

O que é um Middleware?

Nelson Rosa – nsr@cin.ufpe.br

Objetivos da Aula

- **Definir o que é um middleware**
- **Apresentar as principais características de um middleware**
- **Mostrar o passo-a-passo de como construir um sistema distribuído usando um Middleware (RMI)**
- **Socket versus Middleware**

Percepção das Pessoas sobre o que é um Middleware

- API que facilita a construção de sistemas distribuídos
 - Camada de software
 - Sistema (?)
 - Camada de rede
 - Coleção de serviços distribuídos
 - Sistema operacional estendido
 - “Socket turbinado”
 - “Qualquer coisa no meio de ...”
-

O Que é um Middleware?

■ Definição 1

- Um **conjunto de serviços** que ajuda a resolver problemas de heterogeneidade e distribuição" (Bernstein 1996)

■ Definição 2

- "Software projetado para ajudar a gerenciar a **complexidade** e a **heterogeneidade** inerente aos sistemas distribuídos" (Bakken 2001)

■ Definição 3

- "Software localizado entre a aplicação e o SO responsável por resolver a lacuna entre as aplicações e a infraestrutura de software/hardware" (Schmidt 2003)

- **Facilita** o desenvolvimento de aplicações distribuídas
- **Coordena** como as partes da aplicação distribuída interoperam
- Permite e simplifica a **integração** de componentes desenvolvidos em múltiplas tecnologias

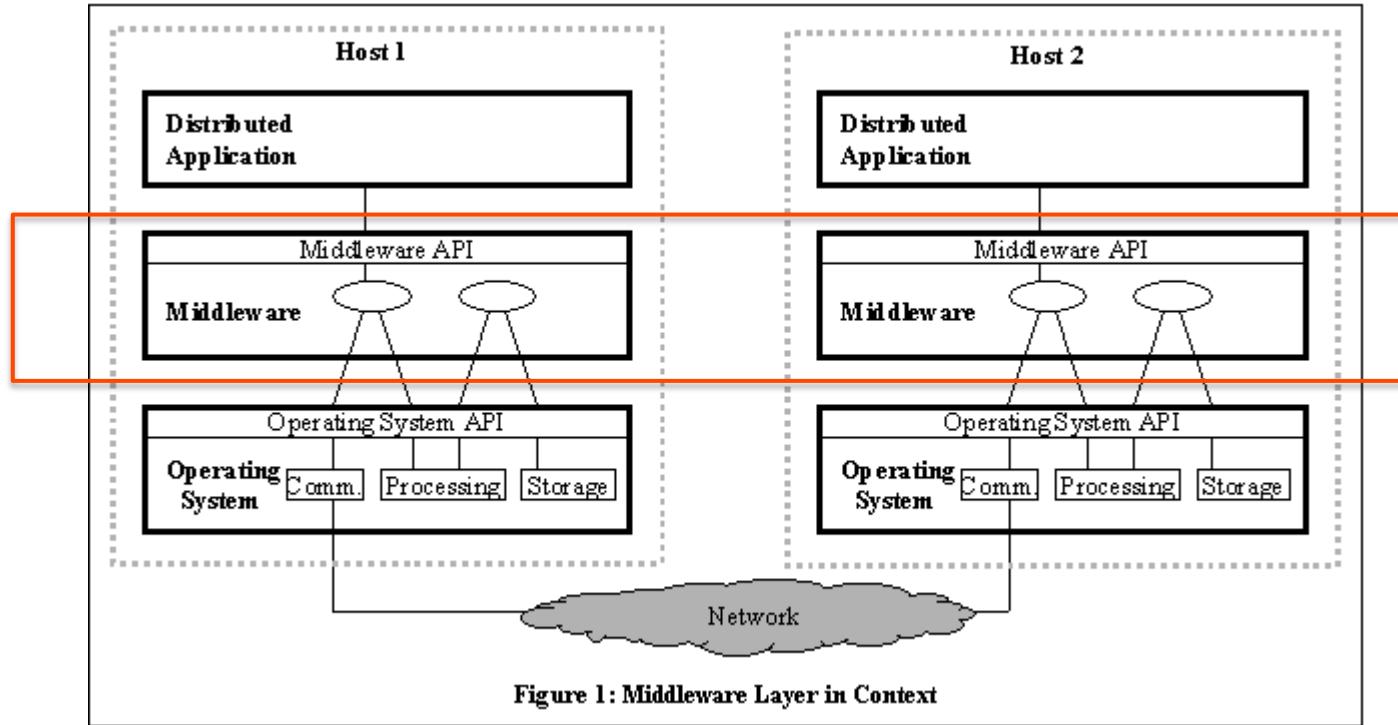
■ Definição 4

- "Software sold to people who don't know how to program by people who know how to program." [Philip Greenspun]

