

# COMPUTAÇÃO DISTRIBUÍDA

## INTRODUÇÃO

Carlos Ferraz <cagf@cin.ufpe.br>

# MOTIVAÇÃO

Computação em evolução

# Antes disso...



# Antes disso...

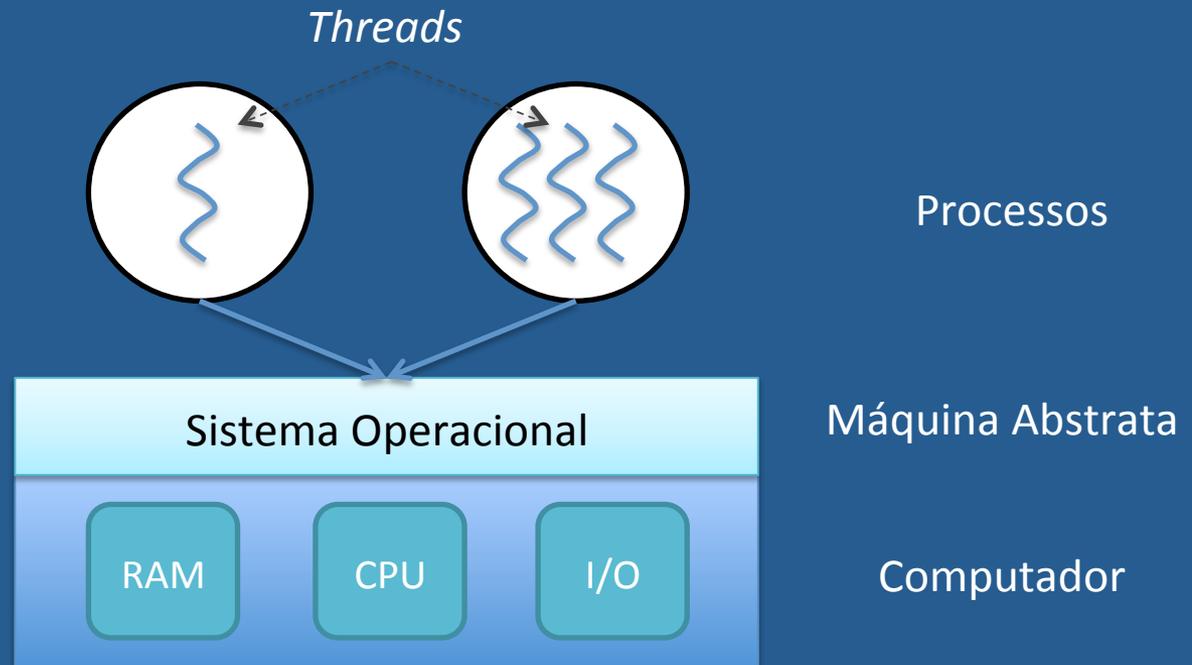


# Evolução



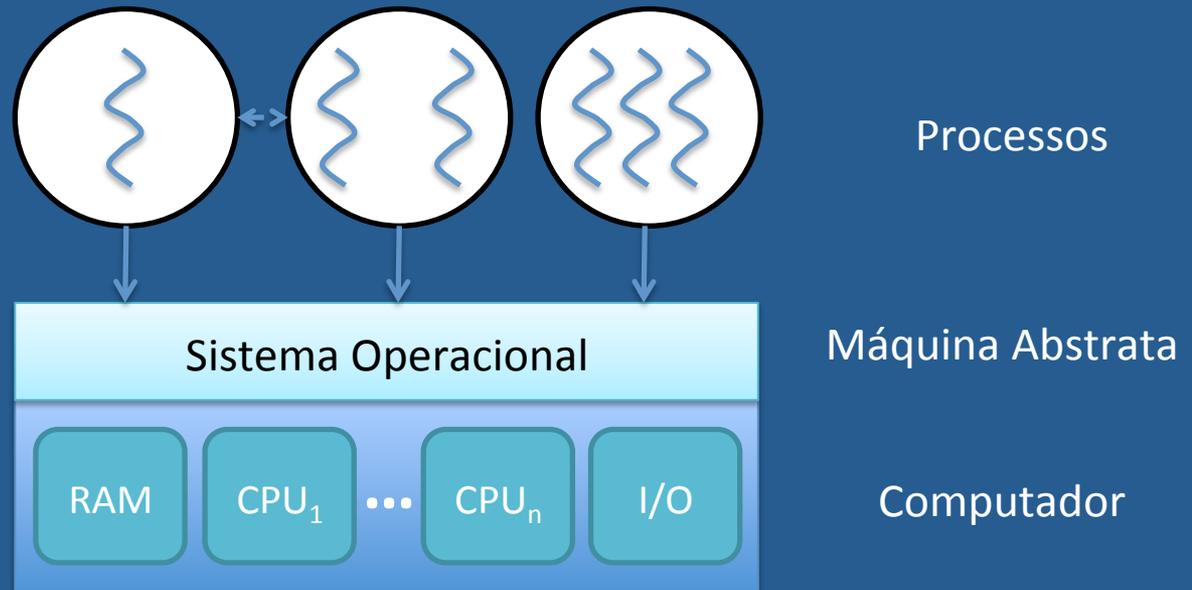
Computação Centralizada

# Evolução



Concorrência

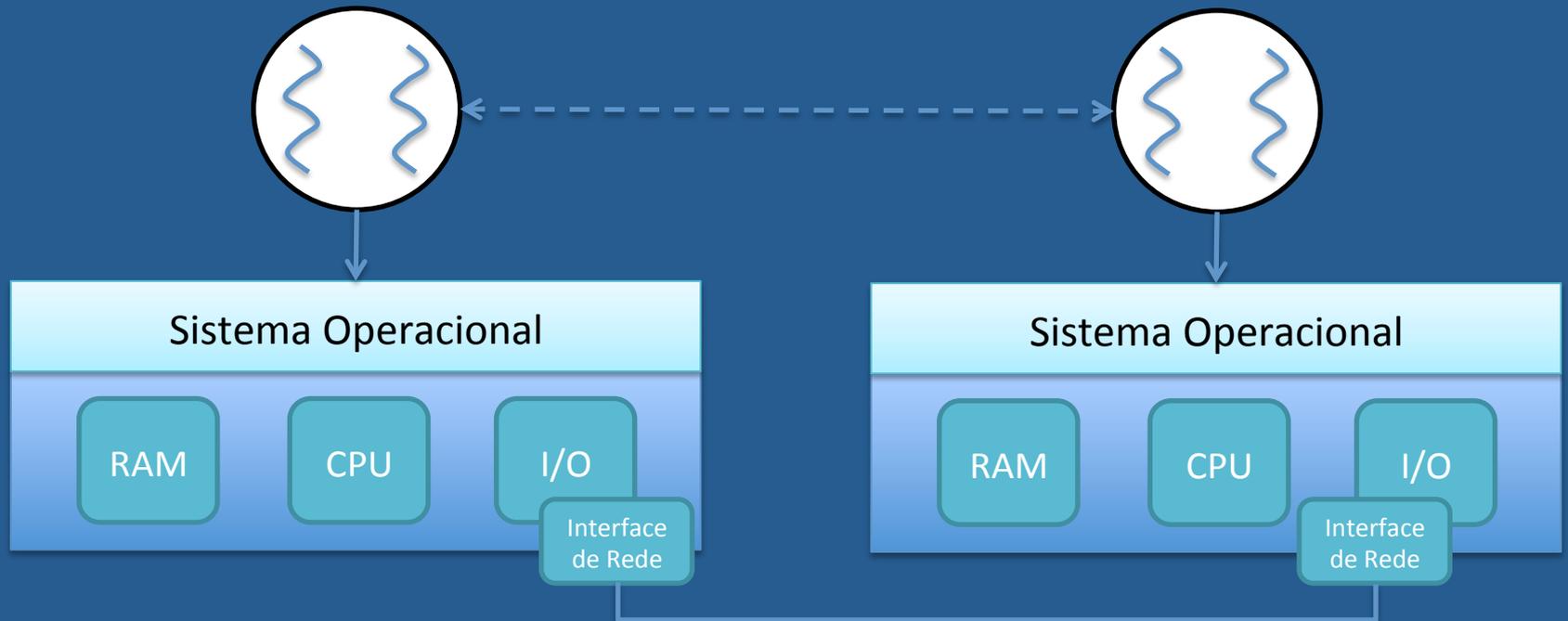
# Evolução



## Computação Paralela

(e comunicação entre processos (IPC) concorrentes/paralelos)

