



**UNIFESSPA**

UNIVERSIDADE FEDERAL DO SUL E SUDESTE DO PARÁ

# Universidade Federal do Sul e Sudeste do Pará

---

## Sistemas Distribuídos

*Prof.: Warley Junior*

[wmvj@unifesspa.edu.br](mailto:wmvj@unifesspa.edu.br)

# Agenda

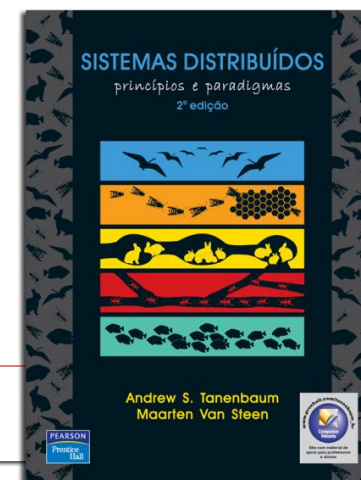
---

- ❑ AULA 7:
- ❑ Comunicação Distribuída
- ❑ Middlewares
  - RPC
  - XML RPC

# Leitura Prévia

---

- ❑ COULOURIS, George. Sistemas distribuídos: conceitos e projetos. 5ª ed. Porto Alegre: Bookman, 2013.
  - Capítulo 5.
- ❑ TANENBAUM, Andrew S. Sistemas distribuídos: princípios e paradigmas. 2ª ed. São Paulo: Pearson Prentice Hall, 2007.
  - Capítulo 4.



# Comunicação entre processos

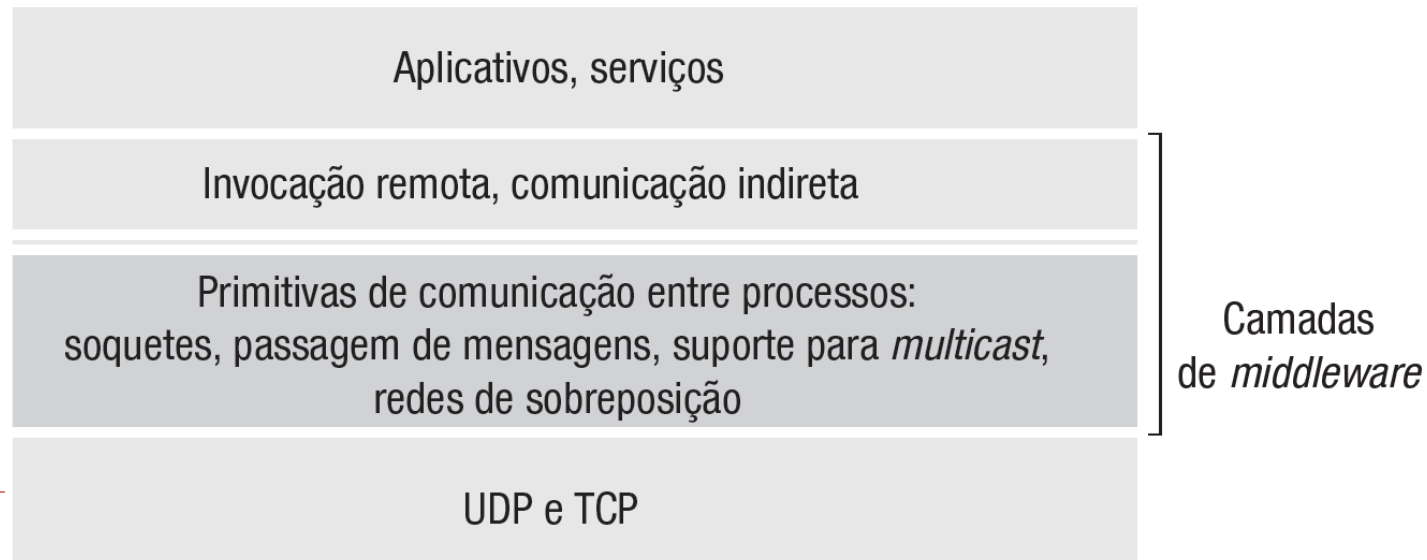
---

- ❑ “Coração” de todo sistema distribuído
- ❑ Como processos de diferentes máquinas trocam informações?
  - Não é uma tarefa fácil
- ❑ O objetivo é prover transparência desta comunicação ao desenvolvedor também

