Universidade Tecnológica Federal do Paraná – UTFPR
Coordenação de Ciência da Computação - COCIC
Ciência da Computação

BCC34G – Sistemas Operacionais

Prof. Rogério A. Gonçalves

rogerioag@utfpr.edu.br

Aula 012

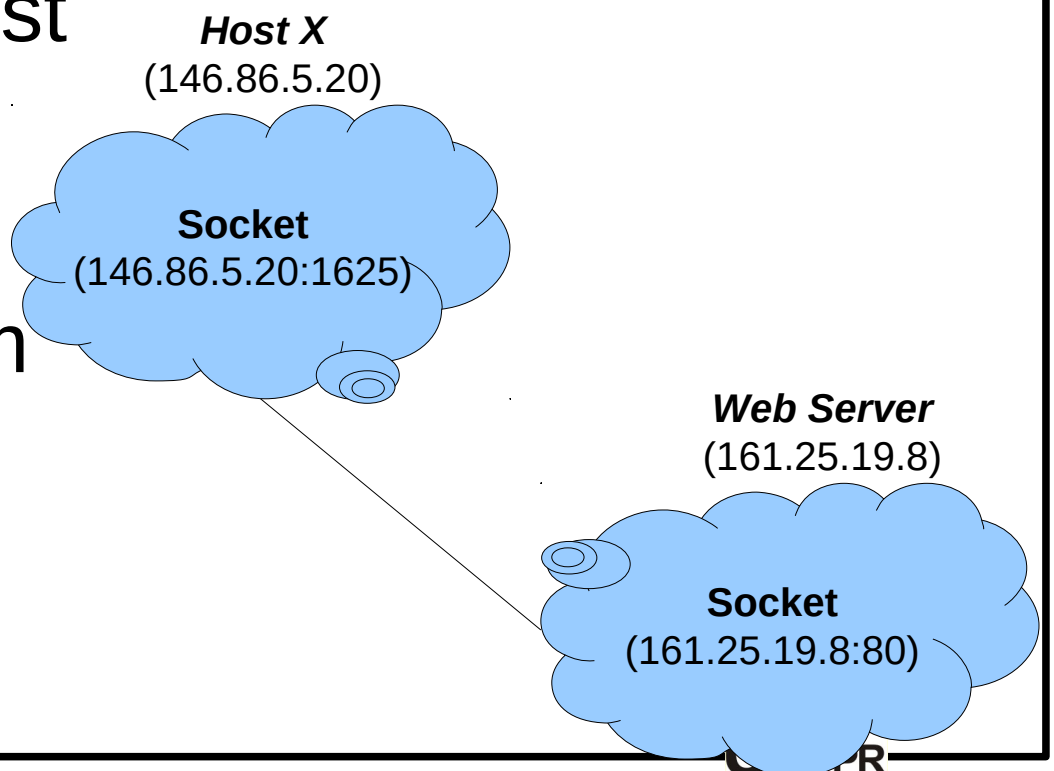
- Comunicação entre processos
  - Sockets
  - Chamada Remota de Procedimento

Mecanismos de passagem de mensagem

- **Sockets**
  - Uma extremidade para comunicação
- **RPC** – *Remote Procedure Call*
  - Rotinas que permitem comunicação de processos em diferentes máquinas
- **MPI** – *Message Passing Interface*
  - *Sistemas paralelos*
- **RMI** (Java) – *Remote Method Invocation*
  - Permite que um objeto ativo em uma máquina virtual Java possa interagir com objetos de outras máquinas virtuais Java, independentemente da localização dessas máquinas virtuais

Socket

- É identificado por um endereço IP concatenado e a porta
- O socket **161.25.19.8:1625** refere-se à porta **1625** no host **161.25.19.8**
- A comunicação consiste entre um par de sockets



Sockets

- Permitem que pares de processos troquem dados estabelecendo canais diretos de comunicação bidirecional.
- Usados principalmente para comunicação bidirecional entre vários processos em sistemas diferentes, mas podem ser usados para processos no mesmo sistema.
- Armazenados internamente como arquivos.
- O nome do arquivo é usado como endereço do soquete, que é acessado por meio do VFS.

