Modelos de Middleware

Nelson Rosa – nsr@cin.ufpe.br

Objetivos da Aula

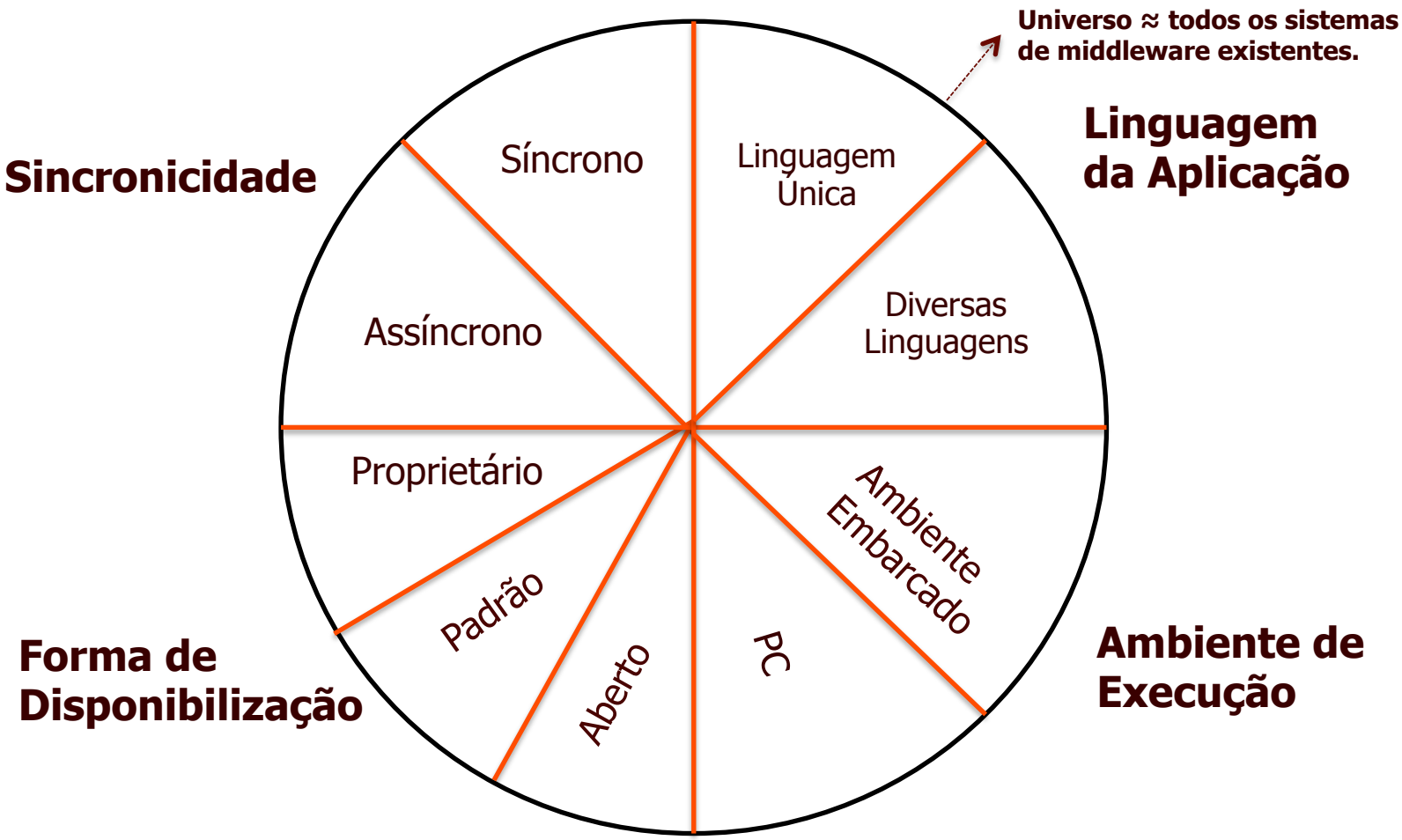
- **Apresentar algumas taxonomias de middleware**
- **Detalhar uma das taxonomias**
  - Middleware Orientado a Objetos
  - Middleware Procedural
  - Middleware Transacional
  - Middleware Orientado à Mensagem
- **Discutir as diferenças entre os Modelos de Middleware**
- **Apresentar o passo-a-passo de uma aplicação construída com o RMI (MOO) e RabbitMQ (MOM)**

Modelos de Middleware:: Fatos Básicos

- Há várias modelos e taxonomias de middleware
- Baseadas em diversas características do middleware
- Não há uma taxonomia mais usada, nem amplamente adotada
- Cada classificação adota critérios diferentes para agrupar os sistemas de middleware

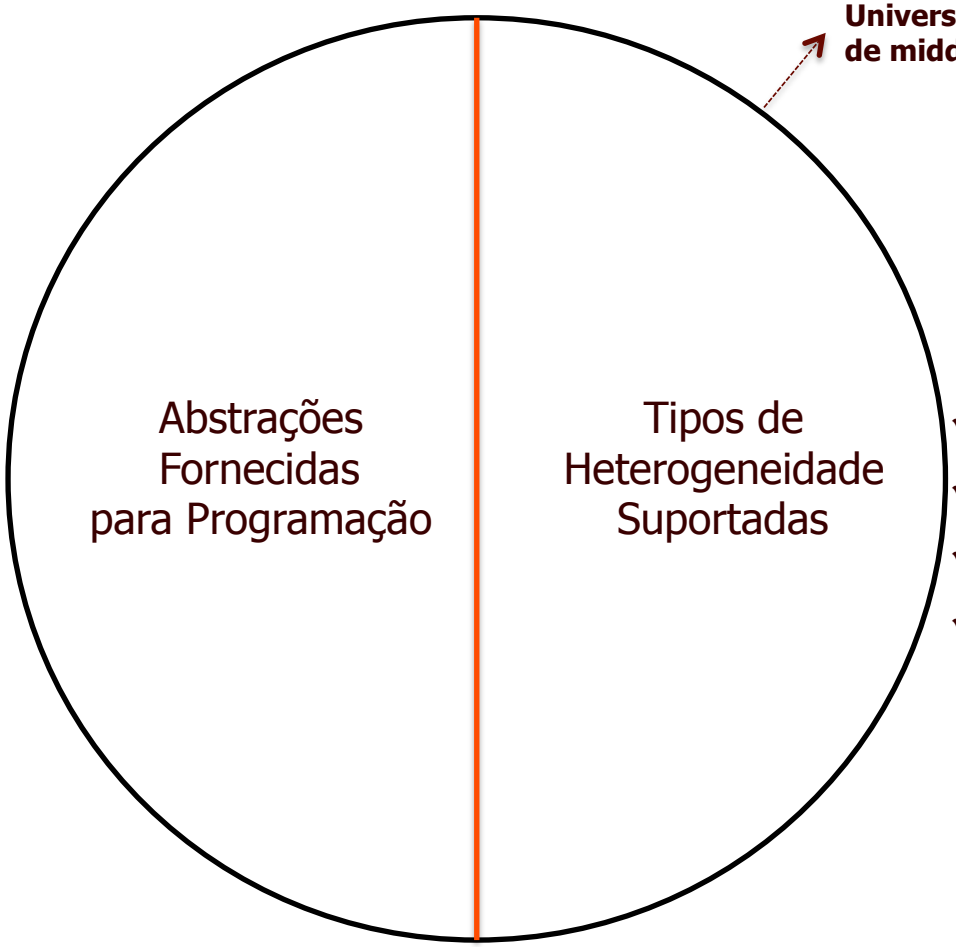
Classificação 1

Critérios: Sincronicidade,Linguagem da aplicação, Ambiente de Execução, Forma de disponibilização