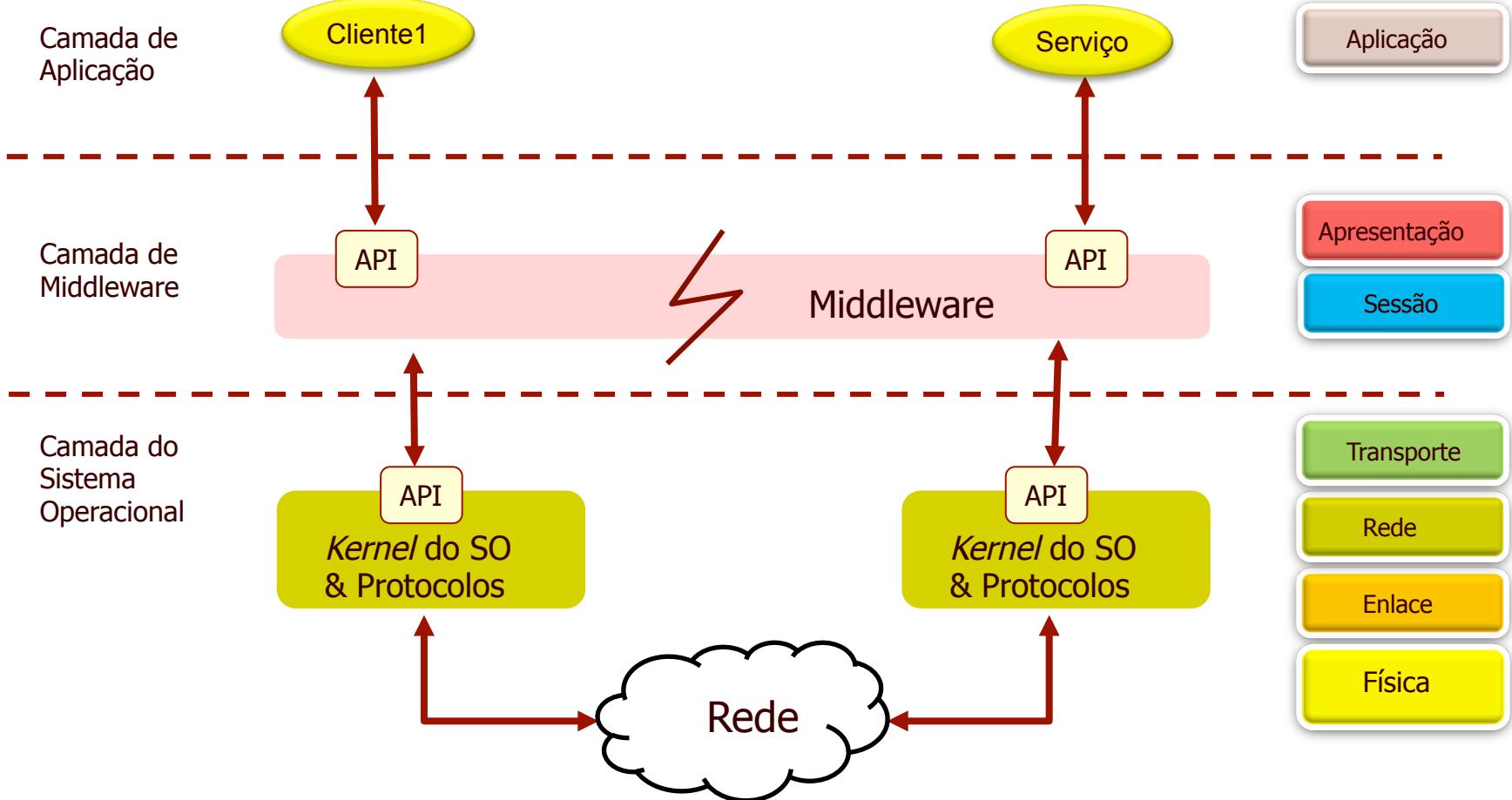
Fatos básicos

- Camada de software localizada entre o **sistema operacional** e a **aplicação**
 - Utiliza-se dos mecanismos de comunicação de “baixo nível” (SO) providos pelo sistema operacional para fornecer uma comunicação de “alto nível” para as aplicações distribuídas
- Deve implementar o máximo de **transparências** e serviços possíveis
- Oferece **serviços** que suportam o desenvolvimento e a execução de aplicações distribuídas
- Toda a **complexidade** da distribuição é “absolvida” pelo middleware

Fatos básicos:: Camada de Software



Fatos básicos:: Camada de Rede



Fatos básicos:: Transparências

O que é Transparência?

Esconder do usuário e dos desenvolvedores de SD “aspectos” de distribuição (Coulouris)

Transparências “Comuns”:

- ✓ Localização
- ✓ Acesso
- ✓ Falha
- ✓ Tecnologia
- ✓ Concorrência

Transparências (Coulouris):

- ✓ Localização
- ✓ Acesso
- ✓ Falha
- ✓ Tecnologia
- ✓ Concorrência
- ✓ Mobilidade
- ✓ Desempenho
- ✓ Escala

Transparências (RM-ODP):

- ✓ Acesso
- ✓ Persistência
- ✓ Localização
- ✓ Relocação
- ✓ Migração
- ✓ Falha
- ✓ Transação
- ✓ Replicação

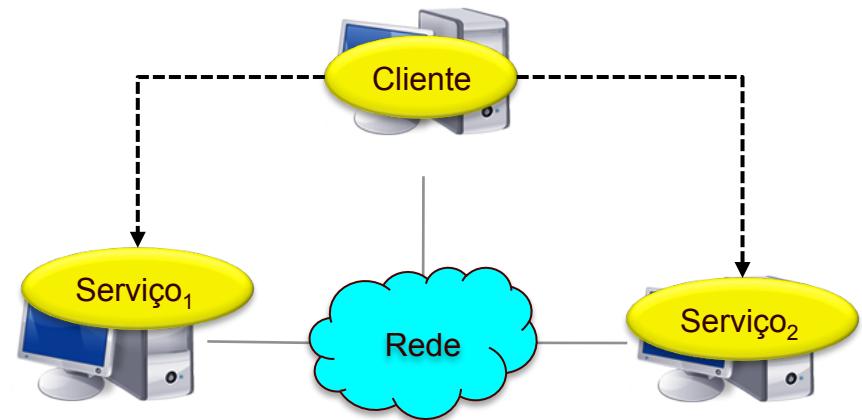
Fatos básicos:: Transparências:: Localização

- ▶ Componentes podem ser acessados sem que se precise saber suas localizações físicas.

- ▶ **Cliente** conhece o nome do serviço (**Serviço₁** ou **Serviço₂**) que ele que usar.

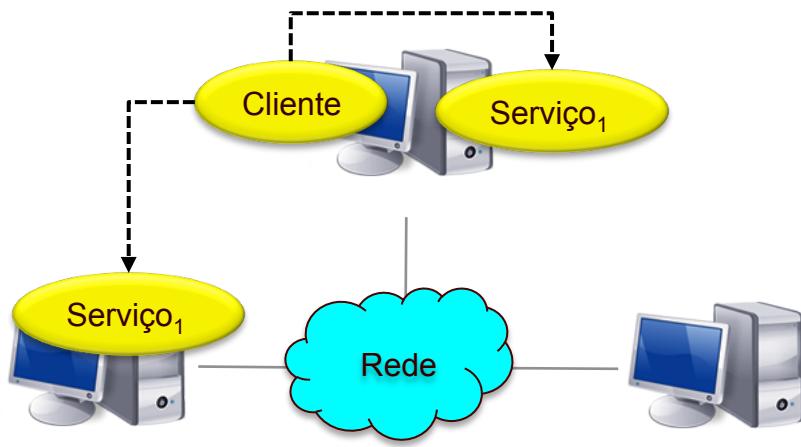
- ▶ **Cliente** não conhece a **localização física** de **Serviço₁** ou **Serviço₂**.

- ▶ **Cliente** conhece a localização de um serviço (serviço de nomes) que é capaz de informar onde o **Serviço₁** está localizado.