# Camada de middleware

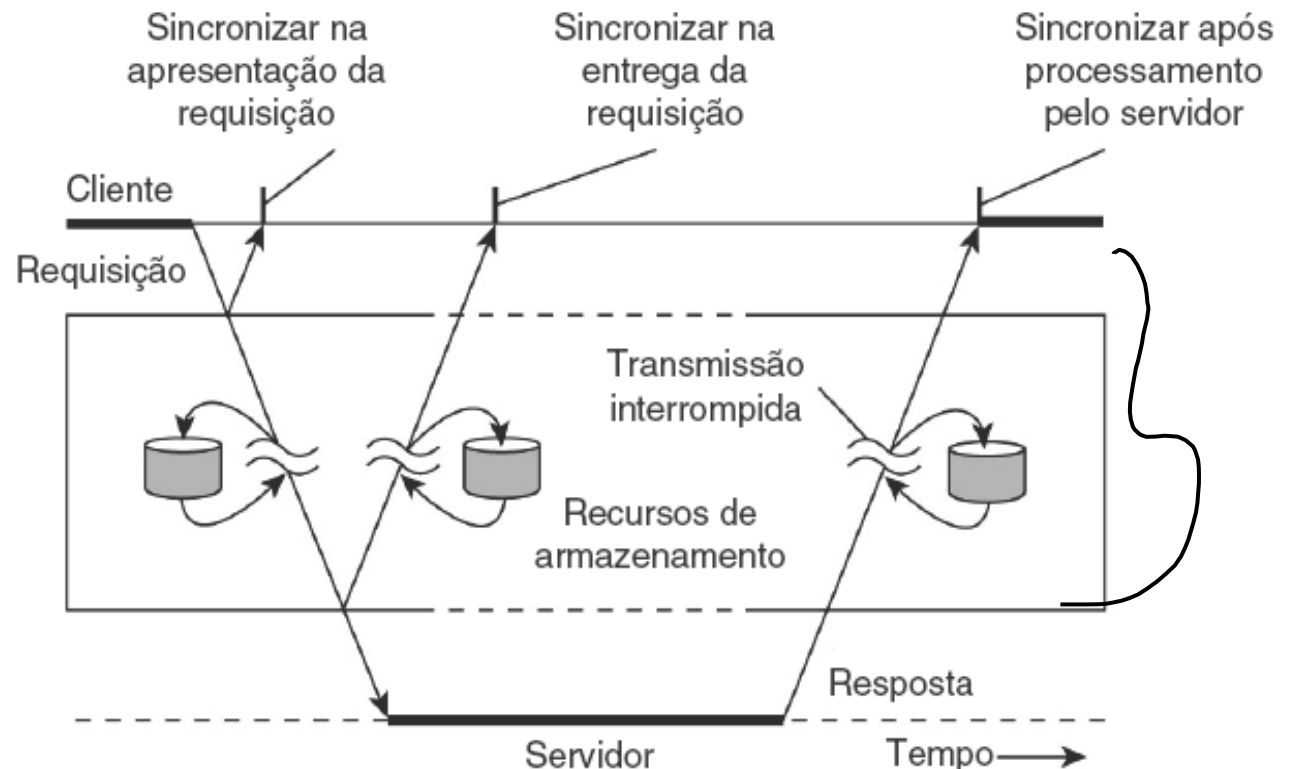
---

- ❑ Camada de software situada logicamente entre a camada de aplicação e a de transporte.
- ❑ Interface **única**



# Tipos de comunicação

- ❑ Middleware visto como serviço intermediário na comunicação de nível de aplicação.



# Tipos de comunicação (Persistência)

---

- ❑ **Persistente:** mensagem armazenada durante o tempo que for necessário para entregá-la ao receptor.
- ❑ **Transiente:** mensagem armazenada somente durante a execução do remetente e do receptor.
- ❑ Mensagens são armazenadas ou não pelo middleware.

