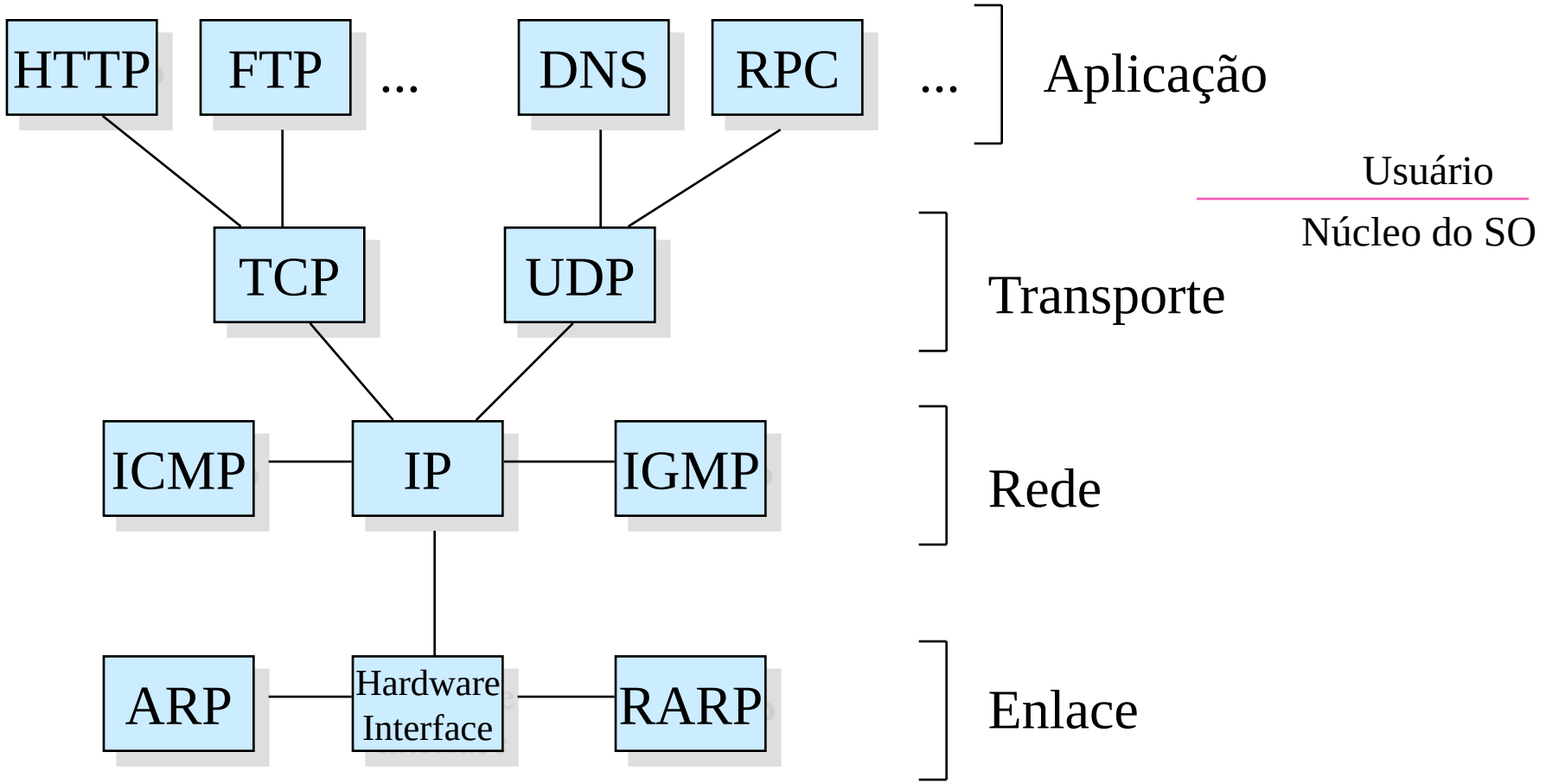
Programação com sockets

Carlos Alberto Kamienski

CIn/UFPE

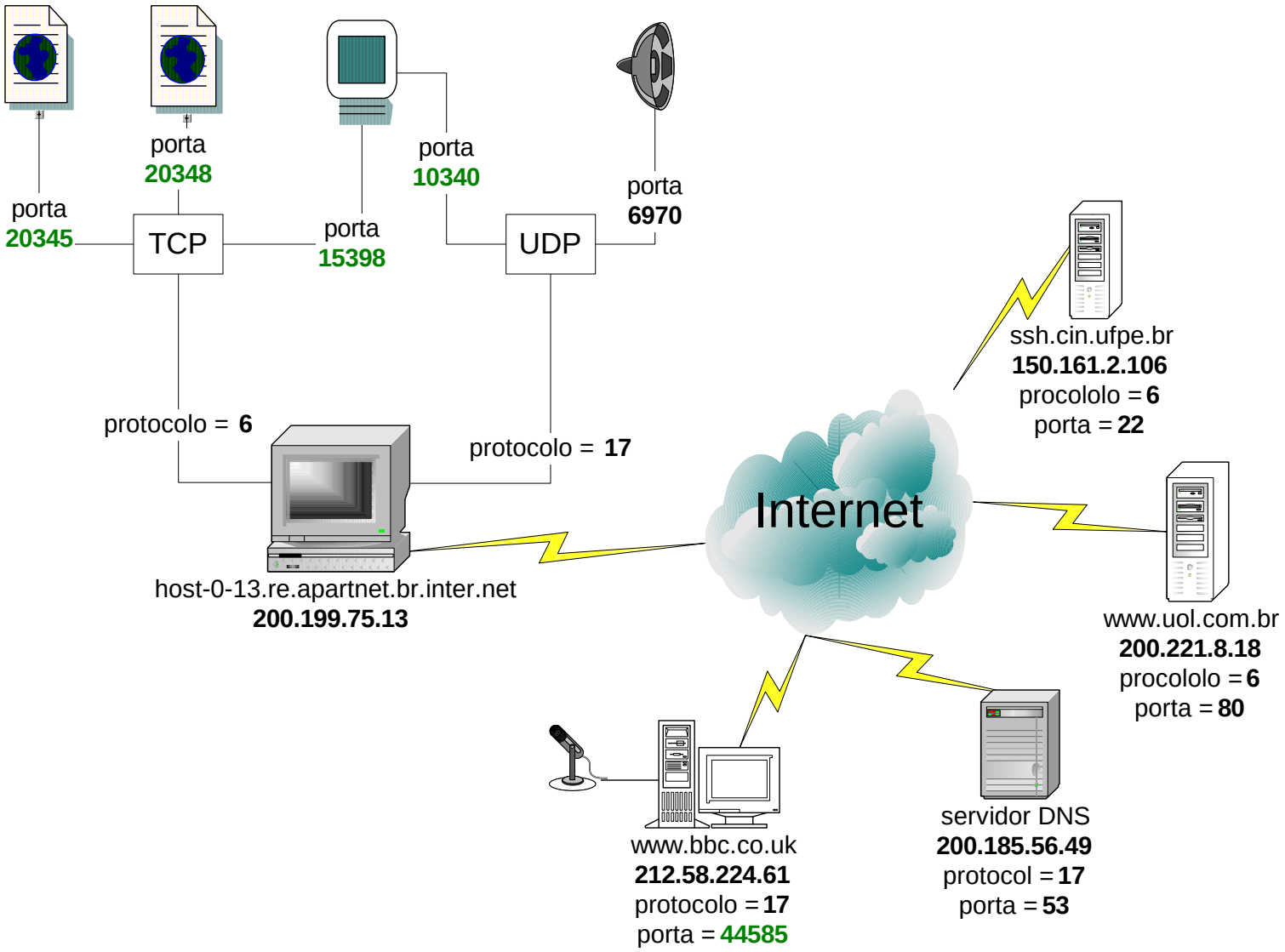
Protocolos da Internet



Identificação de aplicações

- Como cada máquina é identificada unicamente na Internet ?
- Como a entidade de rede (IP) identifica para qual protocolo de transporte está sendo utilizado ?
- Como a entidade de transporte identifica qual aplicação está sendo utilizada ?
- Um cliente pode abrir várias conexões com o mesmo servidor (ex. páginas web). Como o cliente sabe para qual programa enviar os pacotes?

