DNS: Domain Name System

Pessoas: muitos identificadores:

- ✓ CPF, nome, no. da Identidade

hospedeiros, roteadores Internet :

- ✓ endereço IP (32 bit) - usado p/ endereçar datagramas
- ✓ "nome", ex., jambo.ic.uff.br - usado por gente

P: como mapear entre nome e endereço IP?

Domain Name System:

- ➢ *base de dados distribuída* implementada na hierarquia de muitos *servidores de nomes*
- ➢ *protocolo de camada de aplicação* permite que hospedeiros, roteadores, servidores de nomes se comuniquem para *resolver* nomes (tradução endereço/nome)
  - ✓ note: função imprescindível da Internet implementada como protocolo de camada de aplicação
  - ✓ complexidade na borda da rede

DNS

- Roda sobre UDP e usa a porta 53
- Especificado nas RFCs 1034 e 1035 e atualizado em outras RFCs.

- Outros serviços:
  - ✓ apelidos para hospedeiros (aliasing)
  - ✓ apelido para o servidor de mails
  - ✓ distribuição da carga

Servidores de nomes DNS

Por que não centralizar o DNS?

- ponto único de falha
- volume de tráfego
- base de dados centralizada e distante
- manutenção (da BD)

Não é escalável!

- Nenhum servidor mantém todos os mapeamento nome-para-endereço IP

servidor de nomes local:

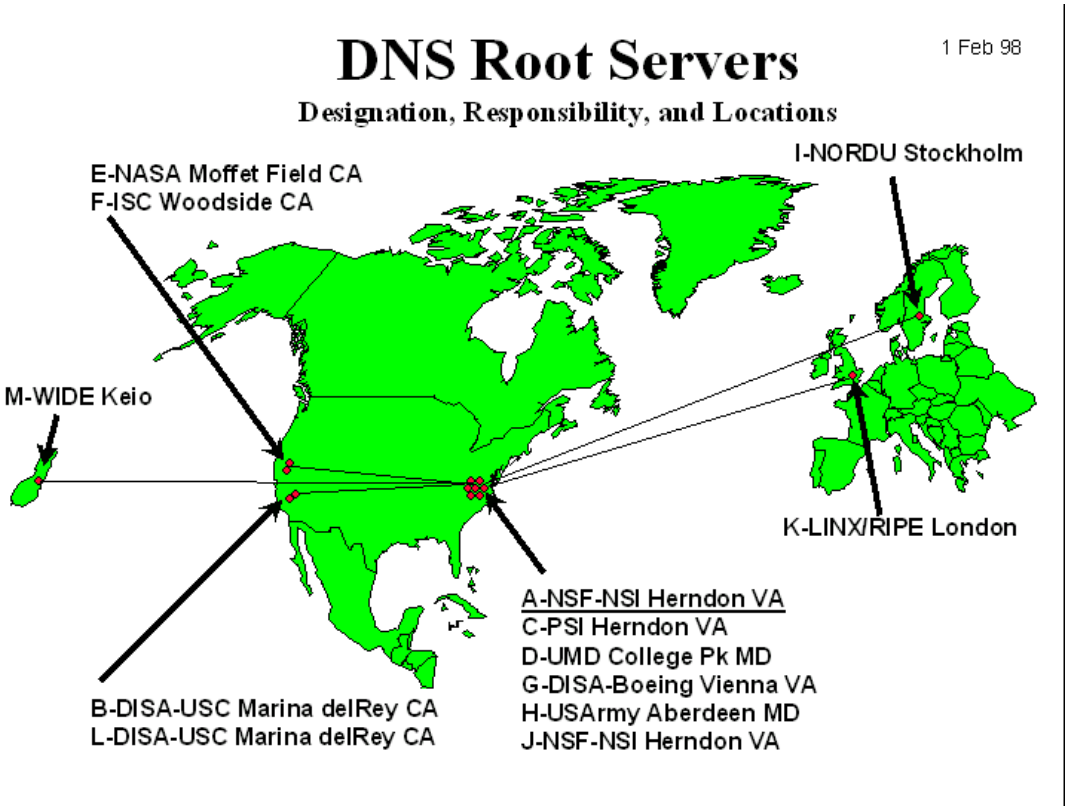
- ✓ cada provedor, empresa tem *servidor de nomes local (default)*
- ✓ pedido DNS de hospedeiro vai primeiro ao servidor de nomes local

servidor de nomes oficial:

- ✓ p/ hospedeiro: guarda nome, endereço IP dele
- ✓ pode realizar tradução nome/endereço para este nome

DNS: Servidores raiz

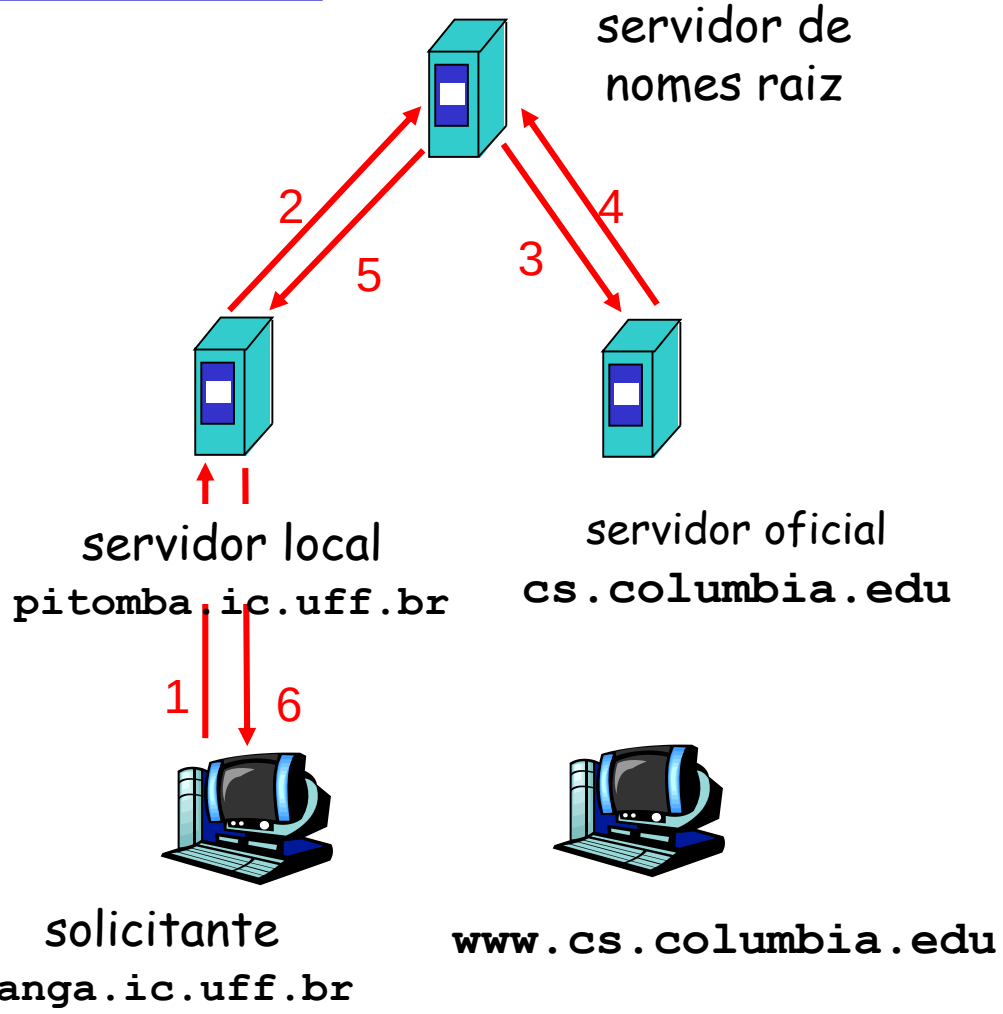
- procurado por servidor local que não consegue resolver o nome
- servidor raiz:
  - ✓ procura servidor oficial se mapeamento desconhecido
  - ✓ obtém tradução
  - ✓ devolve mapeamento ao servidor local
- ~ uma dúzia de servidores raiz no mundo



