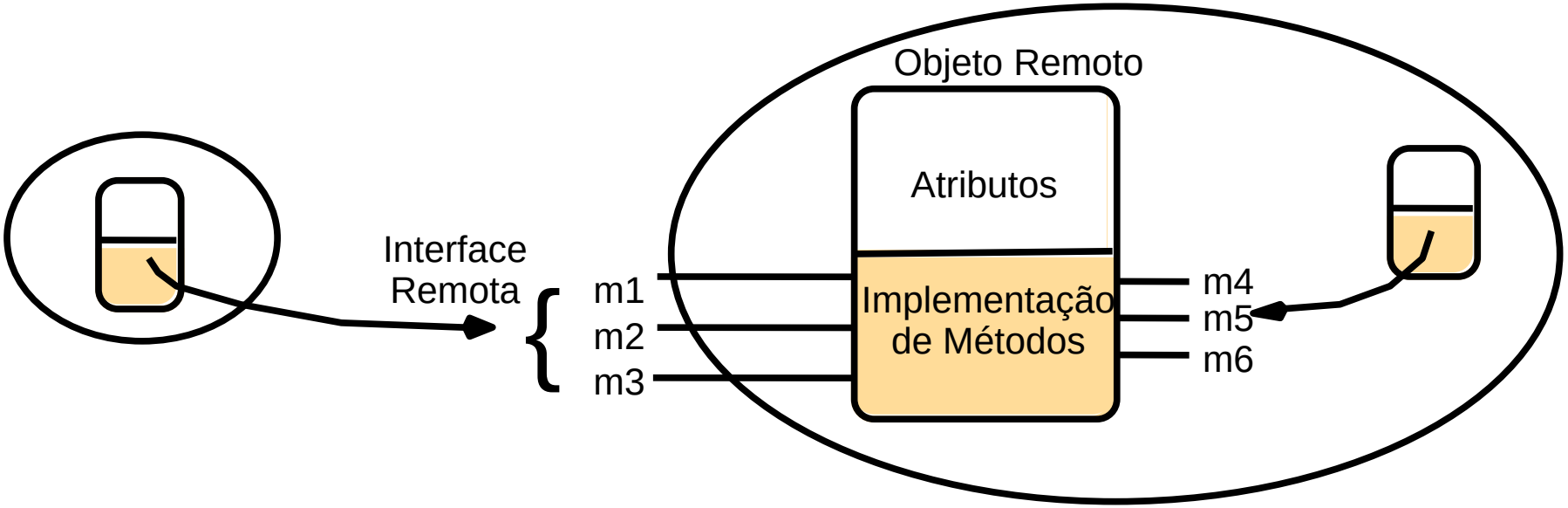
Java RMI (*Remote Method Invocation*)

Microsoft DCOM (*Distributed Common Object Model*)

Apache Thrift

Exemplo Java RMI

Implementando RMI em Java



Serviço Local (Cliente)

```
SampleServer remoteObject;
int s;
…
s = remoteObject.sum(1,2);



System.out.println(s);
```

Serviço Remoto (Servidor)

```
public int sum(int a,int b) {
return a + b;
}
```

Implementando RMI em Java

Interface

```
public interface SampleServer extends Remote{

public int sum (int a, int b) throws
RemoteException;
public int sub (int a, int b) throws
RemoteException;

}
```

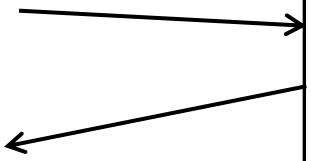
implements

Serviço Local (Cliente)

