Programação Concorrente e Distribuída

Nelson Rosa – nsr@cin.ufpe.br

Objetivos

- **Introduzir os conceitos básicos de ...**
  - Programação Concorrente
  - **Programação Distribuída (ênfase da disciplina)**

- **Como alcançar estes objetivos?**
  - Estudar os conceitos básicos de programação **concorrente** e programação **distribuída**
  - Estudar **padrões de projeto** de middleware
  - Acompanhar passo-a-passo a construção de um middleware em Java
  - Entender os modelos de middleware existentes

Sobre a Disciplina

- **Requisito**
  - Infra-estrutura de Software

- **Site**
  - https://sites.google.com/a/cin.ufpe.br/if711
  - Todas as informações da disciplina estão disponíveis no site: material bibliográfico, slides, avisos, e assim por diante
  - Todo o material da disciplina é acessível apenas através da Intranet do CIn

Sobre a Disciplina

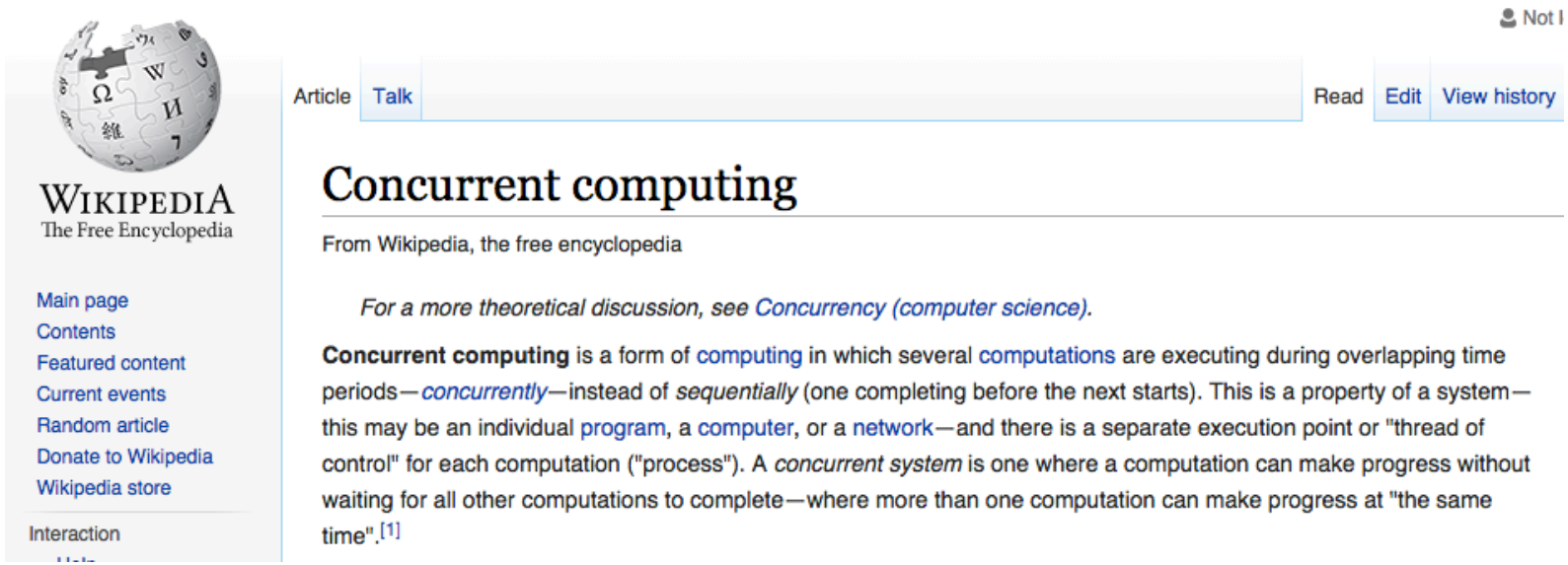
- **Avaliação da Disciplina**
  - Apresentações (50%)
    - Realizadas semanalmente sobre um tópico definido
  - Projeto (50%)
    - alinhamento com os conceitos da disciplina, complexidade, qualidade da apresentação, inovação

- **Projeto**
  - Equipe de XX (TBD) integrantes (no máximo)
  - Projeto e Implementação de um Middleware

Sobre o que “é” e “não é” esta disciplina

- **Não é ...**
  - disciplina de Redes de Computadores
  - disciplina de Engenharia de Software
- **É...**
  - disciplina de Sistemas Distribuídos
- **Será...**
  - ... quase totalmente prática

Do que iremos “falar”?



Not l

WIKIPEDIA
The Free Encyclopedia

Article Talk Read Edit View history

Concurrent computing

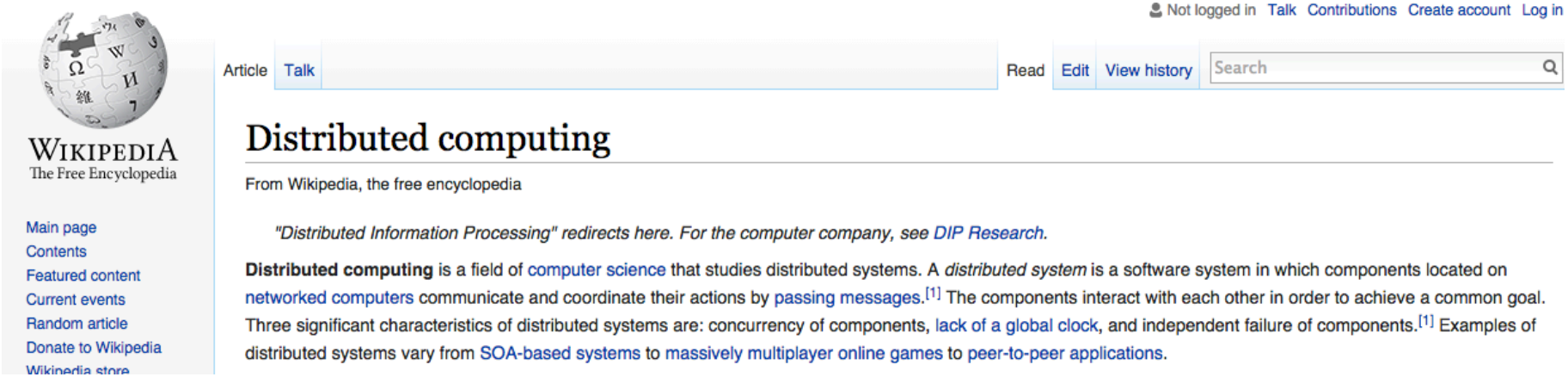
From Wikipedia, the free encyclopedia

For a more theoretical discussion, see Concurrency (computer science).