Exemplo simples do DNS

hospedeiro **`manga.ic.uff.br`** requer endereço IP de **`www.cs.columbia.edu`**

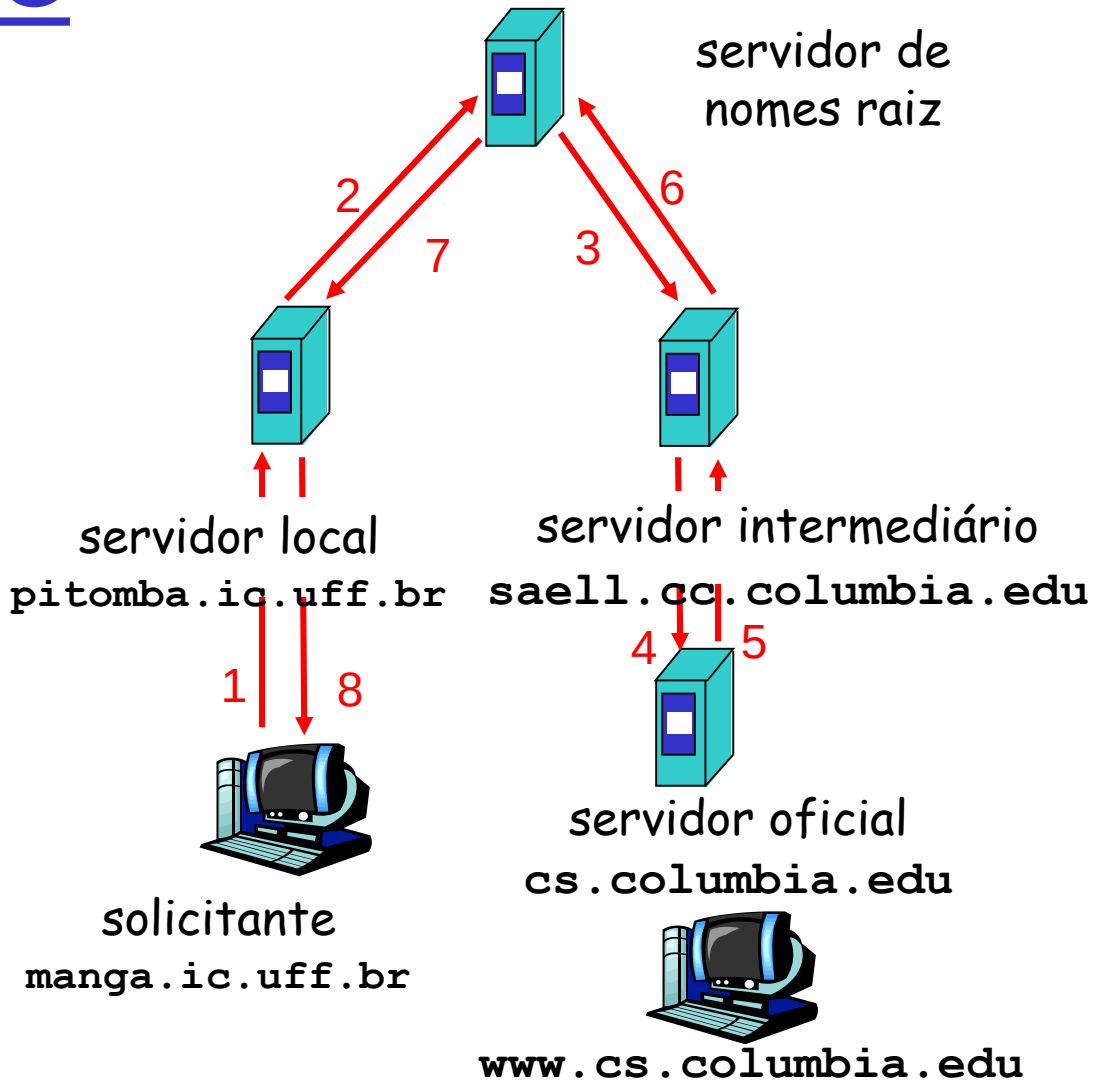
1. Contata servidor DNS local, **`pitomba.ic.uff.br`**
2. **`pitomba.ic.uff.br`** contata servidor raiz, se necessário
3. Servidor raiz contata servidor oficial **`cs.columbia.edu`**, se necessário



Exemplo de DNS

Servidor raiz:

- pode não conhecer o servidor de nomes oficial
- pode conhecer *servidor de nomes intermediário:* a quem contatar para descobrir o servidor de nomes oficial