# Tipos de comunicação (Granularidade)

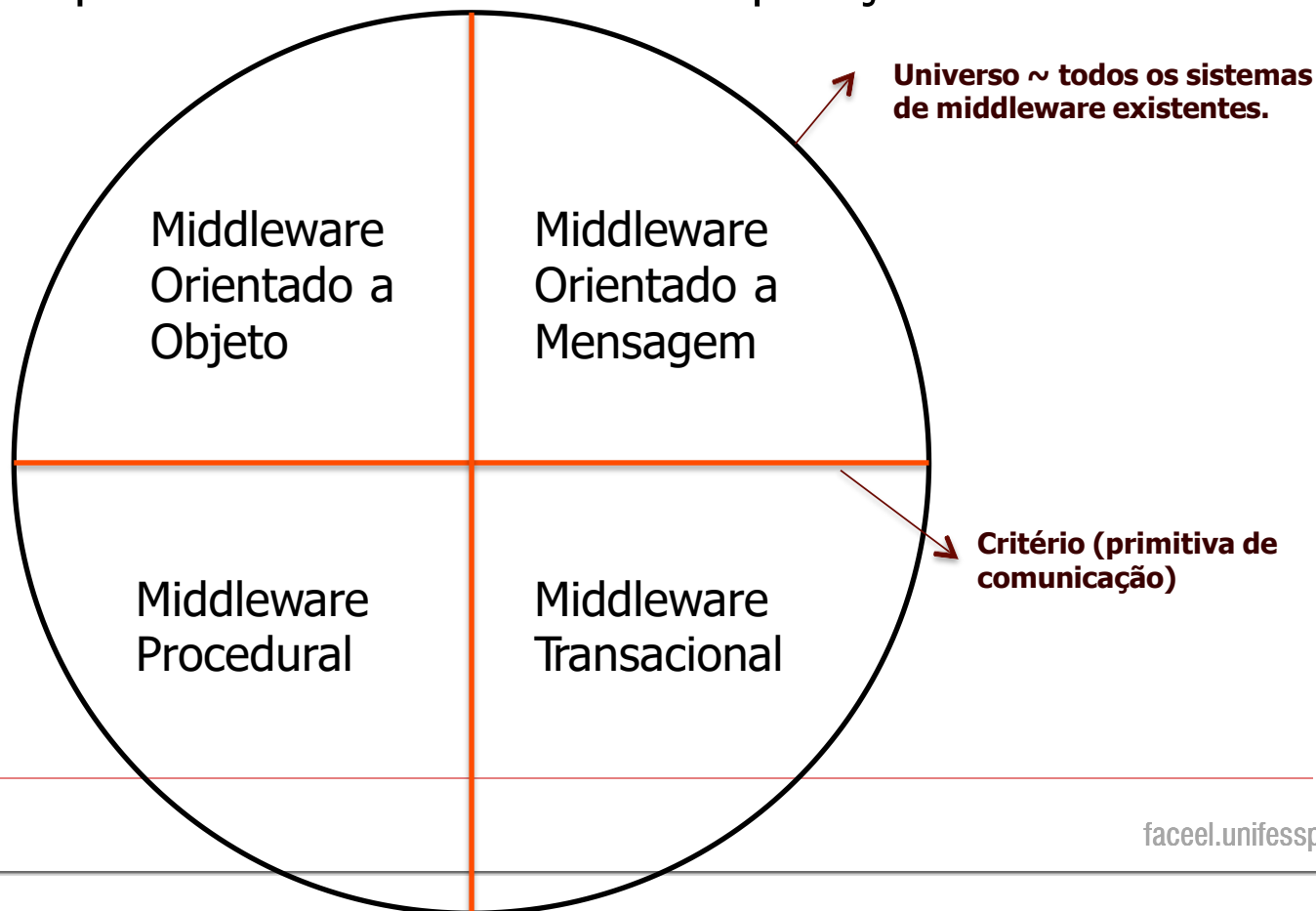
---

- **Discreta:** comunicam por mensagens, onde cada uma delas é uma unidade de informação completa.
- **Fluxo:** comunicam por várias mensagens que estão relacionadas uma com as outras.



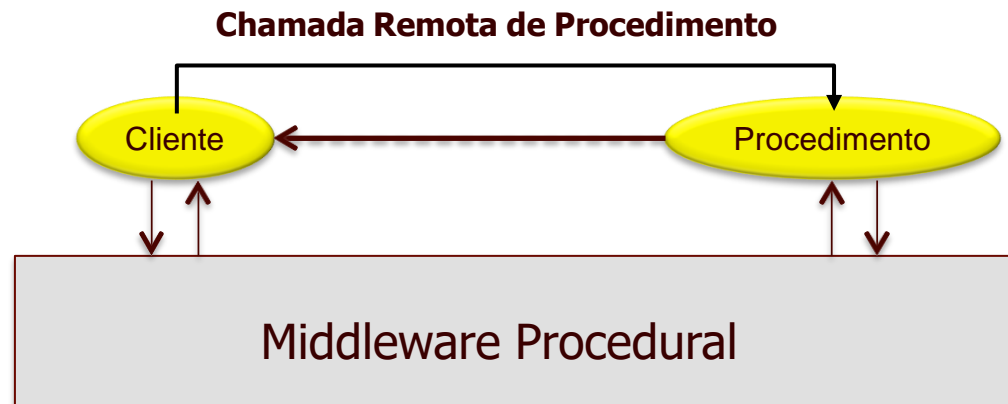
# Middleware - Classificação

- ❑ Critério: **tipo de primitiva de comunicação fornecida** pelo middleware para o desenvolvimento de aplicações distribuídas.