Classificação 2

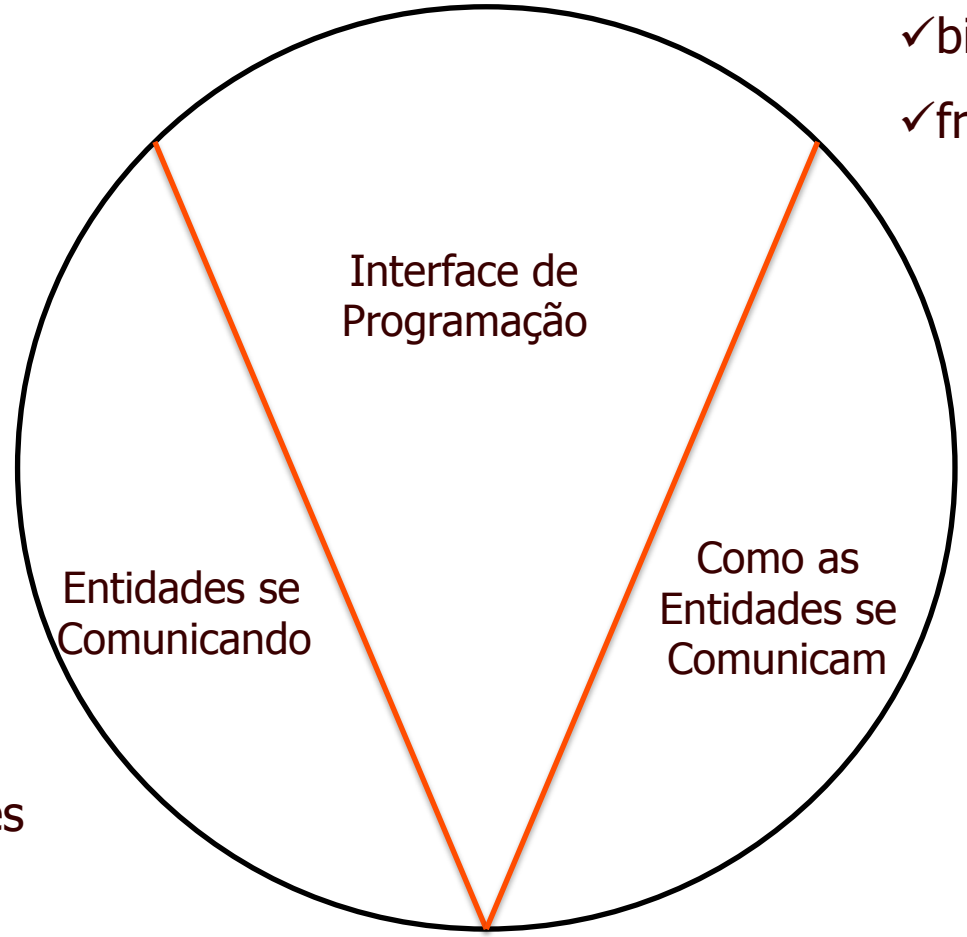
Classificação 3



- ✓Tuplas distribuídas
- ✓Procedimentos
- ✓ Fila de mensagem
- ✓ Objetos distribuídos

- ✓sistema operacional
- ✓linguagem de programação
- ✓ rede (cabeada, sem-fio)
- ✓ hardware

Classificação 4



- ✓APIs
- ✓bibliotecas
- ✓frameworks

- ✓processos
- ✓objetos
- ✓ programas
- ✓ componentes
- ✓ serviços

- ✓request/reply (cliente-servidor)
- ✓peer-to-peer
- ✓push (publish/ subscribe)

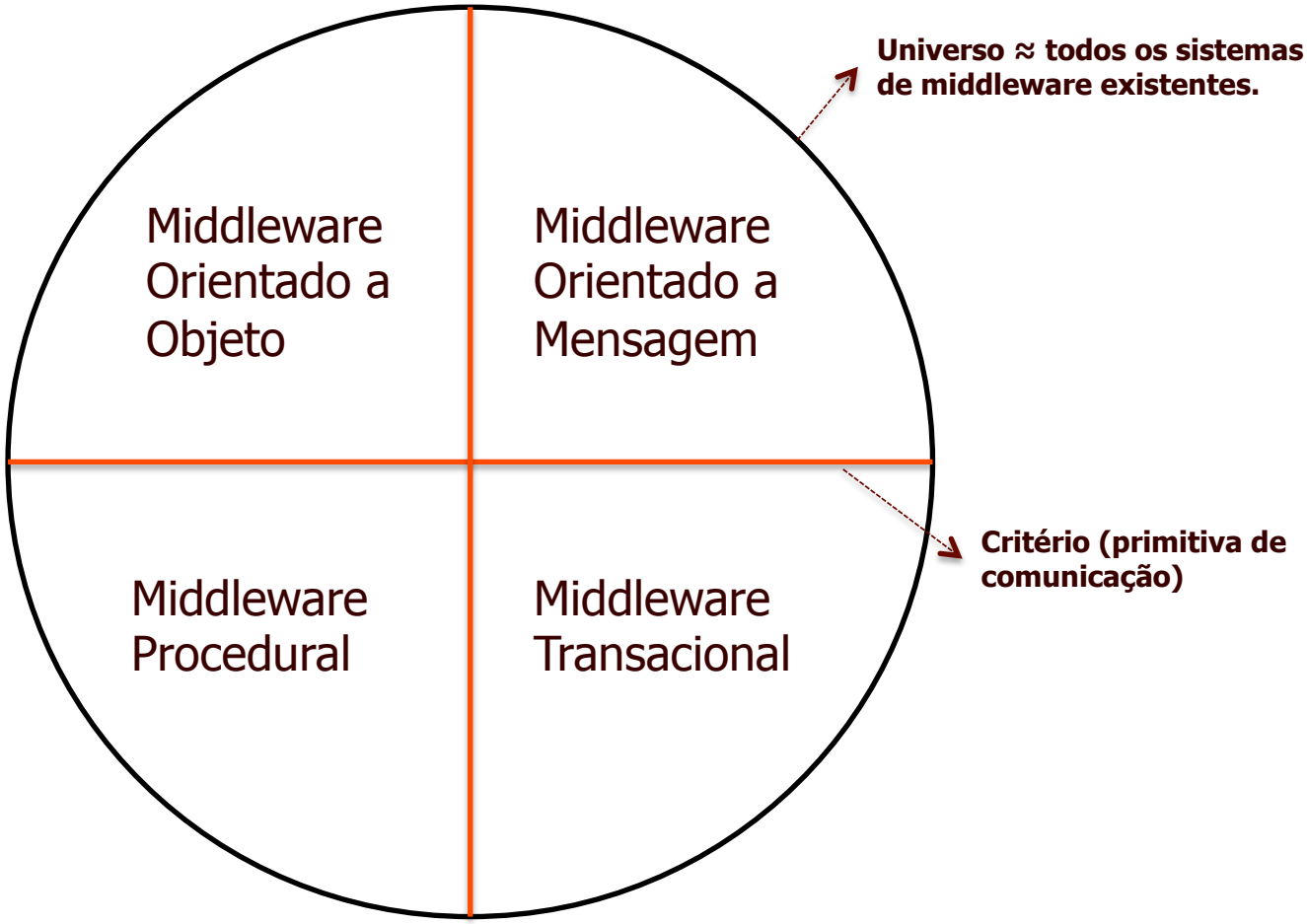
Classificação 5

✓os componentes "não sabem" da intermediação do middleware

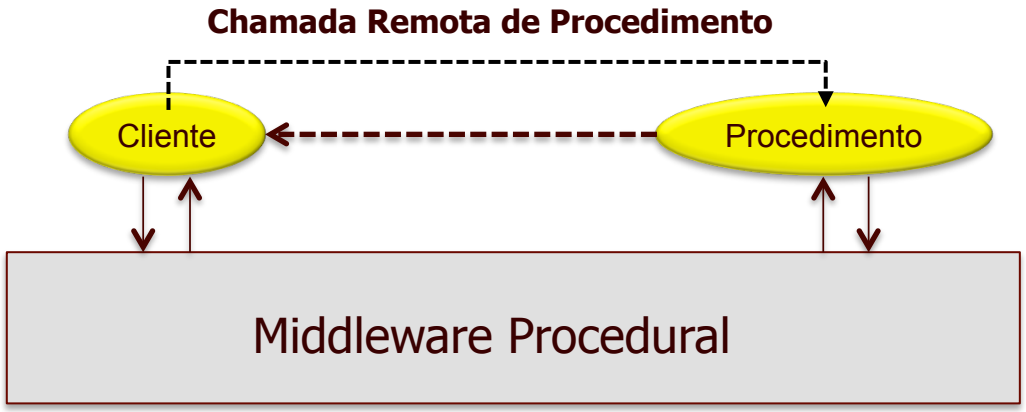
✓os componentes "sabem" da intermediação do middleware