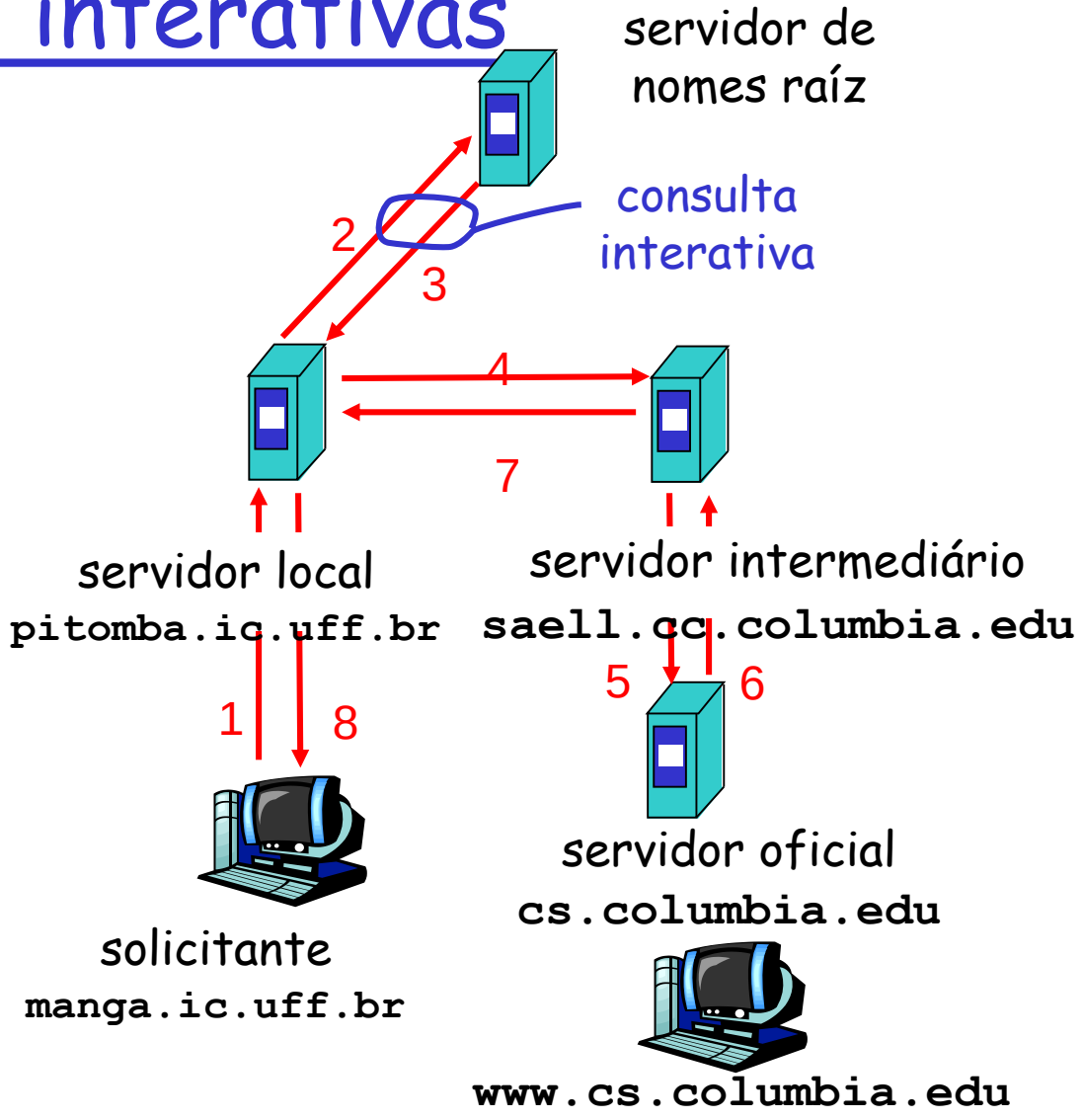
DNS: consultas interativas

consulta recursiva:

- transfere a responsabilidade de resolução do nome para o servidor de nomes contatado
- carga pesada?

consulta interativa:

- servidor consultado responde com o nome de um servidor de contato
- "Não conheço este nome, mas pergunte para esse servidor"



DNS: uso de cache, atualização de dados

- uma vez que um servidor qualquer aprende um mapeamento, ele o coloca numa *cache* local
  - ✓ futuras consultas são resolvidas usando dados da cache
  - ✓ entradas na cache são sujeitas a temporização (desaparecem depois de um certo tempo)
    ttl = time to live (sobrevida)
- estão sendo projetados pela IETF mecanismos de atualização/notificação dos dados
  - ✓ RFC 2136
  - ✓ http://www.ietf.org/html.charters/dnsind-charter.html

Registros DNS

DNS: BD distribuído contendo *registros de recursos* (RR)

formato RR: **(nome, valor, tipo, sobrevida)**

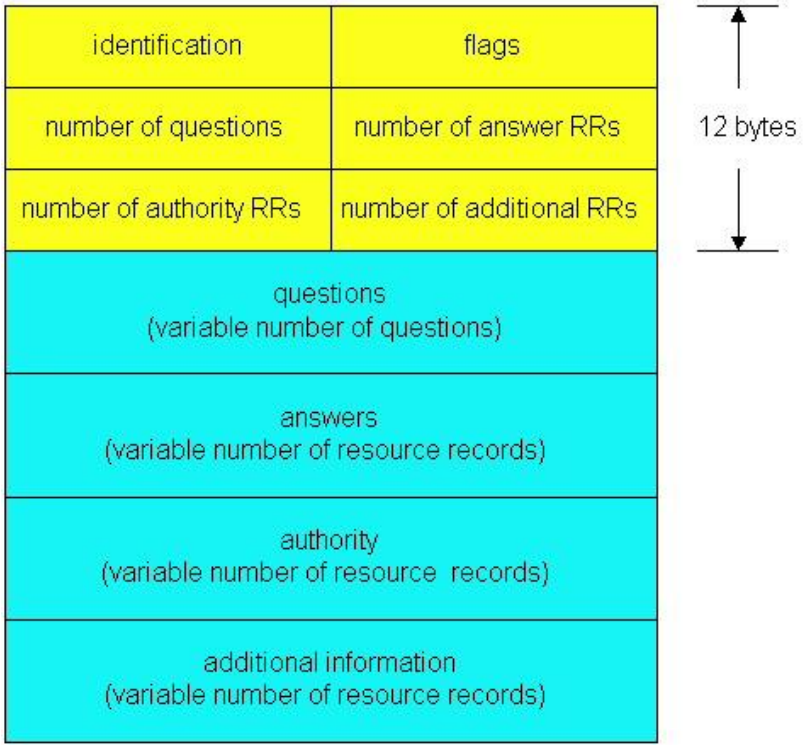
- Tipo=A
  - **nome** é nome de hospedeiro
  - **valor** é o seu endereço IP
- Tipo=NS
  - **nome** é domínio (p.ex. foo.com.br)
  - **valor** é endereço IP de servidor oficial de nomes para este domínio
- Tipo=CNAME
  - **nome** é nome alternativo (alias) para algum nome "canônico" (verdadeiro)
  - **valor** é o nome canônico
- Tipo=MX
  - **nome** é domínio
  - **valor** é nome do servidor de correio para este domínio

DNS: protocolo e mensagens

protocolo DNS: mensagens de *pedido* e *resposta*, ambas com o mesmo *formato de mensagem*

cabeçalho de msg

- identificação: ID de 16 bit para pedido, resposta ao pedido usa mesmo ID
- flags:
  - pedido ou resposta
  - recursão desejada
  - recursão permitida
  - resposta é oficial



