



**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

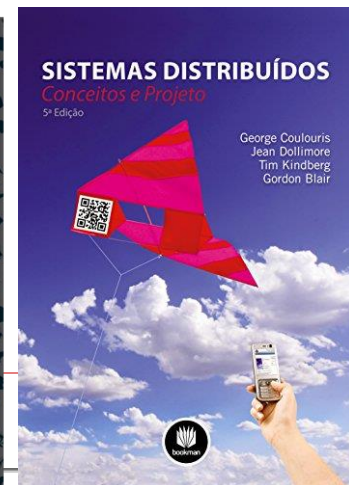
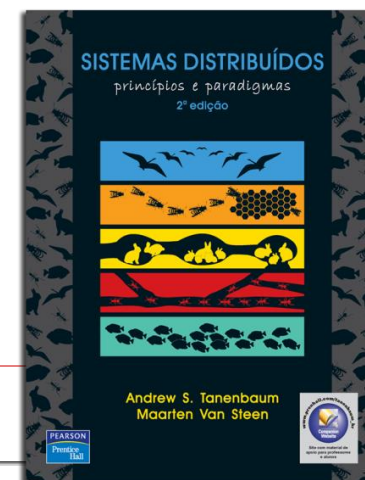
---

- ❑ AULA 8:
- ❑ Middlewares
  - RMI
  - Java RMI

# Leitura Prévia

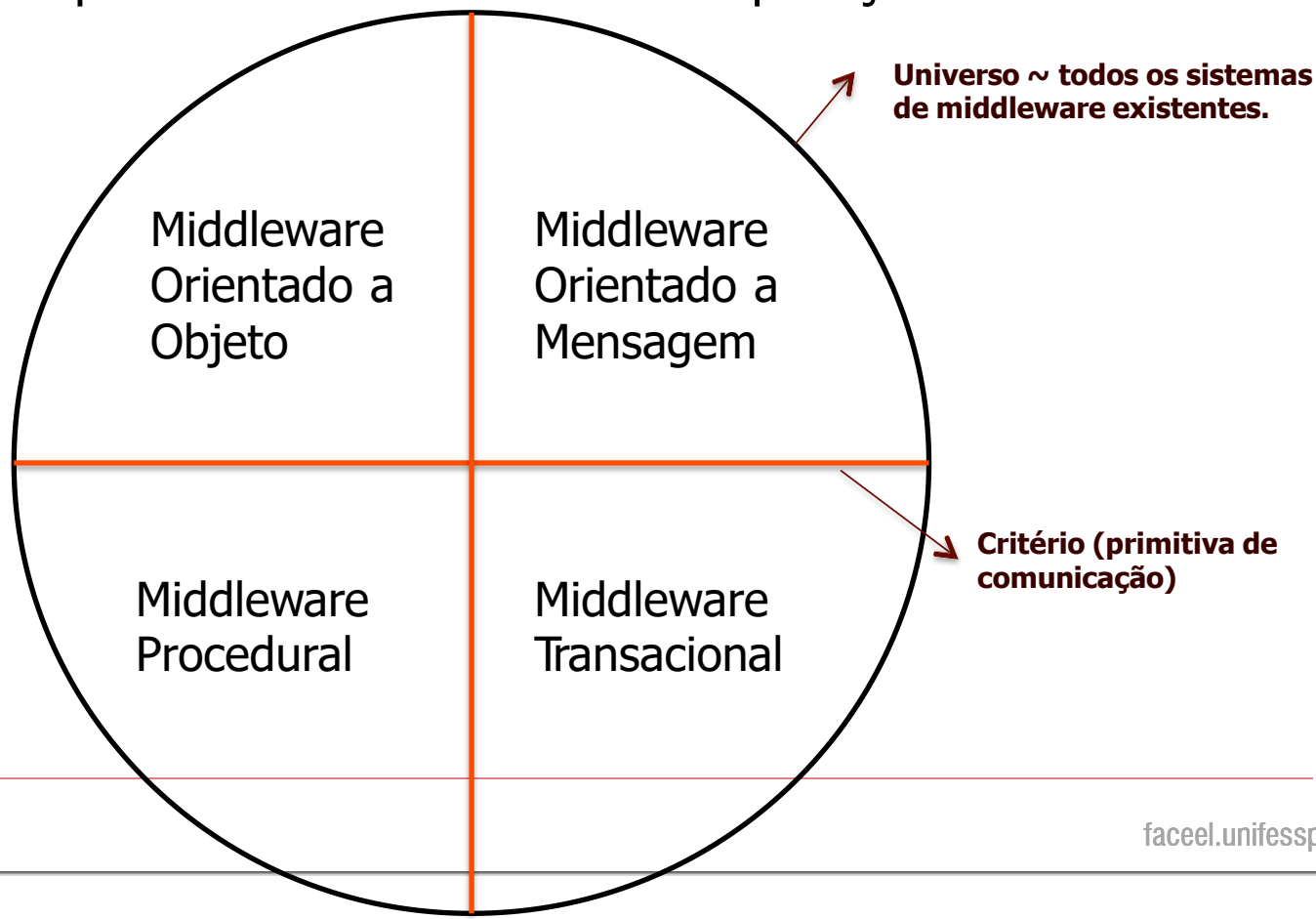
---

- ❑ 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.