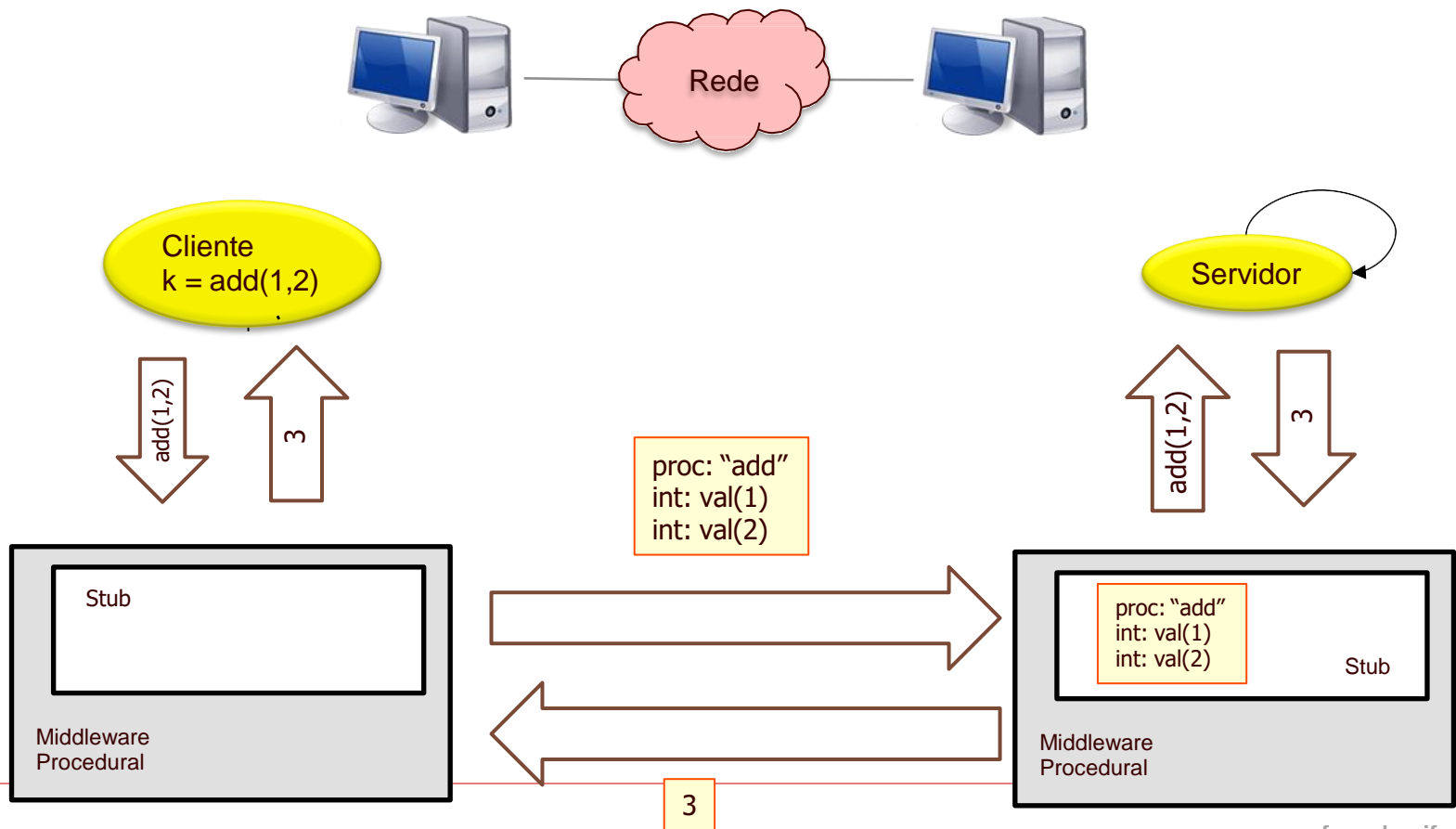


# Middleware Procedural

---



# Middleware Procedural



# Middleware Procedural

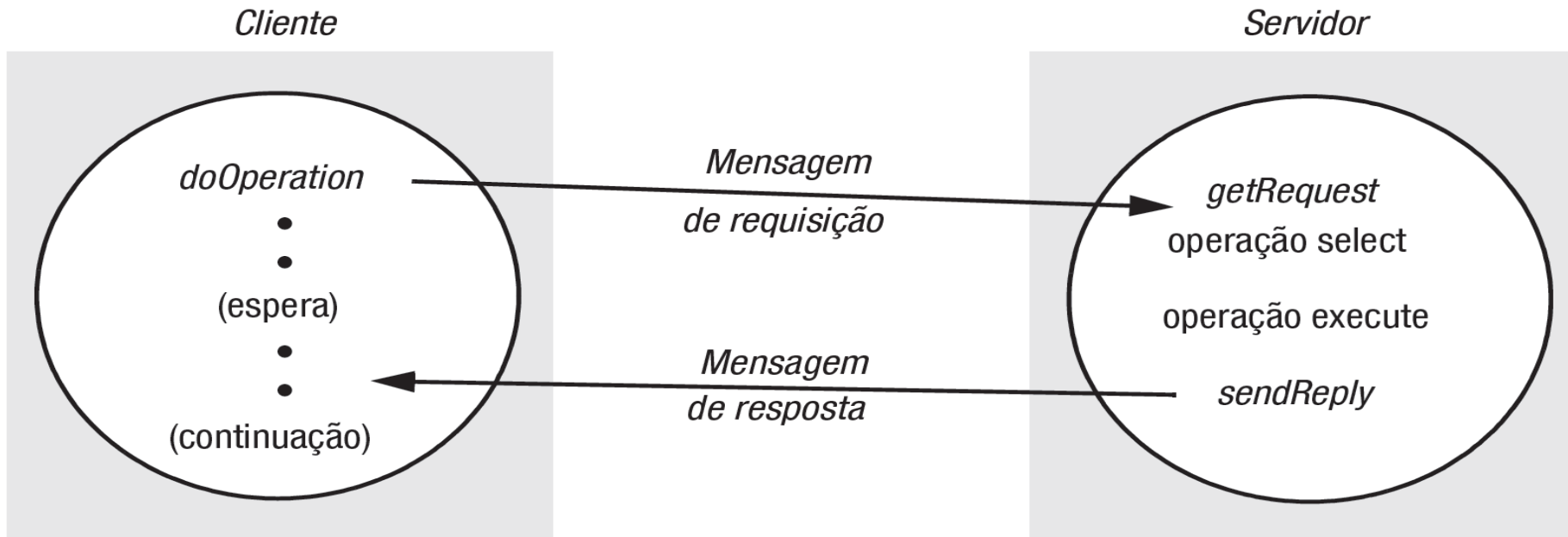
---

- ❑ Primitiva de interação: chamada remota de procedimento
- ❑ Comunicação 1-1
- ❑ Comunicação síncrona suportada naturalmente
- ❑ Protocolo *request/wait-for-reply*
  - e.g., RPC