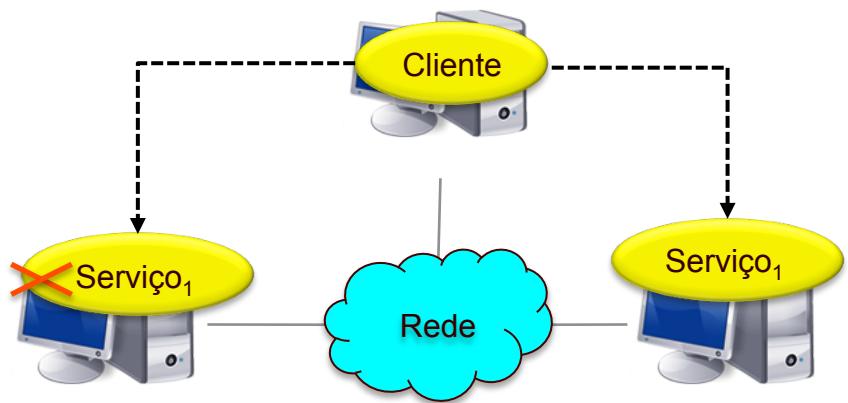
Fatos básicos:: Transparências:: Acesso

- Acesso **local** e **remoto** é programado de forma **similar**.
- A programação do **Cliente** não muda caso o **Serviço₁** esteja executando localmente ou remotamente.



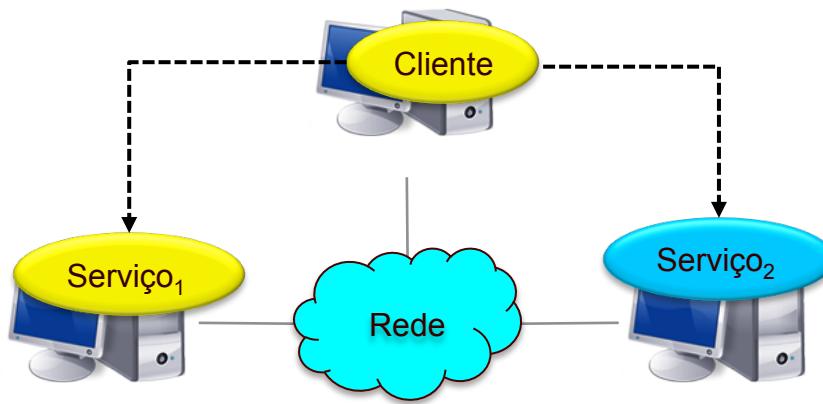
Fatos básicos:: Transparências:: Falha

- ▶ Falhas são escondidas dos componentes.
- ▶ **Cliente** não “fica sabendo” da falha do **Serviço₁**.



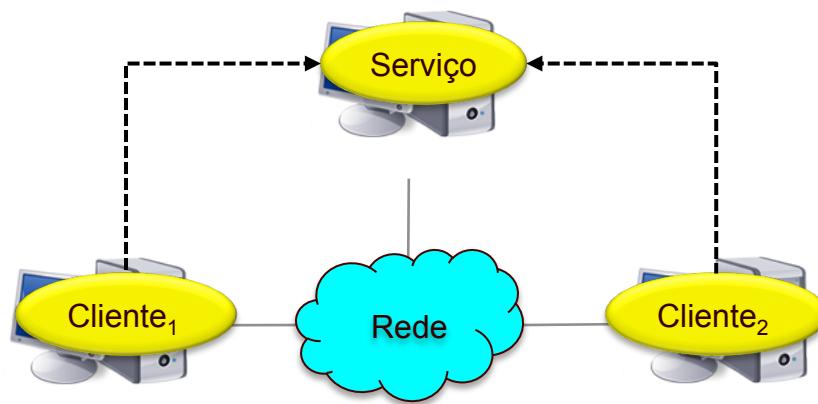
Fatos básicos:: Transparências:: Tecnologia

- ▶ A heterogeneidade tecnológica é transparente ao usuário, e.g., heterogeneidades de hardware, rede, sistema operacional, e linguagem de programação.
- ▶ **Cliente** (Java) invoca **Serviço₁** (C++) ou **Serviço₂** (Java) sem que isto afete os resultados das operações providas por estes serviços.



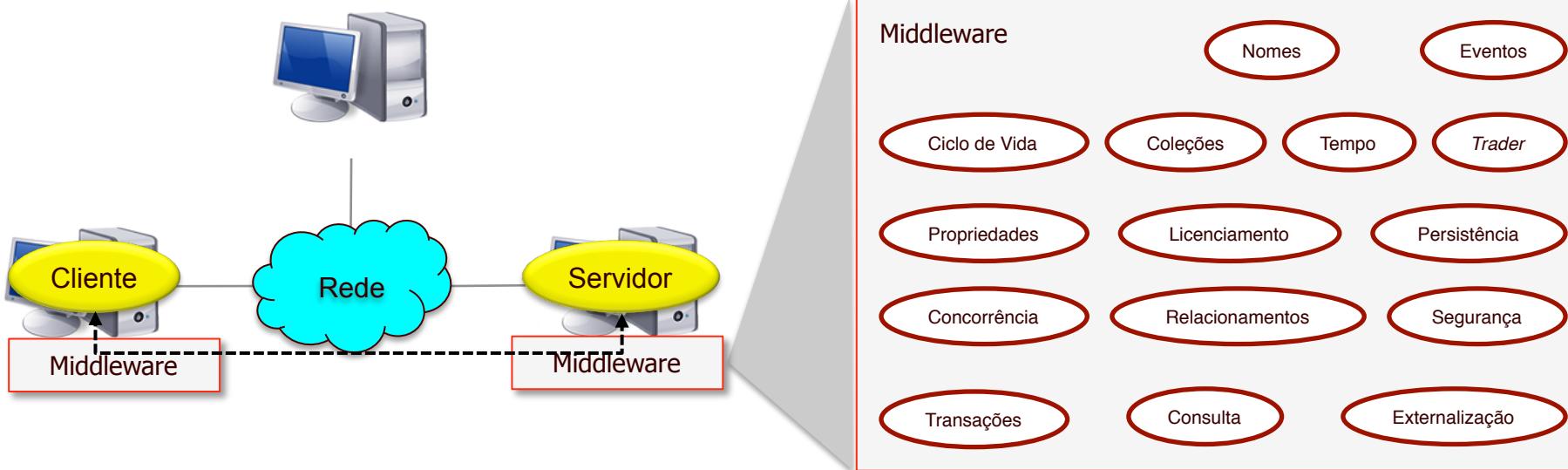
Fatos básicos:: Transparências:: Concorrência

- ▶ Componentes são compartilhados sem que isto afete os resultados produzidos por eles.
- ▶ **Serviço** é compartilhado por **Cliente₁** e **Cliente₂**.