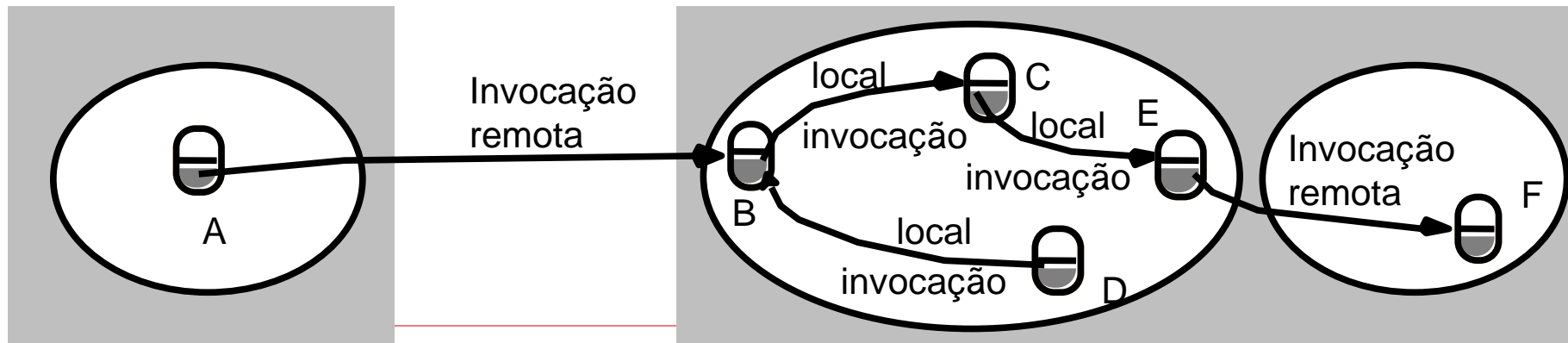
# Middleware - Classificação

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



# Invocação a métodos remotos (RMI)

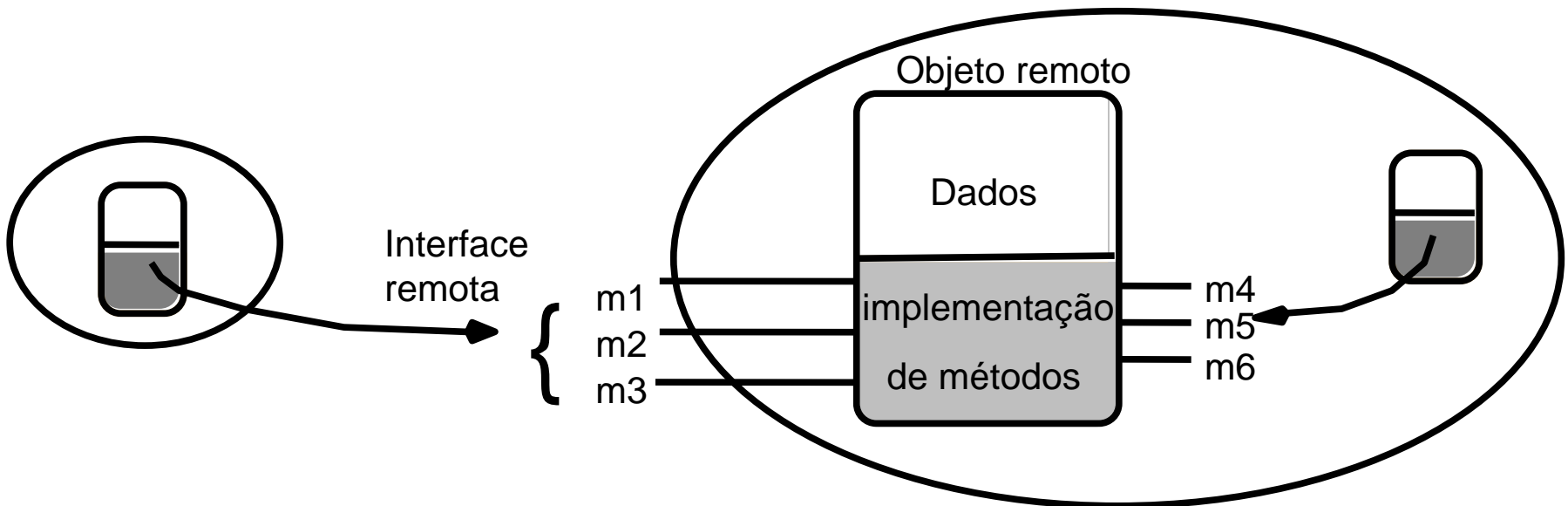
- ❑ Mesmo Processo → Invocações a métodos Locais
- ❑ Diferentes Processos → Invocações a métodos remotos