Concurrent computing is a form of computing in which several computations are executing during overlapping time periods—*concurrently*—instead of *sequentially* (one completing before the next starts). This is a property of a system—this may be an individual program, a computer, or a network—and there is a separate execution point or "thread of control" for each computation ("process"). A *concurrent system* is one where a computation can make progress without waiting for all other computations to complete—where more than one computation can make progress at "the same time".[1]

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction



Not logged in Talk Contributions Create account Log in

WIKIPEDIA
The Free Encyclopedia

Article Talk Read Edit View history Search

Distributed computing

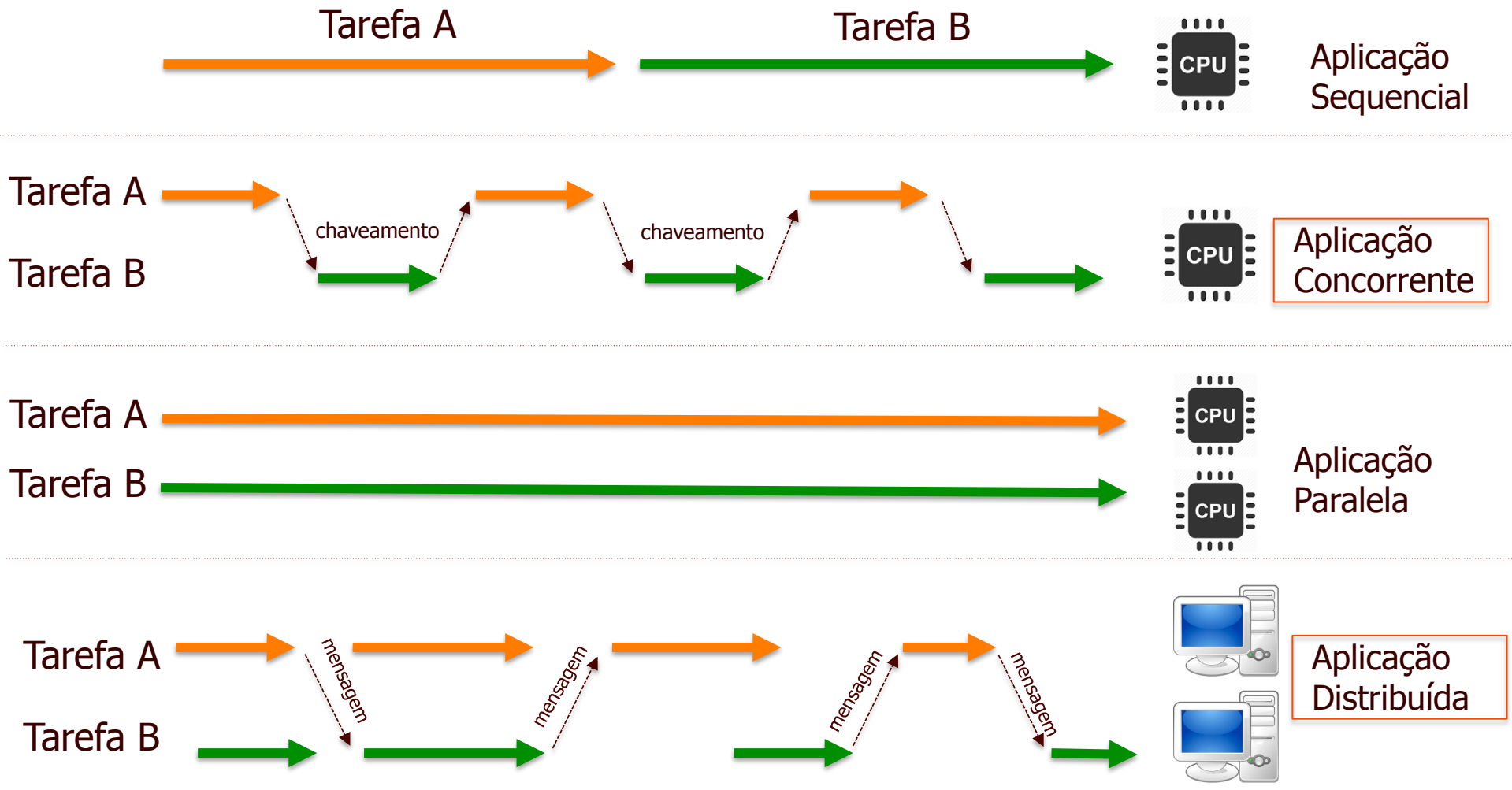
From Wikipedia, the free encyclopedia

"Distributed Information Processing" redirects here. For the computer company, see DIP Research.

Distributed computing is a field of computer science that studies distributed systems. A *distributed system* is a software system in which components located on networked computers communicate and coordinate their actions by passing messages.[1] The components interact with each other in order to achieve a common goal. Three significant characteristics of distributed systems are: concurrency of components, lack of a global clock, and independent failure of components.[1] Examples of distributed systems vary from SOA-based systems to massively multiplayer online games to peer-to-peer applications.

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Do que iremos “falar”?



Do que iremos “falar”?

Not logged in Talk Contributions Create

WIKIPEDIA
The Free Encyclopedia

Article Talk | Read Edit View history | Search

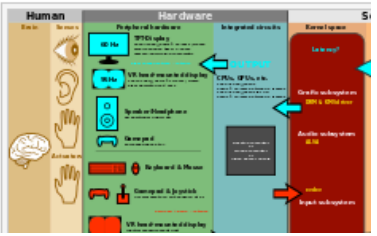
Middleware

From Wikipedia, the free encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

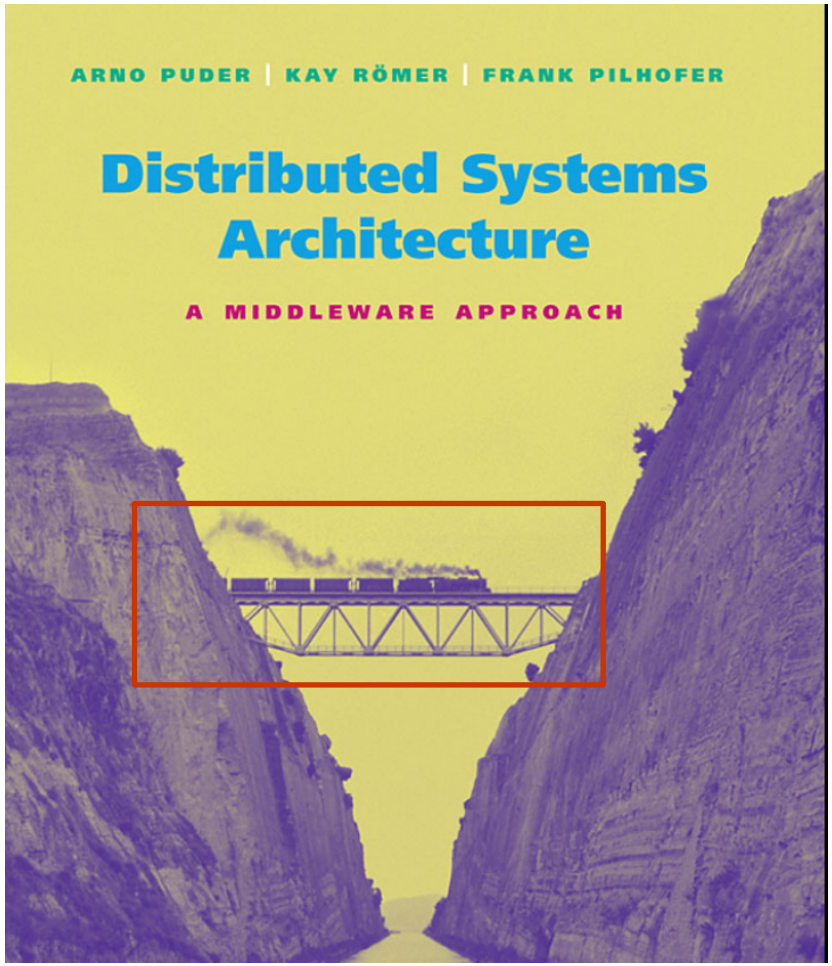
Interaction
Help
About Wikipedia
Community portal