Identificação de aplicações



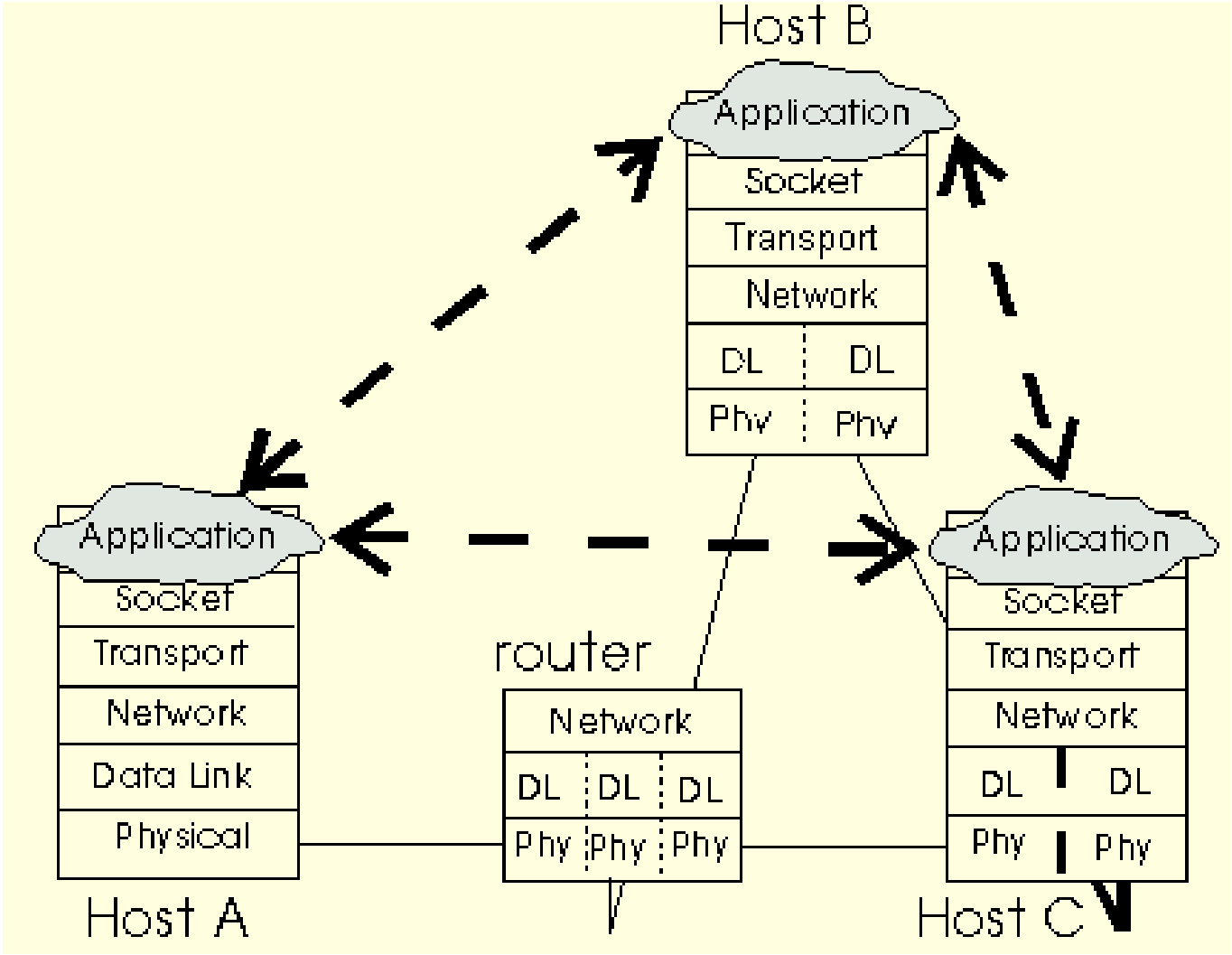
Programação na Internet

- Sockets
  - Estilo: envia/recebe (*send/receive*)
  - Característica: eficiente
- RPC
  - Chamada remota de procedimento
  - Transparência e facilidade de programação
- Objetos distribuídos
  - Transparência, facilidade e todos os benefícios da programação orientada a objetos
  - Execução mais lenta
  - Exemplos:
    - DCOM
    - CORBA
    - Java RMI
    - etc.