Classificação 6

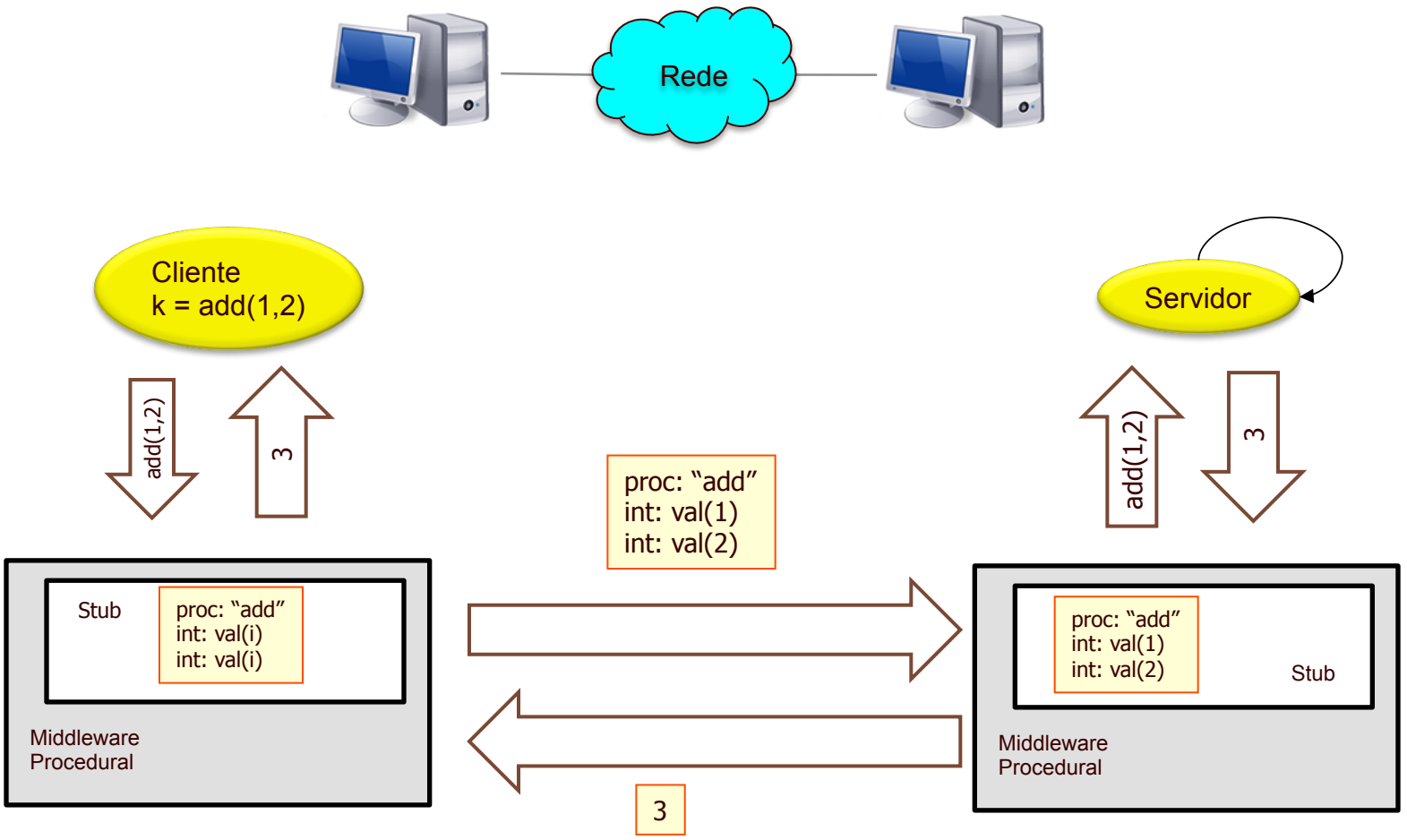
Critério: tipo de primitiva de comunicação fornecida pelo middleware para o desenvolvimento de aplicações distribuídas.



Modelos de Middleware:: Middleware Procedural



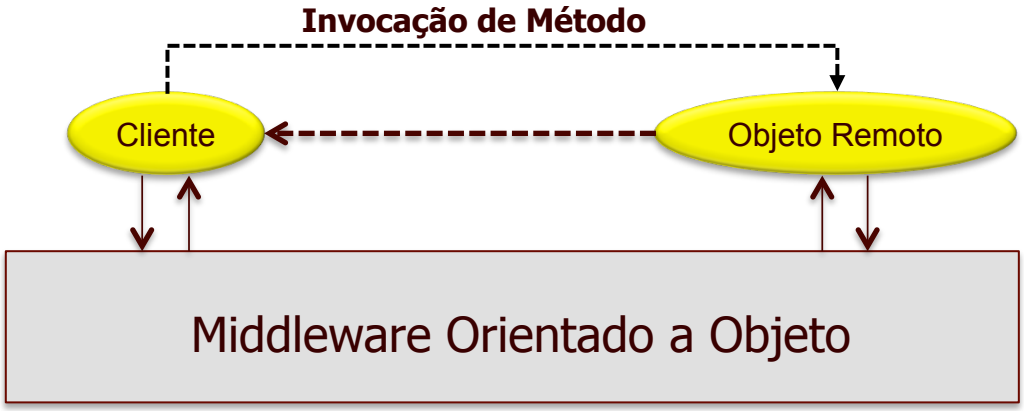
Modelos de Middleware:: Middleware Procedural



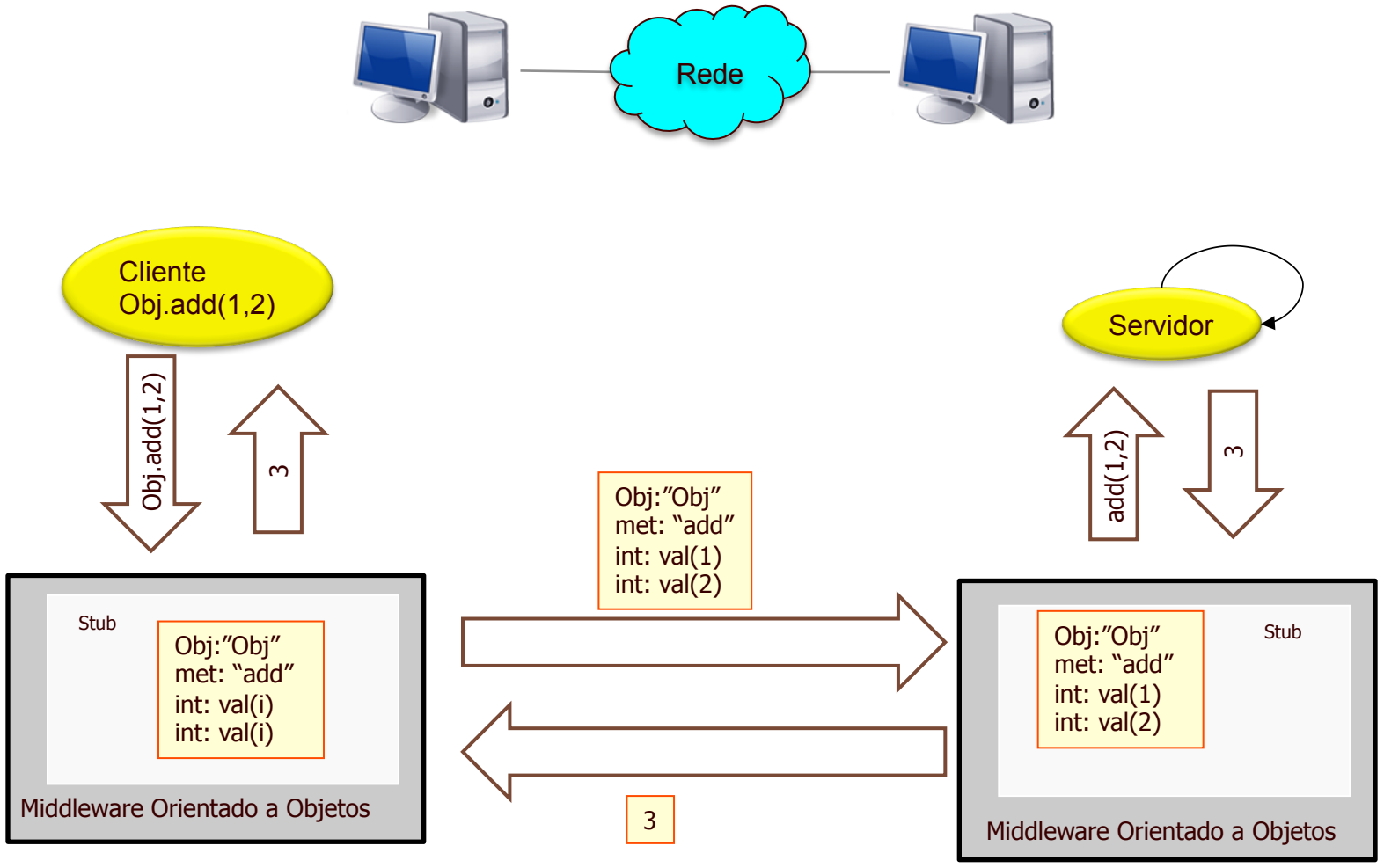
Modelos de Middleware:: Middleware Procedural

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

Modelos de Middleware:: Orientado a Objetos



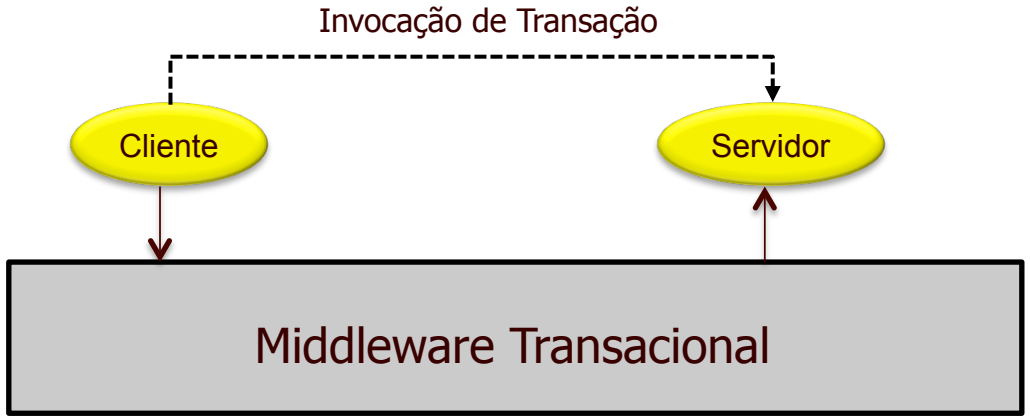
Modelos de Middleware:: Middleware Orientado a Objetos