- ❑ Objetos Remotos: Objetos que podem receber invocações remotas.



# Invocação a métodos remotos (RMI)

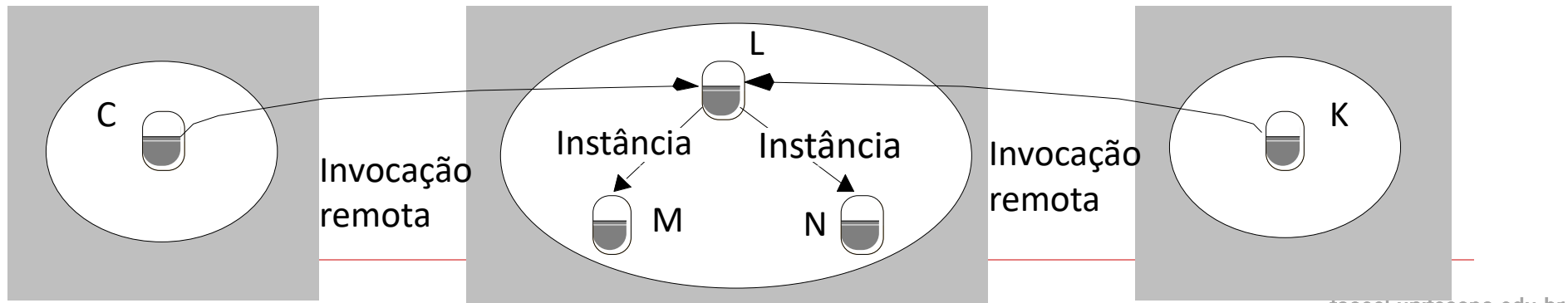
---

- ❑ Referência de objeto remoto
- ❑ Interfaces remotas



# Invocação a métodos remotos (RMI)

- ❑ Ações em um sistema de objeto distribuído
- ❑ Coleta de lixo em um sistema de objeto distribuído
- ❑ Exceções