Sockets

- **Sockets de fluxo**
  - Implementam o tradicional modelo cliente/servidor.
  - Os dados são transferidos como um fluxo de bytes.
  - Usam TCP para comunicação, de modo que são mais apropriados quando a comunicação tem de ser confiável.
- **Sockets de datagrama**
  - Comunicação mais rápida, mas menos confiável.
  - Os dados são transferidos por meio de pacotes de datagramas.
- **Par de Sockets**
  - Par de soquetes conectados e não denominados.
  - Limitado para ser usado por processos que compartilham descritores de arquivo.

Socket em C

- Exemplos.

Tutorial: *http://www-usr.inf.ufsm.br/~giovani/sockets.html*

Exemplo: sockets (so-aula-012-cod.zip)

Comunicação por *Sockets* em Java

Servidor: Código do Livro

```
public class DateServer
{
   public static void main(String[] args) {
      try {
         ServerSocket sock = new ServerSocket(6013);

         // now listen for connections
         while (true) {
            Socket client = sock.accept();

            PrintWriter pout = new
             PrintWriter(client.getOutputStream(), true);

            // write the Date to the socket
            pout.println(new java.util.Date().toString());

            // close the socket and resume
            // listening for connections
            client.close();
         }
      }
      catch (IOException ioe) {
         System.err.println(ioe);
      }
   }
}
```

Comunicação por *Sockets* em Java

Cliente: Código do Livro

```
public class DateClient
{
   public static void main(String[] args) {
      try {
         //make connection to server socket
         Socket sock = new Socket("127.0.0.1",6013);

         InputStream in = sock.getInputStream();
         BufferedReader bin = new
            BufferedReader(new InputStreamReader(in));

         // read the date from the socket
         String line;
         while ( (line = bin.readLine()) != null)
            System.out.println(line);

         // close the socket connection
         sock.close();
      }
      catch (IOException ioe) {
         System.err.println(ioe);
      }
   }
}
```