Modelos de Middlware:: Middleware Orientado a Objetos

- **Primitiva de interação: invocação de método**
- **Comunicação 1 - 1**
- **Evolução do middleware procedural**
- **Comunicação síncrona entre objetos distribuídos suportada naturalmente**
- **Interface dos serviços descritas por linguagens específicas (IDLs)**
  - e.g., CORBA, RMI

Modelos de Middleware:: Middleware Transacional



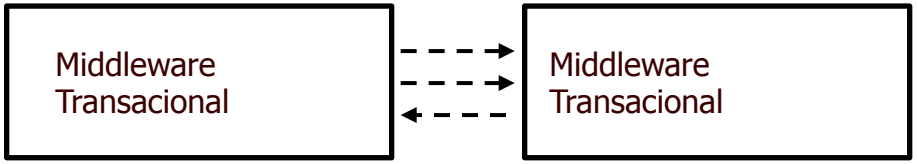
Modelos de Middleware:: Middleware Transacional

- **Primitiva de interação: transação**
  - chamada de procedimento remoto integrada + controle de transações
- **Protocolo comumente usado**
  - two-phase commit
- **Comunicação síncrona suportada naturalmente**
- **Comunicação 1-1**

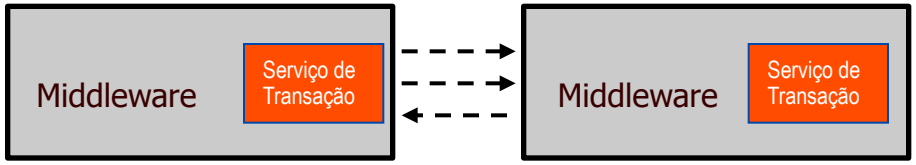
Modelos de Middleware:: Middleware Transacional

- **Fornece suporte à execução de transações de acesso à bases de dados**
- **Esconde a complexidade da implementação de transações em redes**
- **Tipicamente construído sobre outro middleware**
- **Usado em sistemas de processamento de transações de alta escala**

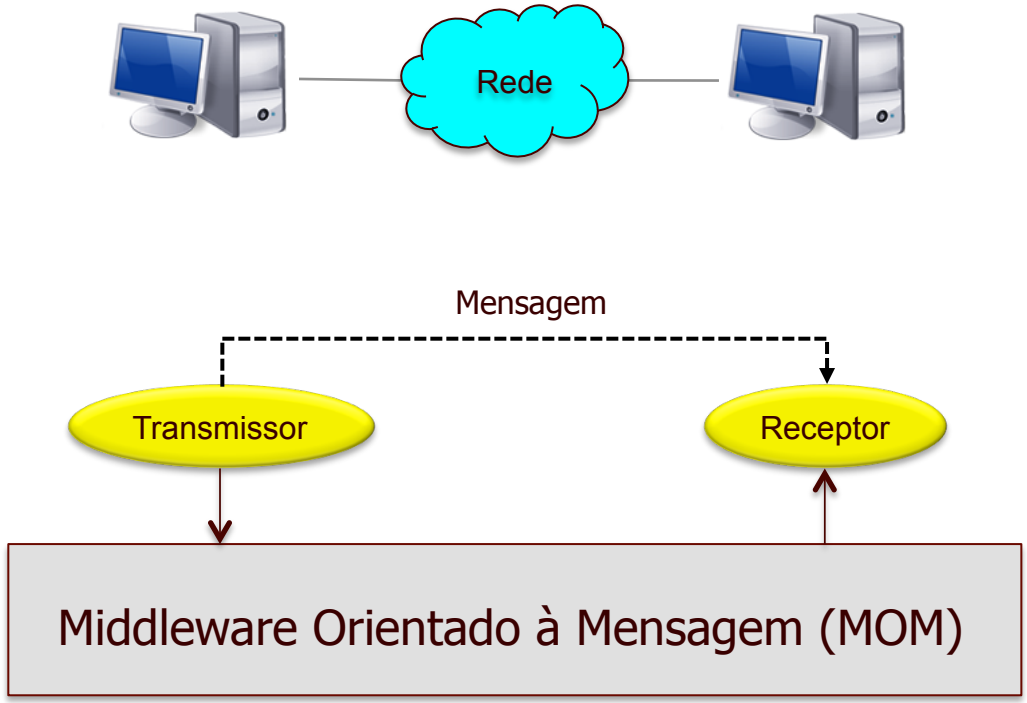
Middleware Transacional



Middleware + serviço de transação

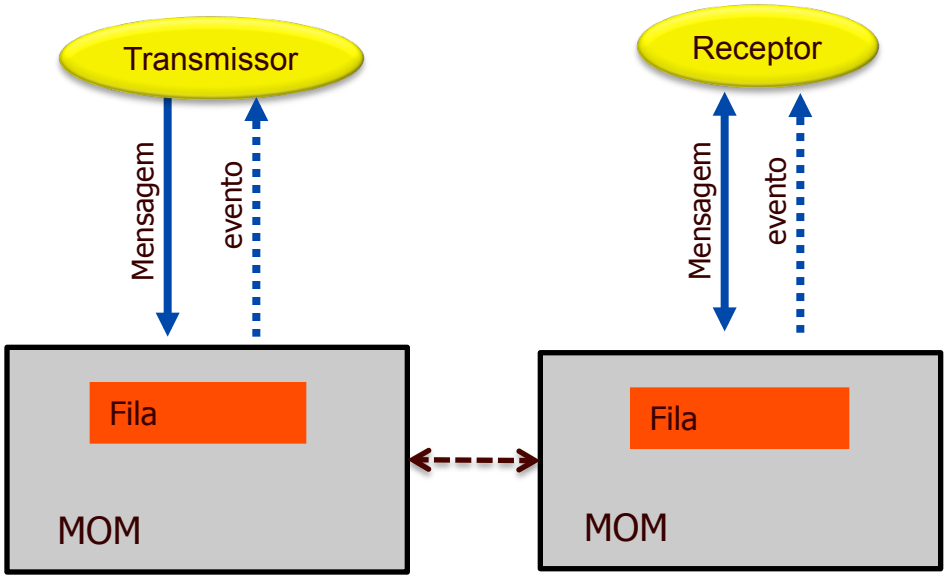


Modelos de Middleware:: Middleware Orientado a Mensagem



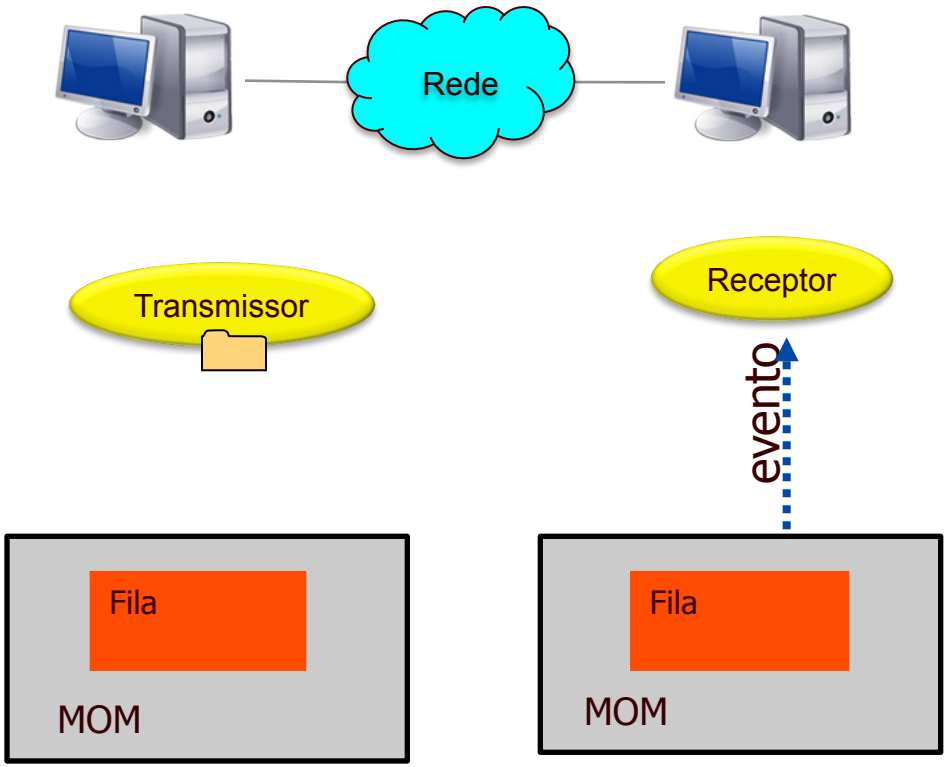
Modelos de Middleware:: Middleware Orientado a Mensagem