# Java RMI

---



- Java é oferecida em três versões
  - J2ME (Java 2 Micro Edition)
    - Para celulares, PDAs, sist. embarcados, ...
  - J2SE (Java 2 Standard Edition)
    - Para desktops
  - J2EE (Java 2 Enterprise Edition)
    - Para servidores
- Versões diferem nas APIs oferecidas
- J2SE e J2EE possuem suporte para invocação remota de métodos (RMI)

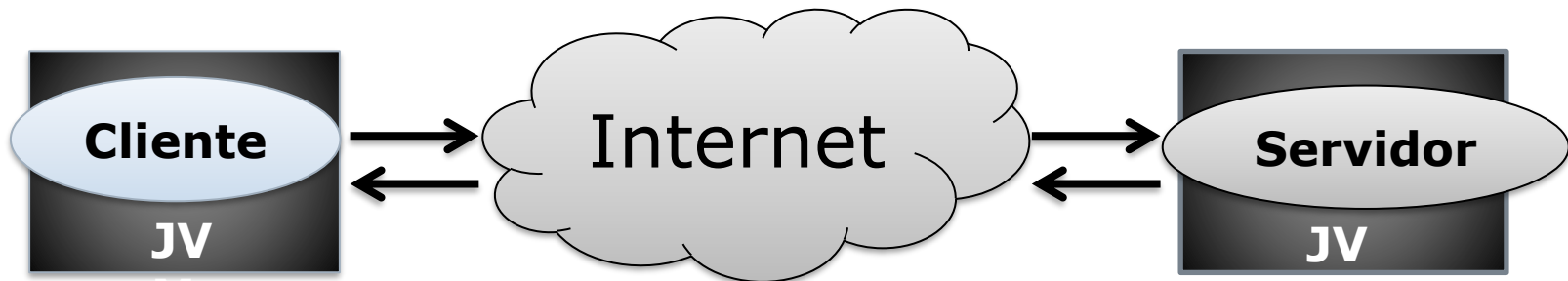


# Java RMI

---

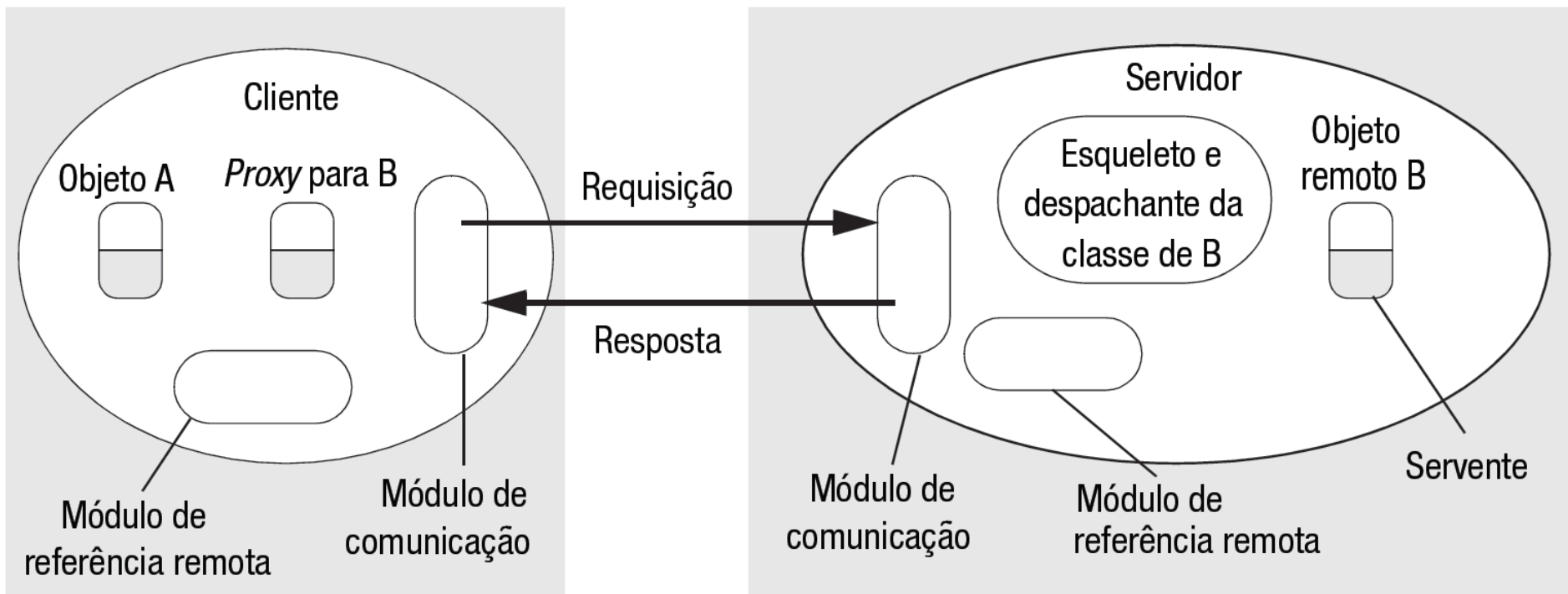
## □ Java RMI (*Remote Method Invocation*)

