

Programando sistemas distribuídos com objetos distribuídos na rede TCP/IP

Prof. Me. Sérgio Carlos Portari Júnior

Conteúdo Programático

- Contextualizando: Aula anterior
- Camada Middleware
- Programar para SD
 - SOCKETS
 - RPC
 - OBJETOS DISTRIBUÍDOS
 - ORB
- Concluindo
- Próxima aula
- Atividades para Fixação
- Referências bibliográficas

Aulas anteriores

- O QUE SÃO SISTEMAS DISTRIBUÍDOS (SD)?
- Tanenbaum: “Um conjunto de máquinas independentes que fornecem uma visão de uma única máquina para os usuários”
- Coulouris: “Sistemas onde componentes de hardware e software localizados em rede, comunicam-se somente através de troca de mensagens”

Aulas anteriores

- POR QUE CONSTRUIR UM SD?
 - Compartilhamento de recursos
 - Custo/desempenho
 - Independência de localização
 - Pessoas e informação são distribuídas
 - Expansão
 - Escalabilidade
 - Disponibilidade
 - Confiabilidade

Aulas anteriores

- DIFICULDADES NO USO DO SD
 - Concorrência
 - Sem relógio global
 - Estados inconsistentes
 - Falhas independentes

Conceito de Middleware

COMO DESENVOLVER SD QUE POSSA
ATENDER TODOS USUÁRIOS?

Existe uma grande variedade de plataformas disponíveis no mercado.

Plataforma: Arquitetura + Sistema Operacional

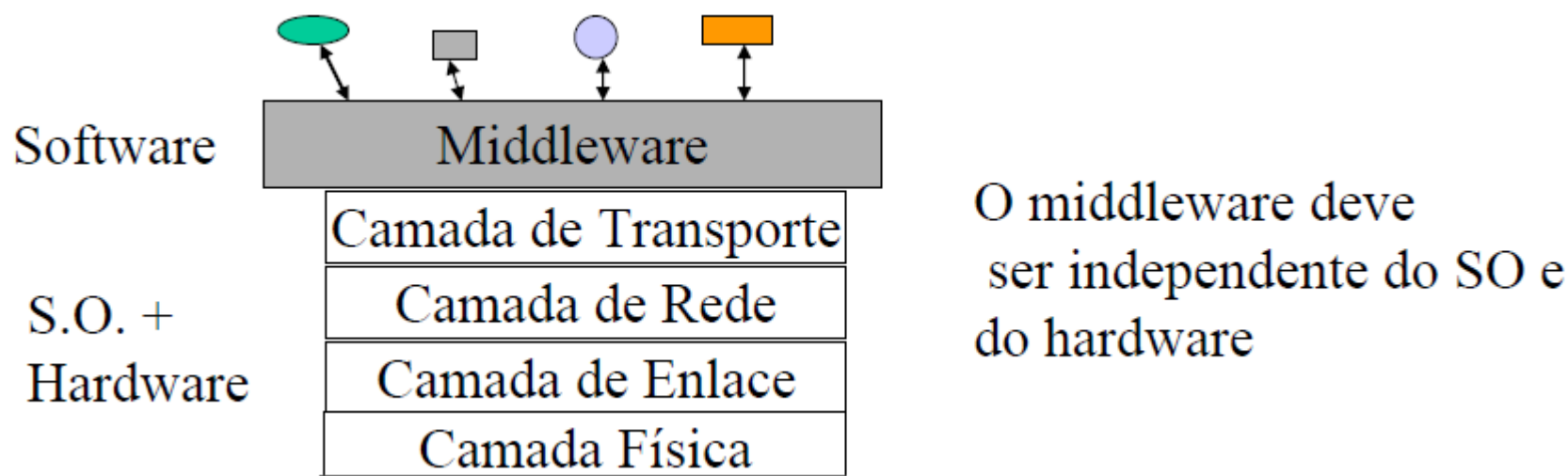
- X86/Linux
- X86/Windows
- Sparc/Linux
- I64/Linux
- I64/Windows