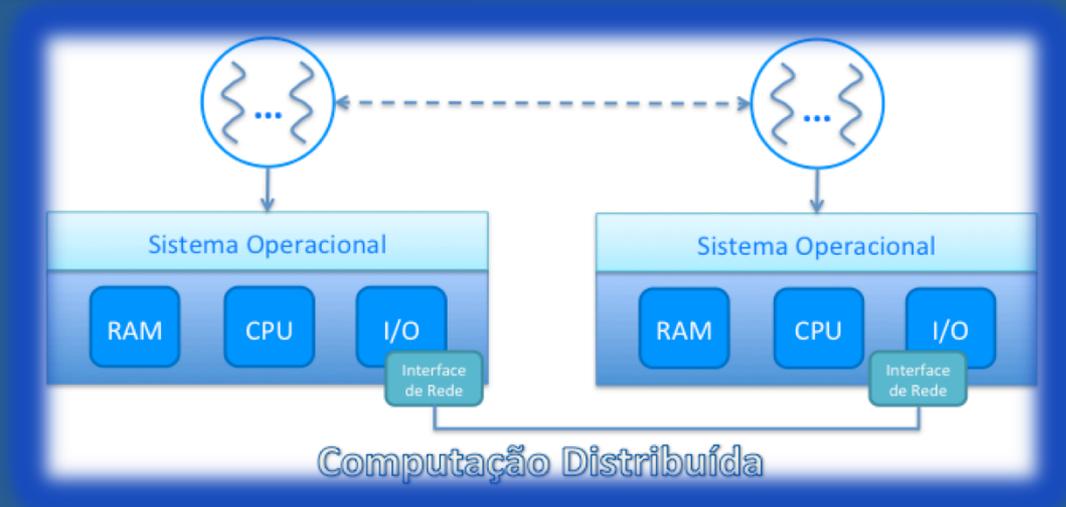
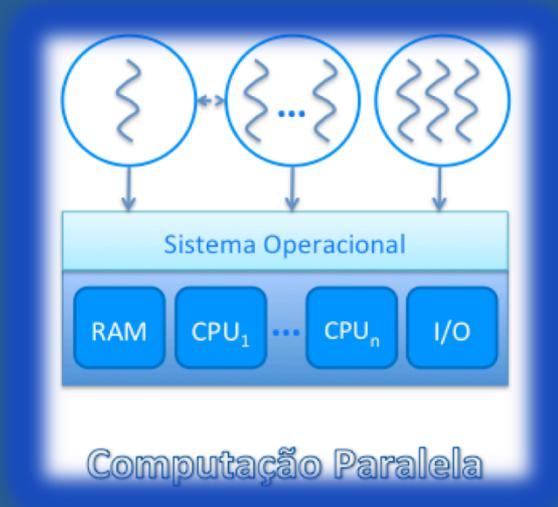
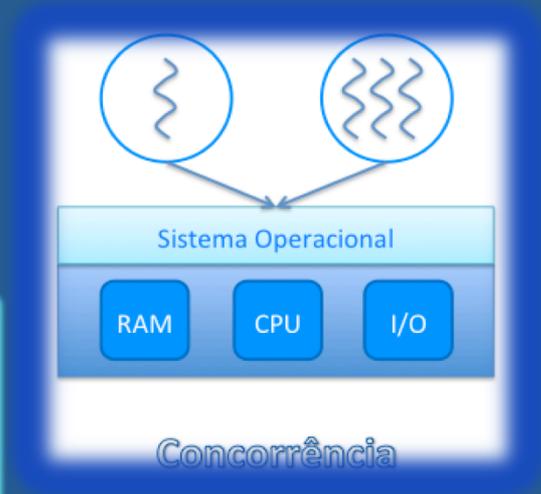
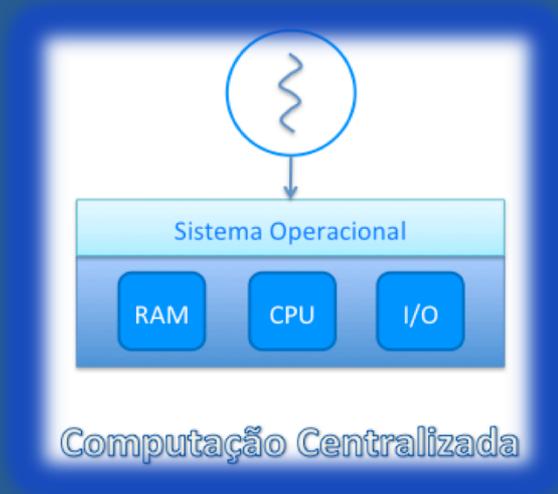
# Evolução



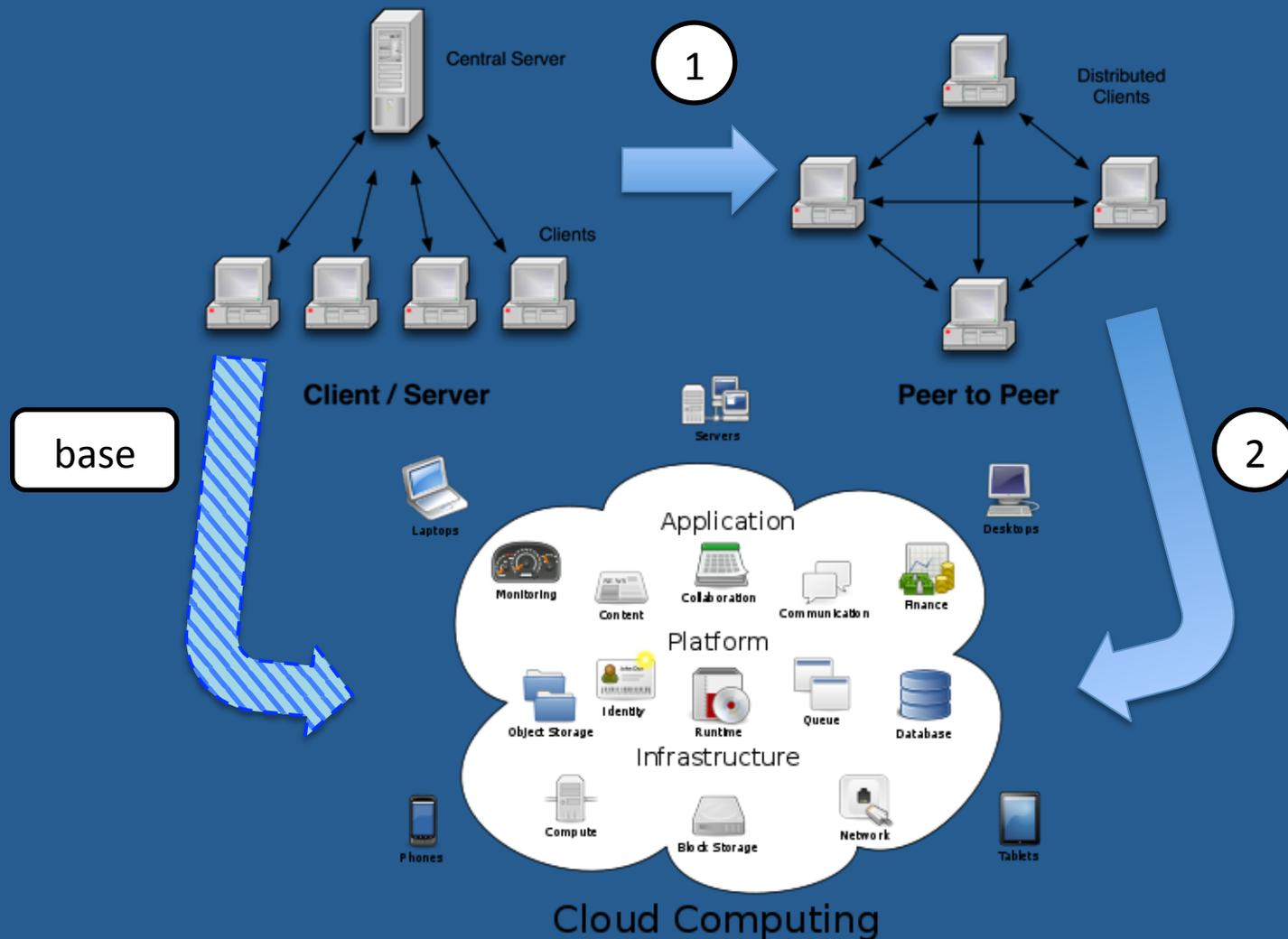
Computação Distribuída

# Evolução da Computação

**Ou qualquer  
combinação  
entre elas !!!**



# Uma história acelerada da evolução da Computação Distribuída

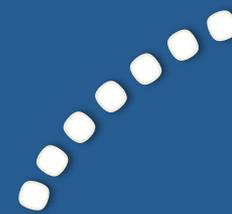


Computação Móvel

Computação Multimídia

Computação na Nuvem

Computação Ubíqua



**Atenção!!!**  
Você fará parte do  
desenvolvimento  
disso

**Computação Distribuída**

# INTRODUÇÃO

Dividir para Conquistar

# Definição

*Um **sistema distribuído** é aquele no qual componentes localizados em uma rede de computadores **se comunicam e coordenam suas ações** através da passagem (troca) de mensagens*

Coulouris et al., 2012

# Características marcantes

Concorrência de componentes

Falta de um relógio global

Componentes falham de forma independente –

falha parcial

# Tendências-chave

O que move os Sistemas Distribuídos hoje?