- **Primitiva de interação: passagem de mensagem**
- **Comunicação assíncrona/em grupo naturalmente implementadas**
- **Uso de fila (temporárias, persistentes) de mensagens**
- **Sub-tipos:**
  - fila de mensagem
  - publish/subscribe
- **e.g., MQSeries, Sun ONE Message Queue (free), JORAM (free)**
- **Comunicação: 1-1, 1-N**

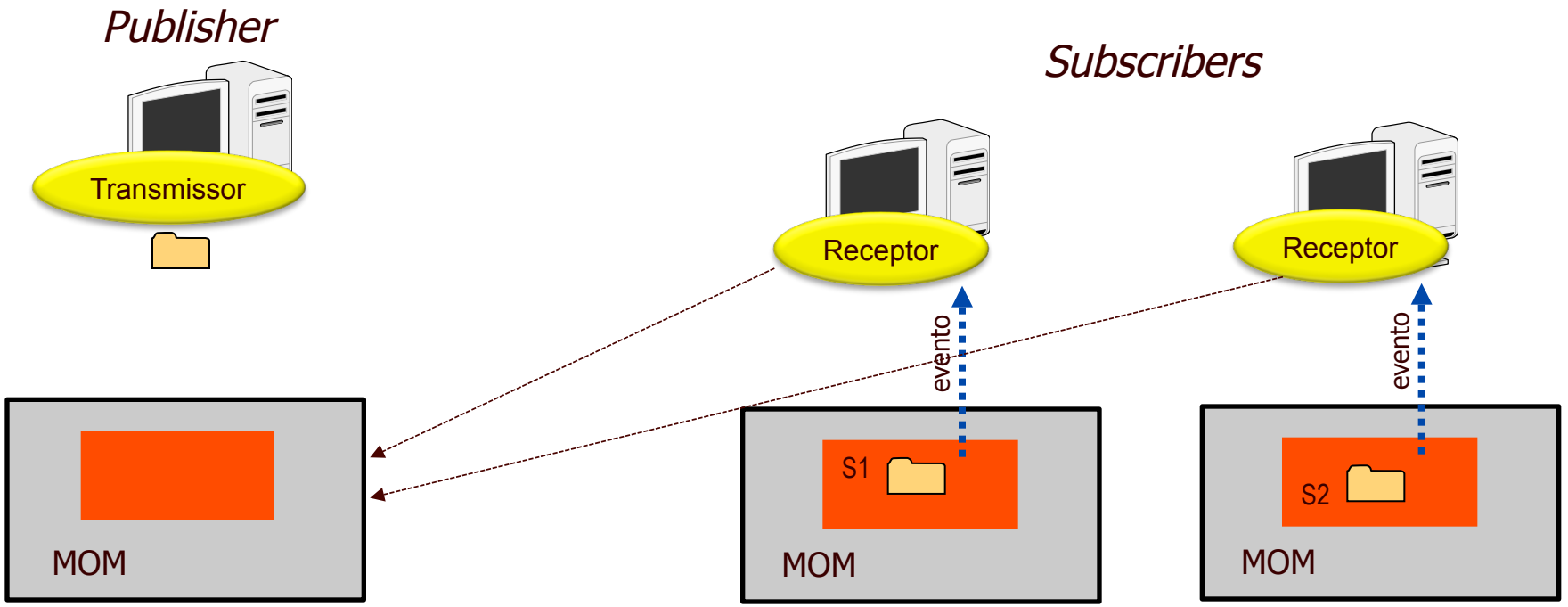


Modelos de Middleware:: Middleware Orientado a Mensagem

Fila de Mensagem

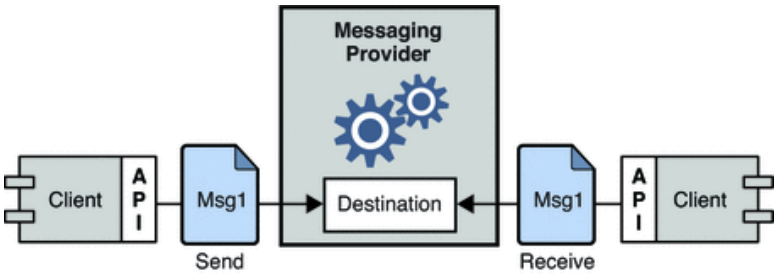


Modelos de Middleware:: Middleware Orientado a Mensagem

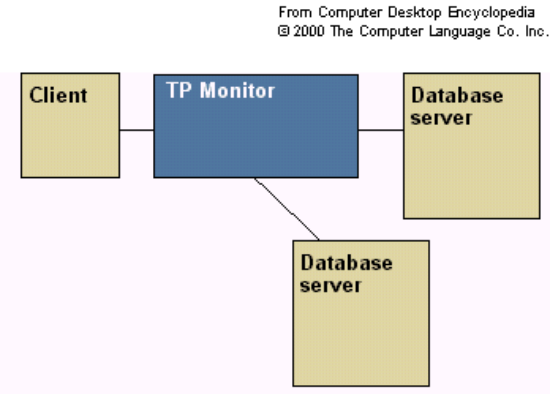


Modelos de Middleware:: Arquiteturas

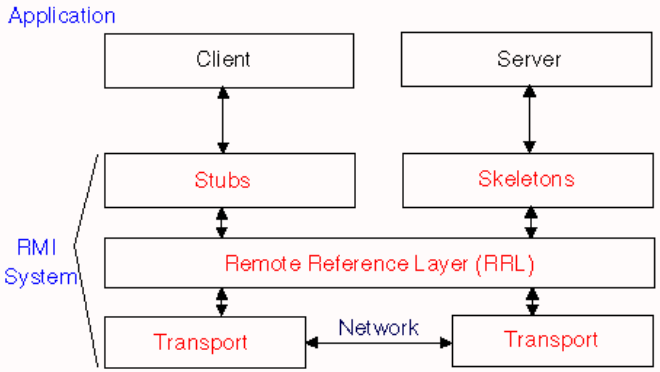
Middleware Orientado a Mensagem



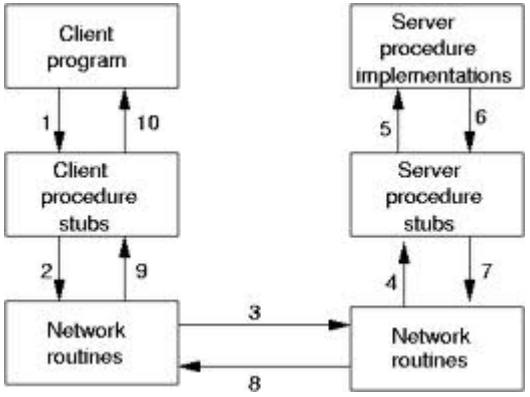
Middleware Transacional



Middleware Orientado a Objetos