Conceito de Middleware

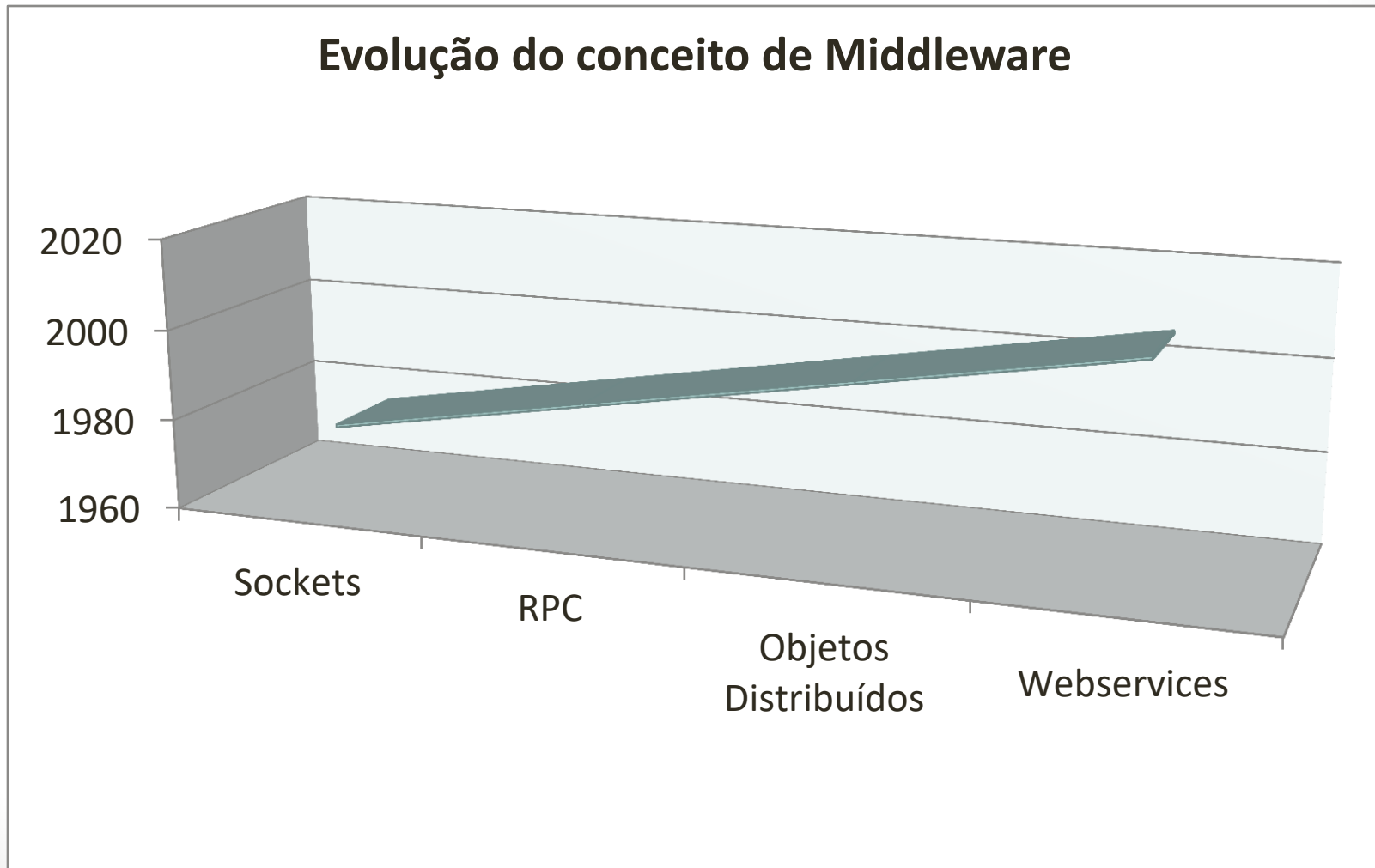
- MIDDLEWARE

Necessidade de agrupar funcionalidades comuns às várias aplicações distribuídas

– Conceito de abstração da programação distribuída



Conceito de Middleware



Fonte: Adaptado de OLIVEIRA (2006)

Programar para SD

- SOCKETS
 - Sua principal função é transmitir mensagens através de portas abertas por processos em diferentes máquinas.
 - Números de portas disponíveis é 2^{16}
 - 2 processos não podem estar utilizando uma porta ao mesmo tempo.
 - Algumas portas são, por padrão, reservadas a serviços específicos, por exemplo, http (80), ftp (20 e 21), ssh (22), smtp (25), dentre outras.