# Protocolo de requisição-resposta

---

- ❑ **Caso normal:** requisição-resposta síncrona.
- ❑ **Confiável:** resposta do servidor é uma confirmação para o cliente.



# Estrutura da mensagem requisição-resposta

---

messageType

*int (0=Request, 1=Reply)*

requestId

*int*

remoteReference

*RemoteRef*

OperationId

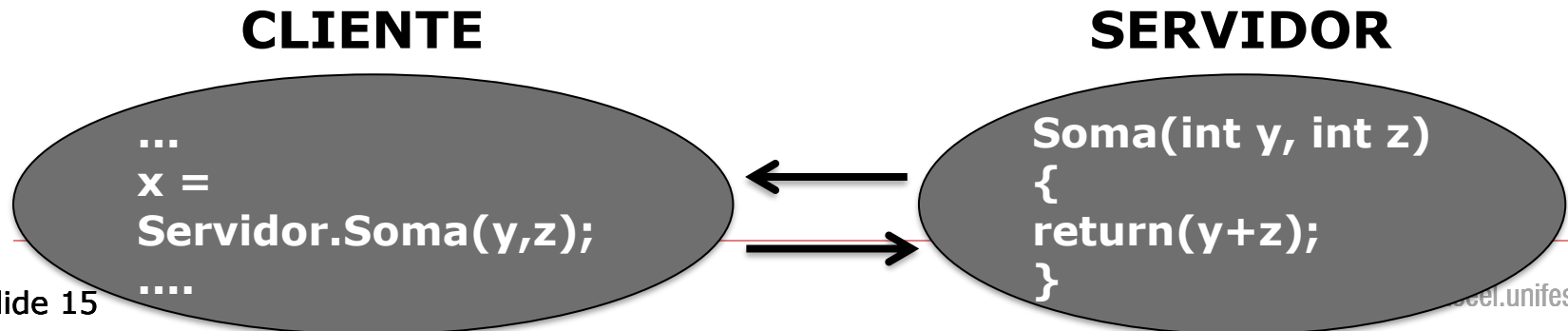
*int ou operação*

arguments

*// vetor de bytes*

# RPC (Chamada de Procedimento Remoto)

- ❑ Segue o **modelo Cliente/Servidor**
- ❑ Programação com **interfaces**
- ❑ Objeto servidor possui interface com procedimentos que podem ser chamados remotamente
- ❑ Objetos clientes usam serviços de servidores
- ❑ Não suporta referências de objeto remoto