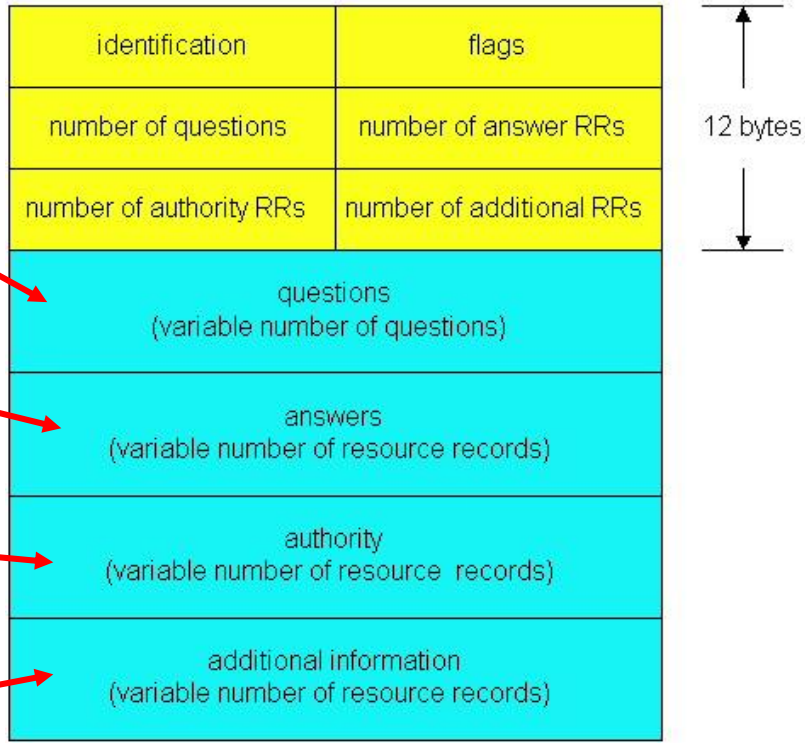
DNS: protocolo e mensagens

campos de nome, e de tipo num pedido

RRs em resposta ao pedido

registros para outros servidores oficiais

info adicional "relevante" que pode ser usada



Programação com sockets

Meta: aprender a construir aplicações cliente/servidor que se comunicam usando sockets

API Sockets

- apareceu no BSD4.1 UNIX em 1981
- são explicitamente criados, usados e liberados por apls
- paradigma cliente/servidor
- dois tipos de serviço de transporte via API Sockets
  - ✓ datagrama não confiável
  - ✓ fluxo de bytes, confiável

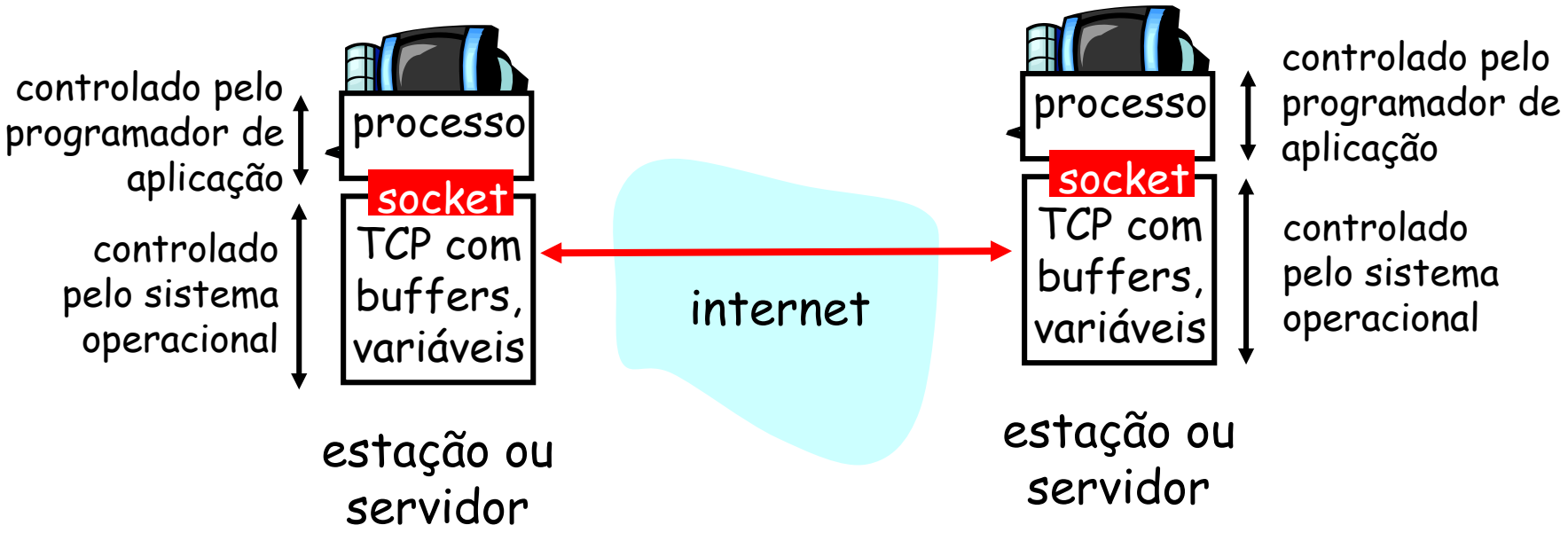
socket

uma interface (uma "porta"), local ao hospedeiro, criada por e pertencente à aplicação, e controlado pelo SO, através da qual um processo de aplicação pode tanto enviar como receber mensagens para/de outro processo de aplicação (remoto ou local)

Programação com sockets usando TCP

Socket: uma porta entre o processo de aplicação e um protocolo de transporte fim-a-fim (UDP ou TCP)

Serviço TCP: transferência confiável de bytes de um processo para outro



Programação com sockets usando TCP

Cliente deve contactar servidor

- processo servidor deve antes estar em execução
- servidor deve antes ter criado socket (porta) que aguarda contato do cliente

Cliente contacta servidor para:

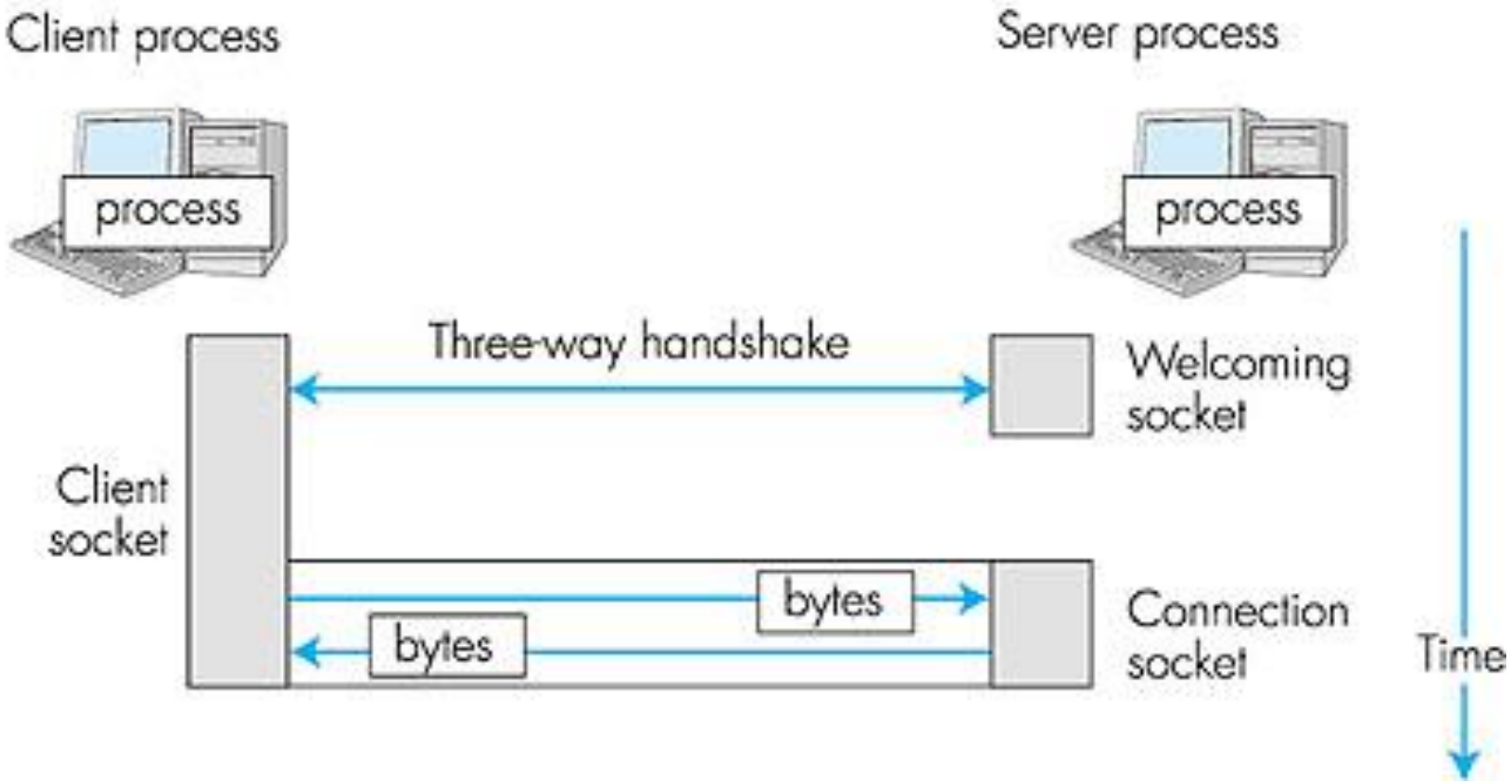
- criar socket TCP local ao cliente
- especificar endereço IP, número de porta do processo servidor

- Quando cliente cria socket: TCP do cliente estabelece conexão com TCP do servidor
- Quando contatado pelo cliente, o TCP do servidor cria socket novo para que o processo servidor possa se comunicar com o cliente
  - ✓ permite que o servidor converse com múltiplos clientes

ponto de vista da aplicação

TCP provê transferência confiável, ordenada de bytes ("tubo") entre cliente e servidor

Comunicação entre sockets

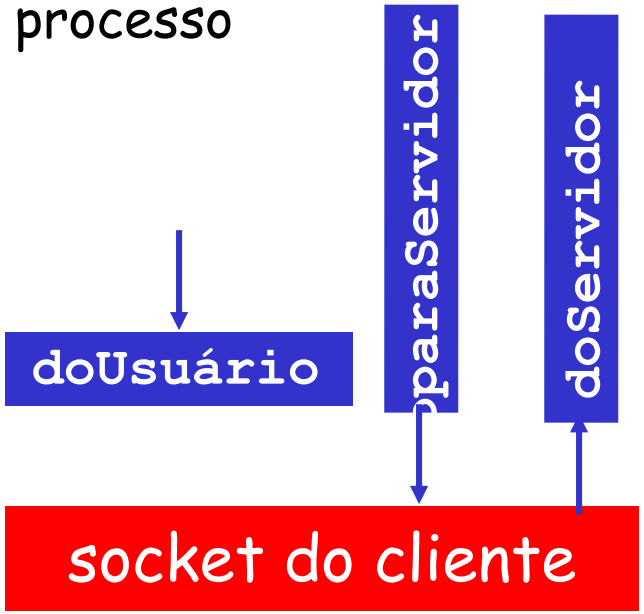


Exemplo de aplicação cliente-servidor

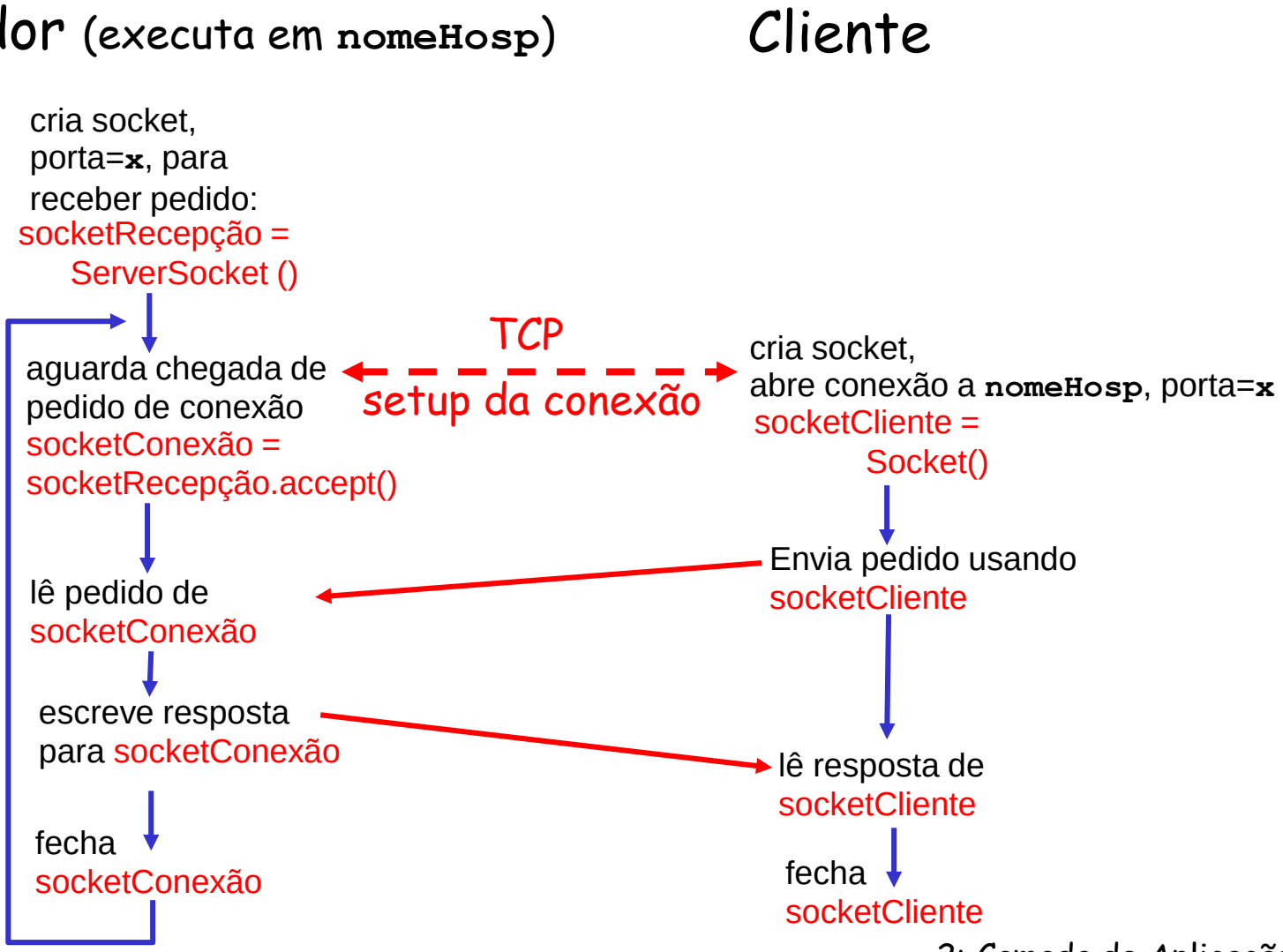
- cliente lê linha da entrada padrão (fluxo **doUsuário**), envia para servidor via socket (fluxo **paraServidor**)
- servidor lê linha do socket
- servidor converte linha para letras maiúsculas, devolve para o cliente
- cliente lê linha modificada do socket (fluxo **doServidor**), imprime-a