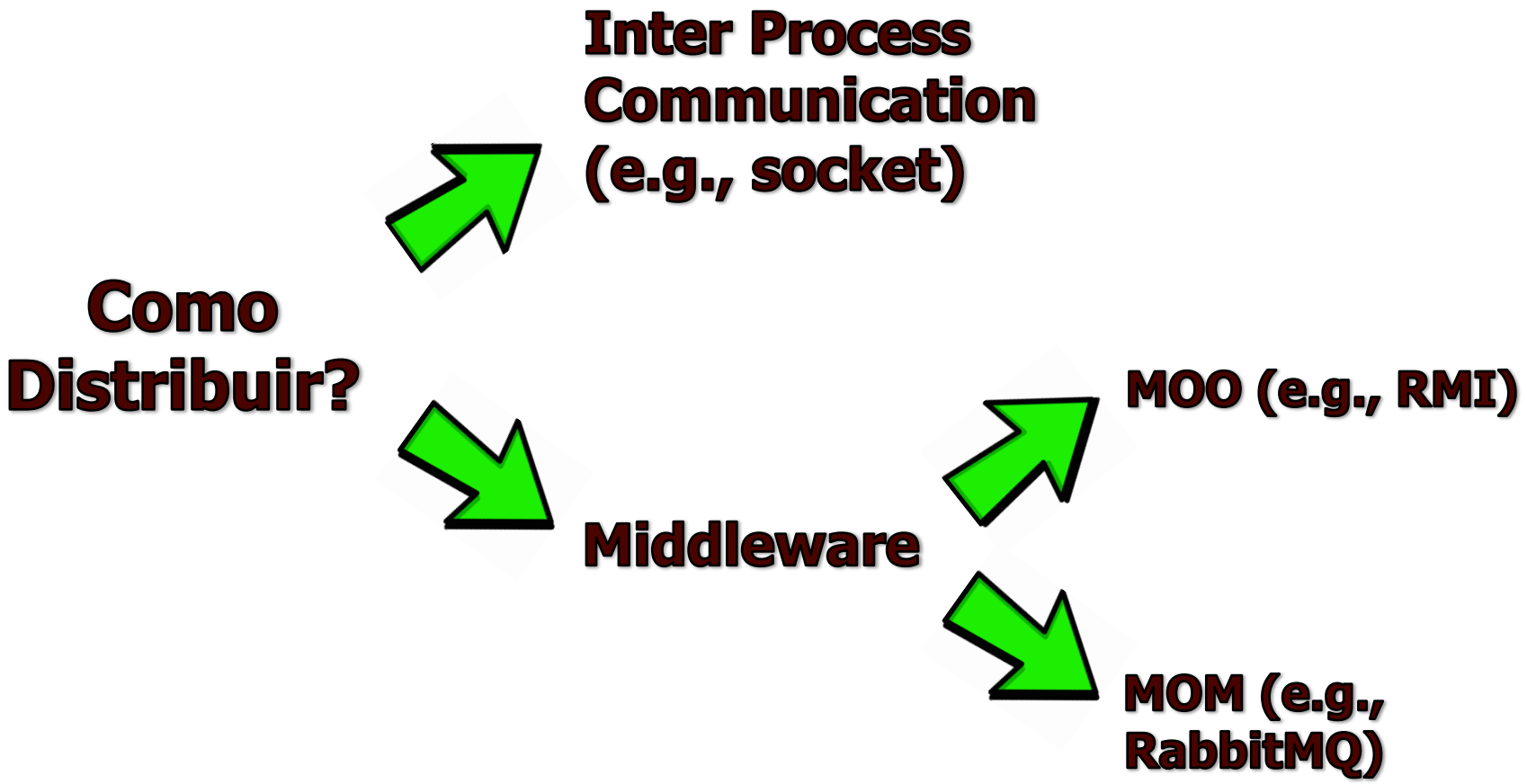


Middleware Procedural



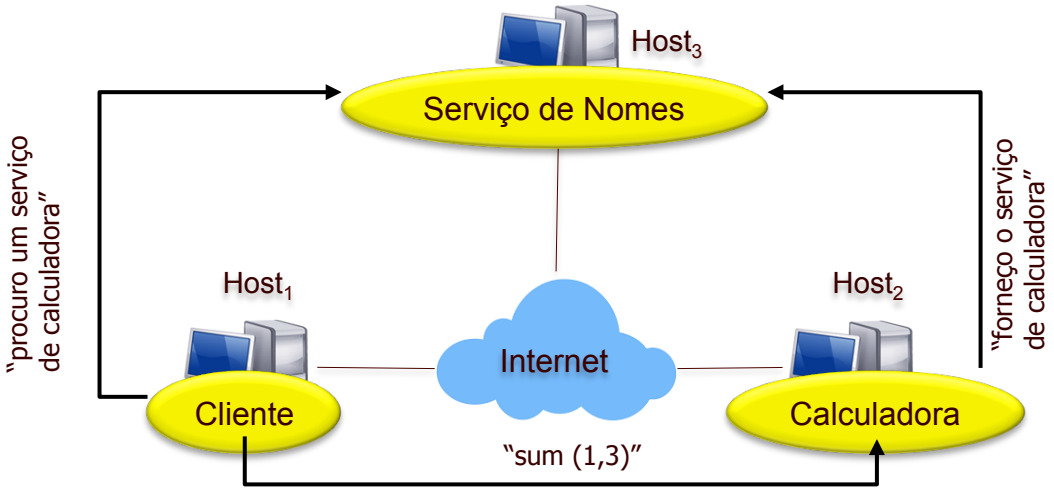
MOO versus MOM
(Aplicações)

Desenvolvimento de SDs

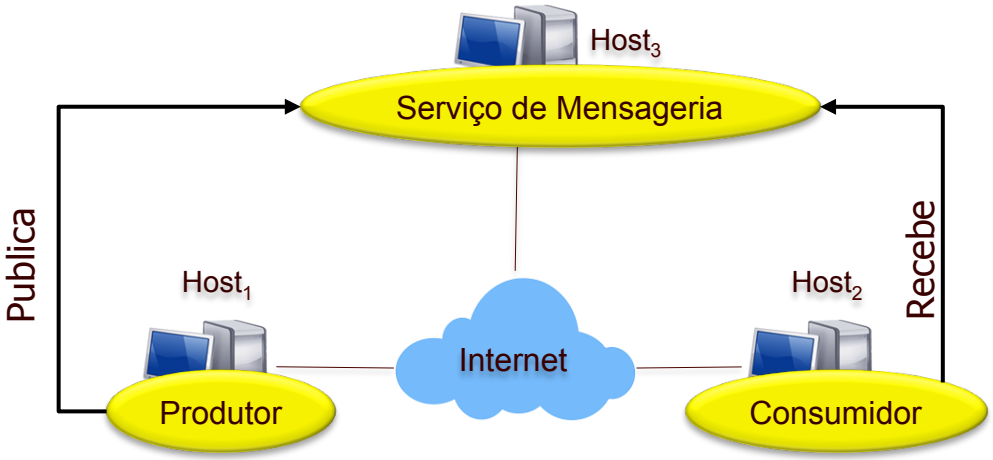


MOO versus MOM

MOO



MOM



Calculadora:: RMI:: Interface

```java
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;
}
```

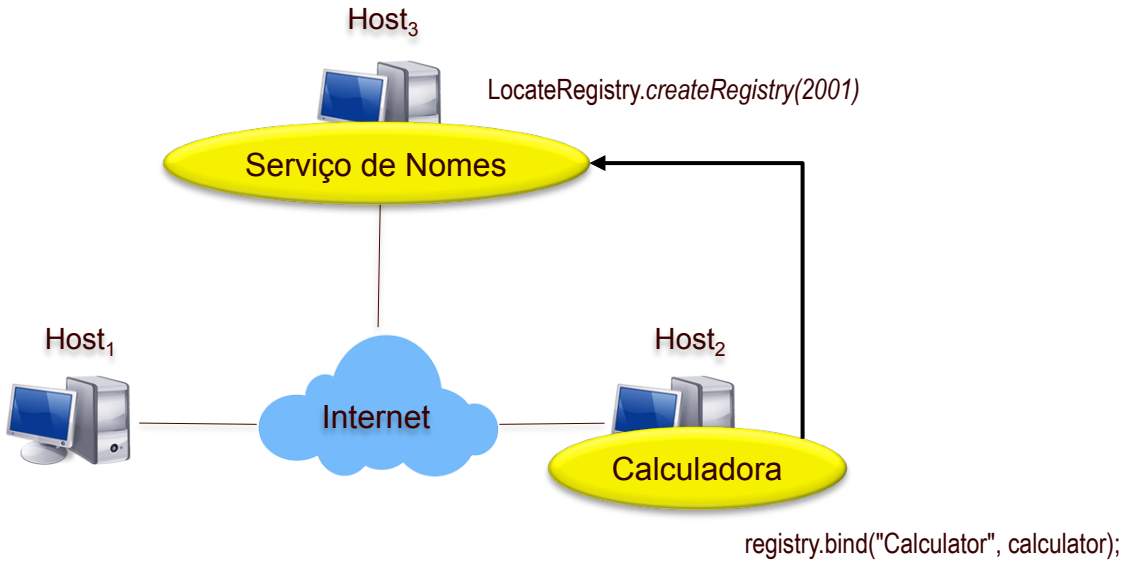
```java
public class CalculatorImpl extends UnicastRemoteObject implements ICalculator {

    protected CalculatorImpl() throws RemoteException {
        super();
        // TODO Auto-generated constructor stub
    }

    private static final long serialVersionUID = 1L;

    public float add(float x, float y) throws RemoteException {
        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;
    }

}
```