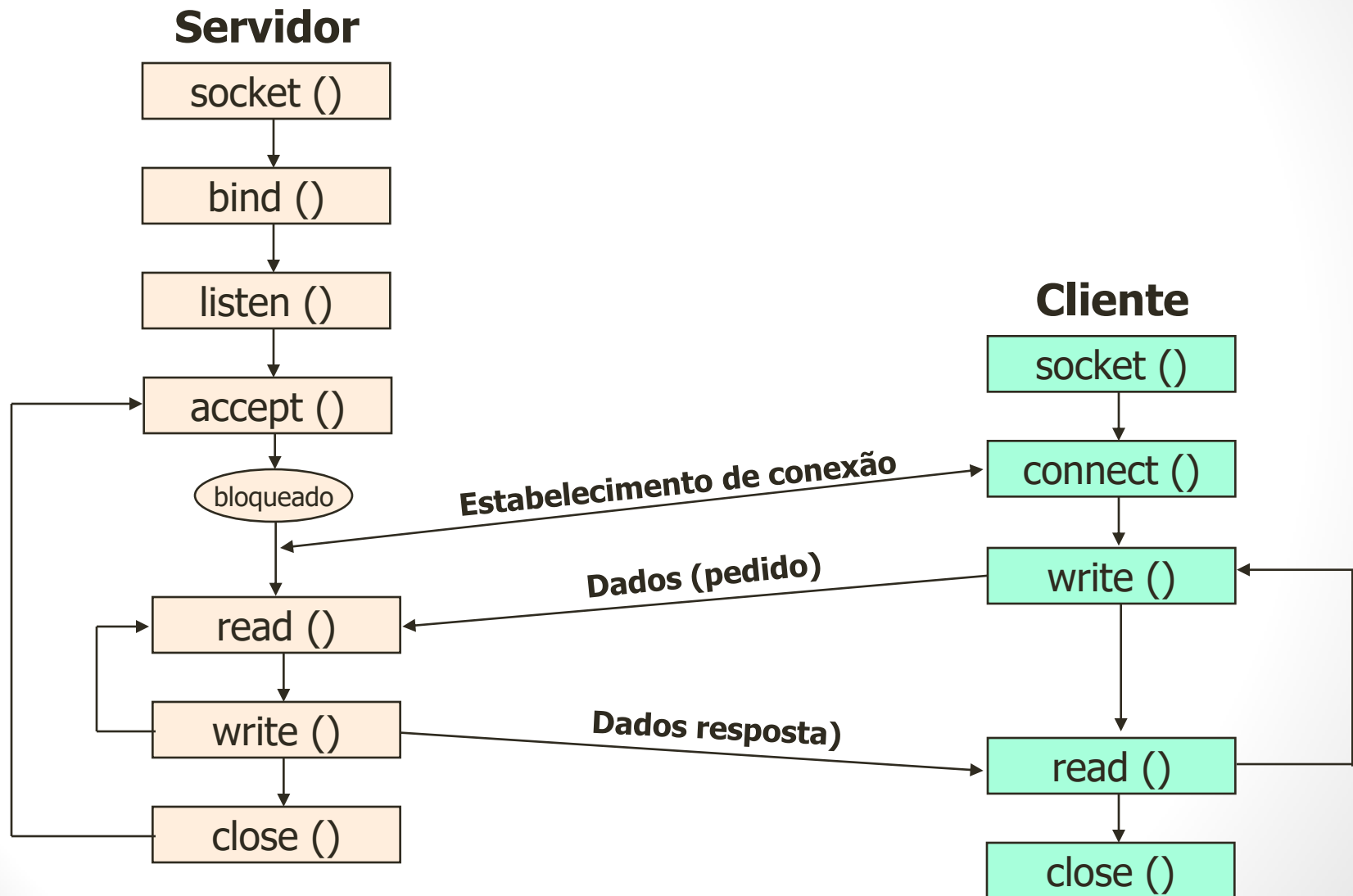
Programar para SD

- SOCKETS
 - Socket Passivo: espera por uma conexão (usado por Servidores)
 - Socket Ativo: Inicia uma conexão (usado pelos Clientes)
 - Complexidade do Socket: parâmetros que o programador pode configurar.