# RPC e os Stubs

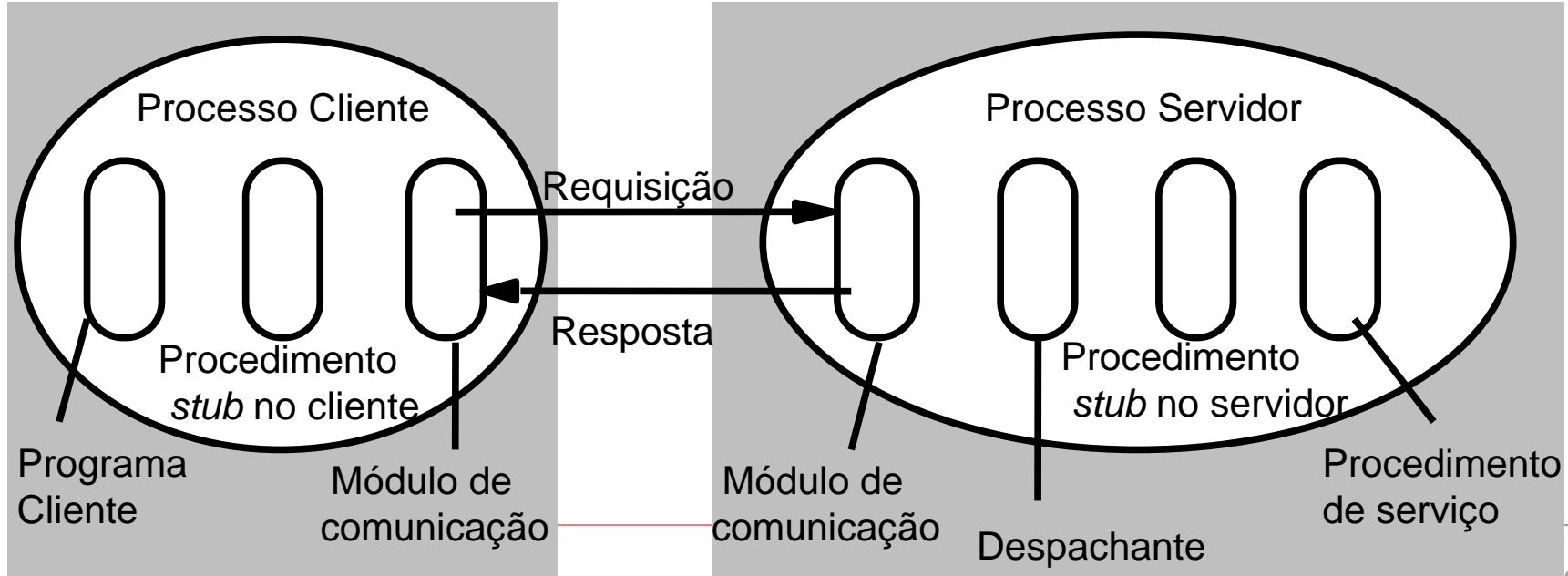
---

- A **transparência** é alcançada através de stubs:
  - **Stub do cliente:** Empacota os parâmetros em uma mensagem e a envia para a máquina do servidor.
  - **Stub do servidor:** Desempacota os parâmetros e invoca o procedimento correto passando os parâmetros.

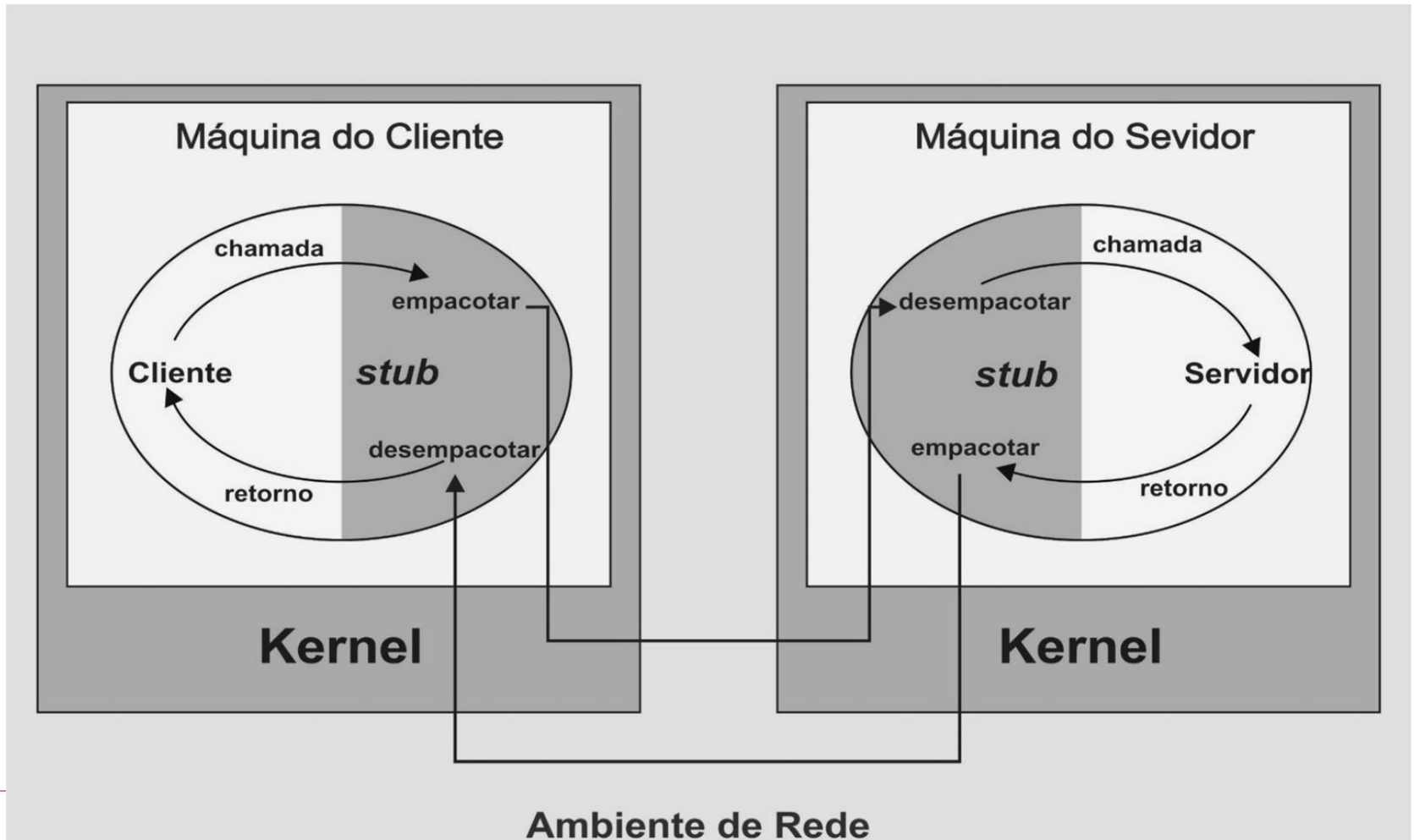


# RPC (Chamada de Procedimento Remoto)

- Uma **chamada de procedimento remoto** é muito parecida com uma invocação a método remoto, pois um processo cliente chama um procedimento que está sendo executado em um processo servidor.