This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed. *(August 2013)*



Middleware is computer software that provides services to software applications beyond those available from the operating system. It can be described as "software glue".[1] Middleware makes it easier for software developers to perform communication and input/output, so they can focus on the specific purpose of their application. Middleware is the software that connects software components or enterprise applications. Middleware is the software layer that lies between the operating system and the applications on each side of a distributed computer network. Typically, it supports complex, distributed business software applications.

Faces do Middleware



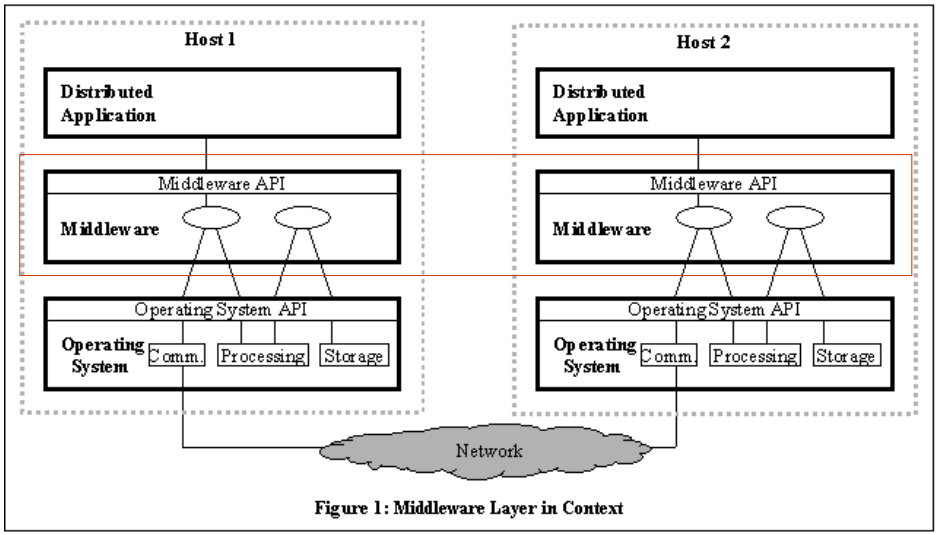
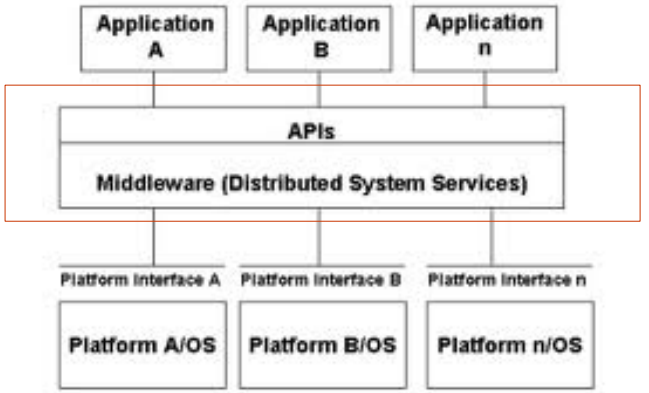
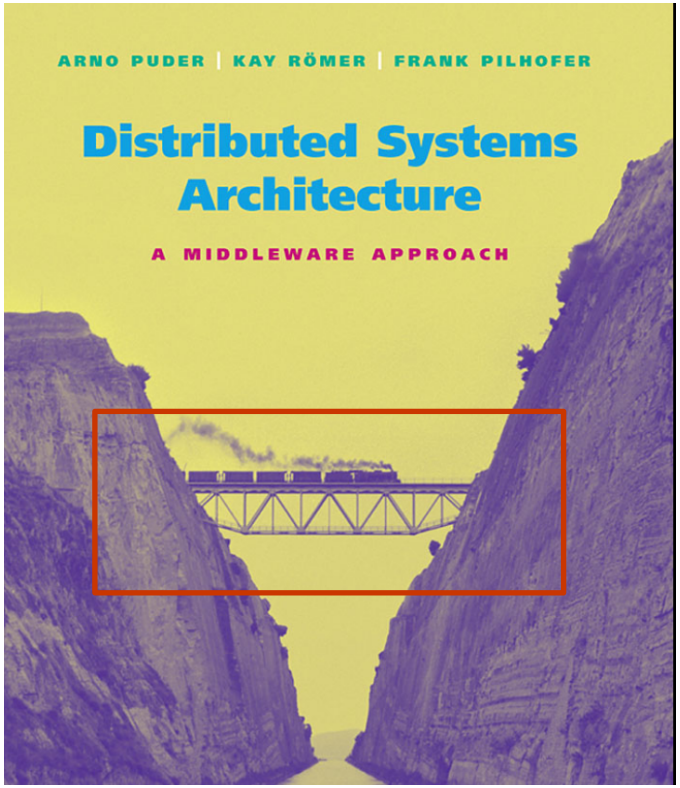
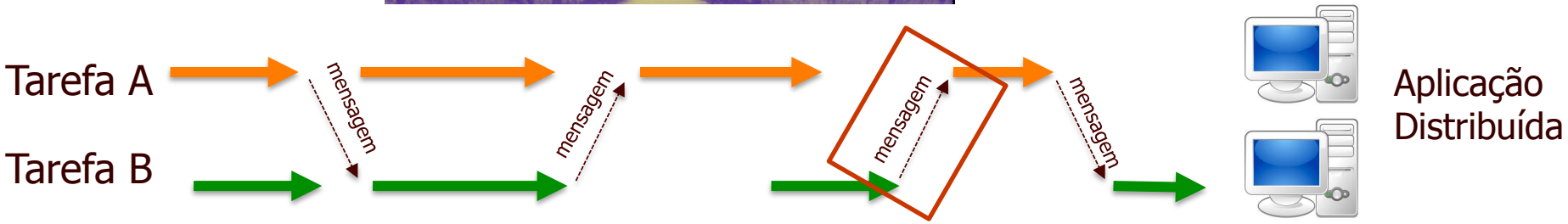