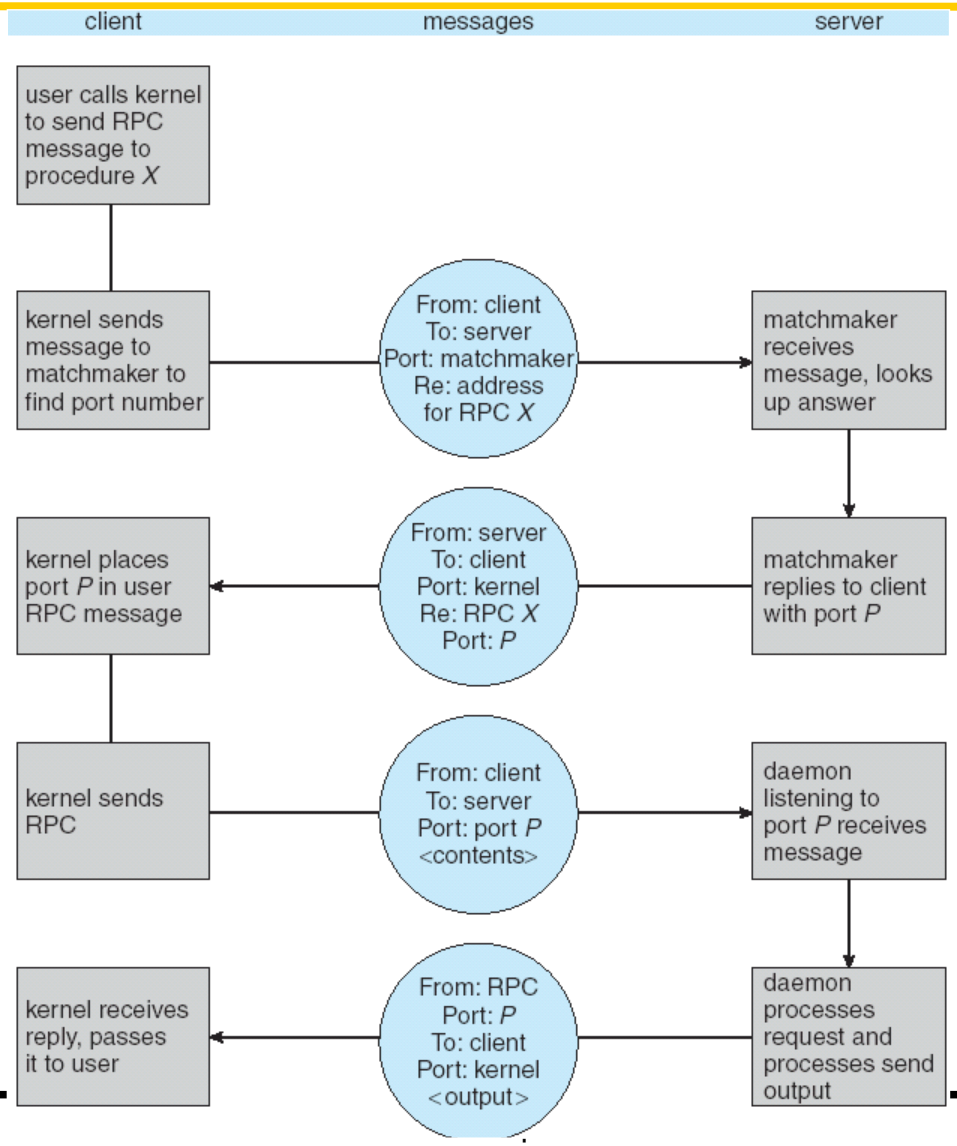
Comunicação por Sockets em Java

Exemplo de Código.

RPC

- Chamada de procedimento remoto (RPC) passa chamadas de procedimento entre processos nos sistemas em rede.
- **Stubs** – proxy no cliente para o procedimento real no servidor.
- O stub no cliente localiza o servidor e *organiza* os parâmetros.
- O stub no servidor recebe essa mensagem, desempacota os parâmetros organizados e realiza o procedimento no servidor.