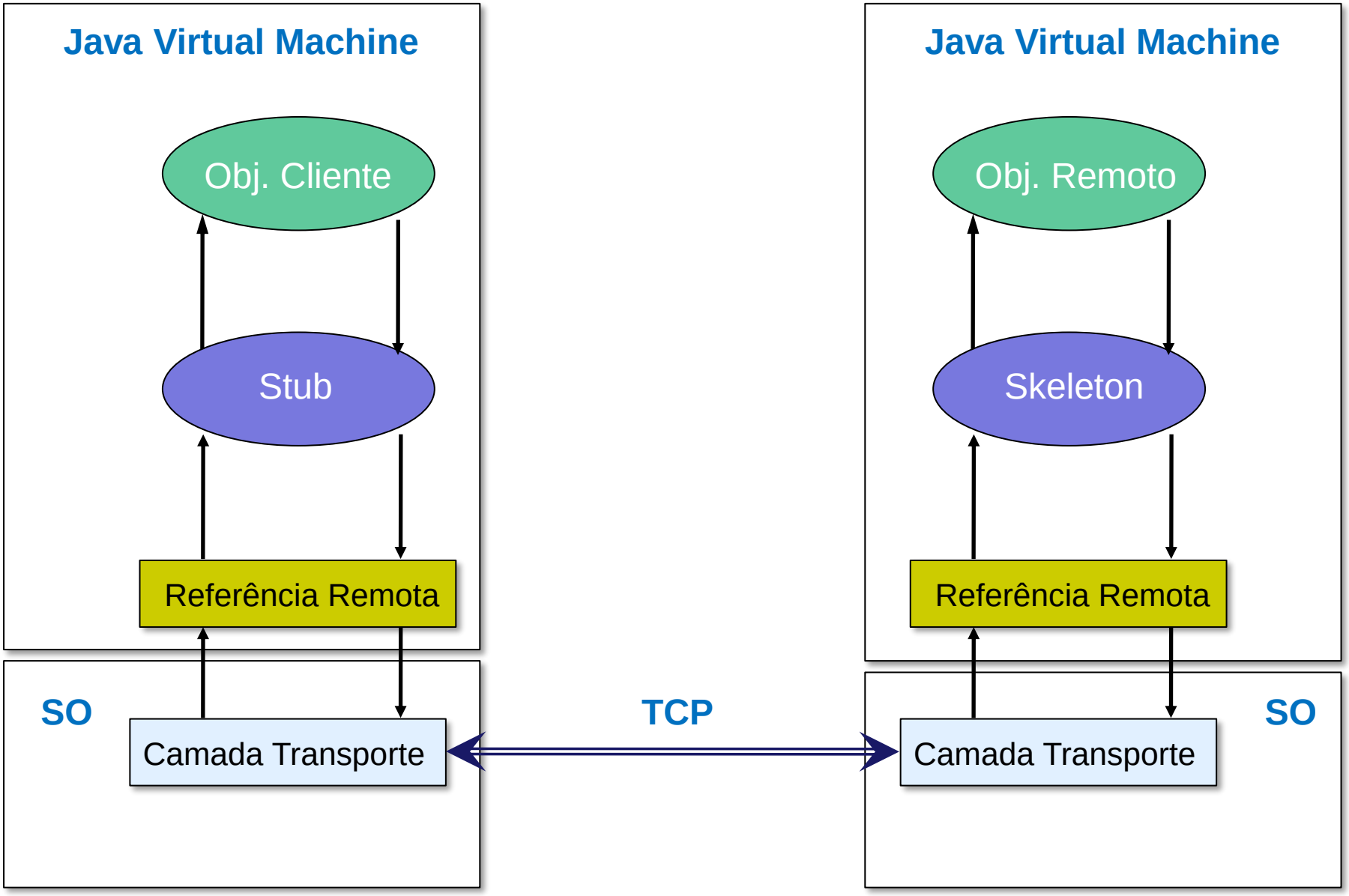
```
SampleServer remoteObject;
int s;
…
s = remoteObject.sum(1,2);



System.out.println(s);
```

Serviço Remoto (Servidor)

```
public int sum(int a,int b) {
return a + b;
}
```



Implementando RMI em Java



Implementando RMI em Java – Exemplo Guiado

Interface Remota

```
import java.rmi.*;

public interface HelloInterface extends Remote {

    public String sayHello() throws RemoteException;

}
```

Implementando RMI em Java – Exemplo Guiado

Implementação do Método Remoto

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

public class Hello extends UnicastRemoteObject implements
HelloInterface {
    private String message; // Strings are serializable
    public String sayHello() throws RemoteException {
    return message;
    }

    public Hello (String msg) throws RemoteException {
    message = msg;
    }
}
```

Implementando RMI em Java – Exemplo Guiado

Registro do Objeto Remoto no Servidor

```
class HelloServer {
    public static void main (String[] argv) {
        try {
            Naming.rebind("rmi://localhost/HelloServer", new
    Hello("Hello, world!"));
        }
         catch (Exception e) { }
    }
}
```

Implementando RMI em Java – Exemplo Guiado

Invocação ao Objeto Remoto pelo Cliente

```
class HelloClient {
    public static void main (String[] args) {
        HelloInterface hello;
        String name = "rmi://localhost/HelloServer";
        String text;
        try {
            hello = (HelloInterface)Naming.lookup(name);
            text = hello.sayHello();
            System.out.println(text);
        }  catch (Exception e) {
            System.out.println("HelloClient exception:“+e);
        }
    }
}
```

Implementando RMI em Java – Exemplo Guiado

Passos de Compilação

Compilar todos os arquivos:
- HelloInterface.java
- Hello.java
- HelloServer.java
- HelloClient.java

```
javac *.java
```

Criar Stubs e Skeletons (proxies)

```
rmic Hello
```

- Hello_Stub.class
- Hello_Skel.class

Arquivos Gerados no Cliente

- HelloInterface.class
- Hello_Stub.class
- HelloClient.class

Arquivos Gerados no Servidor

- HelloInterface.class
- Hello.class
- HelloServer.class

rmic é necessário apenas para versões da Plataforma Java, anteriores a Standard Edition 5.0

Implementando RMI em Java – Exemplo Guiado

Passos para Execução

Servidor