Fluxo de entrada: seqüência de bytes recebida pelo processo

Fluxo de saída: seqüência de bytes transmitida pelo processo



Interações cliente/servidor usando o TCP



Exemplo: cliente Java (TCP)

```
import java.io.*;
import java.net.*;
class ClienteTCP {

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

        BufferedReader doUsuario =
          new BufferedReader(new InputStreamReader(System.in));

        Socket socketCliente = new Socket("nomeHosp", 6789);

        DataOutputStream paraServidor =
          new DataOutputStream(socketCliente.getOutputStream());
```

Cria fluxo de entrada

Cria socket de cliente, conexão ao servidor

Cria fluxo de saída ligado ao socket

Exemplo: cliente Java (TCP), cont.

Cria fluxo de entrada ligado ao socket

```
BufferedReader doServidor =
  new BufferedReader(new
  InputStreamReader(socketCliente.getInputStream()));

frase = doUsuario.readLine();
```

Envia linha ao servidor

```
paraServidor.writeBytes(frase + '\n');
```

Lê linha do servidor

```
fraseModificada = doServidor.readLine();

System.out.println("Do Servidor: " + fraseModificada);

socketCliente.close();

    }
}
```

Exemplo: servidor Java (TCP)

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

class servidorTCP {

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

     ServerSocket socketRecepcao = new ServerSocket(6789);

     while(true) {

       Socket socketConexao = socketRecepcao.accept();

      BufferedReader doCliente =
        new BufferedReader(new
        InputStreamReader(socketConexao.getInputStream()));
```

Cria socket para recepção na porta 6789

Aguarda, no socket para recepção, o contato do cliente

Cria fluxo de entrada, ligado ao socket

Exemplo: servidor Java (TCP), cont

Cria fluxo de saída, ligado ao socket

Lê linha do socket

Escreve linha ao socket

```
DataOutputStream  paraCliente =
  new DataOutputStream(socketConexão.getOutputStream());

fraseCliente= doCliente.readLine();

fraseEmMaiusculas= fraseCliente.toUpperCase() + '\n';