Execução da RPC



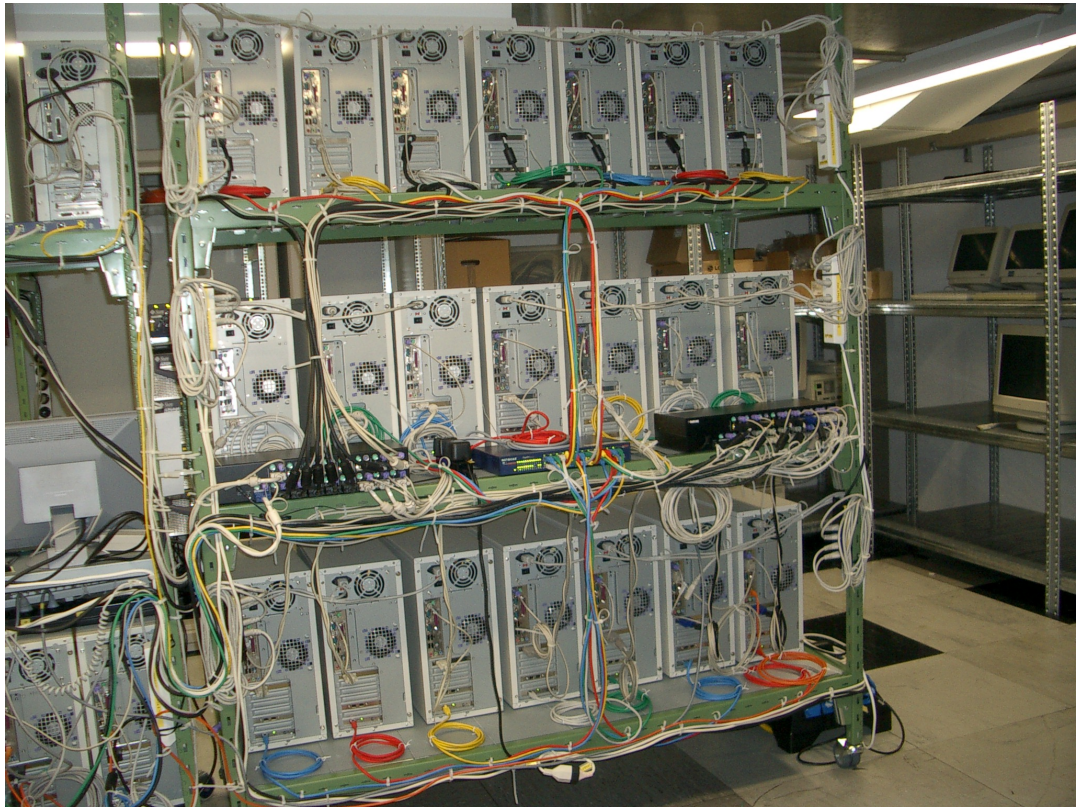
Passagem de mensagem

- Provê troca de mensagens entre processos rodando em máquinas diferentes;
- Utiliza-se de duas primitivas de chamadas de sistema: `send` e `receive`;
- Podem ser implementadas como procedimentos:

```
send (destination,&message);
receive (source,&message);
```

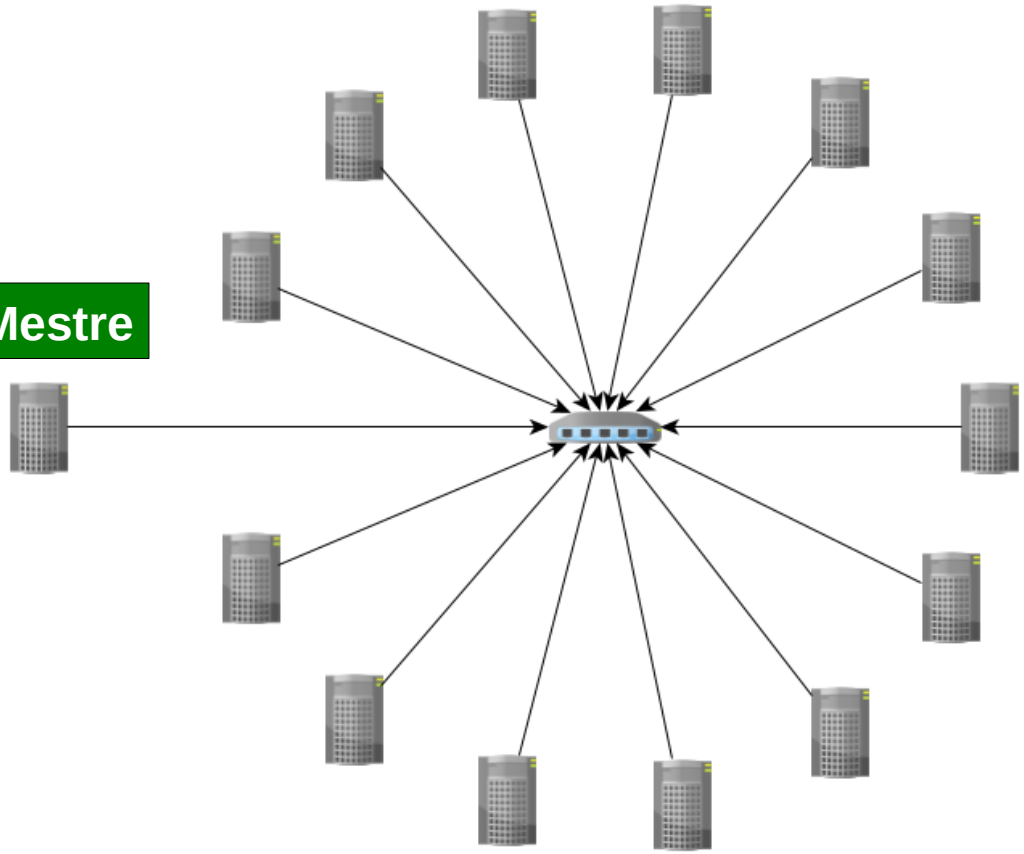
- O procedimento `send` envia para um determinado destino uma mensagem, enquanto que o procedimento `receive` recebe essa mensagem em uma determinada fonte; Se nenhuma mensagem está disponível, o procedimento `receive` é bloqueado até que uma mensagem chegue.