```
$ rmiregistry
$ java HellowServer
```

Cliente

```
$ java HelloClient
```

Implementando RMI em Java – Exemplo Guiado

Passos para Execução

Servidor

```
$ rmiregistry
$ java HellowServer
```

Cliente

```
$ java HelloClient
```

O serviço de registro (*binder*) roda no servidor

```
rmiregistry
```

É necessário registrar o objeto remoto no servidor

```
String url= "rmi://" + host+ ":" +port+ "/" + objectName;
```

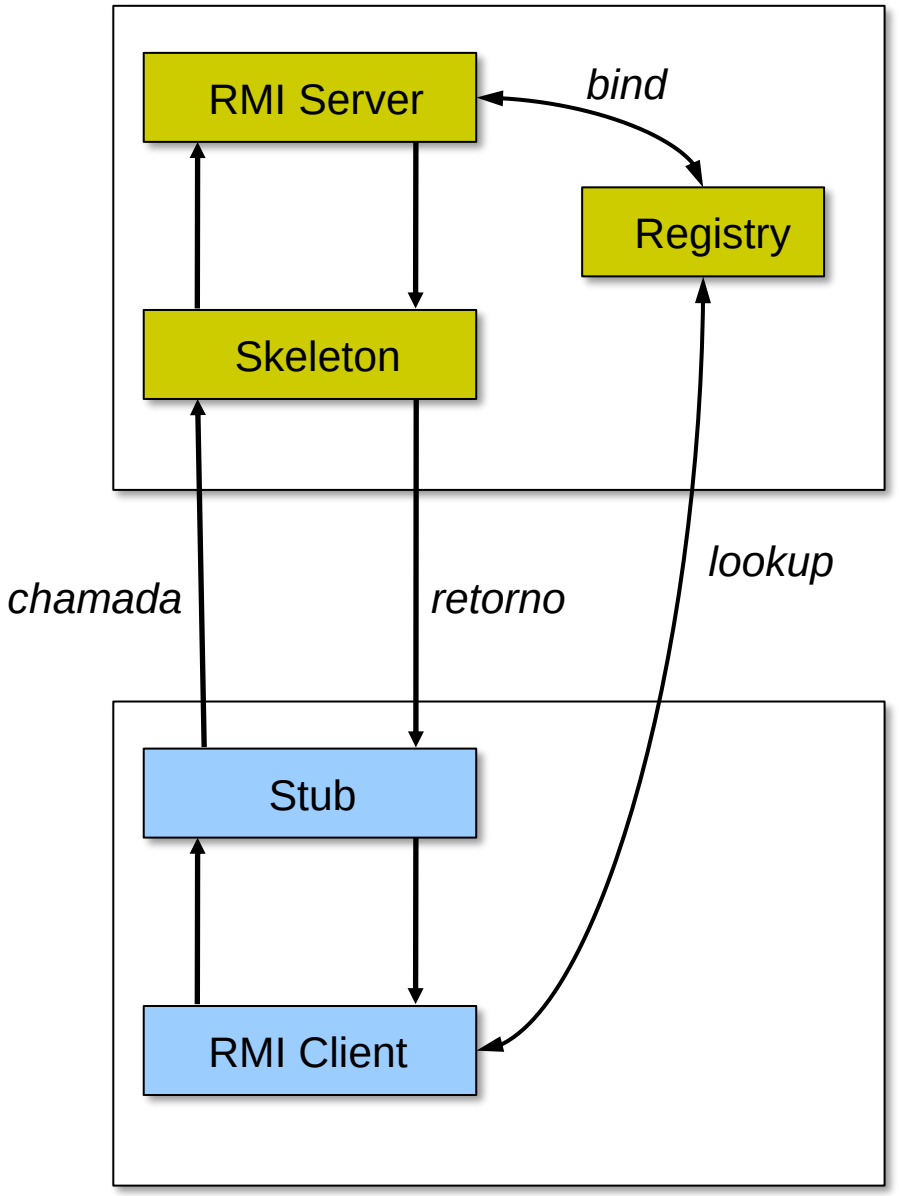
A porta padrão é a 1099

Visão Geral da Arquitetura RMI

Servidor se registra em registry através do método rebind

Cliente contata registry com método lookup

Cliente, através do stub, pode fazer chamadas ao método remoto



Referências

Parte destes slides são baseadas em material de aula dos livros:

- *Coulouris, George; Dollimore, Jean; Kindberg, Tim; Blair, Gordon. Sistemas Distribuídos: Conceitos e Projetos. Bookman; 5ª edição. 2013. ISBN: 8582600534*

- *Tanenbaum, Andrew S.; Van Steen, Maarten. Sistemas Distribuídos: Princípios e Paradigmas. 2007. Pearson Universidades; 2ª edição. ISBN: 8576051427*