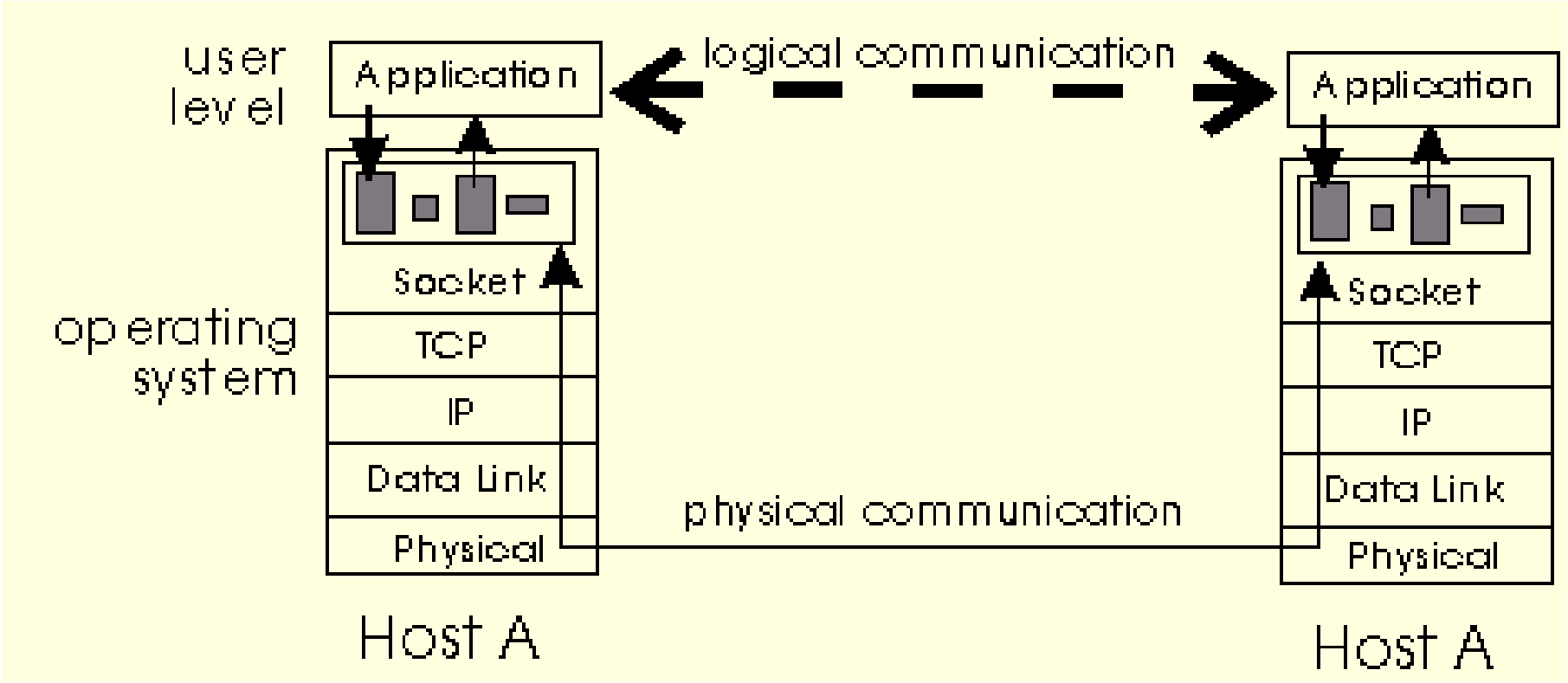
Unix BSD Sockets

- Interface padrão para comunicação entre processos em redes TCP/IP
- Nasceu com o Unix de Berkeley
- Os projetistas tentaram usar ao máximo as chamadas de sistema do Unix
- Implementada hoje em vários Sos
- Programar com sockets pode ser visto como **desenvolver um protocolo de aplicação**