- Fornece um suporte simples para RPC/RMI
- Permite que um objeto Java chame métodos de outro objeto Java rodando em outra JVM
- Solução específica para a plataforma Java



# Java RMI

- ❑ A função do *proxy* e do esqueleto na invocação a método remoto.

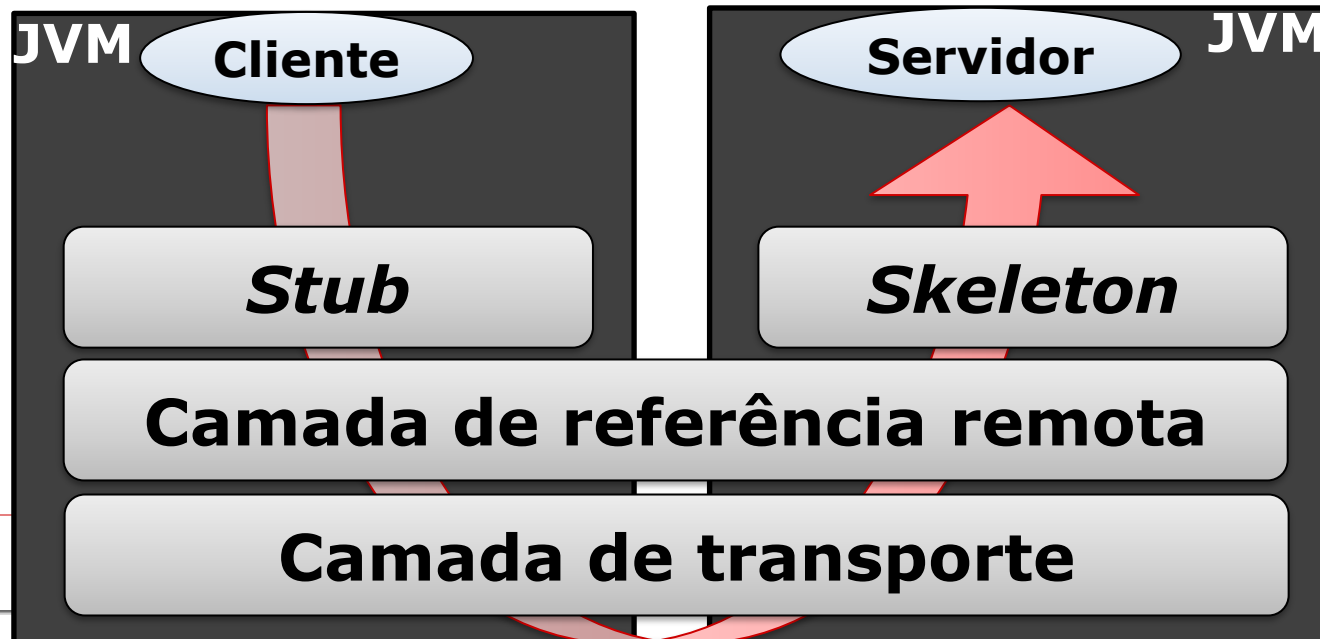


# Java RMI

---

## □ Arquitetura RMI

- Stub e Skeleton
- Camada de referência remota
- Camada de transporte



# Java RMI

---

## □ Stub

- Representa o servidor para o cliente
- Efetua serialização e envio dos parâmetros
- Recebe a resposta do servidor, desserializa e entrega ao cliente