Fatos básicos:: Serviços

- **Middleware visto como uma coleção de serviços fornecidos através APIs**



Fatos básicos:: Serviços

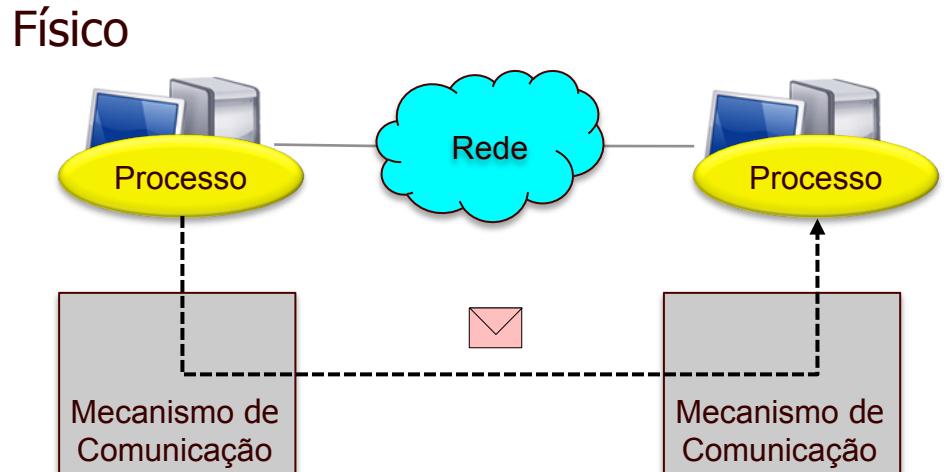
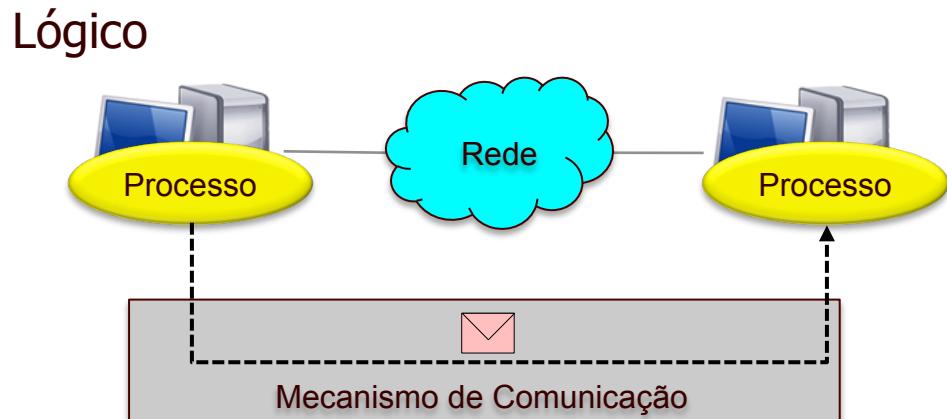
- **Provido pelo middleware**
- **Definido por APIs**
- **Pode suportar vários protocolos**
- **Deve ser genérico para várias plataformas (portáveis)**
- **Distribuído**
- **Deve ser “de facto” integrado com outros serviços**



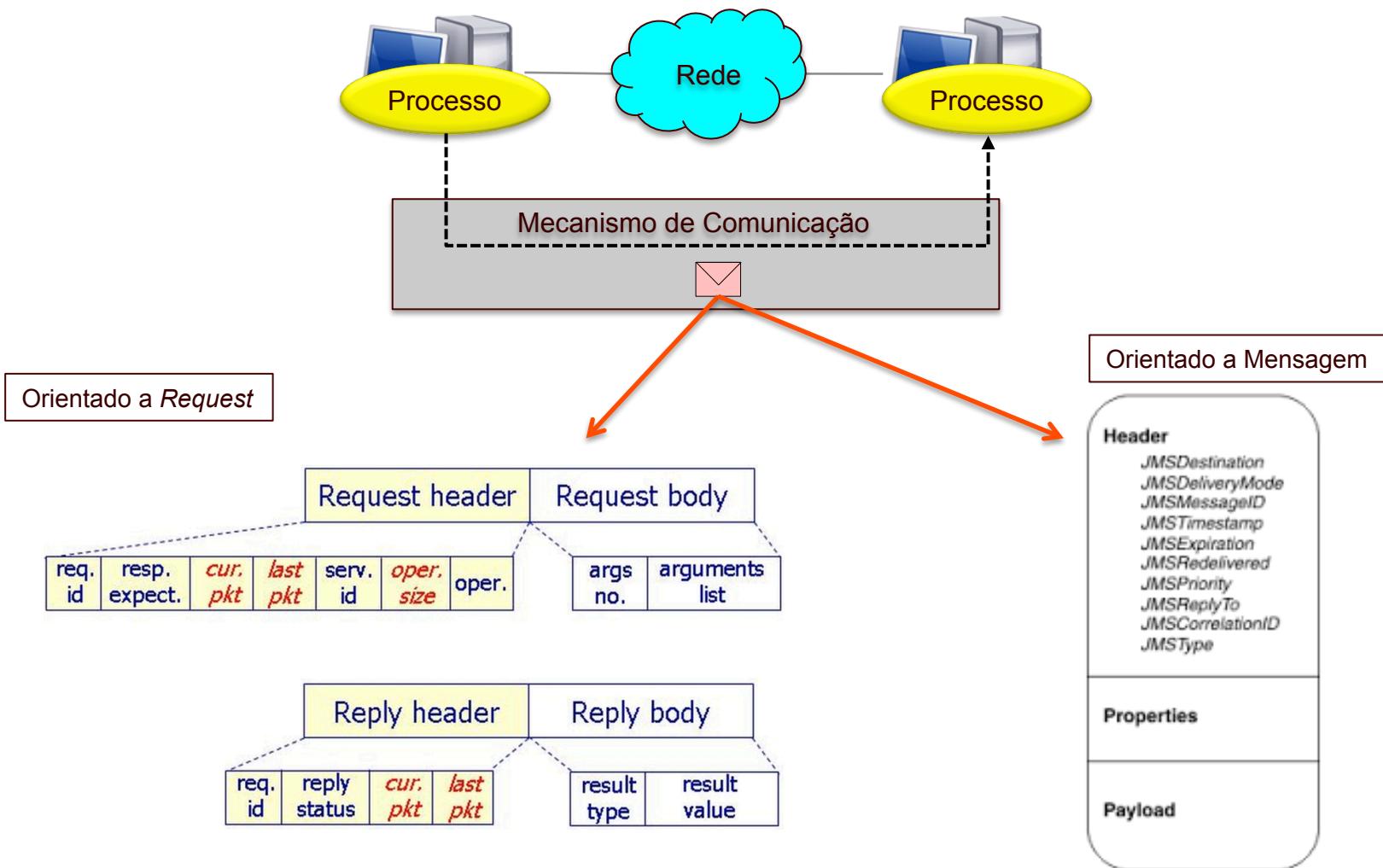
Fatos básicos:: Mecanismos de Comunicação

- **Processos se comunicam através de troca de mensagens**
 - Mensagem = **Sequência de bytes**

- **Mensagens são transportadas por algum mecanismo de comunicação**
 - e.g., TCP, UDP, Bluetooth (?)