paraClient.writeBytes(fraseEmMaiusculas);
            }
        }
    }
```

Final do elo while, volta ao início e aguarda conexão de outro cliente

Programação com sockets usando UDP

UDP: não tem "conexão" entre cliente e servidor

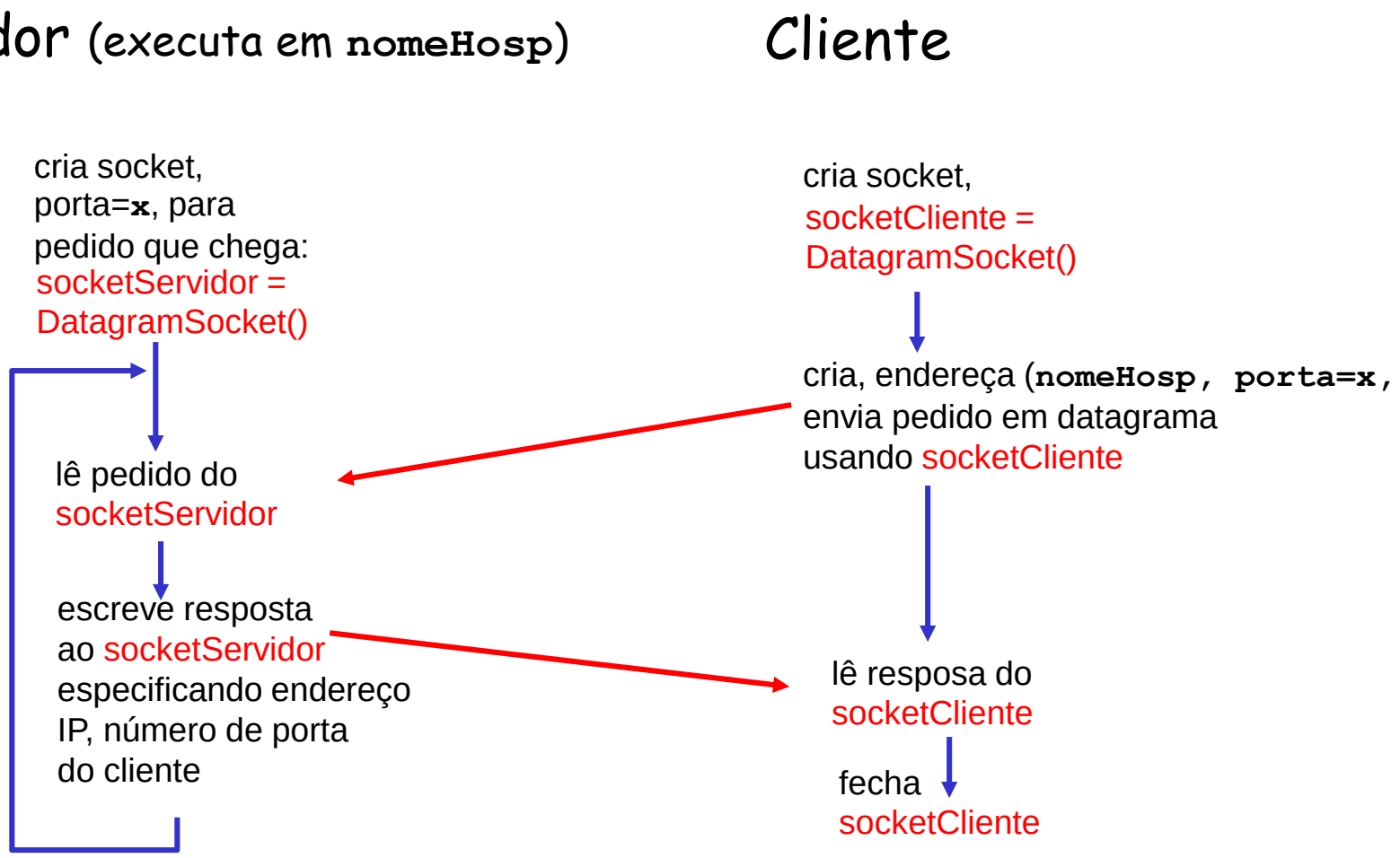
- não tem "handshaking"
- remetente coloca explicitamente endereço IP e porta do destino
- servidor deve extrair endereço IP, porta do remetente do datagrama recebido

UDP: dados transmitidos podem ser recebidos fora de ordem, ou perdidos

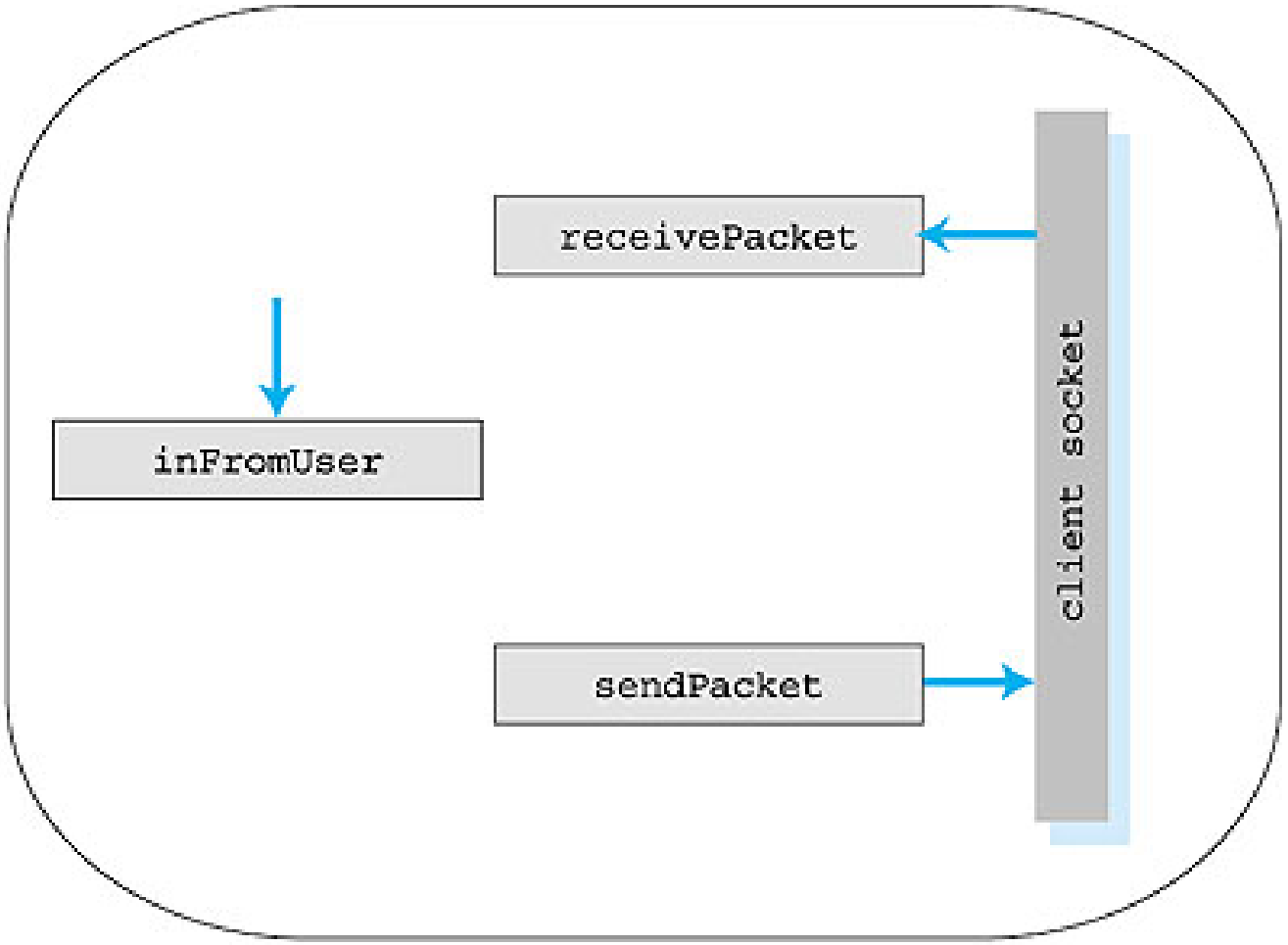
ponto de vista da aplicação

UDP provê transferência não confiável de grupos de bytes ("datagramas") entre cliente e servidor

Interações cliente/servidor usando o UDP



Cliente UDP



Exemplo: cliente Java (UDP)

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

class clienteUDP {
    public static void main(String args[]) throws Exception
    {
      BufferedReader do Usuario=
        new BufferedReader(new InputStreamReader(System.in));

      DatagramSocket socketCliente = new DatagramSocket();

      InetAddress IPAddress = InetAddress.getByName(”nomeHosp");

      byte[] sendData = new byte[1024];
      byte[] receiveData = new byte[1024];

      String frase = doUsuario.readLine();

      sendData = frase.getBytes();
```

Cria fluxo de entrada

Cria socket de cliente

Traduz nome de hospedeiro ao endereço IP usando DNS

Exemplo: cliente Java (UDP) cont.