Pervasividade das redes

Computação móvel e ubíqua

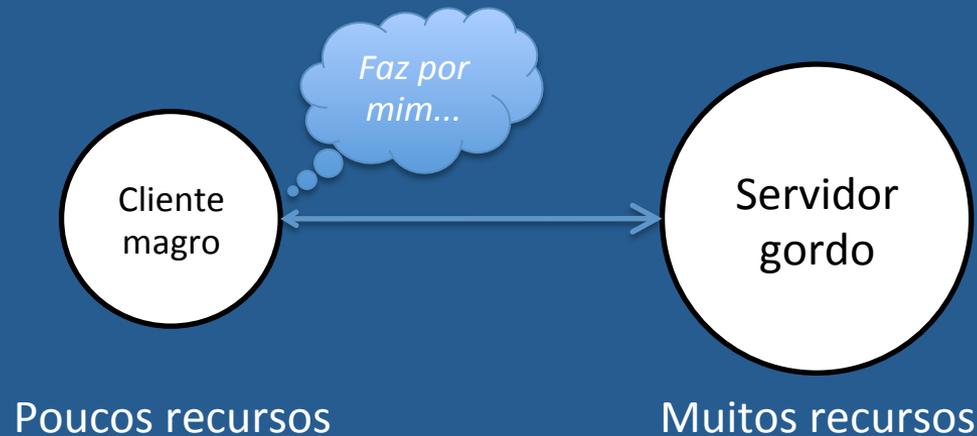
Importância crescente de sistemas multimídia  
(distribuídos)

Sistemas distribuídos como utilidade (serviço) – *Cloud Computing* e seus modelos (IaaS, PaaS, SaaS)

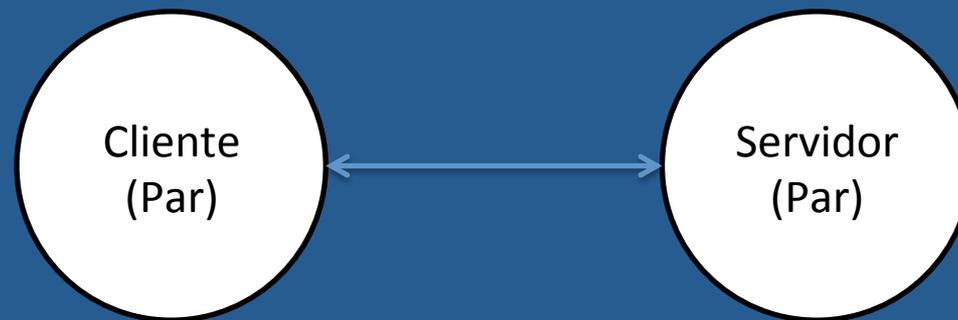
Principal motivação: **compartilhamento de recursos**

# Compartilhamento de recursos

Passado



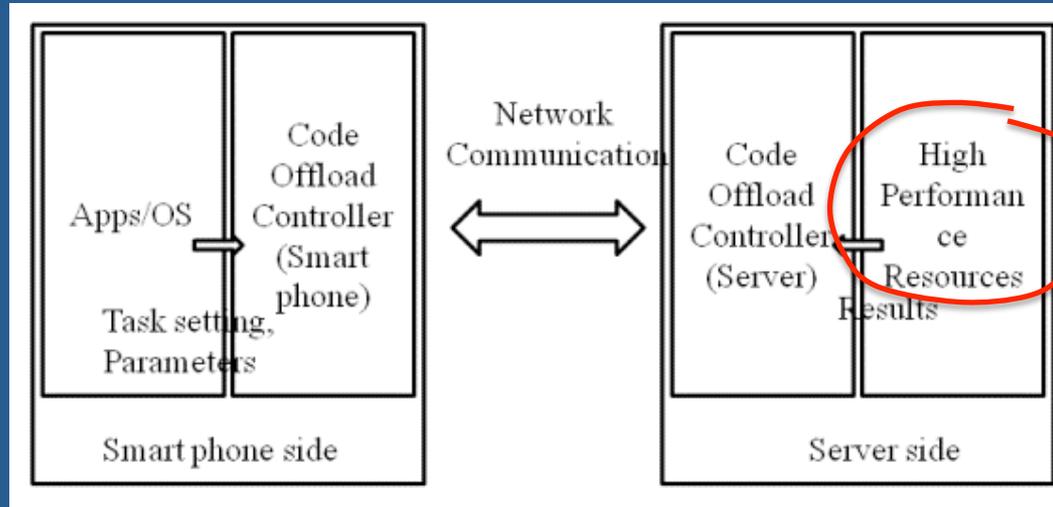
Presente



“Equilíbrio” de recursos (compartilhamento mais “verdadeiro” e justo)

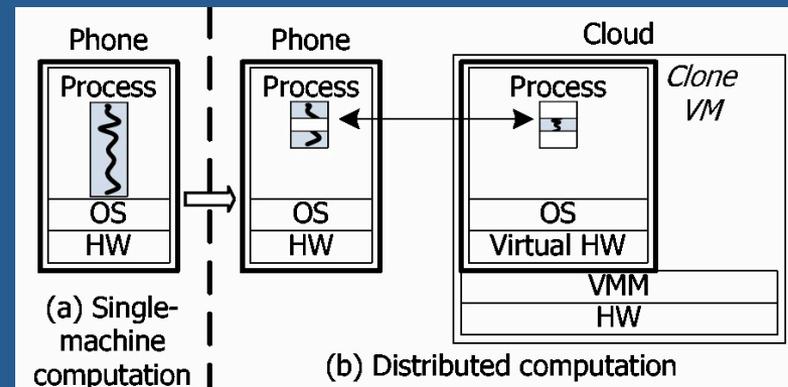
*Abundância, mas...*

# Computation Offloading



## MOBILE CLOUD COMPUTING

Como se faz?



# Objetivos de um projeto de SD

Eficiência – desempenho produtivo

Robustez – resistência a falha

Disponibilidade: o sistema está no ar quando preciso (*instante de tempo*)