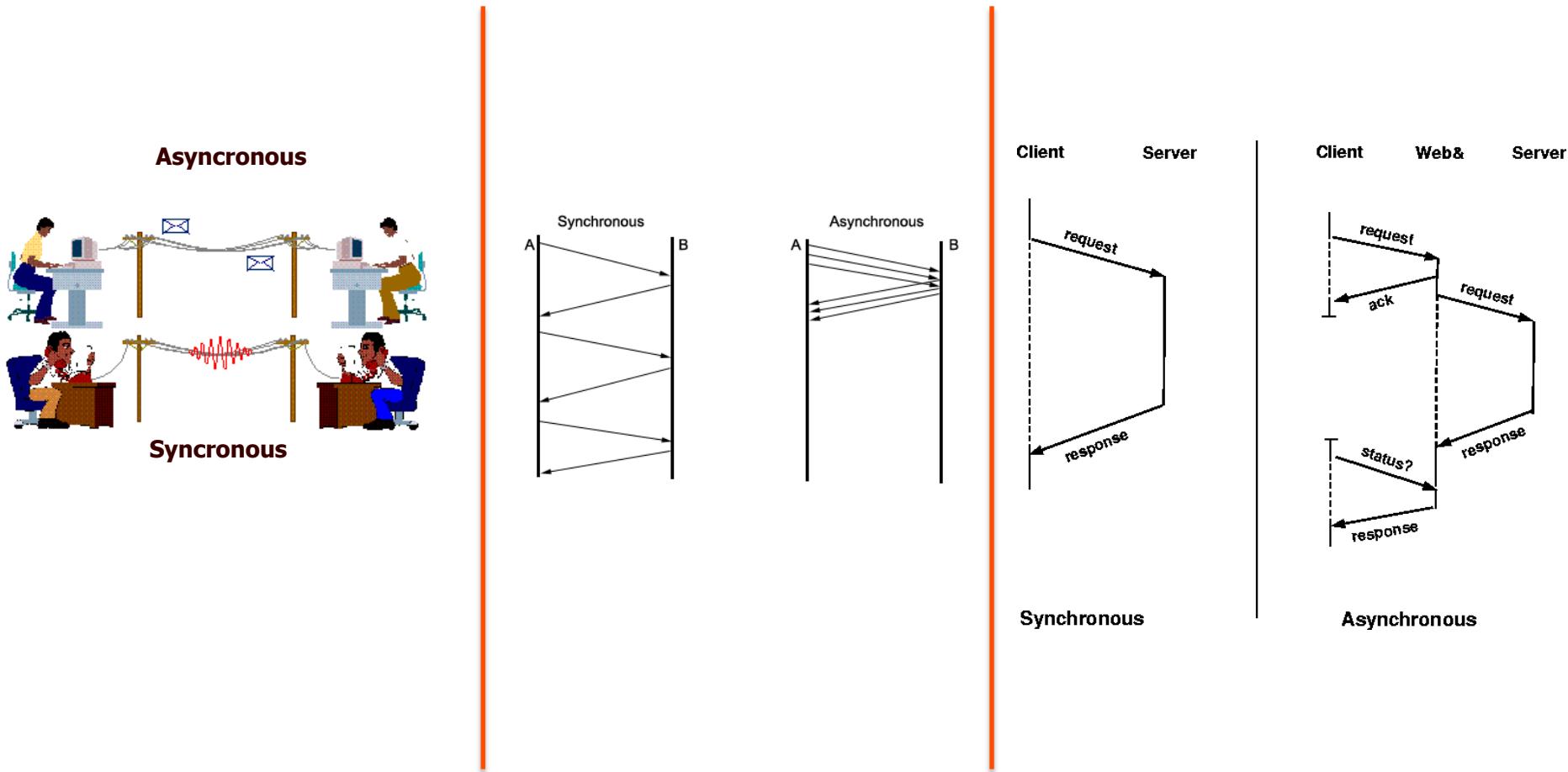


Fatos básicos:: Mecanismos de Comunicação



Fatos básicos:: Sincronicidade

Sincronicidade

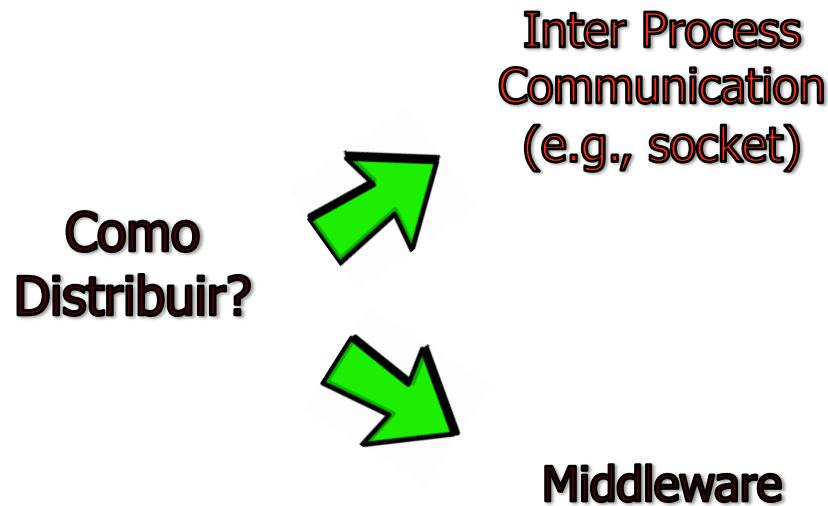


Fatos básicos:: Sincronicidade

■ Sincronicidade

- **Síncrona:** transmissor e receptor sincronizam em toda mensagem.
 - e.g., RPC, RMI
- **Assíncrona:** transmissor ativo depois que a mensagem é enviada
 - Buferização de mensagens
 - e.g., implementações JMS (padrão Java)