Figure 1: Middleware Layer in Context



Faces do Middleware



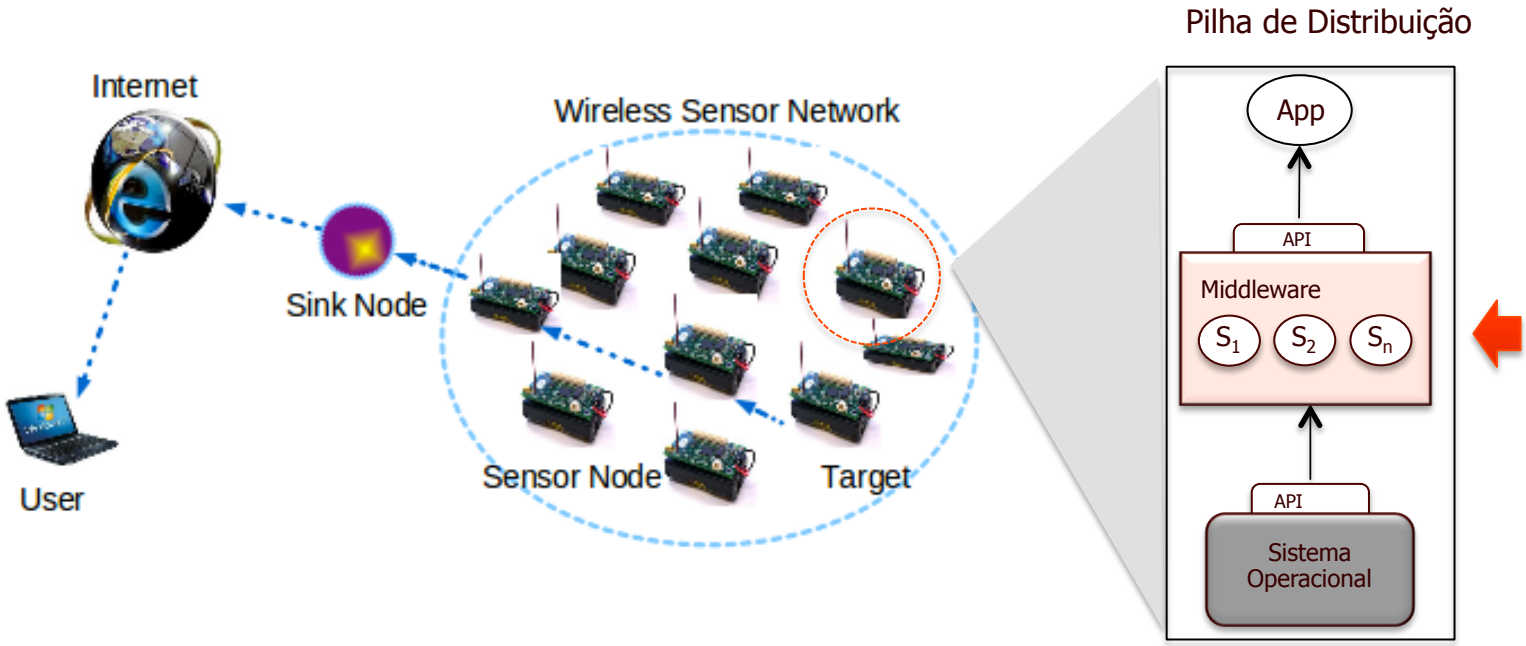


Faces do Middleware

Middleware para **Desktop**

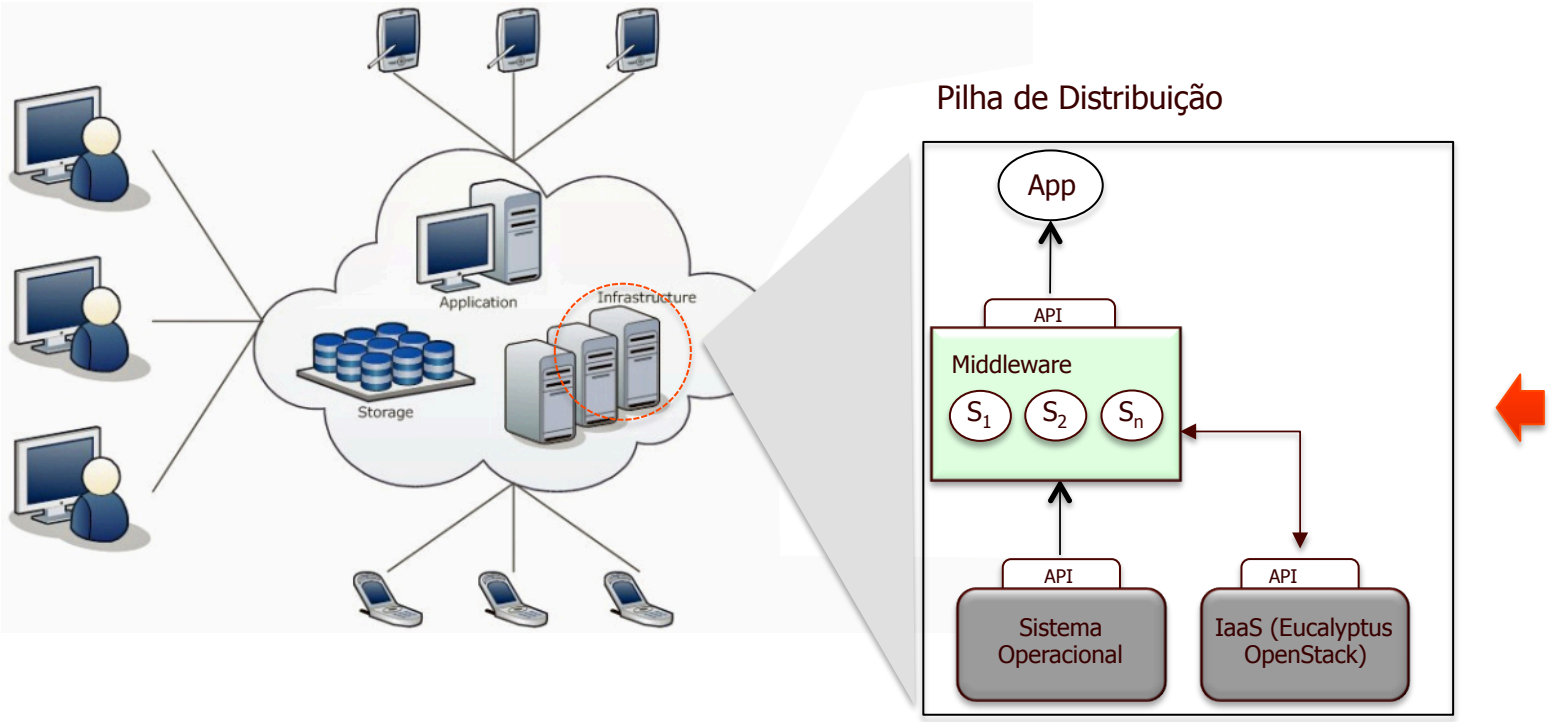
Faces do Middleware

Middleware para **Rede de Sensores**



Faces do Middleware

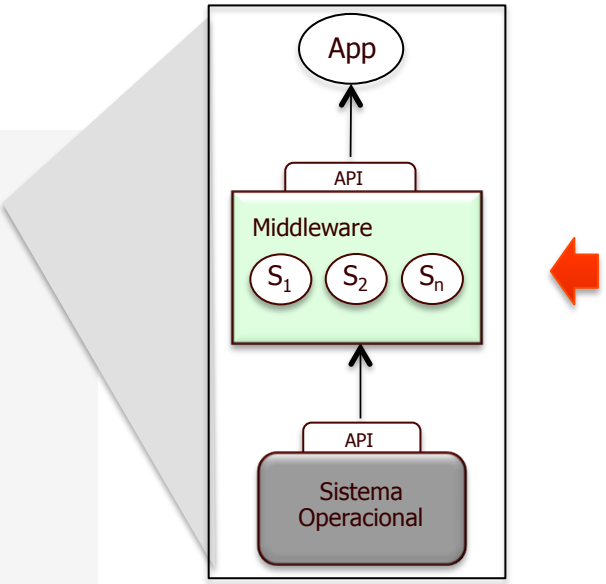
Middleware para **Nuvens Computacionais**



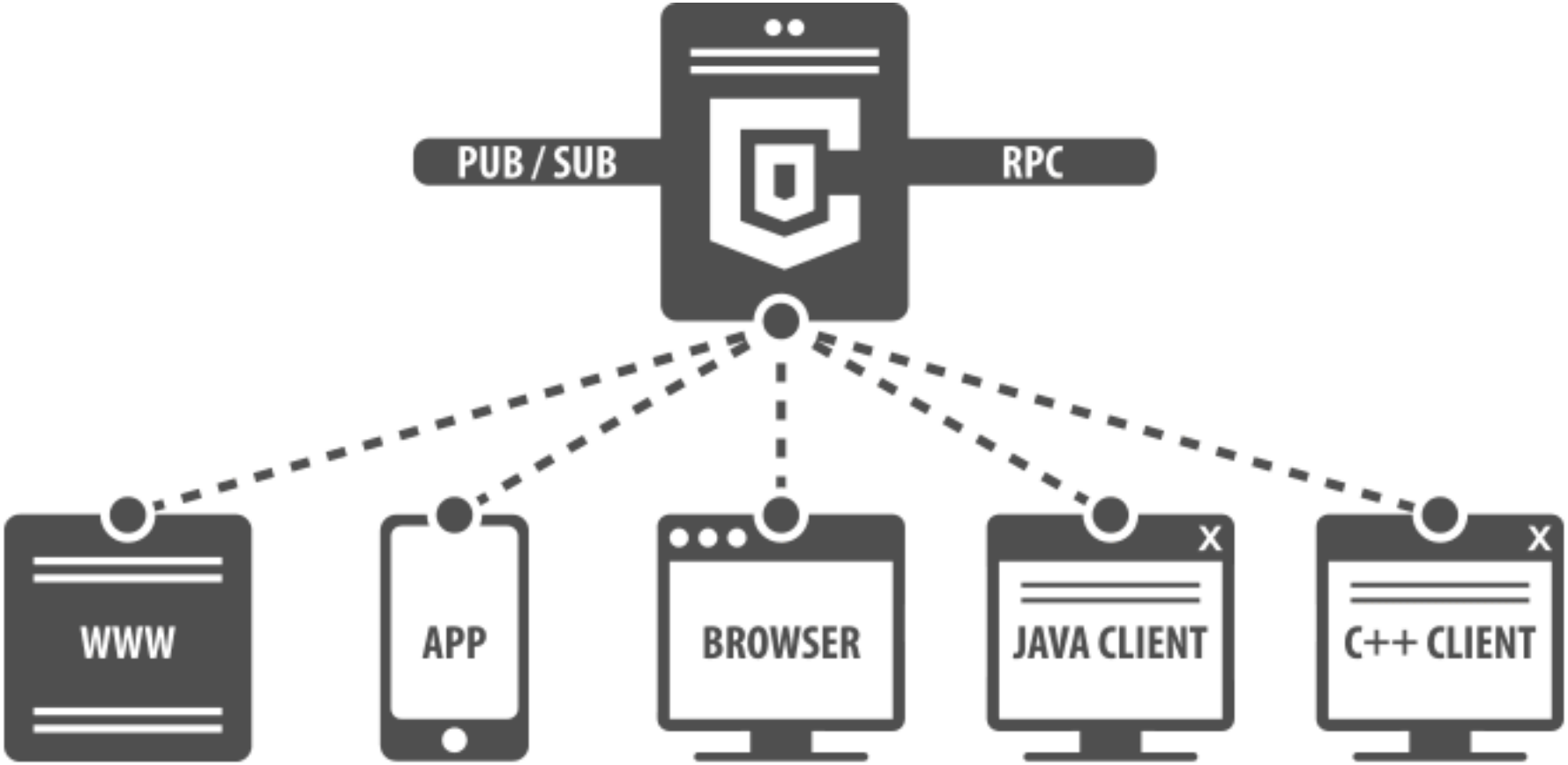
Faces do Middleware

Middleware para **IoT**

Pilha de Distribuição

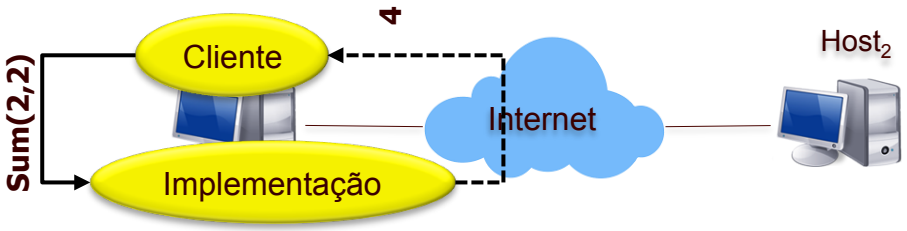


Faces do Middleware



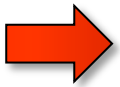
Exemplo:: Calculadora

- **Calculadora**
  - Interface
    - 4 operações
    - sum,sub,div,mul
  - Cliente
    - Usa as operações da interface
  - Implementação
    - Implementa a interface



Exemplo:: Calculadora:: Centralizada

```
public interface ICalculator {

   float sum (float a, float b);
   float sub (float a, float b);
   float div (float a, float b);
   float mul (float a, float b);
}
```



Host 1

```
public class CalculatorImpl
    implements ICalculator {


  public float sum(float a, float b) {
    return a+b;
  }

   public float sub(float a, float b) {
     return a-b;
   }

   public float div(float a, float b) {
     return a/b;
   }

   public float mul(float a, float b) {
     return a * b;
   }
}
```