Confiabilidade: o sistema não falha por um longo *período de tempo*

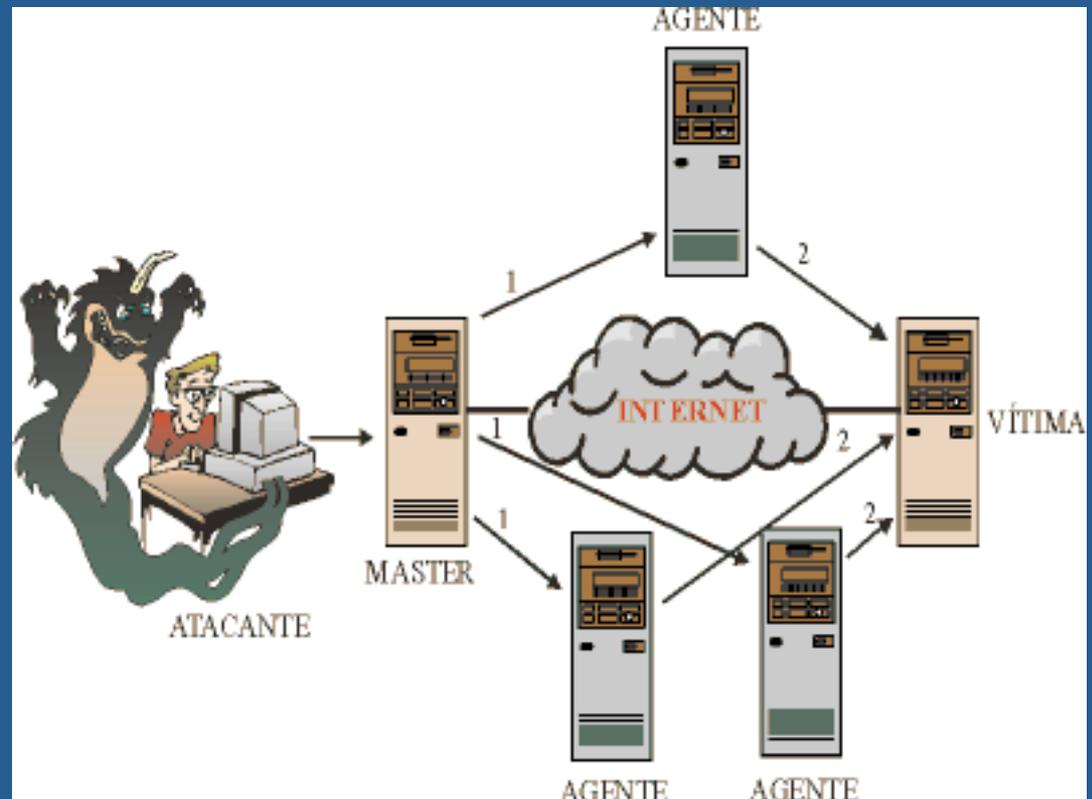
E quando falha, a falha não provoca uma “catástrofe”

Consistência – mesma visão de dados

# Objetivo: Eficiência

Para o bem, baseado em paralelismo

Para o mal, DDoS...



# Desafios

Heterogeneidade

Abertura

Segurança

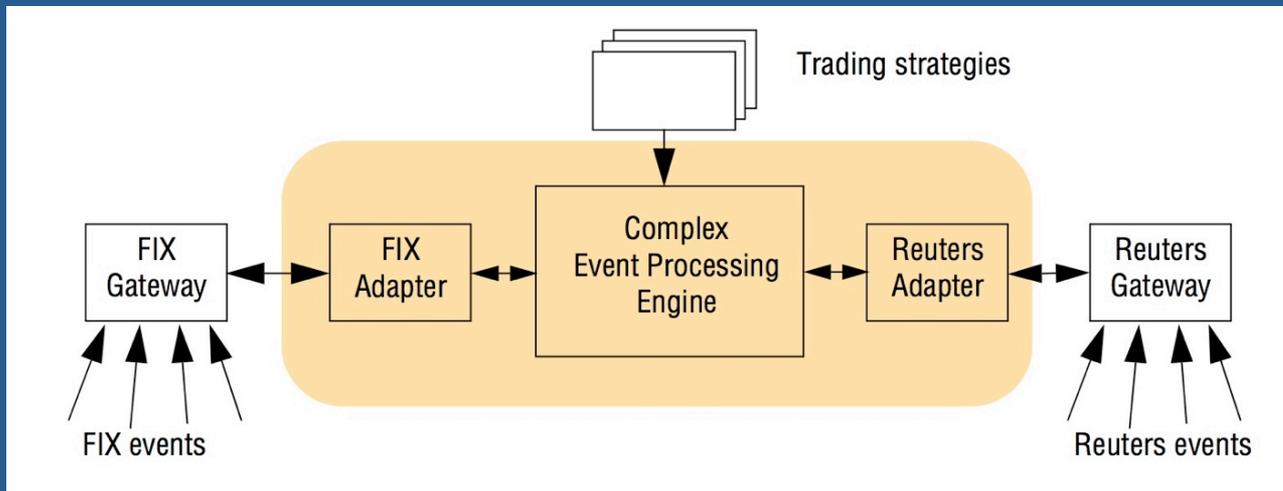
Escalabilidade

Tratamento de falha

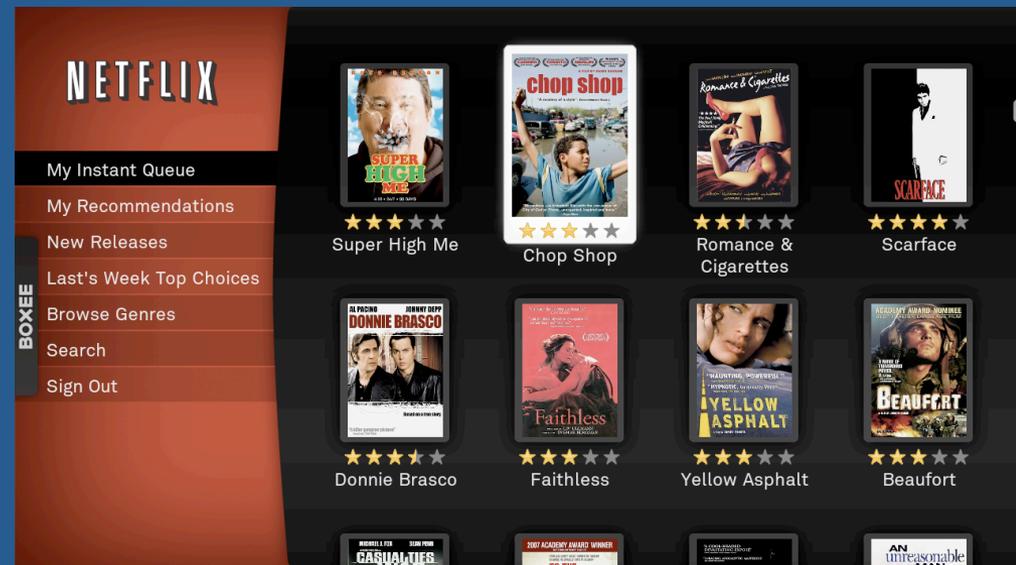
Concorrência

Transparência

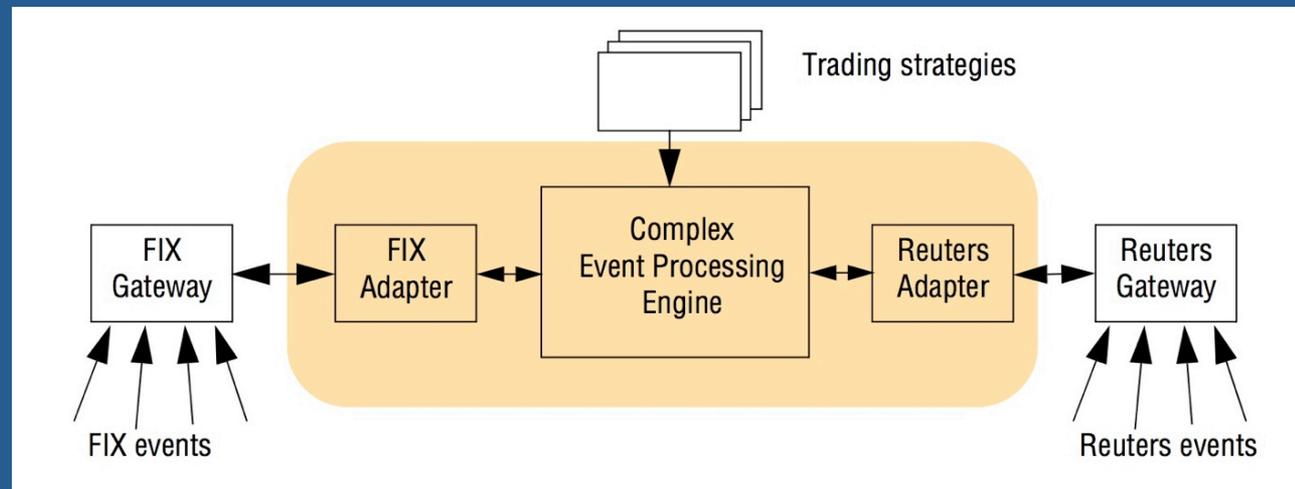
# Dois exemplos de sistemas distribuídos



Sistema de Negociação Financeira [Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design. Edn. 5 © Pearson Education 2012]



## EVENTOS



Acesso em tempo real a **muitas fontes de informação** (ex. Reuters, FIX – Financial Information eXchange), interessantes para **muitos clientes**

Comunicação e processamento de itens de interesse – **eventos**



**Sistemas distribuídos baseados em eventos**

Fontes de informação em diferentes formatos – **heterogeneidade** → adaptadores para traduzir para formato comum (ex. XDR/RPC)