Calculadora:: 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

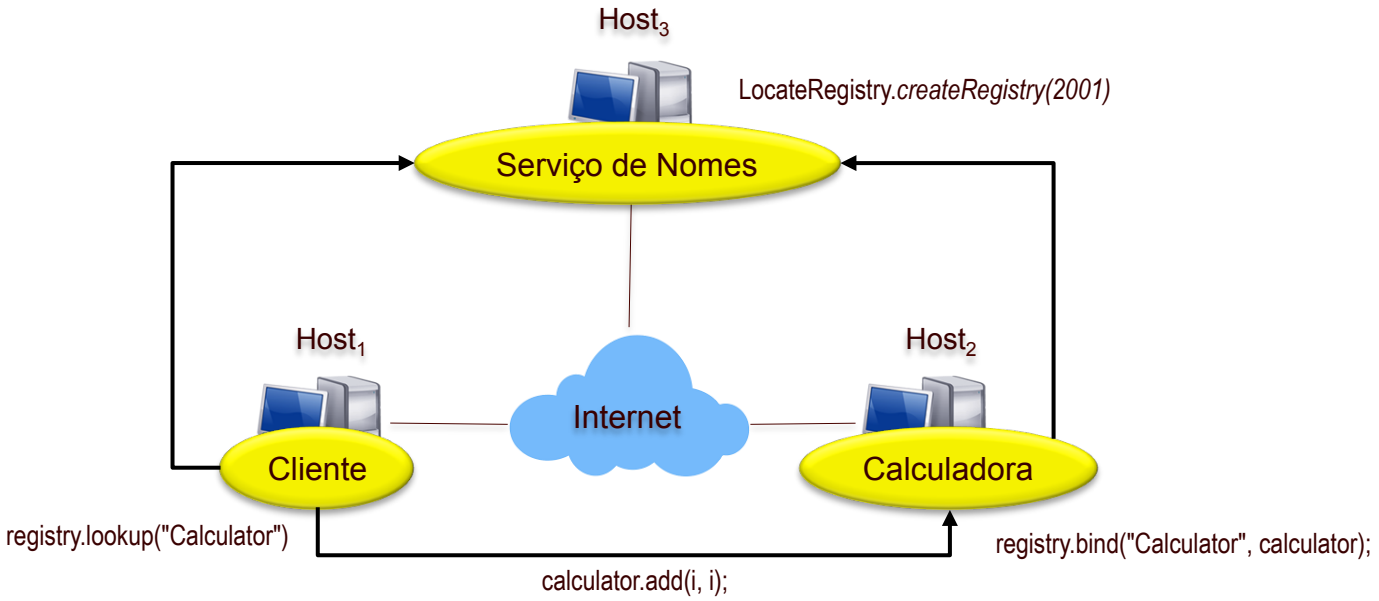
Calculadora:: RMI:: Servidor



Calculadora:: RMI:: Cliente

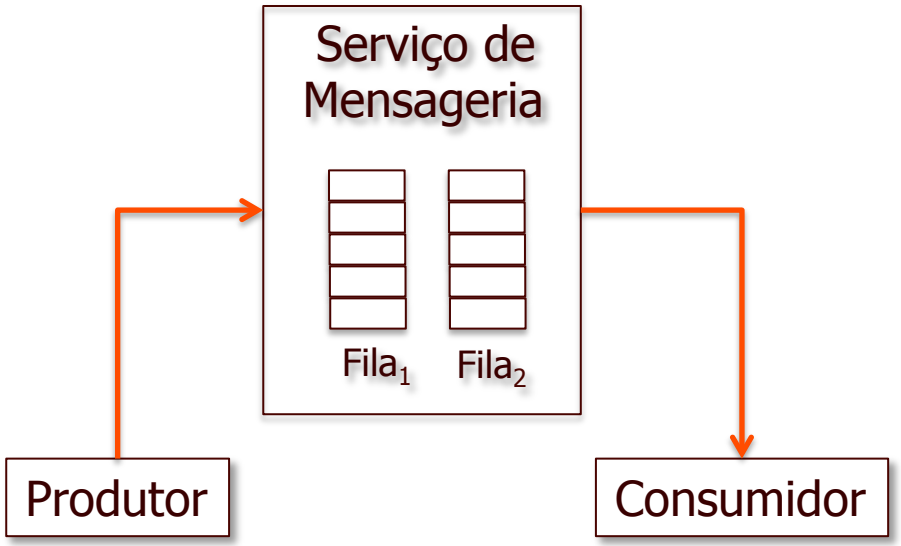
```
3  import java.rmi.NotBoundException;
7
8  public class CalculatorClient {
9
10     public static void main(String[] args) throws RemoteException,
11             NotBoundException {
12
13         // obtain a reference to a bootstrap remote object registry
14         Registry registry = LocateRegistry.getRegistry("localhost", 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:: RMI:: Cliente



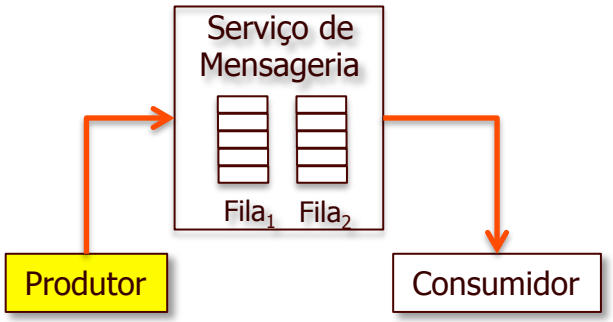
RabbitMQ

- **O que é?**
  - Message-oriented Middleware (MOM) ou
  - Serviço de mensageria ou
  - Broker de mensagens



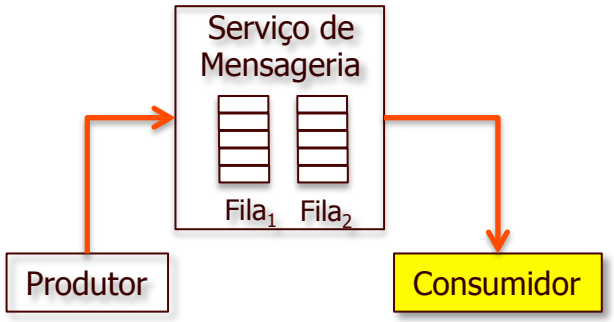
- **Elementos de uma aplicação**
  - Produtor
  - Consumidor
  - Fila(s)

RabbitMQ:: Produtor



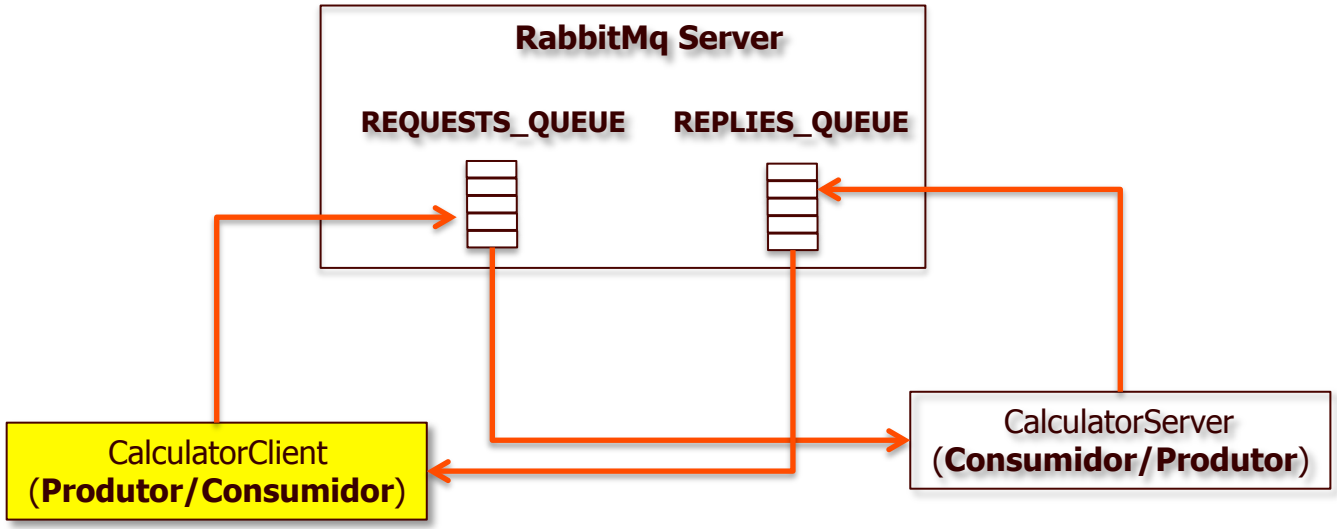

```java
public class Sender {
    private final static String QUEUE_NAME = "hello";

    public static void main(String[] argv) throws Exception {
        // create connection
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("localhost");
        Connection connection = factory.newConnection();

        // create channel
        Channel channel = connection.createChannel();

        // create queue
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        // create message to be sent
        String message = "Hello World!";

        // publish message
        channel.basicPublish("", QUEUE_NAME, null, message.getBytes("UTF-8"));

        // close channel / connection
        channel.close();
        connection.close();
    }
}
```

RabbitMQ:: Receiver




```
public class Receiver {

  private final static String QUEUE_NAME = "hello";

  public static void main(String[] argv) throws Exception {

      // create connection
    ConnectionFactory factory = new ConnectionFactory();
    factory.setHost("localhost");
    Connection connection = factory.newConnection();

    // create channel
    Channel channel = connection.createChannel();

    // create - if not created - the queue
    channel.queueDeclare(QUEUE_NAME, false, false, false, null);