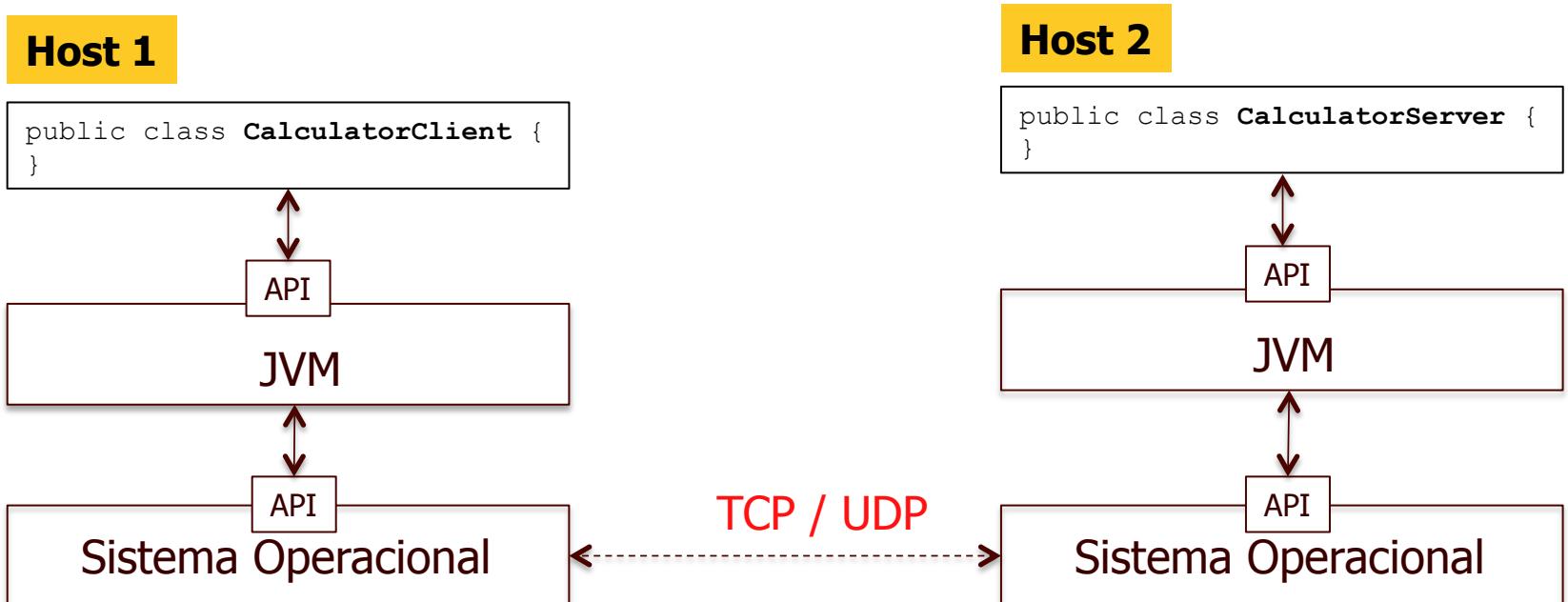
Relembrando:: como implementar SDs?



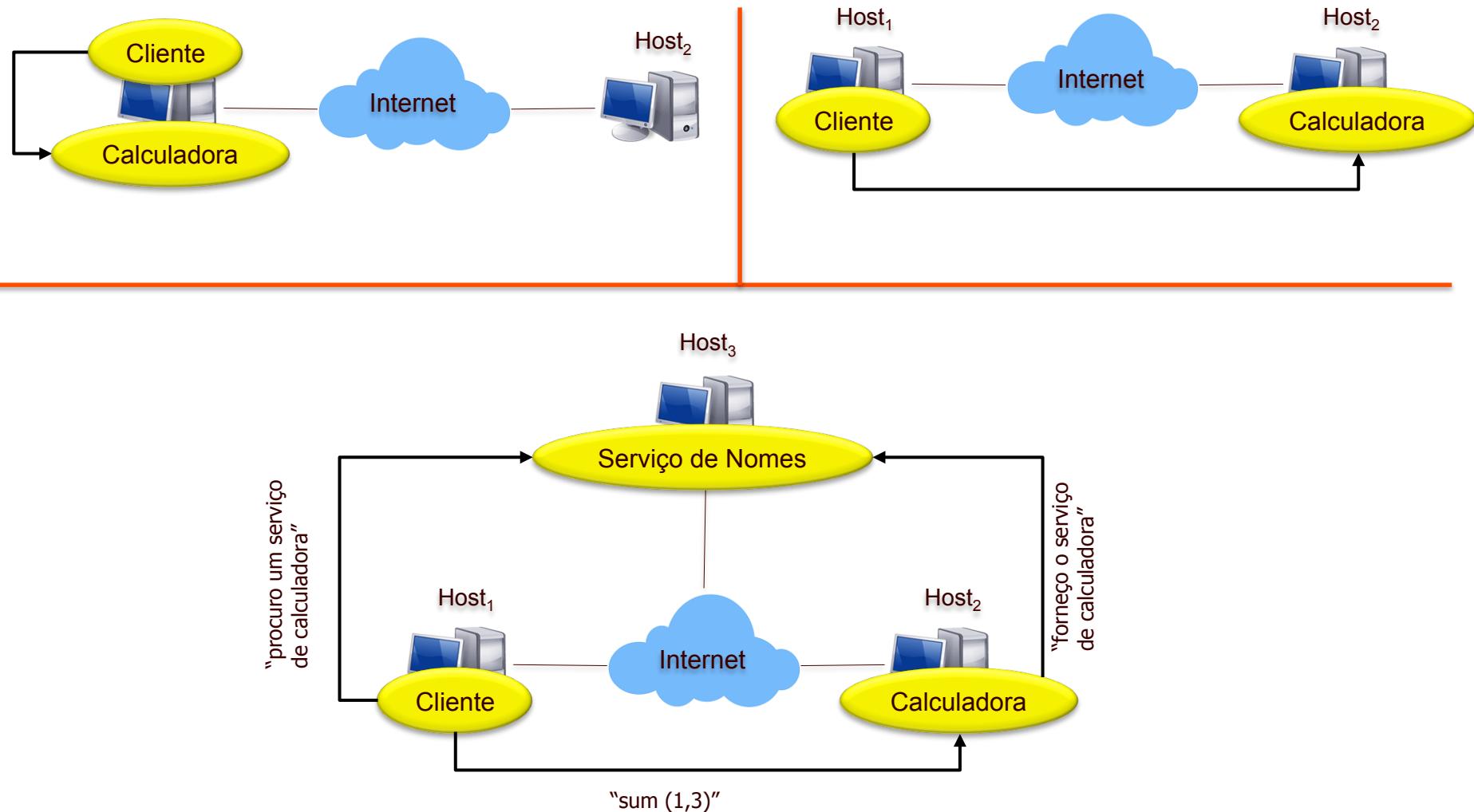
	Interface (*)	Cliente (*)	Servidor (*)
Centralizada	6	6	14
Distribuída (TCP)	6	20	57
Distribuída (UDP)	6	19	55

(*) Linhas de Código excluindo as linhas em branco, comentários, imports e similares.