Diversos fluxos de eventos, chegando a taxas rápidas, muitas vezes requerendo processamento em tempo real – **Complex Event Processing (CEP)**: composição de eventos baseada em padrões lógicos, temporais ou espaciais

# Transparência de distribuição

*WHEN*

*MSFT price moves outside 2% of MSFT Moving Average*

*FOLLOWED-BY (*

*MyBasket moves up by 0.5%*

*AND*

*HPQ's price moves up by 5%*

*OR*

*MSFT's price moves down by 2%*

*)*

*)*

*ALL WITHIN*

*any 2 minute time period*

*THEN*

*BUY MSFT*

*SELL HPQ*

O usuário que escreveu este *script* não precisa ter noção da ocorrência distribuída de eventos

Dados e processamento distribuídos

# Sistemas multimídia distribuídos



# Conceito-chave

Distribuição

Comunicação

Complexidade

Heterogeneidade



**Transparência**

# Tipos de Transparência

**Localização:** esconde onde o recurso está localizado

**Acesso:** operações idênticas para acesso local e remoto

Migração: esconde que um recurso pode se mover para outra localização

Relocação: esconde que um recurso pode ser movido para outra localização enquanto está em uso

Concorrência: compartilhamento de recursos sem interferência entre processos concorrentes

Falha: esconde a falha e recuperação de um recurso

Replicação: esconde de usuários ou programadores de aplicação a existência de réplicas de recursos

Localização + acesso = transparência de rede

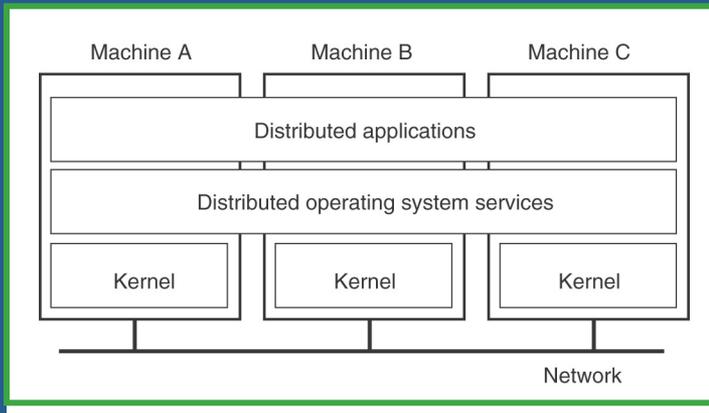
# Infra-estruturas de software para Sistemas Distribuídos

Sistemas Operacionais Distribuídos

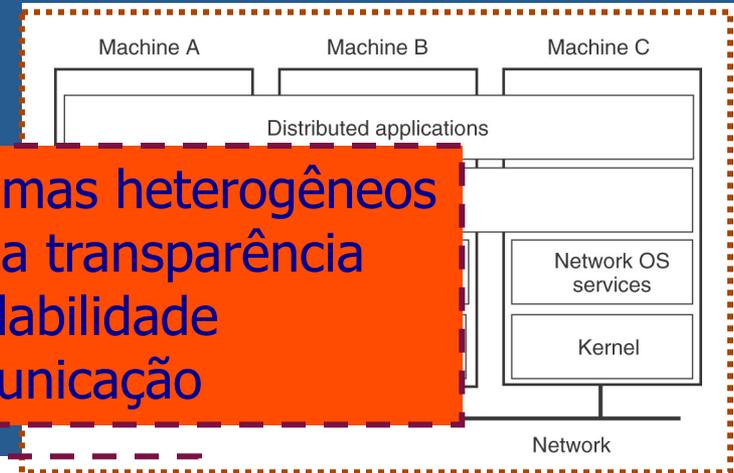
Sistemas Operacionais de Rede

Middleware

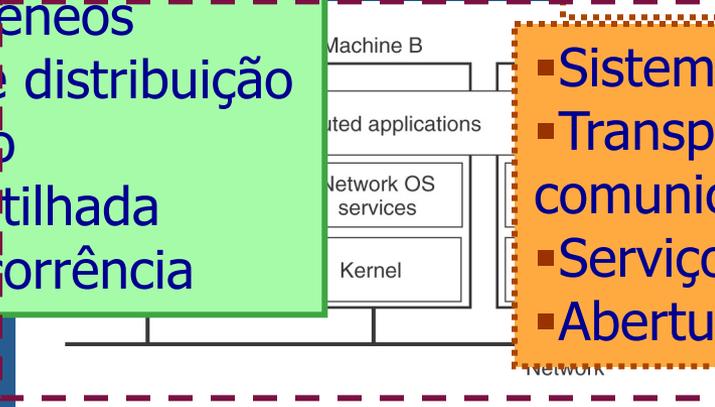
# Infra-estruturas para SDs: diferenças de objetivos e abstrações



- Sistemas homogêneos
- Transparência de distribuição
- Alto desempenho
- Memória compartilhada
- Controle de concorrência



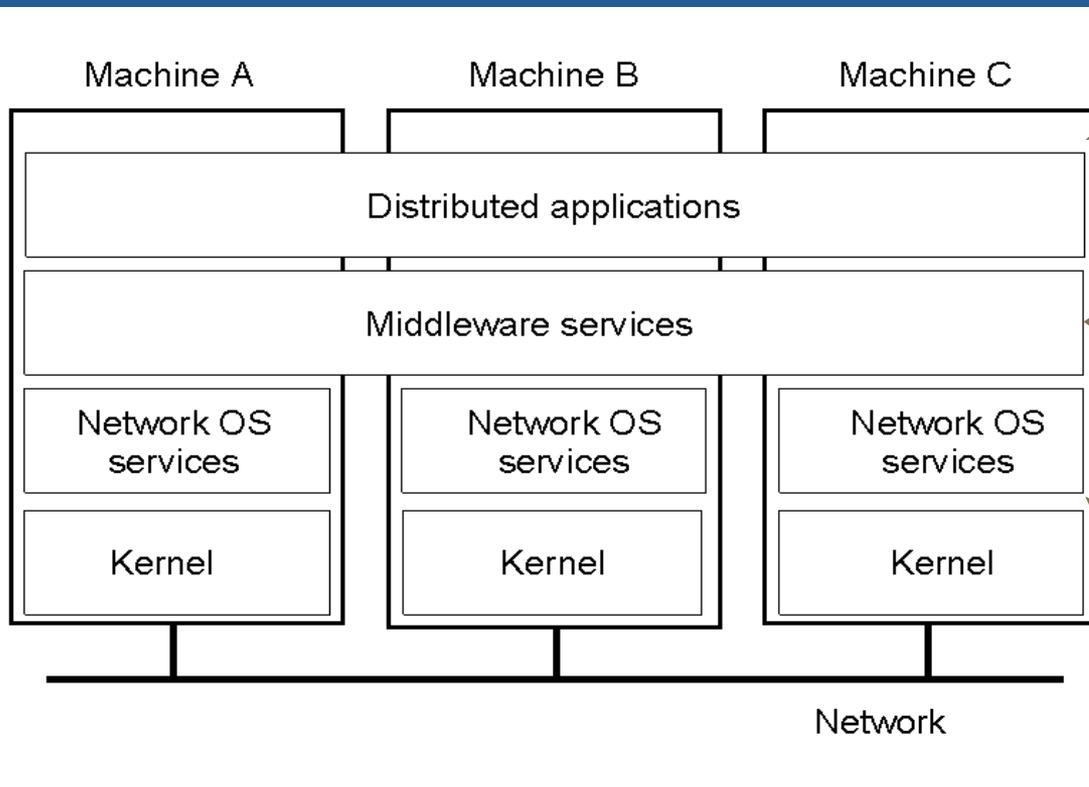
- Sistemas heterogêneos
- Pouca transparência
- Escalabilidade
- Comunicação



- Sistemas heterogêneos
- Transparência de distribuição e comunicação
- Serviços
- Abertura

**MIDDLEWARE**

# Necessidades: quem atende o quê?



## ■ Lógica do negócio

Integração de Sistemas  
Heterogêneos

Interoperabilidade destes sistemas  
Portabilidade

## ■ Comunicação

Uso e gerência de recursos  
de hardware:

CPU  
Memória  
I/O

# Middleware: definição

*um conjunto reusável, expansível de **serviços e funções** ...  
que são comumente **necessários por parte de várias**  
**aplicações distribuídas** ...  
para funcionarem bem em um **ambiente de rede***