## □ Skeleton

- Recebe a chamada e desserializa os parâmetros enviados pelo cliente
- Faz a chamada no servidor e retorna o resultado ao cliente

# Java RMI

---

## ❑ Camada de Referência Remota

- Responsável pela localização dos objetos nas máquinas da rede
- Permite que referências para um objeto servidor remoto sejam usadas pelos clientes para chamar métodos

## ❑ Camada de Transporte

- Cria e gerencia conexões de rede entre objetos remotos
- Elimina a necessidade do código do cliente ou do servidor interagirem com o suporte de rede

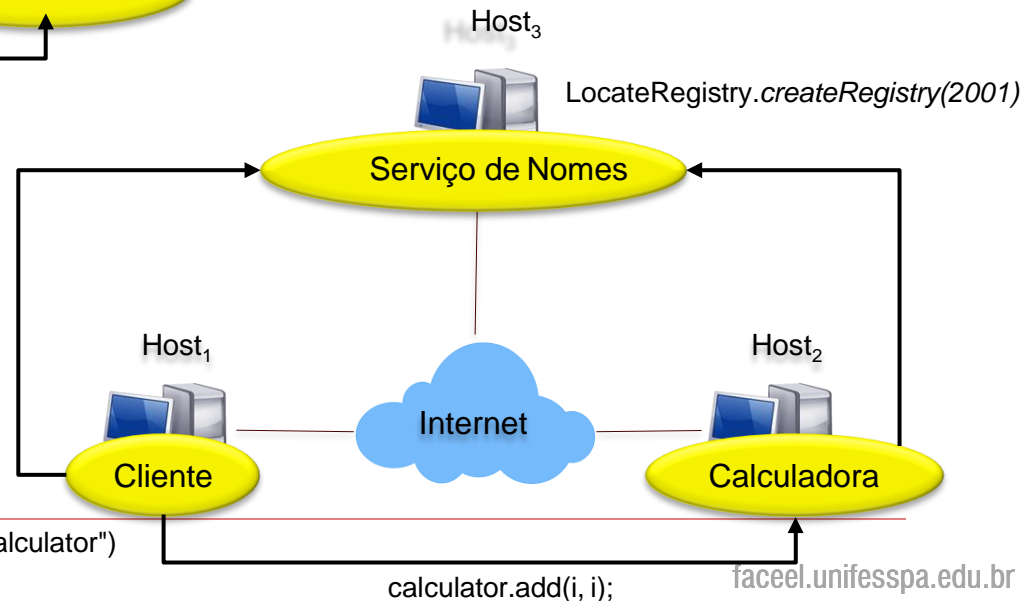
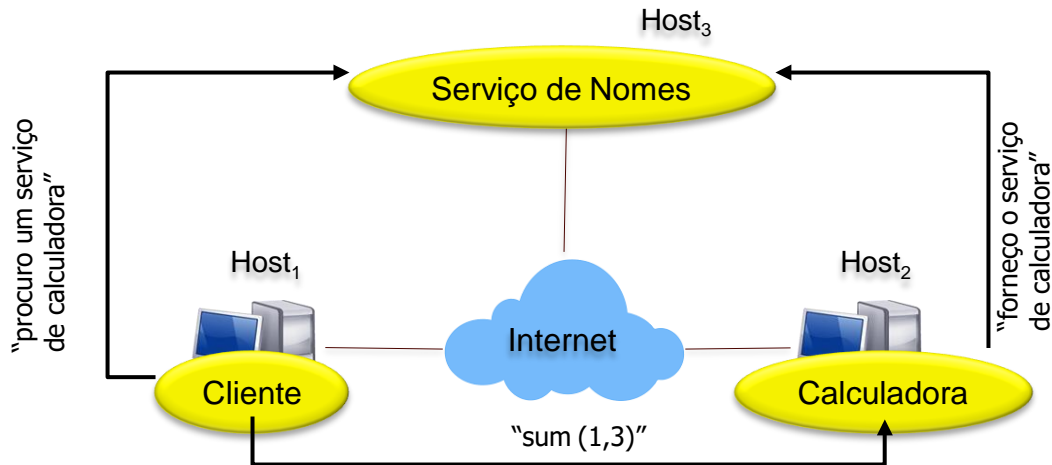
# Java RMI