Cluster



Fonte: TIK Experimental Cluster “Scylla”

Cluster



Cada máquina (nó) do cluster executam o mesmo código, sendo chaveado pelo id (rank) do processo,

```
main(int argc, char** argv){
  ...
  /* start up MPI */
  MPI_Init(&argc, &argv);

  /* Get my process rank */
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

  /* Find out how many process are being used */
  MPI_Comm_size(MPI_COMM_WORLD, &p);

  // Faz cálculos...

  /* Chaveamento no código pelo id */
  if(my_rank == 0){
   for(source = 1; source < p; source++){
     MPI_Recv(&variavel, 1, MPI_FLOAT, source, tag, MPI_COMM_WORLD, &status);
   }
  }else{
    MPI_Send(&variavel, 1, MPI_FLOAT, dest, tag, MPI_COMM_WORLD);
  }

  /* Imprimir o resultado só no Mestre */
  if(my_rank == 0){
    printf("Resultado");
  }

  /* Shutdown MPI */
  MPI_Finalize();
}
```

O Mestre tem rank = 0
Os slaves tem rank > 0.
Desta forma é possível determinar qual trecho de código será executado no mestre e em cada um dos slaves.

Passagem de mensagem - Problemas

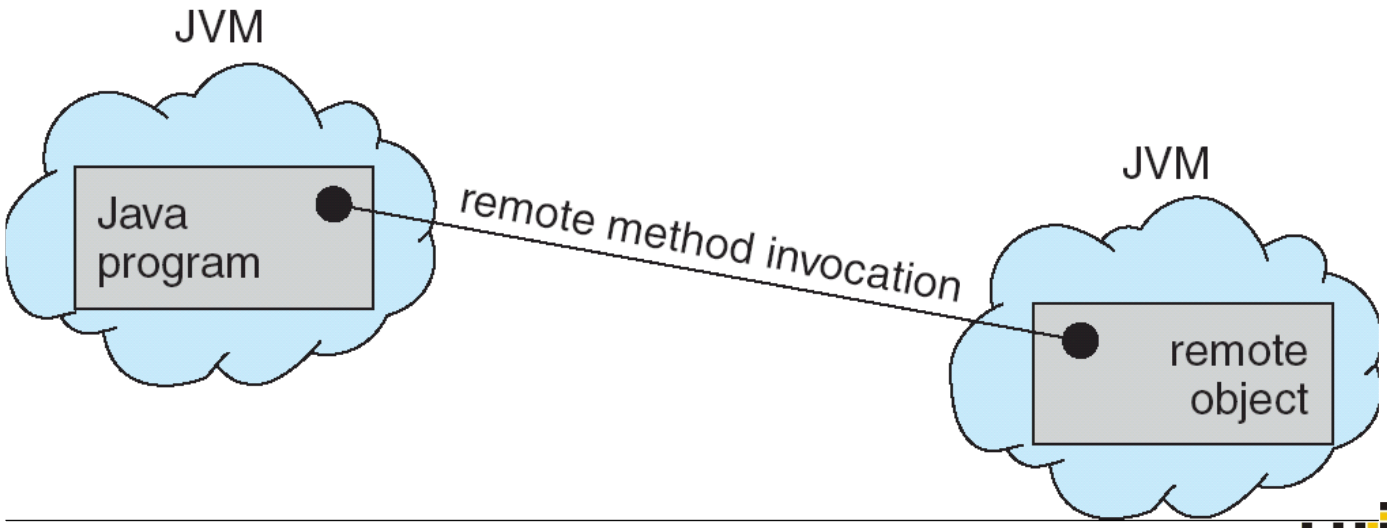
- Mensagens podem ser perder na transmissão;
  - Mensagem especial **acknowledgement (ack)** → o procedimento `receive` envia um **ack** para o procedimento `send`. Se esse **ack** não chega no procedimento `send`, esse procedimento retransmite a mensagem já enviada;