Host 1

```
public class CalculatorClient {

  public static void main(String[] args) {

    CalculatorImpl calculator = new CalculatorImpl();

    calculator.sum(1,3);
  }
}
```

Exemplo:: Calculadora:: Centralizada

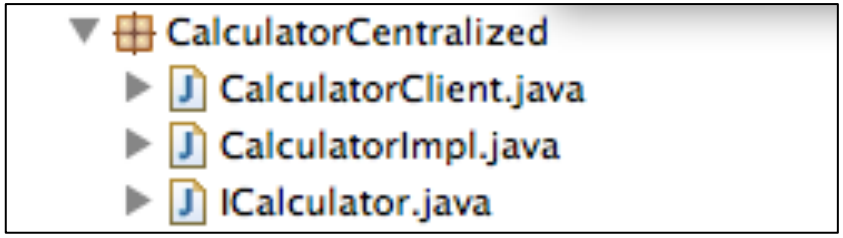
Host 1

```
public class CalculatorClient {
}
```

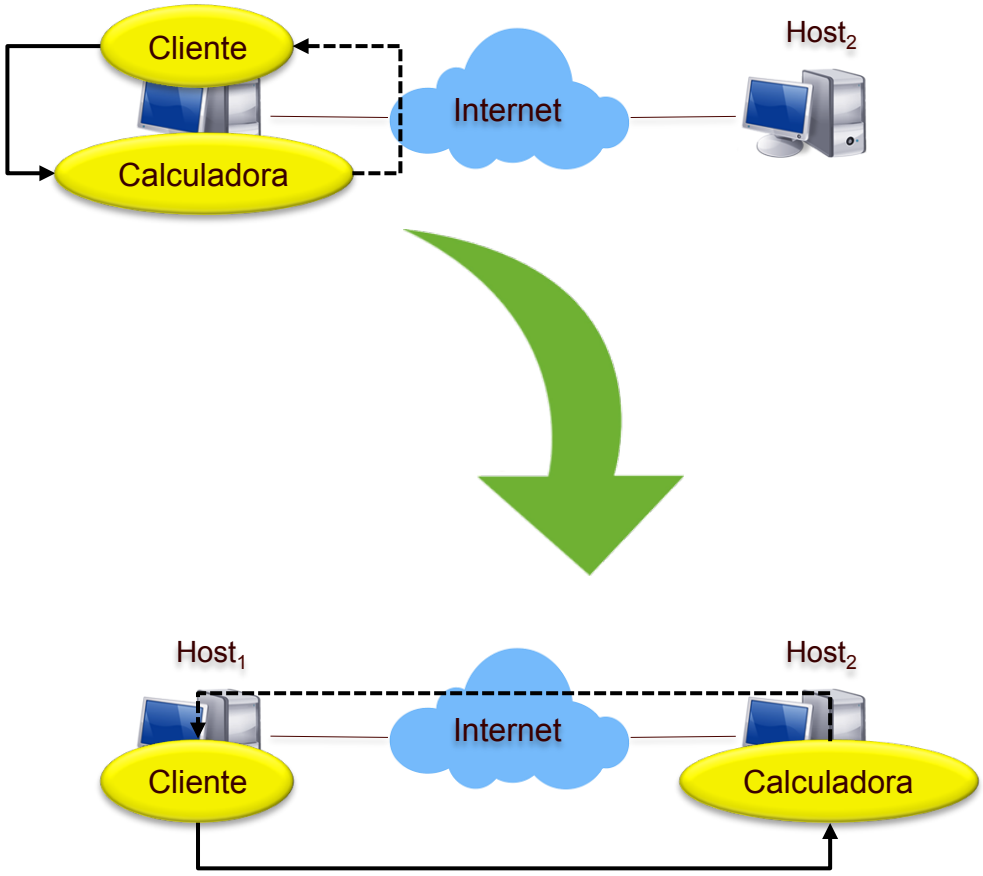
Host 1

```
public class CalculatorImpl
    implements ICalculator {
}
```

JVM

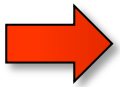


Exemplo:: Calculadora:: Distribuída



Exemplo:: Calculadora:: Distribuída

```
public interface ICalculator {

   float sum (float a, float b);
   float sub (float a, float b);
   float div (float a, float b);
   float mul (float a, float b);
}
```



Host 2

```
public class CalculatorImpl
    implements ICalculator {


  public float sum(float a, float b) {
    return a+b;
  }

   public float sub(float a, float b) {
     return a-b;
   }

   public float div(float a, float b) {
     return a/b;
   }

   public float mul(float a, float b) {
     return a * b;
   }
}
```

Host 1

```
public class CalculatorClient {

  public static void main(String[] args) {

    CalculatorImpl calculator = new CalculatorImpl();

    calculator.sum(1,3);
  }
}
```

Exemplo:: Calculadora:: Distribuída

Como Distribuir?

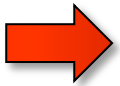
Inter Process Communication (e.g., socket)

X

Middleware

Exemplo:: Calculadora:: Distribuída:: Socket

```
public interface ICalculator {

   float sum (float a, float b);
   float sub (float a, float b);
   float div (float a, float b);
   float mul (float a, float b);
}
```



Host 1

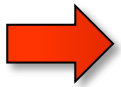
```
public class CalculatorClient {

  public static void main(String[] args) {

    CalculatorImpl calculator = new CalculatorImpl();

    calculator.sum(1,3);
    /* programação com socket */
  }
}
```

Host 2

```
public class CalculatorImpl
    implements ICalculator {

 /* programação com socket */

  public float sum(float a, float b) {
    return a+b;
  }

   public float sub(float a, float b) {
     return a-b;
   }

   public float div(float a, float b) {
     return a/b;
   }

   public float mul(float a, float b) {
     return a * b;
   }
}
```

Exemplo:: Calculadora:: Distribuída:: Middleware

```
public interface ICalculator {

   float sum (float a, float b);
   float sub (float a, float b);
   float div (float a, float b);
   float mul (float a, float b);
}
```