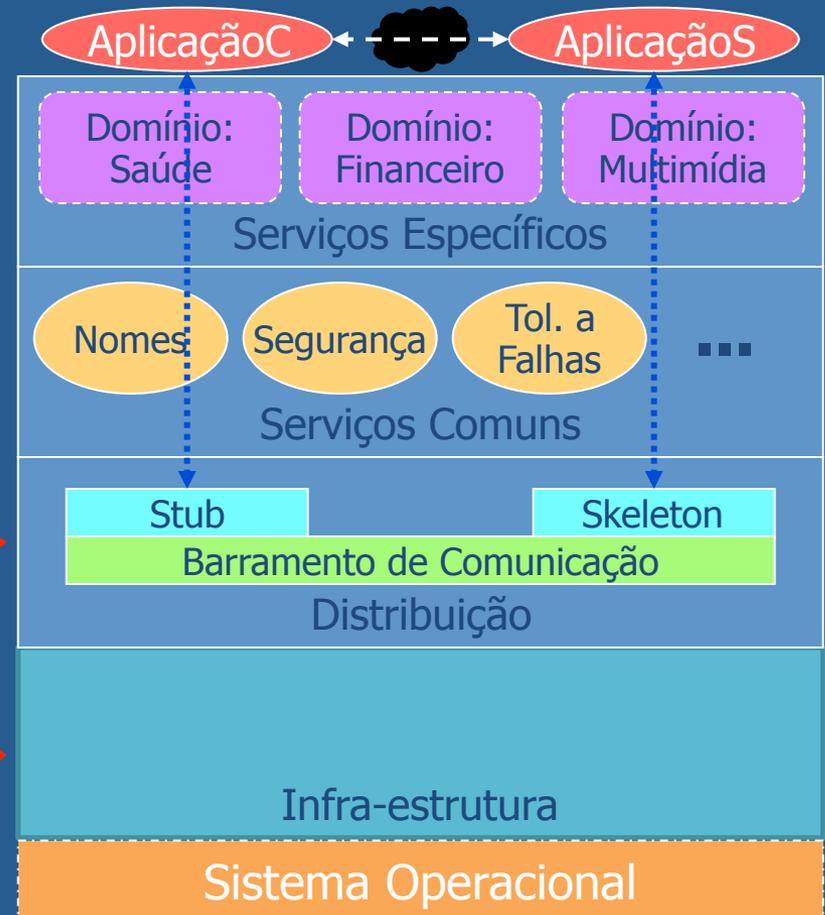
# Infra-Estrutura de SW para SDs

## Middleware: Arquitetura Básica em Camadas

Middleware: coleção de serviços (**serviços de middleware**) fornecidos através de interfaces padrões (**APIs**) – **visão “unificada” de redes e engenharia de software, respectivamente** + formado sobre camadas de infra-estrutura

modelos de programação, onde a comunicação é abstraída, por ex. – ORB CORBA, incluindo RPC

abstrai as peculiaridades dos sistemas operacionais, encapsulando e melhorando os mecanismos de concorrência, por ex. – ex. JVM



# Middleware “Tradicional”

Message-Oriented Middleware (MOM)

Transaction Processing Monitors (TPMON)

Forte associação com acesso distribuído a BD

Propriedades “ACID”

Remote Procedure Calls (RPC)

Object-Oriented Middleware (ORB, ...)

# Próximas Datas

2º. EE: 10/12

Revisão de notas: 14/12

FINAL: 17/12

# INVOCACÃO REMOTA E COMUNICAÇÃO INDIRETA

RPC, RMI

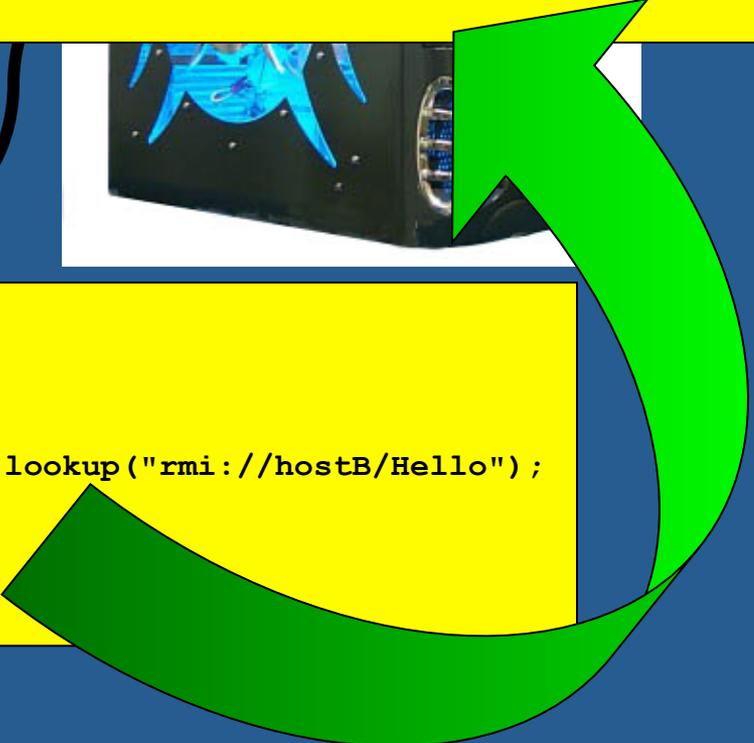
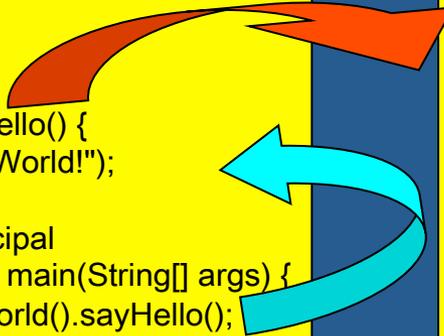
PUB-SUB, FILAS DE MENSAGENS

```
// quase Java....
class HelloWorld {
    // método local
    public void sayHello() {
        print("Hello World!");
    }
    // programa principal
    public static void main(String[] args) {
        new HelloWorld().sayHello();
    }
}
```

```
public class HelloWorld {
    public HelloWorld ( String name ) {
        Naming.rebind( name, this );
    }
    // método remoto
    public String sayHello () {
        return "Hello World!";
    }
}
public class HelloWorldServer {
    public static void main(String[] args) {
        HelloWorld object = new HelloWorld( "Hello" );
    }
}
```

```
public class HelloWorldClient {

    public static void main(String[] args) {
        // conexão
        HelloWorld hello_server = (HelloWorld)Naming.lookup("rmi://hostB/Hello");
        // chamada remota
        print( hello_server.sayHello() );
    }
}
```



# Camadas

Aplicações e serviços

RMI e RPC

Protocolo Request-Reply  
Marshalling e eXternal Data Representation

TCP e UDP (sockets)

Middleware  
(camada de  
sessão/  
apresentação  
[OSI])

# Transmissão de dados

Dados em programas são **estruturados**

Enquanto isso, mensagens carregam informação **sequencial**:

Linearização/Restauração de dados

**Heterogeneidade** na representação de dados em computadores

Uso de um **formato externo comum**

Inclusão de uma identificação de arquitetura na mensagem

# Marshalling/Unmarshalling

## *Marshalling:*

Linearização de uma coleção de itens de dados estruturados

Tradução dos dados em formato externo (ex: **XDR** – eXternal Data Representation)

## *Unmarshalling:*

Tradução do formato externo para o local

Restauração dos itens de dados de acordo com sua estrutura

# Chamada de Procedimentos Remotos

## Remote Procedure Call (RPC)

Ideal: programar um sistema distribuído como se fosse centralizado

RPC objetiva permitir chamada de procedimento remoto como se fosse local, ocultando entrada/saída de mensagens

RPC: considerações em função da distribuição

evitar passagem de endereço e variáveis globais

novos tipos de erros

Ex.: tratamento de exceções

Se atraso de comunicação ➡ *timeout*

# RPC: processamento de interface

**Objetivo:** integração dos mecanismos de RPC com os programas cliente e servidor escritos em uma linguagem de programação convencional