Passagem de mensagem - Problemas

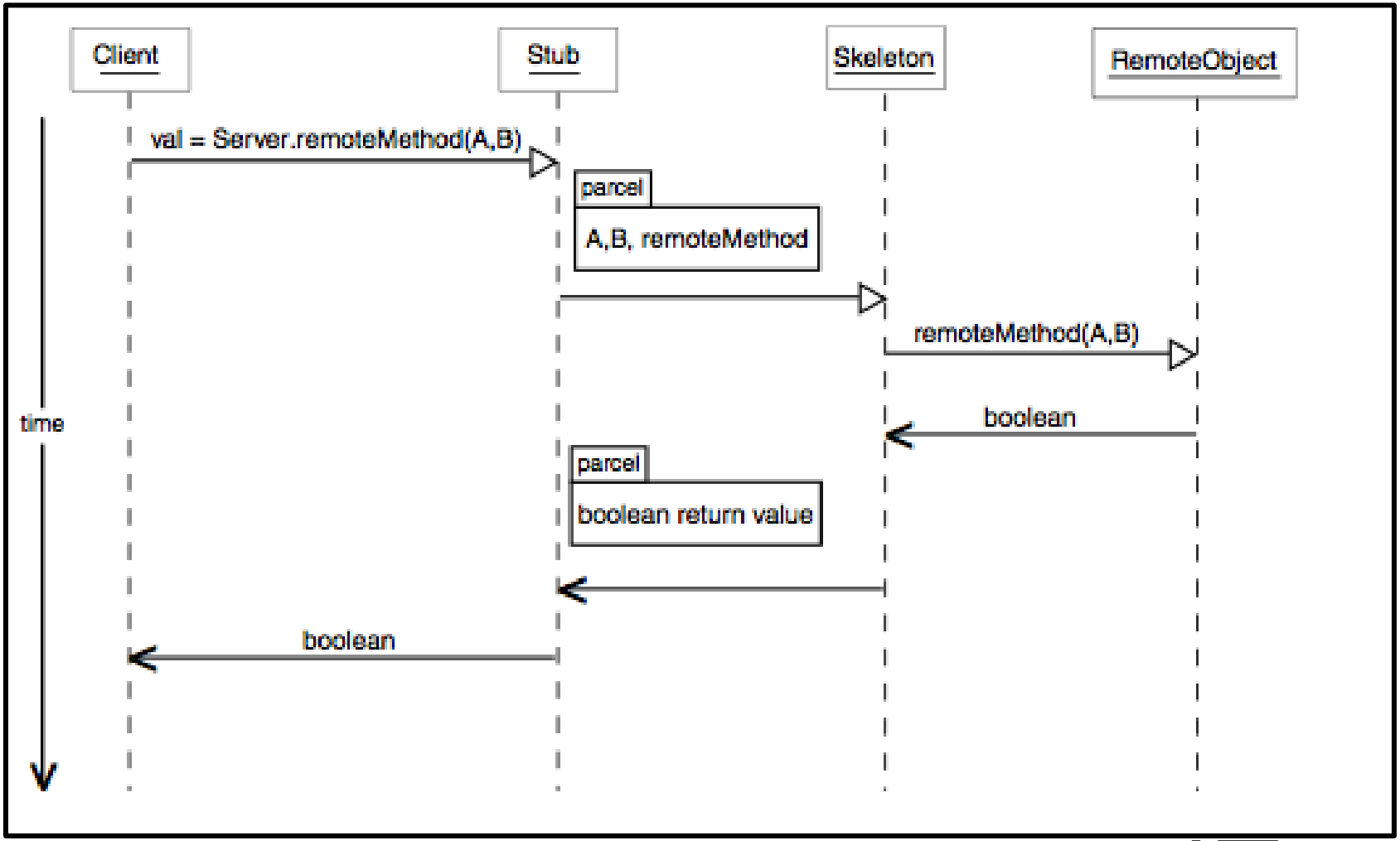
- A mensagem é recebida mas o **ack** se perde.
  - `receive` checa se cada mensagem enviada pelo `send` satisfaz uma seqüência de números. Ao receber uma nova mensagem, `receive` verifica essa identificação, se ela for semelhante a de alguma mensagem já recebida, `receive` descarta a mensagem.
  - Desempenho: copiar mensagens de um processo para o outro é mais lento do que operações com semáforos e monitores;
  - Autenticação → Segurança;