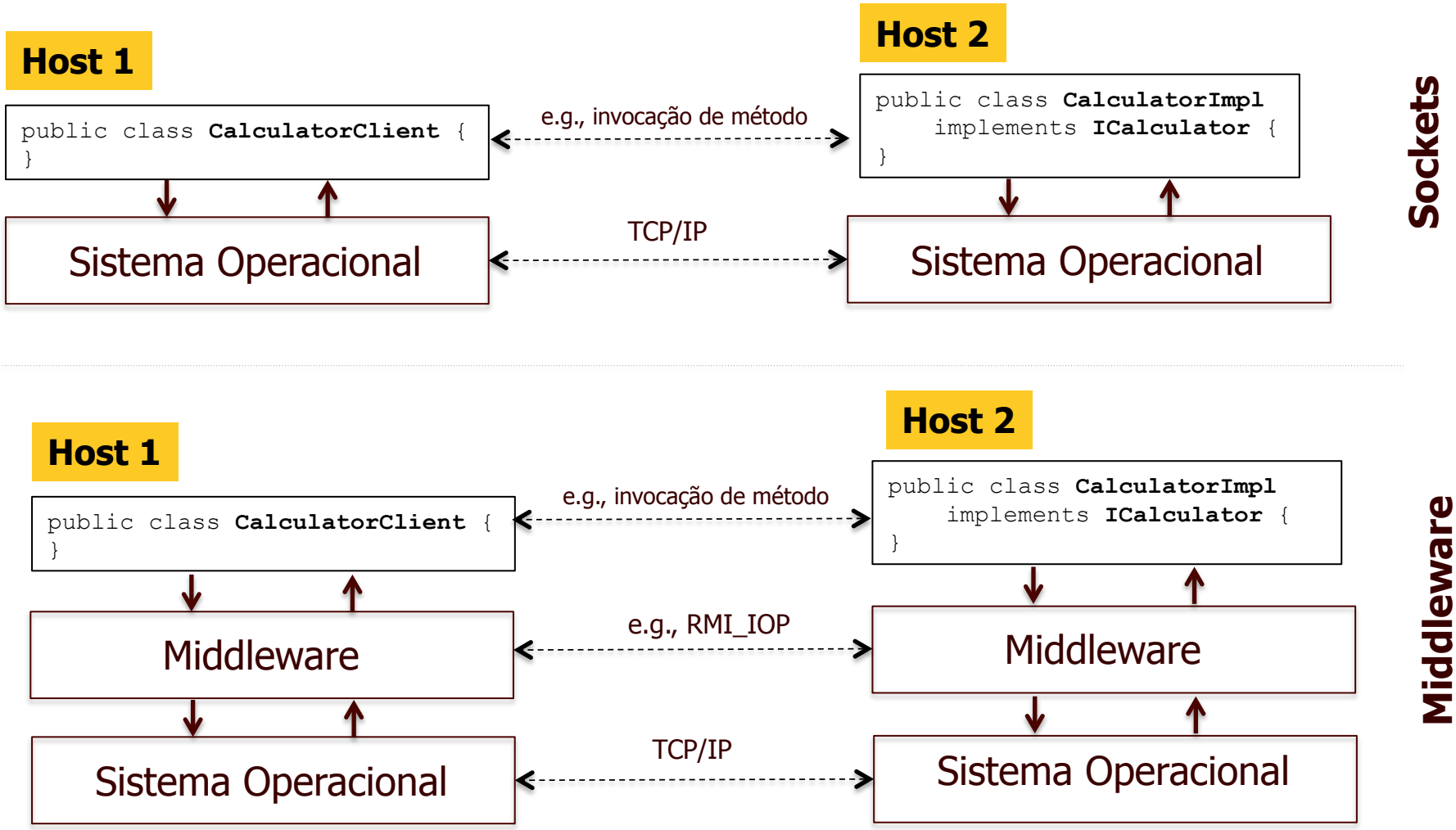
Host 1

```
public class CalculatorClient {

  public static void main(String[] args) {

    CalculatorImpl calculator = new CalculatorImpl();

    calculator.sum(1,3);

  }
}
```

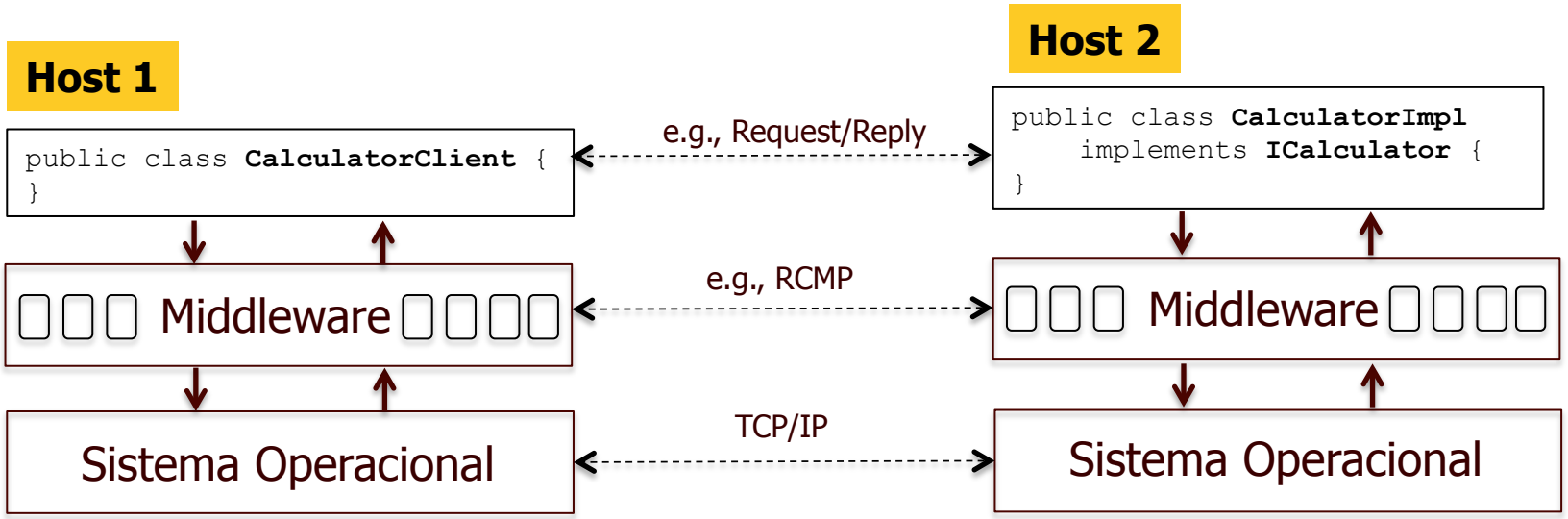
Host 2

```
public class CalculatorImpl
    implements ICalculator {


  public float sum(float a, float b) {
    return a+b;
  }

   public float sub(float a, float b) {
     return a-b;
   }

   public float div(float a, float b) {
     return a/b;
   }

   public float mul(float a, float b) {
     return a * b;
   }
}
```

Exemplo:: Calculadora:: Distribuída



Exemplo:: Calculadora:: Distribuída:: Middleware



Transparências "Comuns":

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

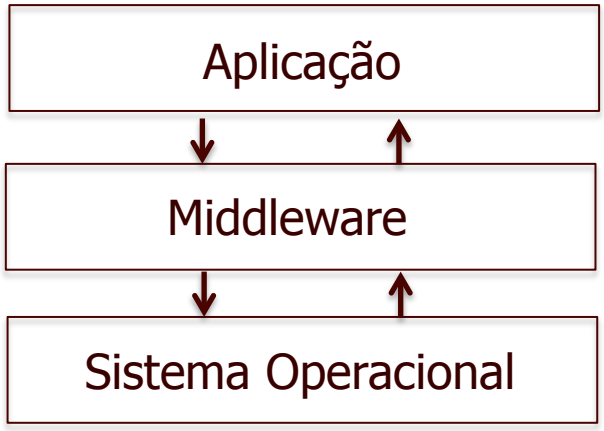
Transparências (Coulouris):