***Stubs*** (no cliente e no servidor): **transparência de acesso**

tratamento de algumas exceções no local

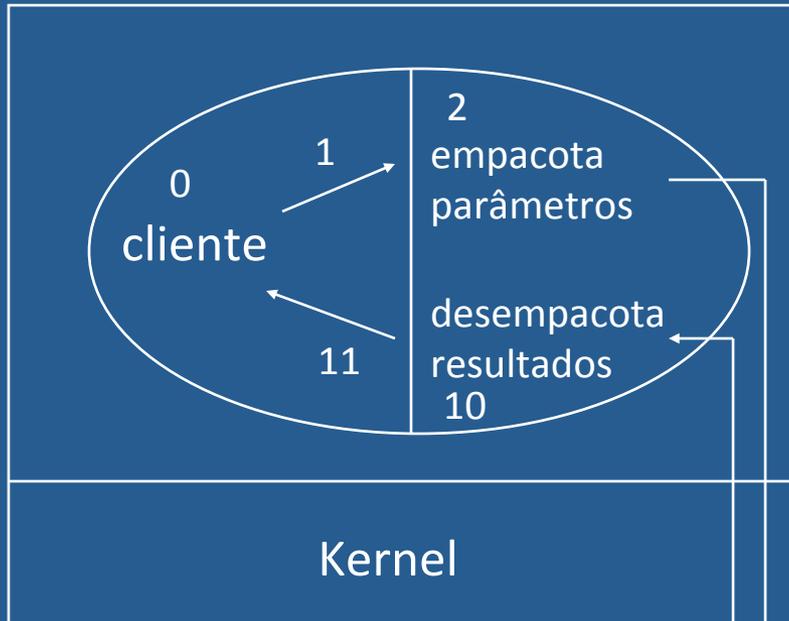
*marshalling*

*unmarshalling*

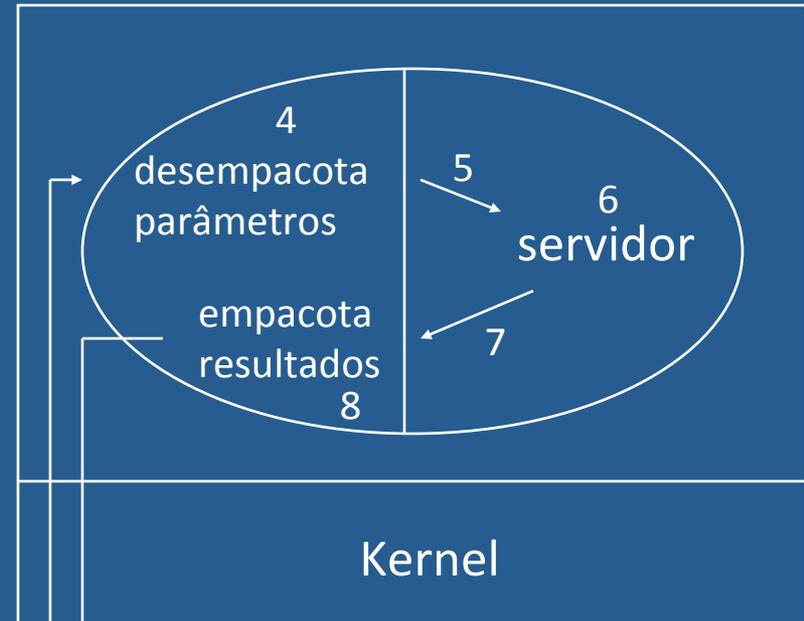


# Chamadas e mensagens em RPC

Máquina do Cliente



Máquina do Servidor



3

9

transporte de mensagens  
via rede

# Funções dos Stubs

## Client stub

1. intercepta a chamada
2. empacota os parâmetros (marshalling)
3. envia mensagem de request ao servidor (através do núcleo)

## Server stub

4. recebe a mensagem de request (através do núcleo)
5. desempacota os parâmetros (unmarshalling)
6. chama o procedimento, passando os parâmetros
7. empacota o resultado
8. envia mensagem de reply ao cliente (através do núcleo)

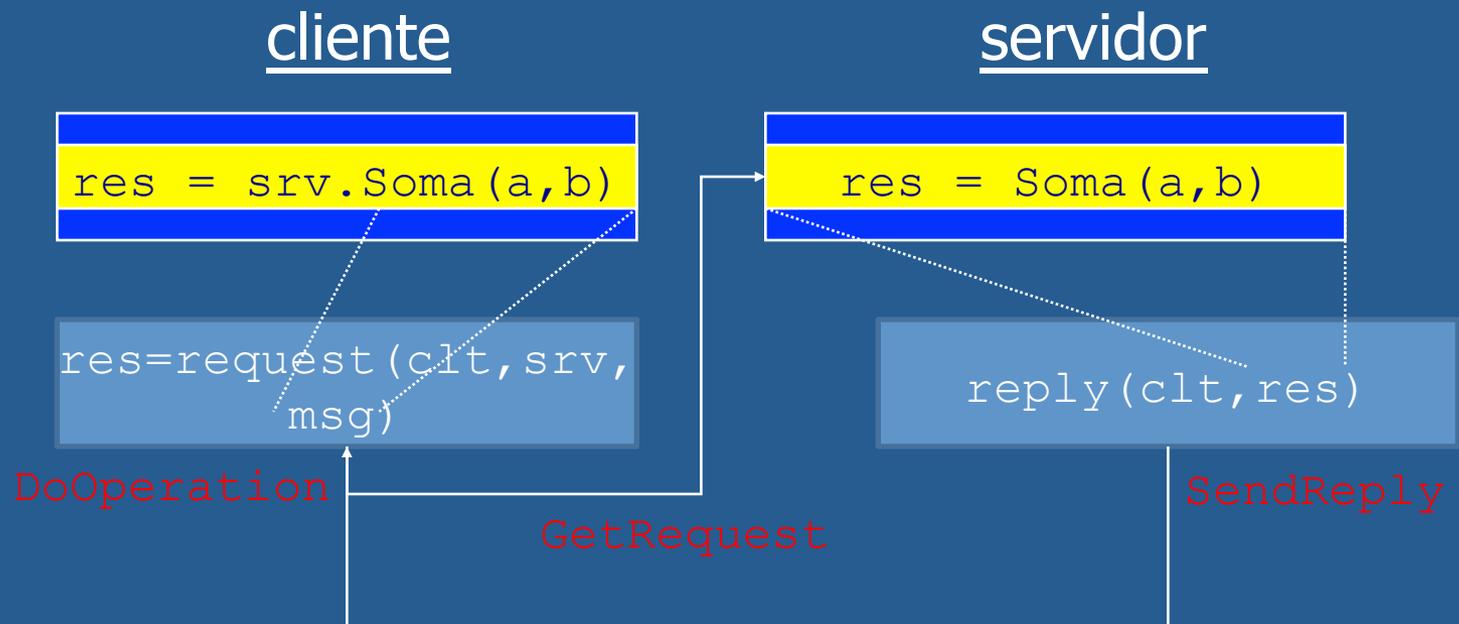
## Client stub

9. recebe a mensagem de reply (através do núcleo)
10. desempacota o resultado
11. passa o resultado para o cliente