    System.out.println(" [*] Waiting for messages. To exit press CTRL+C");

    // callback object - used by the broker to notify the receiver and store messages they can be consumed by the receiver
    Consumer consumer = new DefaultConsumer(channel) {
      @Override
      public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body)
          throws IOException {
        Object message = new String(body, "UTF-8");
        System.out.println(" [x] Received '" + message + "'");
      }
    };
    channel.basicConsume(QUEUE_NAME, true, consumer);
  }
}
```

Calculadora:: RabbitMQ:: Client



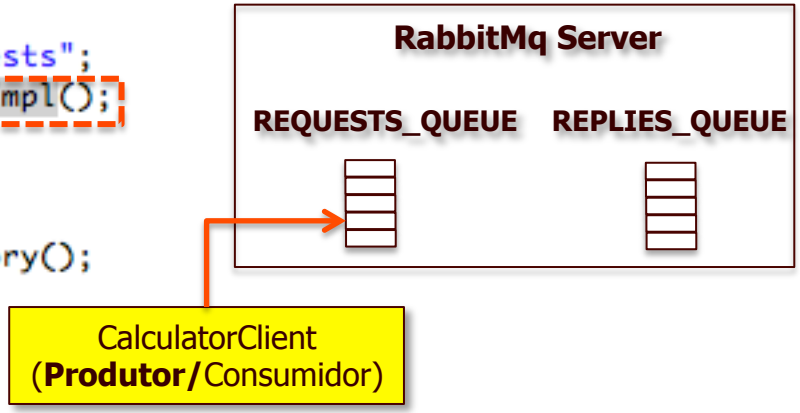

```
77  public class CalculatorClient {
78      public static void main(String[] argv) throws Exception {
79          Thread sender = new Thread(new ThreadSend());
80          Thread receiver = new Thread(new ThreadReceive());
81
82          sender.start();     // send requests
83          receiver.start();   // receive responses
84      }
```

To Server

From Server

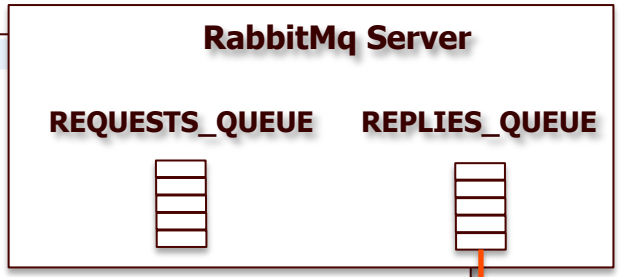
Calculadora:: RabbitMQ:: Sender

```
class ThreadSend implements Runnable {
    private final static String REQUESTS_QUEUE = "requests";
    private MarshallerImpl marshaller = new MarshallerImpl();

    public void run() {
        // create connection
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("localhost");
        Connection connection;
        Channel channelReq = null;

        try {
            connection = factory.newConnection();
            channelReq = connection.createChannel();
            channelReq.queueDeclare(REQUESTS_QUEUE, false, false, false, null);
            for (int idx = 0; idx < 1000; idx++) {
                Request message = new Request("sum", idx, idx);
                channelReq.basicPublish("", REQUESTS_QUEUE, null, marshaller.marshall(message));
            }
        } catch (IOException | TimeoutException | InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```



Calculadora:: RabbitMQ:: Receiver

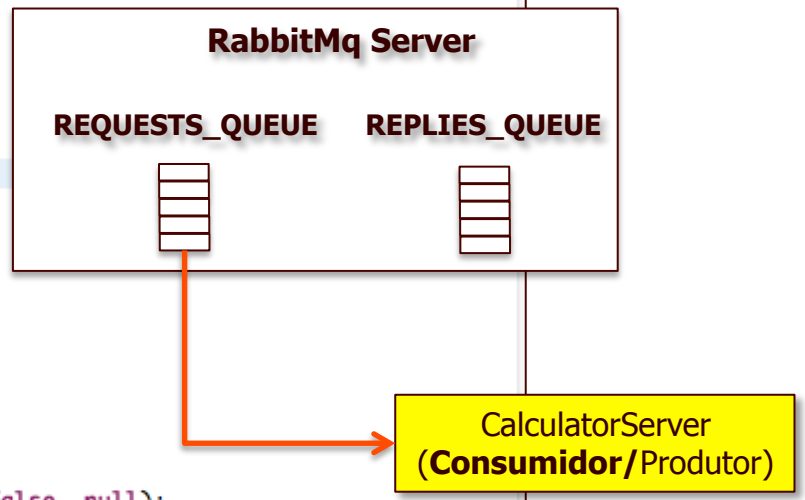
```
class ThreadReceive implements Runnable {
    private final static String REPLIES_QUEUE = "replies";
    private MarshallerImpl marshaller = new MarshallerImpl();

    public void run() {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("localhost");
        Connection connection = null;
        Channel channelRep = null;

        try {
            connection = factory.newConnection();
            channelRep = connection.createChannel();
            channelRep.queueDeclare(REPLIES_QUEUE, false, false, false, null);
        } catch (IOException | TimeoutException e) {
            e.printStackTrace();
        }

        Consumer consumer = new DefaultConsumer(channelRep) {
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties,
                    byte[] body) throws IOException {
                Object message = marshaller.unmarshall(body);
                Reply reply = (Reply) message;
                System.out.println(reply.getReply());
            }
        };
        try {
            channelRep. basicConsume(REPLIES_QUEUE, true, consumer);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



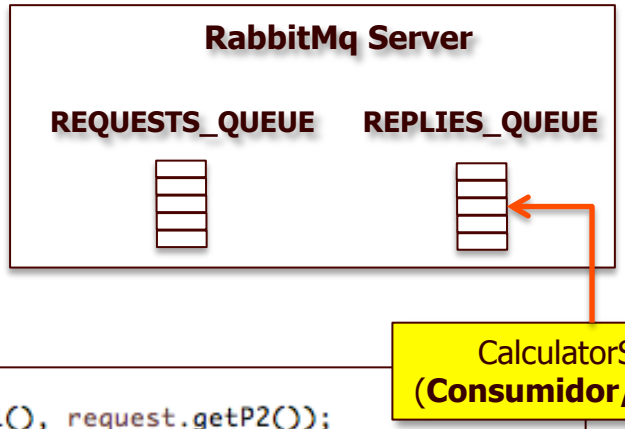
CalculatorClient
(Produtor/**Consumidor**)

Calculadora:: RabbitMQ:: Server

```
8  public class CalculatorServer {
9      private final static String REQUESTS_QUEUE = "requests";
10     private final static String REPLIES_QUEUE = "replies";
11
12     public static void main(String[] argv) throws Exception {
13         MarshallerImpl marshaller = new MarshallerImpl();
14
15         // create connection
16         ConnectionFactory factory = new ConnectionFactory();
17         factory.setHost("localhost");
18         Connection connection = factory.newConnection();
19
20         // create channel
21         Channel channelReq = connection.createChannel();
22         Channel channelRep = connection.createChannel();
23
24         // create - if not created - the queue
25         channelReq.queueDeclare(REQUESTS_QUEUE, false, false, false, null);
26         channelRep.queueDeclare(REPLIES_QUEUE, false, false, false, null);
27
28         System.out.println(" [*] Waiting for messages. To exit press CTRL+C");
29
30         Consumer consumer = new DefaultConsumer(channelReq) {
31             @Override
32             public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties p
33                     byte[] body) throws IOException {
34                 Object message = marshaller.unmarshall(body);
35                 Request request = (Request) message;
36                 float r = 0;
37
38                 switch (request.getOp()) {
39                 case "sub":
40                     r = new CalculatorImpl().sub(request.getP1(), request.getP2());
41                     break;
```



Calculadora:: RabbitMQ:: Server (cont.)




```
42                  case "sum":
43                      r = new CalculatorImpl().add(request.getP1(), request.getP2());
44                      break;
45                  case "div":
46                      r = new CalculatorImpl().div(request.getP1(), request.getP2());
47                      break;
48                  case "mul":
49                      r = new CalculatorImpl().mul(request.getP1(), request.getP2());
50                      break;
51                  default:
52                          System.out.println("Operation is not implemented in the server");
53                          System.exit(0);
54                      break;
55                  }
56                  Reply rep = new Reply(r);
57                  try {
58                      channelRep.basicPublish("", REPLIES_QUEUE, null, marshaller.marshall(rep));
59                  } catch (InterruptedException e) {
60                      // TODO Auto-generated catch block
61                      e.printStackTrace();
62                  }
```

Códigos

- **RMI**
  - https://www.dropbox.com/sh/1nxeiiyddgg04f4/AADnYztKbjGgL7E974-HaTpVa?dl=0

- **RabbitMQ**
  - https://www.dropbox.com/sh/rh39ecrpewdbd0f/AABCqhh-hwcQj16gFPVnIF8Xa?dl=0

Fim dos Slides