Programar para SD

- SOCKETS
 - Socket pode utilizar dois tipos de canais conexão:
 - Não-confiável (usando protocolo UDP)
 - Confiável (usando protocolo TCP)

Programar para SD



Fonte: Adaptado de Tanenbaum (2003)

Programar para SD

- EXEMPLO DE UM SERVIDOR SOCKET EM JAVA

```
import java.net.*;
import java.io.*;
// cria um socket UDP usando a porta 6789
DatagramSocket s = new DatagramSocket(6789);
byte[] buffer = new byte[1000];
System.out.println("*** Servidor aguardando requisição do cliente");
// cria datagrama para receber requisição do cliente
DatagramPacket r = new DatagramPacket(buffer, buffer.length);
s.receive(r);
System.out.println("*** Requisição recebida de: " + r.getAddress());
// envia resposta
DatagramPacket resp = new DatagramPacket(r.getData(), r.getLength(),
r.getAddress(), r.getPort());
s.send(resp);
s.close();
```

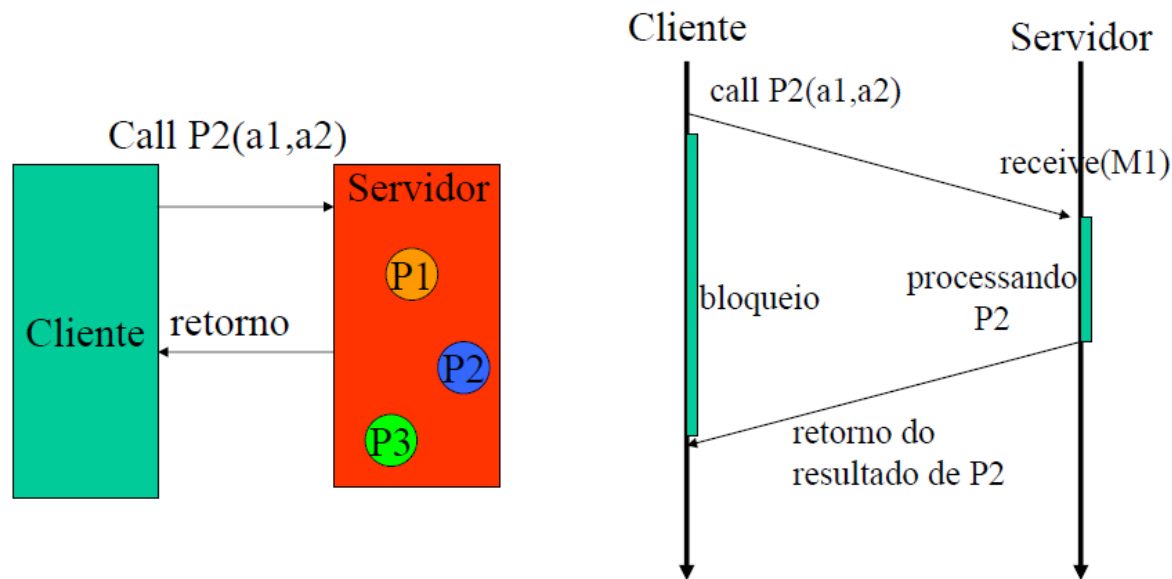
Programar para SD

- EXEMPLO DE UM CLIENTE SOCKET EM JAVA

```
import java.net.*;
import java.io.*;
// cria um socket UDP
s = new DatagramSocket();
System.out.println("* Socket criado na porta: " + s.getLocalPort());
byte[] m = args[0].getBytes(); // transforma arg em bytes
InetAddress serv = InetAddress.getByName(args[1]);
int porta = 6789;
DatagramPacket req = new DatagramPacket(m, args[0].length(), serv, porta);
s.send(req); // envia datagrama contendo a mensagem m
byte[] buffer = new byte[1000];
DatagramPacket resp = new DatagramPacket(buffer, buffer.length);
s.setSoTimeout(10000); // timeout em ms
// recebe resposta do servidor – fica em wait ateh chegada
s.receive(resp);
System.out.println("* Resposta do servidor:" + new String(resp.getData()));
// fecha socket
s.close();
```