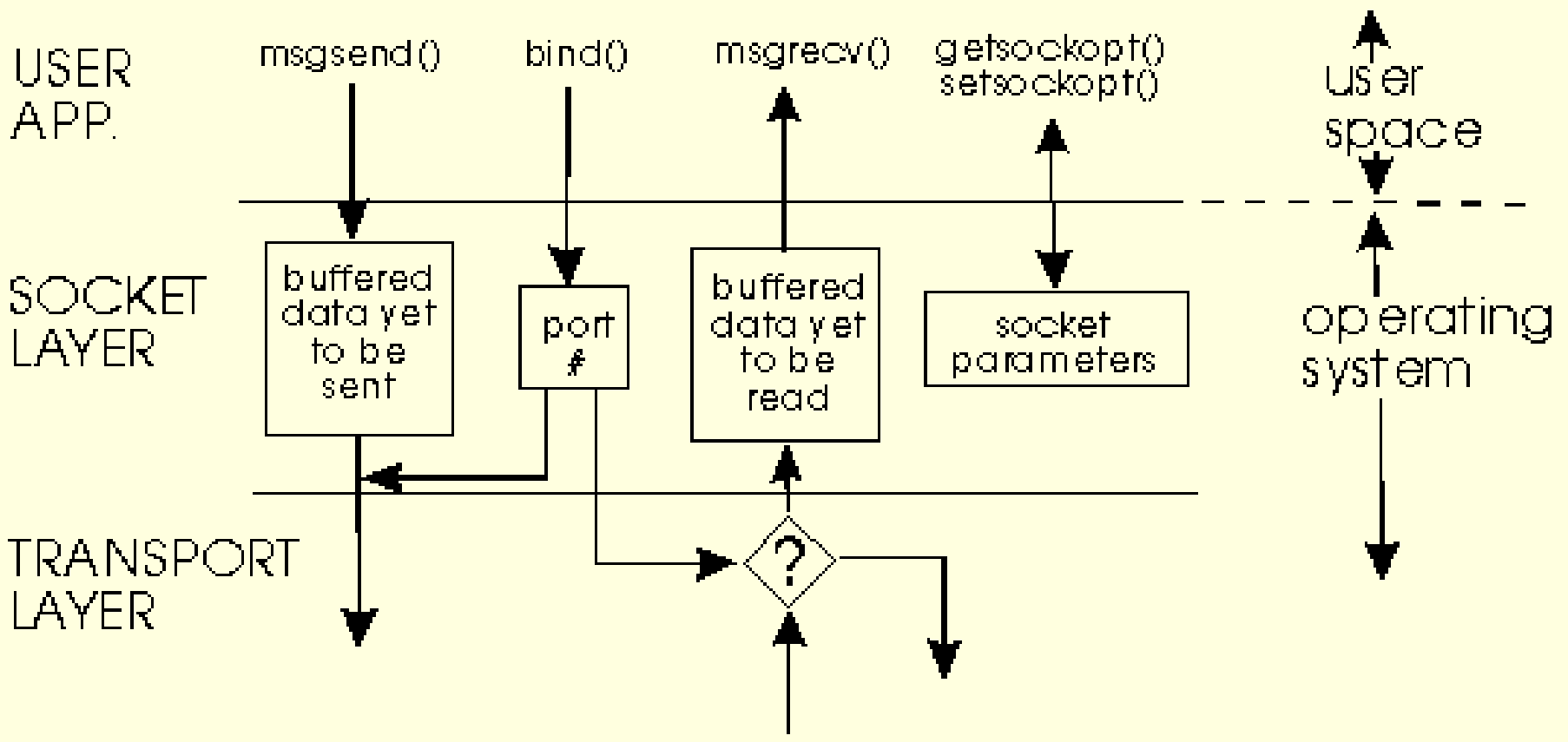
Berkeley Sockets



Berkeley Sockets



Sockets - visão conceitual



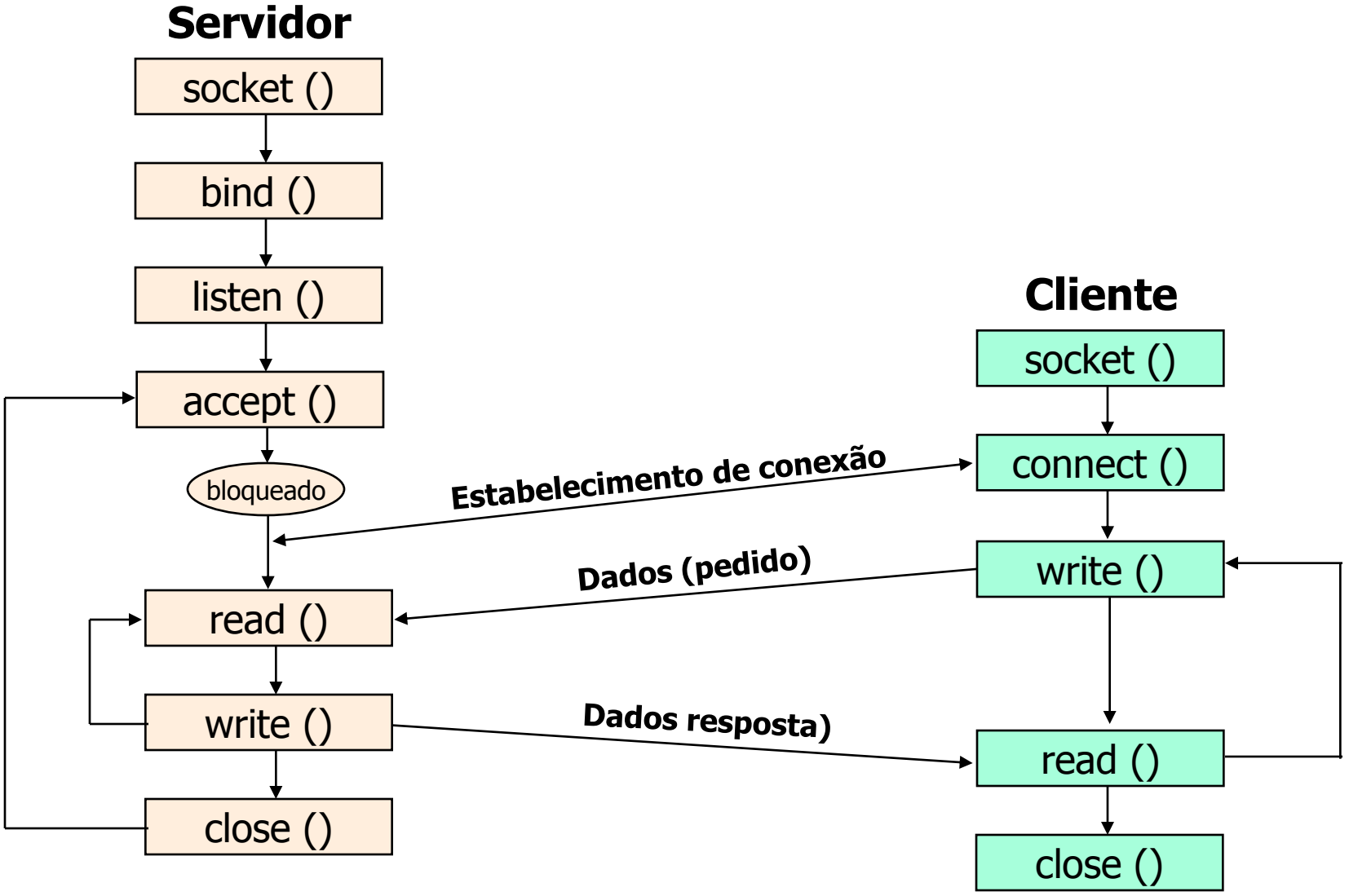
Tipos de sockets

- Serviço com conexão
  - » Implementa um *stream* de dados (SOCK_STREAM)
  - » Protocolo TCP (tipicamente)
- Serviço sem conexão
  - » Implementa um serviço de datagramas (SOCK_DGRAM)
  - » Protocolo UDP (tipicamente)
- Serviço de baixo nível
  - » Acessa diretamente a camada de rede (SOCK_RAW)
  - » Protocolo IP (tipicamente)