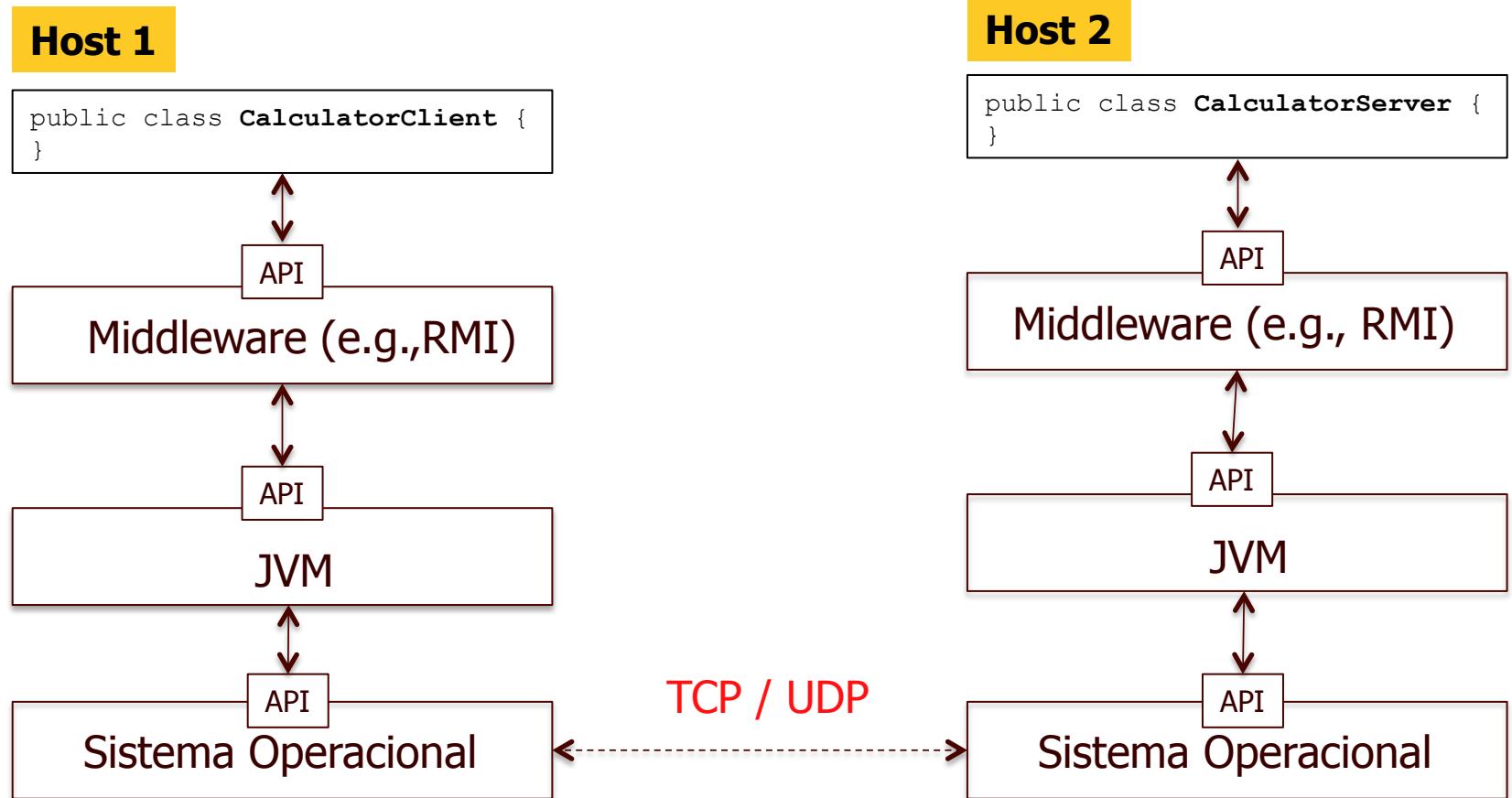
Relembrando:: Camadas



Relembrando:: Distribuição



A partir de agora:: SD implementados em RMI



Calculadora em RMI:: Interface

```
6 public interface ICalculator extends Remote {  
7     float add (float x, float y) throws RemoteException;  
8     float sub (float x, float y) throws RemoteException;  
9     float mul (float x, float y) throws RemoteException;  
10    float div (float x, float y) throws RemoteException;  
11 }
```

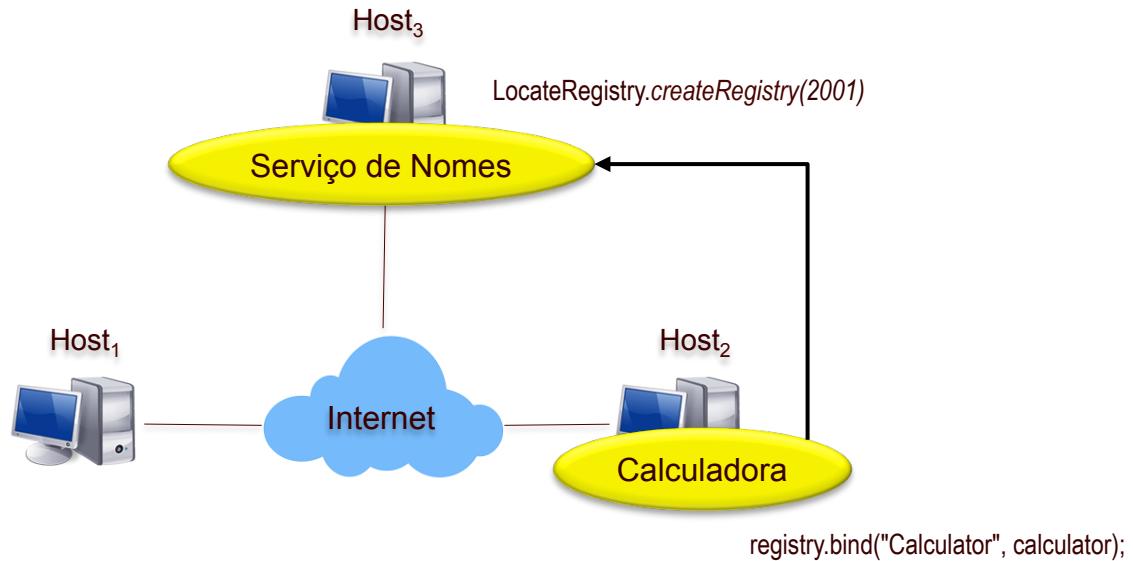
```
6 public class CalculatorImpl extends UnicastRemoteObject implements ICalculator {  
7  
8     protected CalculatorImpl() throws RemoteException {  
9         super();  
10        // TODO Auto-generated constructor stub  
11    }  
12  
13    private static final long serialVersionUID = 1L;  
14  
15    public float add(float x, float y) throws RemoteException {  
16        return x + y;  
17    }  
18  
19    public float sub(float x, float y) throws RemoteException {  
20        return x - y;  
21    }  
22  
23    public float mul(float x, float y) throws RemoteException {  
24        return x * y;  
25    }  
26  
27    public float div(float x, float y) throws RemoteException {  
28        return x / y;  
29    }  
30  
31 }
```