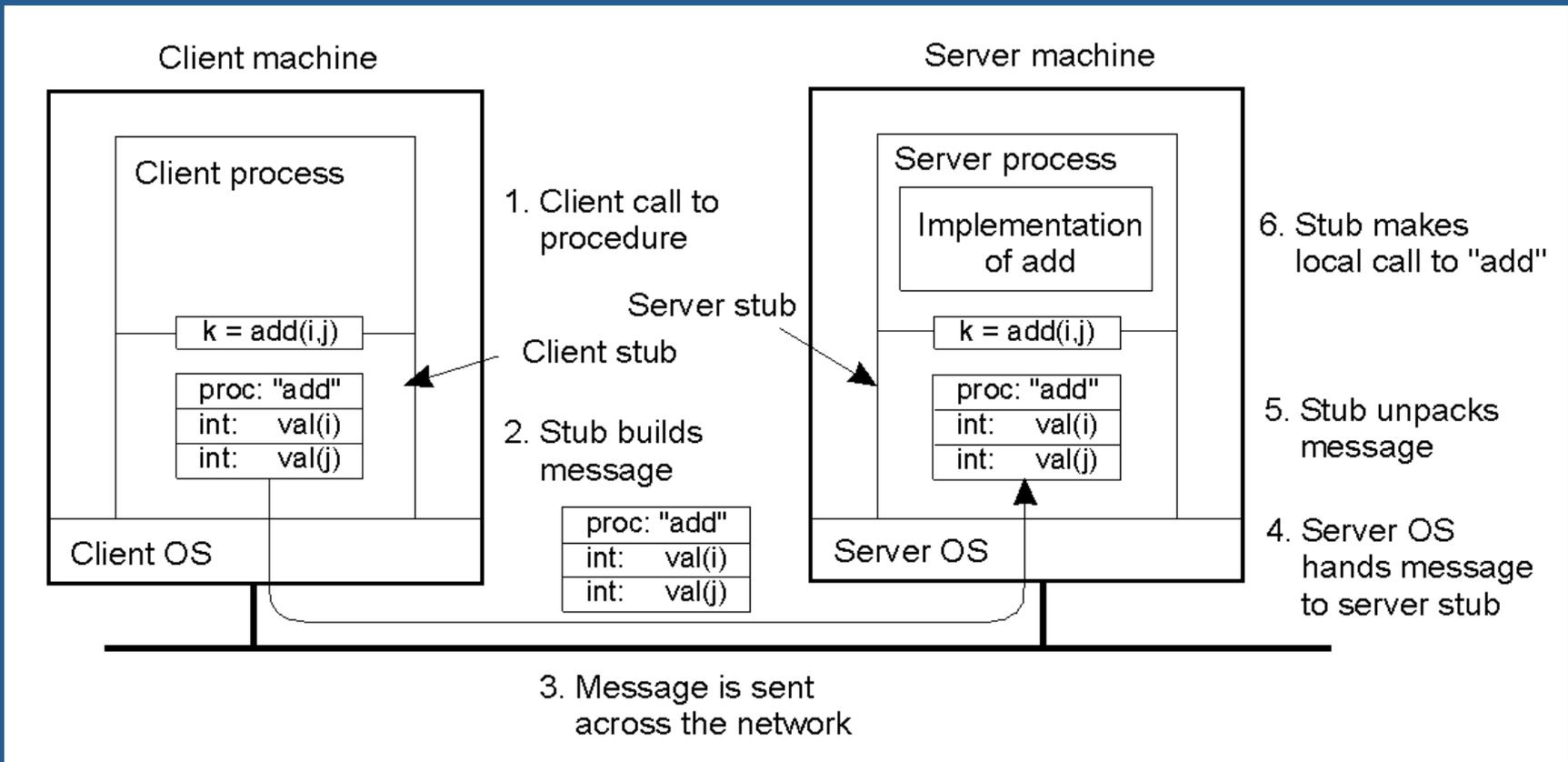
# RPC: tratamento da comunicação

Módulo de comunicação usa protocolo *pedido-resposta* / *request-reply* para troca de mensagens entre cliente e servidor

Mensagem de aplicação encapsulada em um request/reply



# RPC: Passagem de Parâmetros



# RPC: ligação

(antes da comunicação...)

O mecanismo possui um *binder* para resolução de nomes, permitindo

Ligação dinâmica

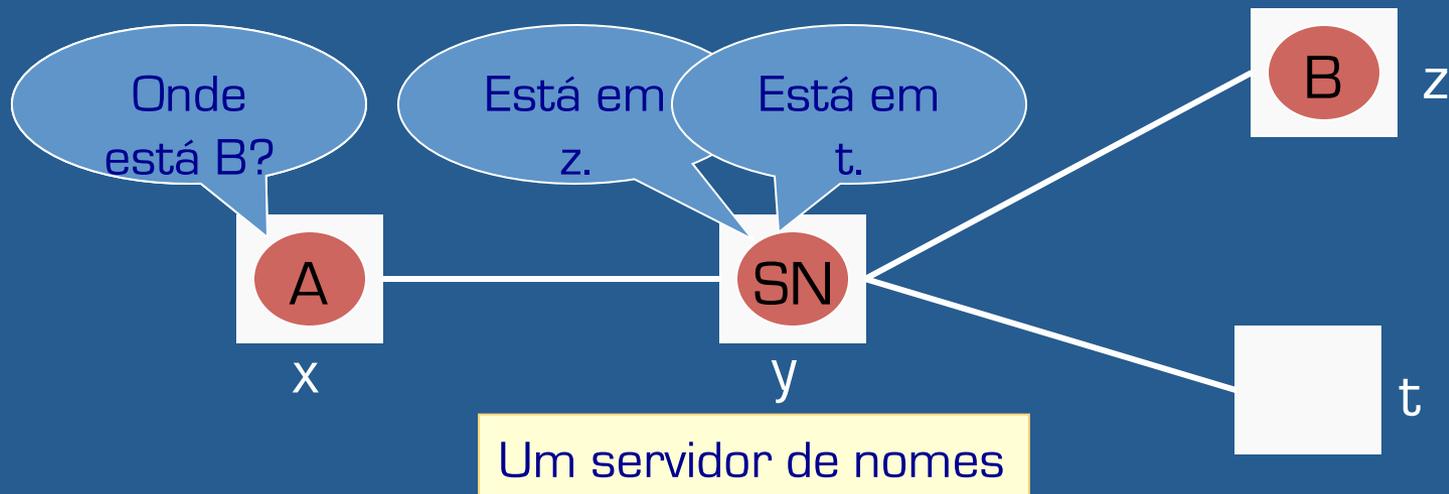
Transparência de localização

# Ligação Dinâmica e Nomes

Nomes podem ser a solução, mas na realidade os endereços físicos é que são necessários...

Daí, faz-se necessário mapear nomes em endereços... como **serviço**

Geralmente, algum agente intermediário tem este papel específico de *resolvedor de nomes*



# Servidores de Nomes

Os servidores de nomes, com este papel de mapeamento de nomes para endereços, contribuem para implementar a

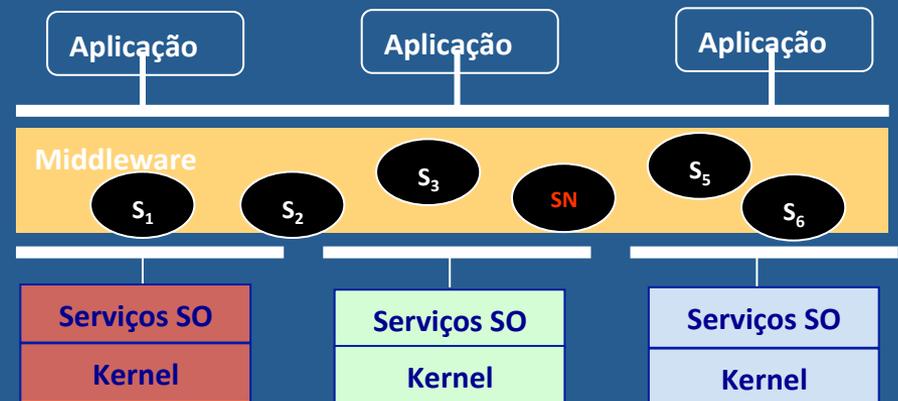
Transparência de localização

Os servidores de nomes são serviços executados no ambiente de suporte a aplicações distribuídas

Exs:

Rmiregistry: JavaRMI

tnameserv: CORBA (JDK)



# Exemplo: Servidor

```
import java.rmi.Naming;  
public class CalculatorServer {  
  
    public CalculatorServer() {  
        try {  
            Calculator c = new CalculatorImpl();  
            Naming.rebind("rmi://localhost:1099/CalculatorService", c);  
        } catch (Exception e) {  
            System.out.println("Trouble: " + e);  
        }  
    }  
  
    public static void main(String args[]) {  
        new CalculatorServer();  
    }  
}
```

2 Cria referência do serviço a oferecer;

Registra (exporta) referência "c" com nome "...CalculatorService" no servidor de nomes.

# Exemplo: Cliente

```
import java.rmi.Naming;
```

```
...
```

```
public class CalculatorClient {  
    public static void main(String[] args) {  
        try {
```

```
            Calculator c = (Calculator) Naming.lookup( "rmi://localhost  
                                                    /CalculatorService");
```

```
            System.out.println( c.sub(4, 3) );
```

```
            System.out.println( c.add(4, 5) );
```

```
            System.out.println( c.mul(3, 6) );
```

```
            System.out.println( c.div(9, 3) );
```

```
        }
```

```
        catch ...
```

Procura (importa) no servidor de nomes o endereço do servidor usando o seu nome.

3

4

Obtido o endereço, o cliente está ligado (conectado) ao servidor e passa a chamar as operações do serviço.

# Objetivos de serviços de nomes

Ser **escalável**

Ter um **longo tempo de vida**

Ser altamente **disponível**

Ter **isolamento de falhas**

Tolerar **desconfiança**

# COMPUTAÇÃO DISTRIBUÍDA

CONCLUSÕES

# Características marcantes

Concorrência de componentes – possibilidade de  
paralelismo

Compartilhamento de recursos

Componentes falham de forma independente –  
falha parcial

# Próximas Datas

2º. EE: 10/12

Revisão de notas: 14/12

FINAL: 17/12