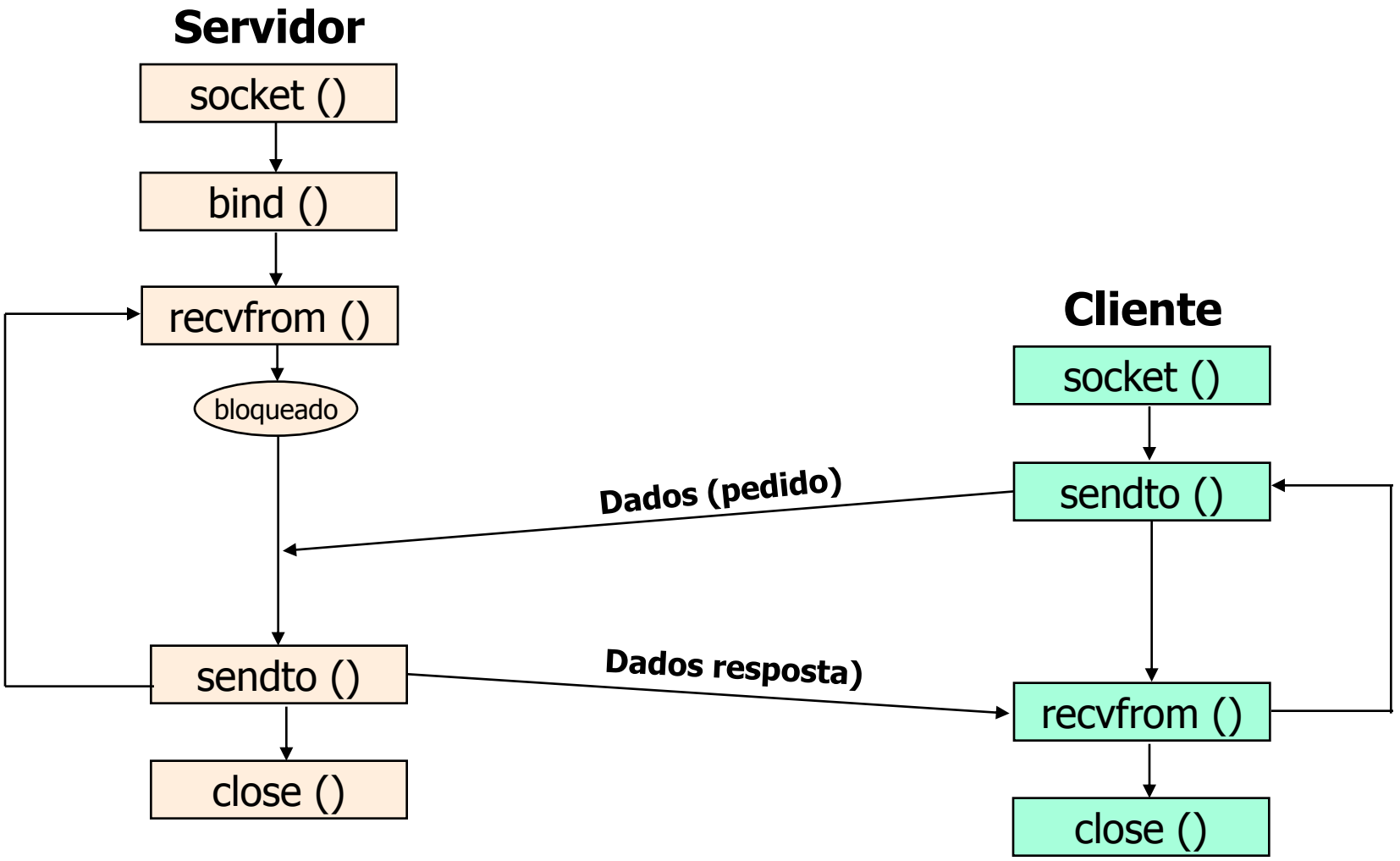
Principais funções da API

socket	Cria um novo descritor para comunicação
connect	Iniciar conexão com servidor
write	Escreve dados em uma conexão
read	Lê dados de uma conexão
close	Fecha a conexão
bind	Atribui um endereço IP e uma porta a um socket
listen	Coloca o socket em modo passivo, para “escutar” portas
accept	Bloqueia o servidor até chegada de requisição de conexão
recvfrom	Recebe um datagrama e guarda o endereço do emissor
sendto	Envia um datagrama especificando o endereço

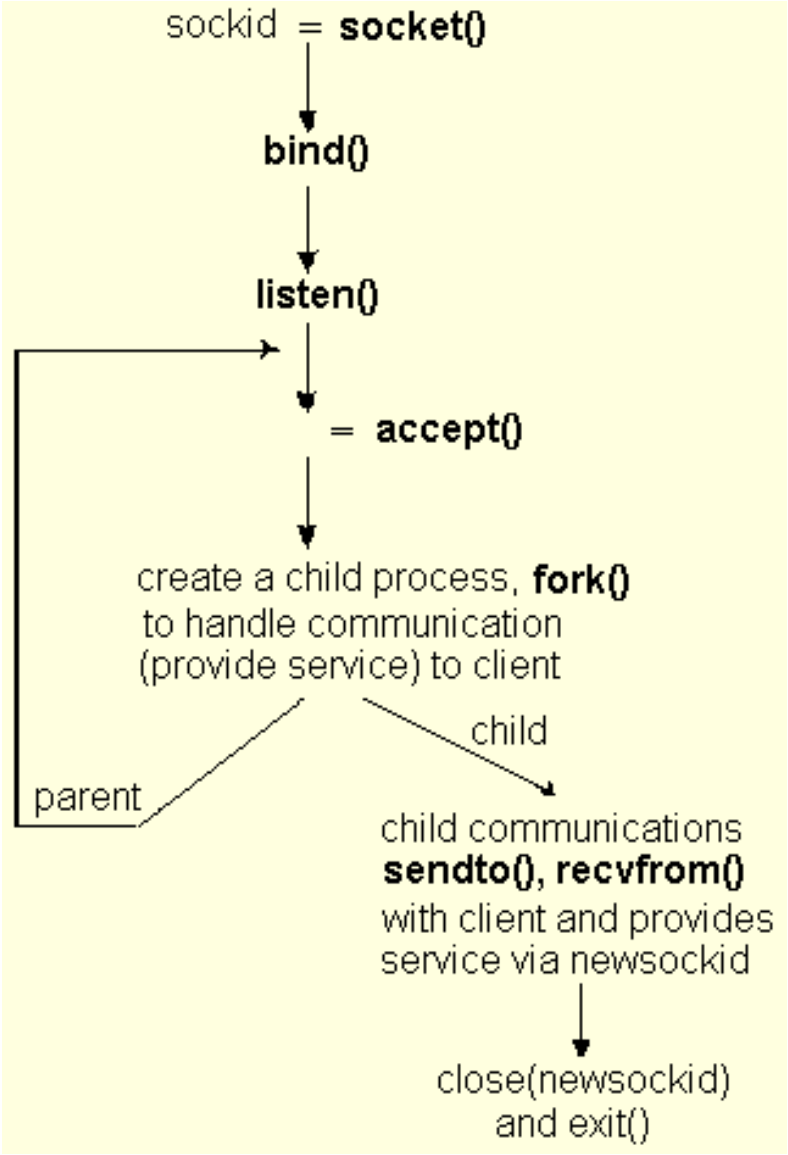
Serviço com Conexão (TCP)



Serviço sem Conexão (UDP)



Estrutura Típica de um Servidor



Números de portas

- 1-255 reservadas para serviços padrão portas “bem conhecidas”
- 256-1023 reservado para serviços Unix
- 1-1023 Somente podem ser usadas por usuários privilegiados (super-usuário)
- 1024-4999 Usadas por processos de sistema e de usuário
- 5000- Usadas somente por processos de usuário