---

## □ Desenvolvimento de Aplicações com RMI

- Devemos definir a interface do servidor
  - A interface do servidor deve estender `java.rmi.Remote`
  - Todos os métodos da interface devem prever a exceção `java.rmi.RemoteException`
  - O Servidor irá implementar esta interface
- Stubs e skeletons são gerados pelo compilador RMI (`rmic`) com base na interface do servidor

# Exemplo - Calculadora



# Exemplo – Calculadora (Interface)


```
public interface ICalculator extends Remote {  
    float add(float x, float y) throws RemoteException;  
    float sub(float x, float y) throws RemoteException;  
    float mul(float x, float y) throws RemoteException;  
    float div(float x, float y) throws RemoteException;  
}
```

```
public class CalculatorImpl extends UnicastRemoteObject implements ICalculator {  
  
    protected CalculatorImpl() throws RemoteException {  
        super();  
    }  
  
    private static final long serialVersionUID = 1L;  
  
    public float add(float x, float y) {  
        return x + y;  
    }  
  
    public float sub(float x, float y) throws RemoteException {  
        return x - y;  
    }  
  
    public float mul(float x, float y) throws RemoteException {  
        return x * y;  
    }  
  
    public float div(float x, float y) throws RemoteException {  
        return x / y;  
    }  
}
```



# Exemplo – Calculadora (Cliente)

---



```
public class CalculatorClient {  
  
    public static void main(String[] args) throws RemoteException,  
        NotBoundException, InterruptedException {  
  
        // obtain a reference to a bootstrap remote object registry  
        Registry registry = LocateRegistry.getRegistry("localhost", 1313);  
  
        // look for an instance of Calculator in the Naming service  
        ICalculator calculator = (ICalculator) registry.lookup("Calculator");  
  
        // invoke the remote operation  
        float result = calculator.add(1, 3);  
        System.out.println("add (1,3) = " + result);  
    }  
}
```

# Exemplo – Calculadora (Servidor)

---



```
public class CalculatorServer {  
  
    protected CalculatorServer() throws RemoteException {  
        super();  
    }  
  
    public static void main(String args[]) throws RemoteException,  
        AlreadyBoundException {  
  
        // create an instance of Calculator  
        CalculatorImpl calculator = new CalculatorImpl();  
        //ICalculator calculatorI = (ICalculator) UnicastRemoteObject.exportObject(calculator, 0);  
  
        // create a Registry instance on the local host  
        Registry registry = LocateRegistry.getRegistry("localhost",1313);  
  
        // register the instance of Calculator in the Naming Service  
        registry.bind("Calculator", calculator);  
    }  
}
```