RMI

- Invocação de método remoto (RMI) é um mecanismo da Java semelhante às RPCs.
- RMI permite que um programa Java em uma máquina chame um método em um objeto remoto.



Organização de parâmetros



Exemplo RMI

Definição da Interface Remota

RMICalc.java | RMICalcImpl.java | RMICliente.java

```
1  package br.utfpr.rag.exeaulas.rmi;
2
3  // Declaracao da interface para o objeto remoto.
4
5  import java.rmi.Remote;
6
7  // Interface para objetos que implementam uma calculadora Remota.
8
9
10 public interface RMICalc extends Remote
11 {
12     double somar( double x, double y ) throws RemoteException;
13
14     double subtrair( double x, double y ) throws RemoteException;
15
16     double multiplicar( double x, double y ) throws RemoteException;
17
18     double dividir( double x, double y ) throws RemoteException;
19 }
20
```

Exemplo RMI

Implementação da Interface Remota

RMICalc.java | RMICalcImpl.java | RMICliente.java

```
1  package br.utfpr.rag.exeaulas.rmi;
2  // Implementacao da classe do objeto remoto.
7
8  import java.rmi.Naming;
12
13 public class RMICalcImpl
14         extends UnicastRemoteObject
15         implements RMICalc
16
17 {
18     // Construtor.
19     public RMICalcImpl ( ) throws RemoteException { }
20
21     // Implementacao dos metodos da interface.
22     public double somar( double x, double y ) throws RemoteException
23     {
24         System.out.println("Servidor: Método somar(" + x +","+ y +") invoked...'
25         System.out.println(" -> retornando somar(" + x +","+ y +") = " + (x + y
26         return x + y;
27     }
```

Exemplo RMI

Chamada ao método remoto no Cliente

RMICalc.java | RMICalcImpl.java | RMICliente.java

```
39
40 {
41 // Faz o lookup do objeto remoto.
42 calculadoraRemota = (RMICalc) Naming.lookup("rmi://" + args[0] + "/CalculadoraZ
43
44 // Invoca os metodos para efetuar cada uma das operacoes.
45 soma = calculadoraRemota.somar(valorA, valorB);
46 diferenca = calculadoraRemota.subtrair(valorA, valorB);
47 produto = calculadoraRemota.multiplicar(valorA, valorB);
48 quociente = calculadoraRemota.dividir(valorA, valorB);
49
50 // Imprime os Resultados.
51 System.out.println("Resultados:");
52 System.out.println("Soma de " + valorA + " e " + valorB + " = " + soma);
53 System.out.println("Sutracao entre " + valorA + " e " + valorB + " = " + difere
54 System.out.println("Produto de " + valorA + " por " + valorB + " = " + produto)
55 System.out.println("Divisao de " + valorA + " por " + valorB + " = " + quociente
56
57 tch (Exception e) {
```

Próxima Aula

- ***Sincronismo***