Calculadora em RMI:: Servidor

```
3+import java.rmi.AlreadyBoundException;[]
8
9 public class CalculatorServer {
10
11-    protected CalculatorServer() throws RemoteException {
12        super();
13    }
14
15-    public static void main(String args[]) throws RemoteException,
16        AlreadyBoundException {
17
18        // create an instance of Calculator
19        CalculatorImpl calculator = new CalculatorImpl(); } }
20
21        // create a Registry instance on the local host
22        Registry registry = LocateRegistry.getRegistry("localhost",1313); } }
23
24        // register the instance of Calculator in the Naming Service
25        registry.bind("Calculator", calculator); } }
26    }
27 }
```

Registry ≈ Serviço de nomes

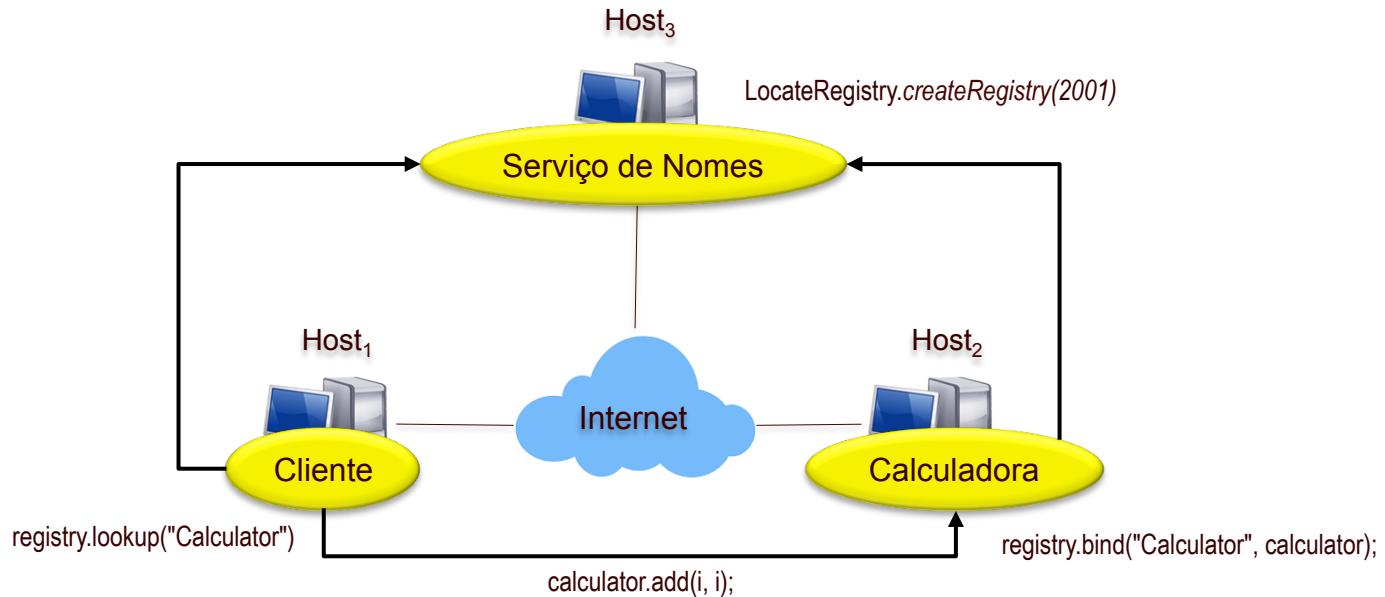
Calculadora em RMI:: Servidor



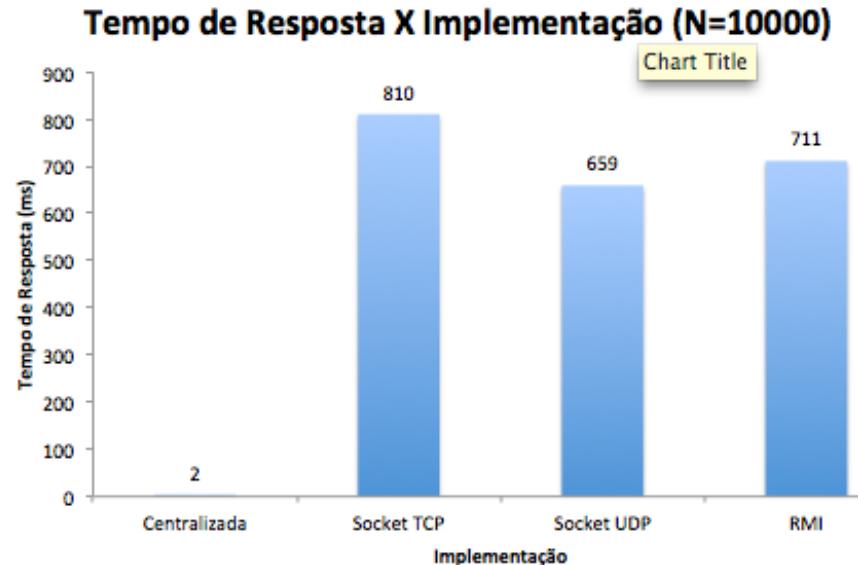
Calculadora em RMI:: Cliente

```
3+import java.rmi.NotBoundException;□
7
8 public class CalculatorClient {
9
10 ⊕ public static void main(String[] args) throws RemoteException,
11     NotBoundException {
12
13     // obtain a reference to a bootstrap remote object registry
14     Registry registry = LocateRegistry.getRegistry("localhost", 2001); } }
15
16     // look for an instance of Calculator in the Naming service
17     ICalculator calculator = (ICalculator) registry.lookup("Calculator"); } }
18
19     // invoke the remote operation
20     float result = calculator.add(1, 3); } }
21
22     System.out.println("add (1, 3) = " + result);
23 }
24
25 }
```

Calculadora em RMI:: Cliente



Socket versus RMI



	Interface (*)	Cliente (*)	Servidor (*)
Centralizada	6	6	14
Distribuída (TCP)	6	20	57
Distribuída (UDP)	6	19	55
RMI	6	9	30

(*) Linhas de Código excluindo as linhas em branco, comentários, imports e similares.

“X da Questão”

A complexidade da distribuição deve ser transparente (ou invisível) ao programador da aplicação distribuída

Sonho dos programadores de aplicações distribuídas!

Pesadelo de quem constrói middleware !

Códigos Java

■ Calculadora em RMI

- [https://www.dropbox.com/sh/1nxieiyyddgg04f4/
AADnYztKbjGgL7E974-HaTpVa?dl=0](https://www.dropbox.com/sh/1nxieiyyddgg04f4/AADnYztKbjGgL7E974-HaTpVa?dl=0)

Fim da Aula