Sockets em Java

- Java modernizou a API para trabalhar com sockets
- O programador não precisa chamar todas as funções, algumas chamadas são automáticas
- Exemplos
  - » Socket: equivalente a *socket* e *bind*
  - » ServerSocket: equivalente a *socket*, *bind* e *listen*
- Sockets são implementados no pacote ***java.net***
- A transmissão e a recepção de dados são feitos através de classes do pacote ***java.io*** de maneira semelhante à escrita e leitura em arquivos
  - » Classes *DataInputStream*, *DataOutputStream*, etc.,

```
import java.net.*;
import java.io.*;

public class SimpleJavaClient {
   public static void main(String[] args)       {
        try {
                Socket s = new Socket("127.0.0.1", 9999);
                InputStream i = s.getInputStream();
                OutputStream o = s.getOutputStream();
                String str;
                do {
                        byte[] line = new byte[100];
                        System.in.read(line);
                        o.write(line);
                        i.read(line);
                        str = new String(line);
                        System.out.println(str.trim());
                } while ( !str.trim().equals("tchau") );
                s.close();
        }
        catch (Exception err) {
                System.err.println(err);
        }
   }
}
```

```
import java.io.*;
import java.net.*;

public class SimpleJavaServer {
   public static void main(String[] args)       {
        try {
                ServerSocket s = new ServerSocket(9999);
                String str;
                while (true) {
                        Socket c = s.accept();
                        InputStream i = c.getInputStream();
                        OutputStream o = c.getOutputStream();
                        do {
                                byte[] line = new byte[100];
                                i.read(line);
                                o.write(line);
                                str = new String(line);
                        } while ( !str.trim().equals("tchau") );
                        c.close();
                }
        }
        catch (Exception err){
           System.err.println(err);
        }
   }
}
```

Sockets em C/C++

- C é a linguagem “básica” para programação com sockets
- De maneira diferente de Java, programar com sockets em C/C++ envolve utilizar todas as chamadas da API

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

int main(int argc, char **argv) {
        int s;
        struct sockaddr_in dest;
        char msg_write[100], msg_read[100];
        s = socket(AF_INET, SOCK_STREAM, 0);
        bzero(&dest, sizeof(dest));
        dest.sin_family = AF_INET;
        dest.sin_port = htons(9999);
        inet_aton("127.0.0.1", (struct in_addr *) &(dest.sin_addr.s_addr));
        connect(s, (struct sockaddr*)&dest, sizeof(dest));
        do {
                bzero(&msg_write, sizeof(msg_write));
                scanf("%s", msg_write);
                write (s, msg_write, strlen(msg_write)+1);
                bzero(&msg_read, sizeof(msg_read));
                read (s, msg_read, 100);
                printf("%s\n", msg_read);
        } while (strcmp(msg_read, "!"));
   close(s);
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>

int main(void) {
 struct sockaddr_in info, info_cliente;
 int socket_entrada, socket_conexao;
 char mensagem; int tamanho_estrutura;
 socket_entrada = socket(AF_INET, SOCK_STREAM, 0);
 if(socket_entrada < 0) { printf("Vixe! Erro no socket!\n"); exit(1); }
 info.sin_family = AF_INET;
 info.sin_port = htons(9999);
 info.sin_addr.s_addr = INADDR_ANY;
 tamanho_estrutura = sizeof(info_cliente);
 if(bind(socket_entrada, (struct sockaddr *)&info, tamanho_estrutura)==0) {
  listen(socket_entrada, 5);
  while(1) {
   socket_conexao = accept(socket_entrada, (struct sockaddr *)&info_cliente, &tamanho_estrutura);
   mensagem='.';
   while(mensagem!='!') { // Finaliza conexao se mensagem e' um "!"
    read(socket_conexao, &mensagem, 1);
//    if(mensagem=='\n') printf("\n"); else printf("%c", mensagem);
    fflush(stdout);
    write(socket_conexao, &mensagem, 1);
   }
   close(socket_conexao);
//   printf("\n");
  }
 } else { printf("Vixe! Nao deu para vincular endereco ou porta!\n"); exit(1); }
 return(0);
}
```

Sockets sem Conexão (Java)

- Cliente:
  - `socket = new DatagramSocket( );`
  - `message = new DatagramPacket(msg,length,Addr,Port);`
  - `reply   = new DatagramPacket( new byte[100], 100 );`
  - `socket.send( message );`
  - `socket.receive( reply );`
  - `socket.close();`
- Servidor:
  - `socket = new DatagramSocket(porta);`
  - `socket.receive( message );`
  - `socket.send( message );`

Sockets sem Conexão (C)

- Cliente:
  - `s = socket(AF_INET, SOCK_DGRAM, 0);`
  - `sendto(s, msg, length, flags, destaddr, addrlen);`
  - `recvfrom(s, msg, length, flags, fromaddr, addrlen);`
- Servidor:
  - `s = socket(AF_INET, SOCK_DGRAM, 0);`
  - `bind(s, dest, sizeof(dest));`
  - `recvfrom(s, msg, length, flags, fromaddr, addrlen);`
  - `sendto(s, msg, length, flags, destaddr, addrlen);`