Programar para SD

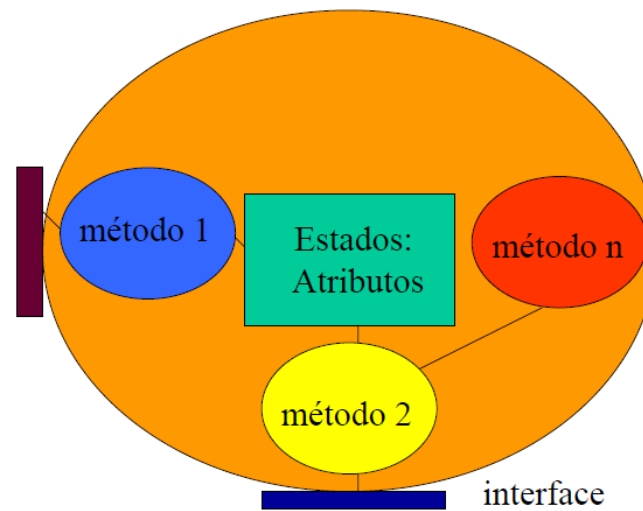
- RPC
- Baseada nas chamadas de procedimentos estruturadas



Fonte: Adaptado de Tanenbaum (2008)

Programar para SD

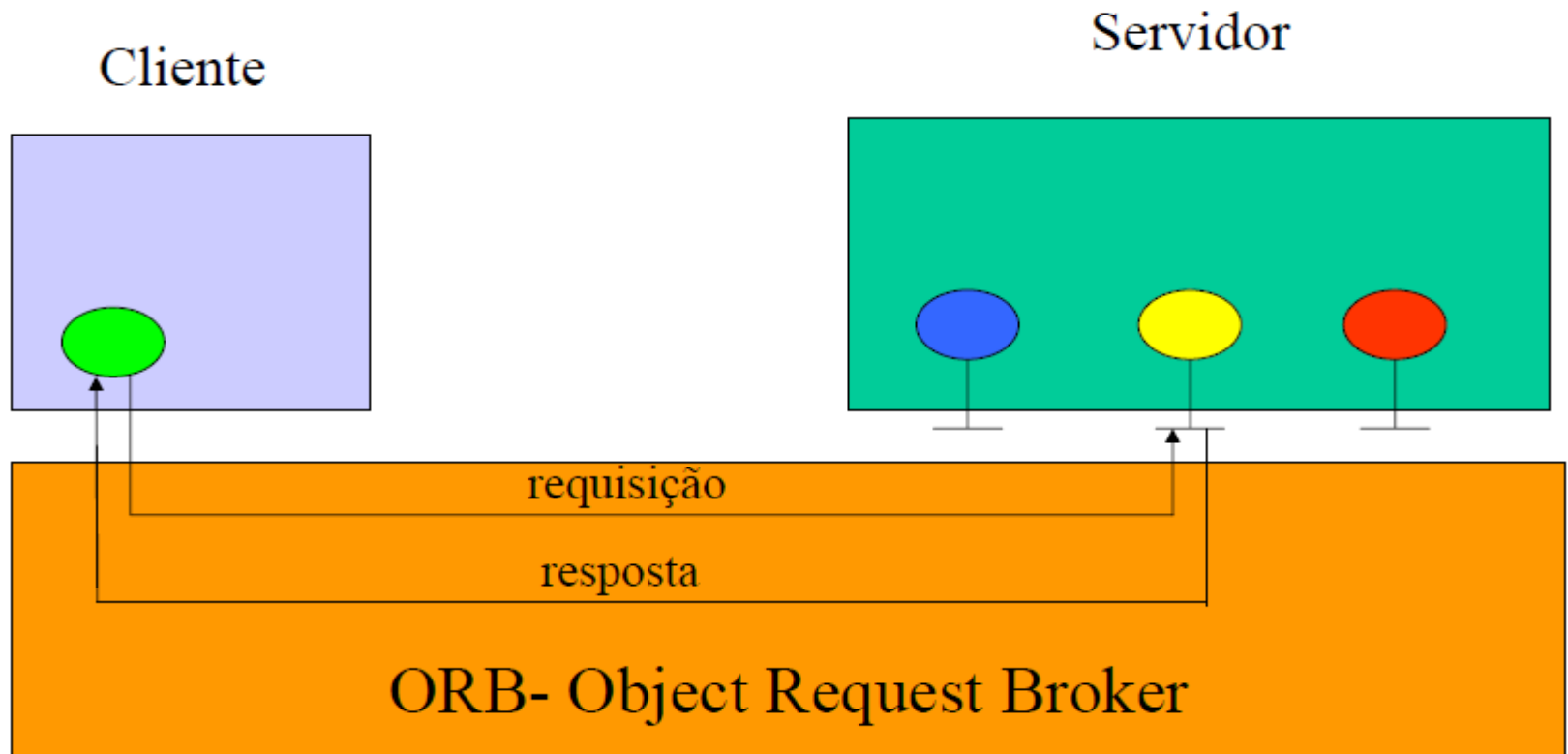
- OBJETOS DISTRIBUÍDOS
- Objetos Distribuídos = Programação Orientada a Objetos + Programação Distribuída