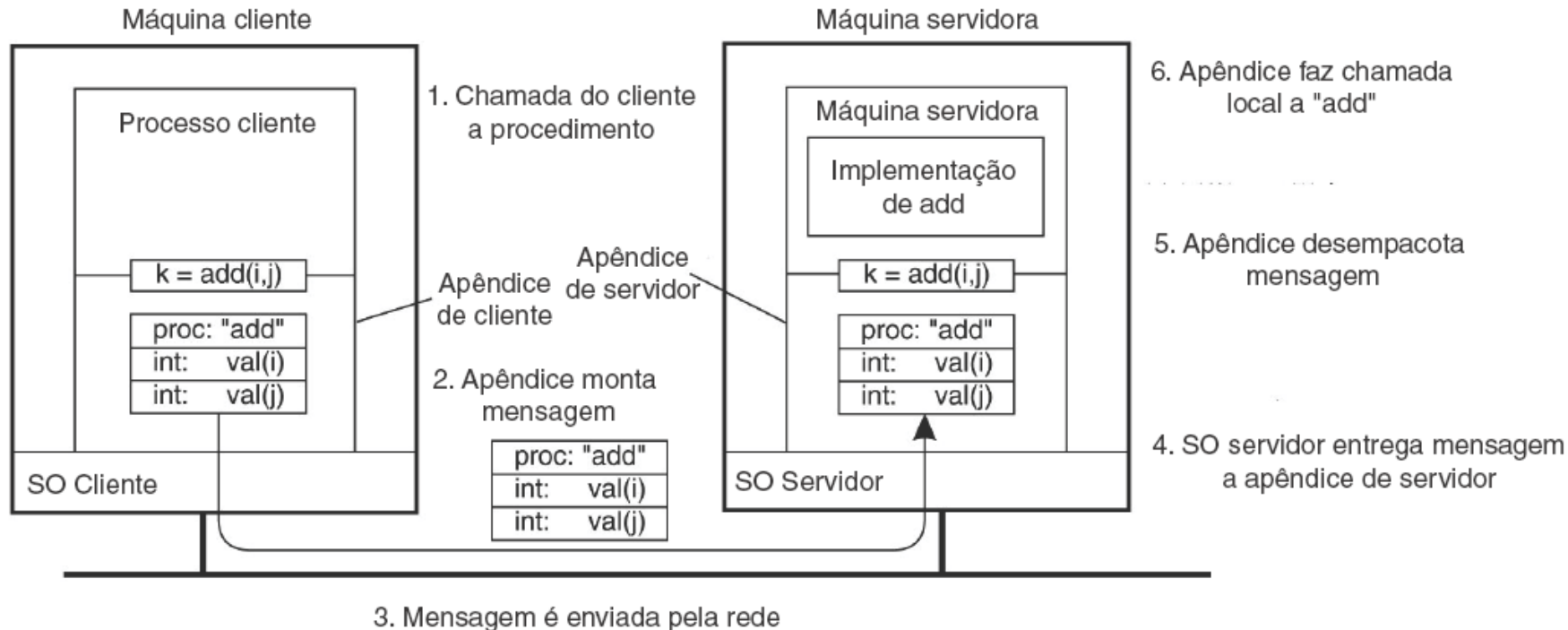


# RPC (Chamada de Procedimento Remoto)



# Passos para uma RPC

- ❑ Empacotar parâmetros em uma mensagem é conhecido como *marshalling de parâmetro*.
- ❑ RPC suporta:
  - Passagem por valor e passagem por referência.



# RPC – Linguagem de Programação de Interface (IDL)

---

- ❑ **Interface:** É um conjunto de procedimentos que pode ser chamado por um cliente e implementado por servidor.
- ❑ IDL é a linguagem voltada para especificar a interface e permite definir procedimentos como **idempotentes**.