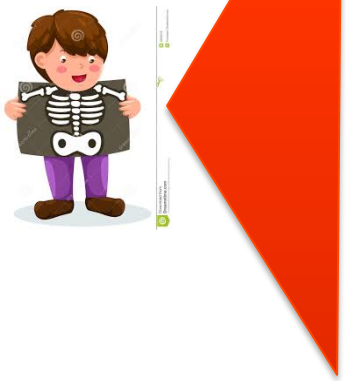
- ✓ Localização
- ✓ Acesso
- ✓ Falha
- ✓ Tecnologia
- ✓ Concorrência
- ✓ Mobilidade
- ✓ Desempenho
- ✓ Escala

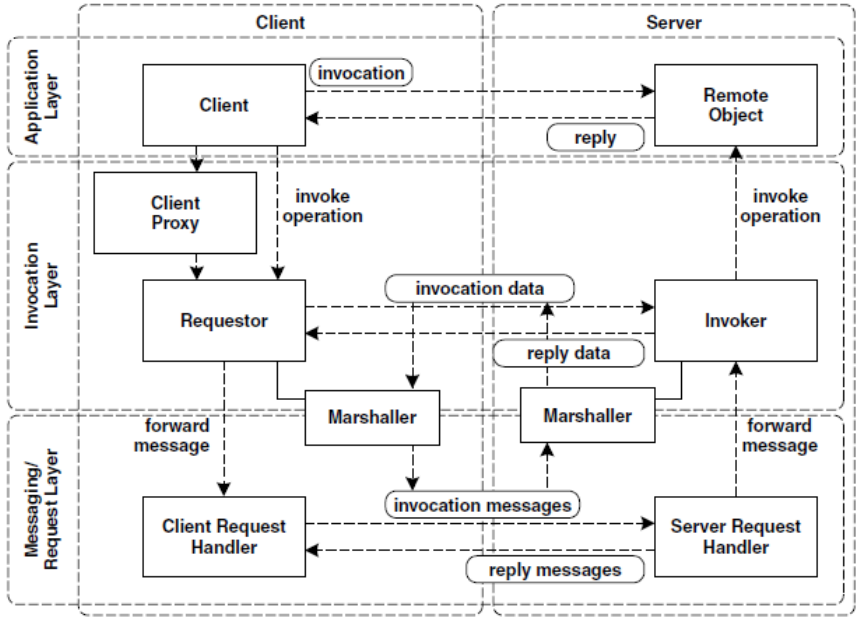
Transparências (RM-ODP):

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

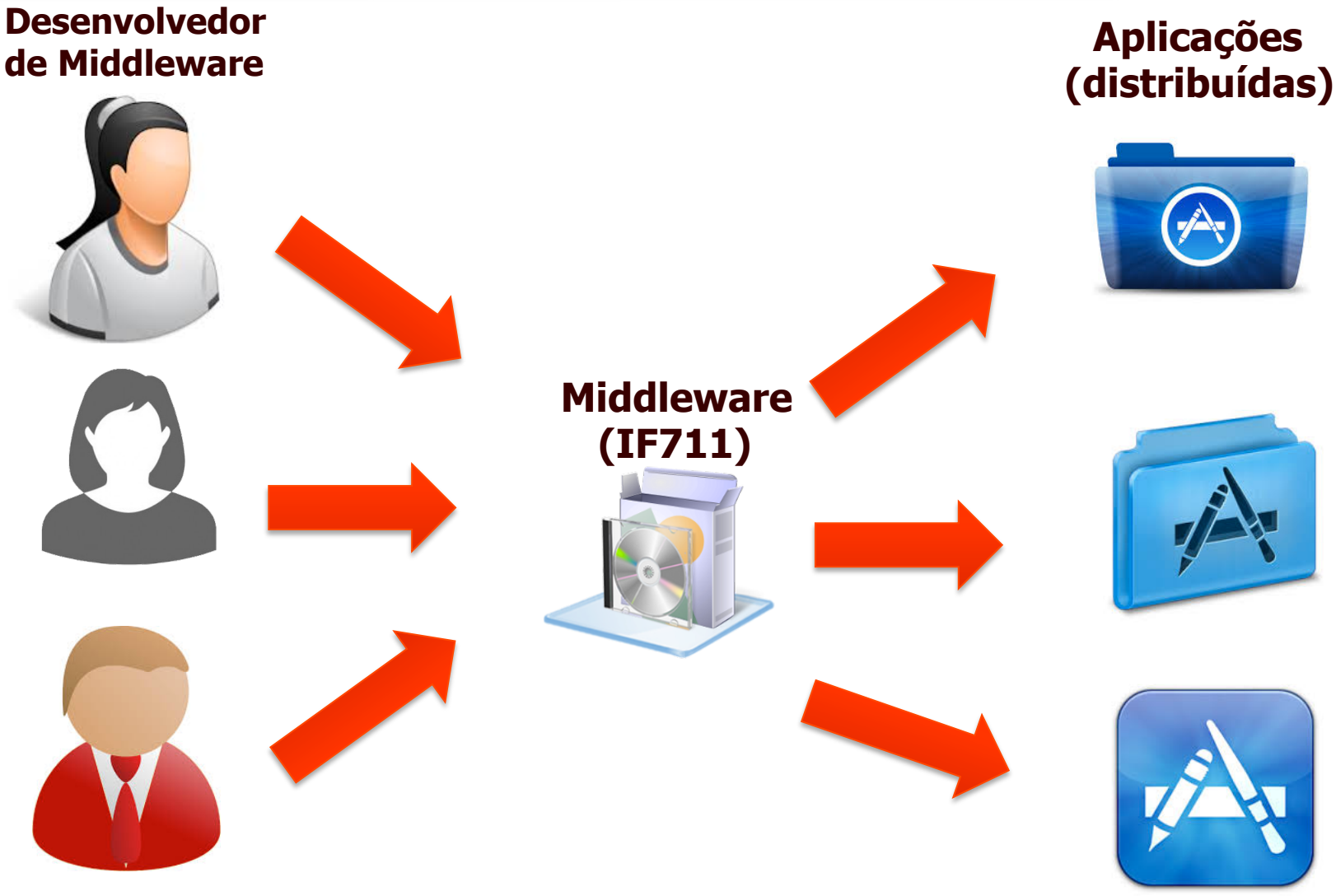
Middleware


Aplicação
Middleware
Sistema Operacional




Client
Server
Application Layer
Invocation Layer
Messaging/ Request Layer
Client
invocation
reply
Remote Object
Client Proxy
invoke operation
invoke operation
Requestor
invocation data
reply data
Invoker
Marshaller
Marshaller
forward message
forward message
Client Request Handler
invocation messages
reply messages
Server Request Handler

Middleware


Desenvolvedor
de Middleware
Middleware
(IF711)
Aplicações
(distribuídas)

Fim dos Slides