Fonte: Adaptado de Deitel (2005)

Programar para SD

- OBJETOS DISTRIBUÍDOS



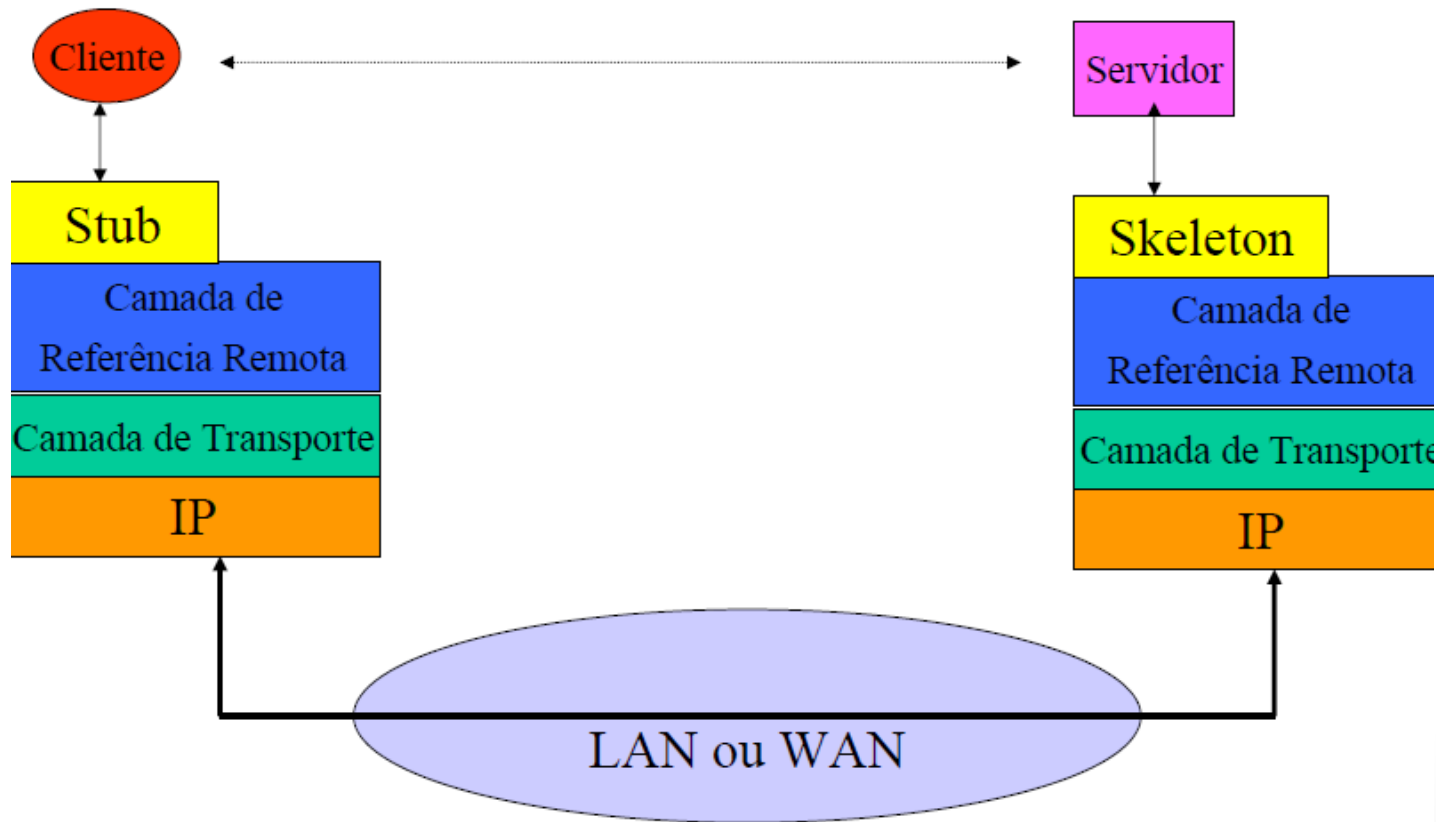
Fonte: Adaptado de Tanenbaum (2008)