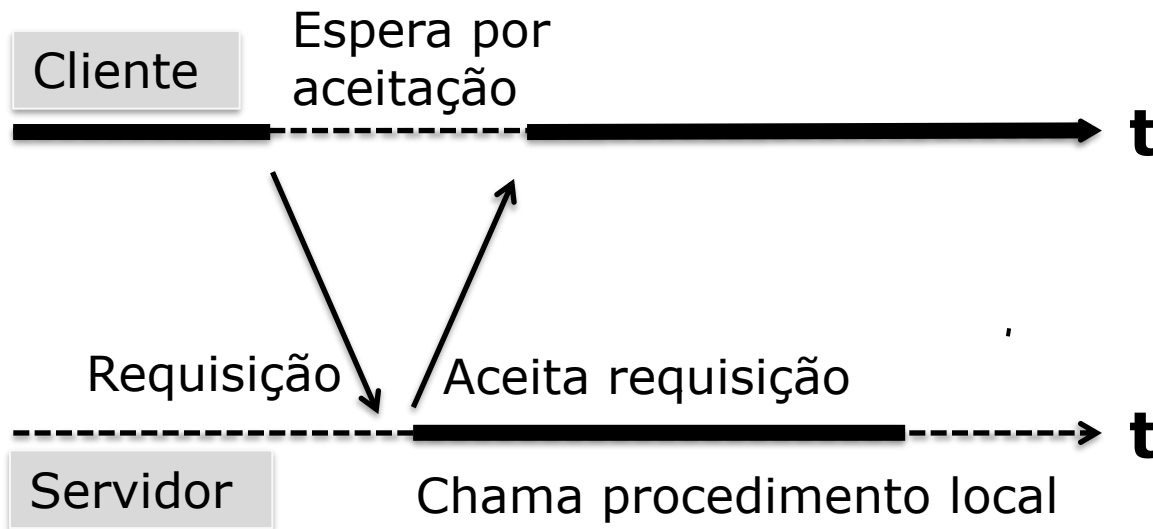
# RPC (Chamada de Procedimento Remoto)

- ❑ **Chamada síncrona:** cliente fica bloqueado aguardando o término da execução do método.



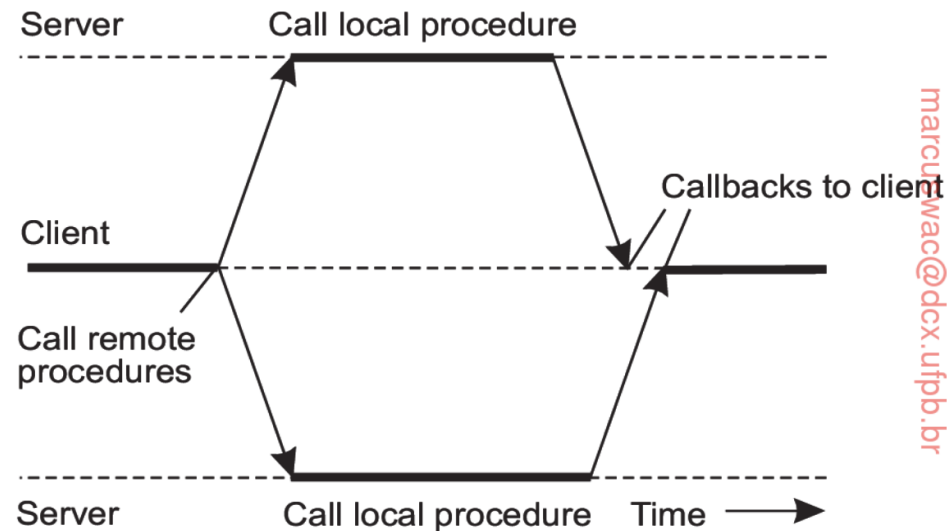
# RPC (Chamada de Procedimento Remoto)

- **Chamadas assíncronas:** cliente continua a execução sem aguardar o retorno do método.



# RPC (Chamada de Procedimento Remoto)

- **RPC multicast:**  
múltiplos RPCs executados simultaneamente
  - Tolerância a falhas: espera primeiro ou maioria responder
  - Computação distribuída: cliente espera todas as respostas para juntar




# Apache XML-RPC

---

- ❑ Um protocolo popular que usa XML sobre o HTTP para implementar chamada de procedimento remoto.
- ❑ Antes, era conhecido como Helma XML-RPC.



# XML-RPC Client



```
public class JavaClient {  
  
    public static void main(String[] args) {  
        String serverURL = "http://localhost:1093";  
        try {  
            XmlRpcClientConfigImpl cliConfig = new XmlRpcClientConfigImpl();  
            cliConfig.setServerURL(new URL(serverURL));  
            XmlRpcClient client = new XmlRpcClient();  
            client.setConfig(cliConfig);  
            Vector params = new Vector();  
            params.addElement(new Integer(17));  
            params.addElement(new Integer(13));  
            Object result = client.execute("Calculator.sum", params);  
            int sum = ((Integer) result).intValue();  
            System.out.println("The sum is: " + sum);  
        } catch (Exception exception) {  
            System.err.println("JavaClient: " + exception);  
        }  
    }  
}
```

# XML-RPC Server

```
public class JavaServer {  
  
    public Integer sum(int x, int y) {  
        return new Integer(x + y);  
    }  
  
    public static void main(String[] args) {  
        try {  
            System.out.println("Attempting to start XML-RPC Server...");  
            WebServer webServer = new WebServer(1093);  
            XmlRpcServer xmlRpcServer = webServer.getXmlRpcServer();  
            PropertyHandlerMapping pm = new PropertyHandlerMapping();  
            pm.addHandler("Calculator", (new JavaServer()).getClass()); //  
  
            webServer.start ();  
  
            System.out.println ("Started successfully.");  
            System.out.println ("Accepting requests. (Halt program to stop.)");  
        } catch (Exception exception) {  
            System.err.println("JavaServer: " + exception);  
        }  
    }  
}
```