Programar para SD

- ORB
 - Exemplos de Middlewares ORBs
 - CORBA – Common ORB Architecture
 - RMI – Remote Method Invocation (Java)
 - DCOM - Distributed Component Object Model
 - SOAP - Simple Object Access Protocol

Programar para SD

- ESTRUTURA DO ORB



Fonte: Adaptado de Tanenbaum (2008)

Programar para SD

- RMI
 - Exemplo de utilização de RMI - Servidor

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

```
public interface Hello extends Remote  
{  
    public String sayHello() throws RemoteException;  
}
```

Programar para SD

- RMI

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

```
public class servidor  
    extends UnicastRemoteObject  
    implements Hello{ //Implementa a interface
```

```
    public servidor() throws RemoteException { // Construtor  
        super(); //chama o construtor da classe pai  
    }
```

```
    // Método remoto
```

```
        public String sayHello() throws RemoteException {  
            return("Oi cliente");  
        } //continua...
```

Programar para SD

- RMI

//continuação

```
public static void main(String args[]) {  
    try{  
        servidor serv=new servidor();  
        // Registra nome do servidor  
        Naming.rebind("ServidorHello",serv);  
        System.out.println("Servidor remoto pronto.");  
    }  
    catch(RemoteException e) {  
        System.out.println("Exceção remota:"+e);  
    }  
    catch(MalformedURLException e) { } ;  
}  
}
```

Programar para SD

- RMI

- Exemplo de utilização de RMI – Cliente

```
import java.rmi.*;
class cliente{
    public static void main(String args[]){
        try{
            Servidor serv= (Servidor) Naming.lookup
("rmi://localhost/ServidorHello");
            String retorno=serv.sayHello();
        }
        catch(Exception e);
    }
}
```

Concluindo

- Podemos fazer uma comparação de Sockets x RMI
 - Na construção
 - Velocidade de execução
 - Interface e Nomes
 - RMI usa Sockets
 - TCP / UDP
 - Localização

Atividades para Fixação

1. Utilizando Sockets, crie uma aplicação cliente/servidor em que o cliente envia uma string e o servidor retorna uma string invertida.

Referências Bibliográficas

- COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Sistemas distribuídos: conceitos e projeto**. 4 edição. Bookman, Porto Alegre-RS, 2007.
- DEITEL, H. M.; DEITEL, P. J. **Java: Como Programar**. 6.ed. São Paulo: Pearson Education, 2005.
- OLIVEIRA, L. A. H. G. **Programação em Rede**. Notas de Aula (2006) Disponível em http://www.dca.ufrn.br/~affonso/DCA2401/2004_1/notas_de_aulas.html. Acesso em 29/04/2017.
- SOBRAL, J. B. M, **Programação Paralela e Distribuída**. Notas de Aula (2017). (Disponível em <http://www.inf.ufsc.br/~bosco/ensino/ine5645.html>. Acesso em 29/04/2017).
- TACLA, C. A., **Sistemas distribuídos II**. Notas de Aula (2006) Disponível em <http://www.dainf.ct.utfpr.edu.br/~tacla/SDII>. Acesso em 29/04/2017.
- TANENBAUM, A. S. **Redes de Computadores**. 4 edição. Campus, Rio de Janeiro- RJ, 2003.
- TANENBAUM, A. S.; STEEN, M. V. **Sistemas Distribuídos: princípios e paradigmas**. 2 edição. Pearson Prentice Hall, São Paulo-SP, 2008.