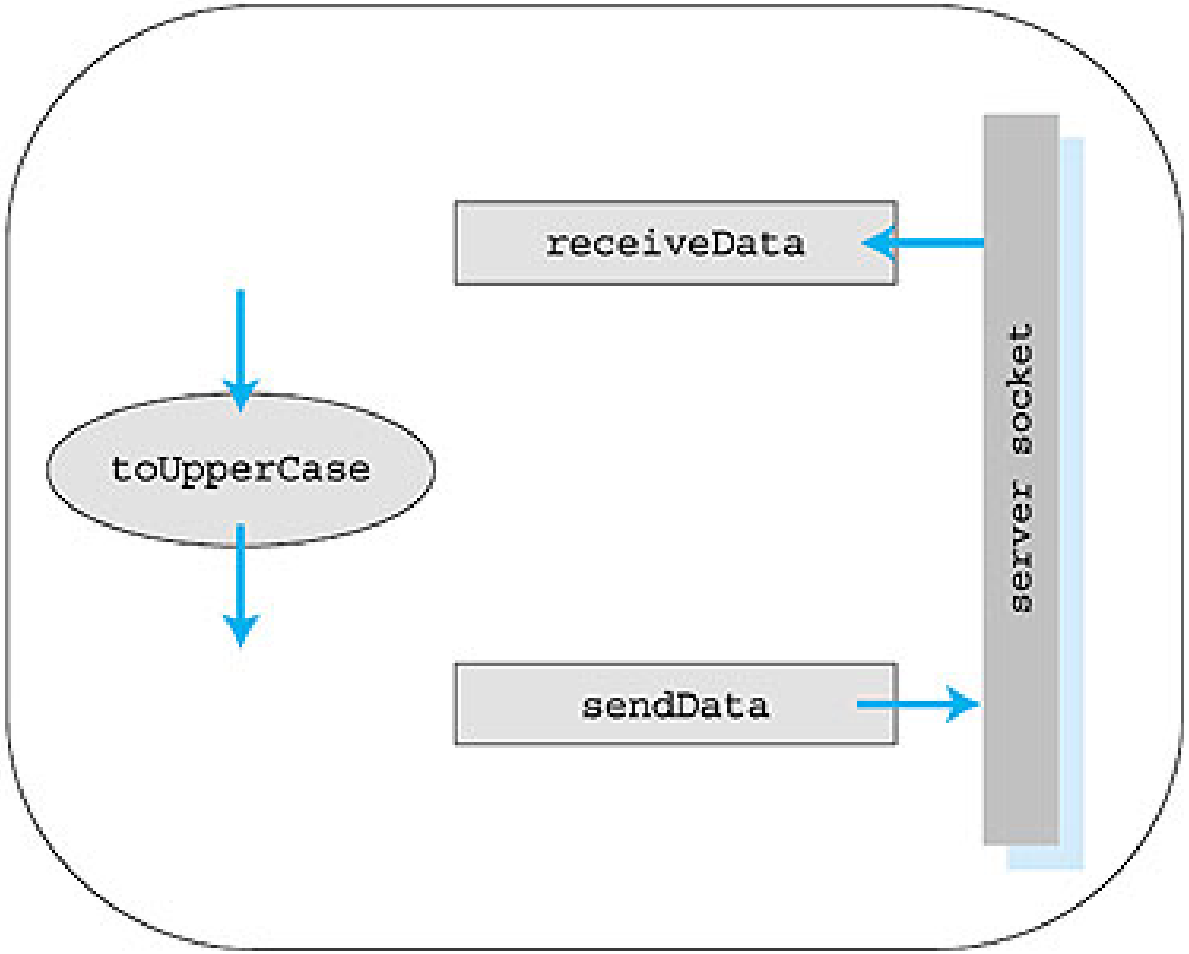
Cria datagrama com dados para enviar, comprimento, endereço IP, porta

Envia datagrama ao servidor

Lê datagrama do servidor

```
                DatagramPacket pacoteEnviado =
                  new DatagramPacket(dadosEnvio, dadosEnvio.length,
                         IPAddress, 9876);

                socketCliente.send(pacoteEnviado);

                DatagramPacket pacoteRecebido =
                   new DatagramPacket(dadosRecebidos, dadosRecebidos.length);

                socketCliente.receive(pacoteRecebido);

                String fraseModificada =
                    new String(pacoteRecebido.getData());

                System.out.println("Do Servidor:" + fraseModificada);
                socketCliente.close();
                }

        }
```

Servidor UDP



Exemplo: servidor Java (UDP)

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

class servidorUDP  {
  public static void main(String args[]) throws Exception
   {

      DatagramSocket socketServidor = new DatagramSocket(9876);

      byte[] dadosRecebidos = new byte[1024];
      byte[] dadosEnviados  = new byte[1024];

      while(true)
       {
          DatagramPacket pacoteRecebido =
           new DatagramPacket(dadosRecebidos,
                              dadosRecebidos.length);

          socketServidor.receive(pacoteRecebido);
```

Cria socket para datagramas na porta 9876

Aloca memória para receber datagrama

Recebe datagrama

Exemplo: servidor Java (UDP), cont

```
String frase = new String(pacoteRecebido.getData());

InetAddress IPAddress = pacoteRecebido.getAddress();

int porta = pacoteRecebido.getPort();

String fraseEmMaiusculas = frase.toUpperCase();

dadosEnviados = fraseEmMaiusculas.getBytes();

DatagramPacket pacoteEnviado =
   new DatagramPacket(dadosEnviados,
            dadosEnviados.length, IPAddress, porta);

socketServidor.send(pacoteEnviado);
    }
  }

}
```

Obtém endereço IP, no. de porta do remetente

Cria datagrama p/ enviar ao cliente

Escreve datagrama no socket

Fim do elo while, volta ao início e aguarda chegar outro datagrama

Servidor Web Simples

- Funções do servidor Web:
  - Trata apenas um pedido HTTP por vez
  - Aceita e examina o pedido HTTP
  - Recupera o arquivo pedido do sistema de arquivos do servidor
  - Cria uma mensagem de resposta HTTP consistindo do arquivo solicitado precedido por linhas de cabeçalho
  - Envia a resposta diretamente ao cliente.

Servidor Web Simples

Contém a classe StringTokenizer que é usada para examinar o pedido

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

class WebServer {
   public static void main(String argv[]) throws Exception
   {
      String requestMessageLine;
      String fileName;

      ServerSocket listenSocket = new ServerSocket(6789);
      Socket connectionSocket = listenSocket.accept();

      BufferedReader inFromClient =
        new BufferedReader(new InputStreamReader(
              connectionSocket.getInputStream()));

      DataOutputStream outToClient =
        new DataOutputStream(
              connectionSocket.getOutputStream());
```

Primeira linha da mensagem de pedido HTTP e Nome do arquivo solicitado

Aguarda conexão do cliente

Cria fluxo de Entrada

Cria fluxo de Saída

Servidor Web Simples, cont

Lê a primeira linha do pedido HTTP que deveria ter o seguinte formato: GET file_name HTTP/1.0

```
requestMessageLine = inFromClient.readLine();
```

Examina a primeira linha da mensagem para extrair o nome do arquivo

```
StringTokenizer tokenizedLine =
   new StringTokenizer(requestMessageLine);
if (tokenizedLine.nextToken().equals("GET")){
   fileName = tokenizedLine.nextToken();
   if (fileName.startsWith("/") == true )
         fileName = fileName.substring(1);

   File file = new File(fileName);
   int numOfBytes = (int) file.length();
```

Associa o fluxo inFile ao arquivo fileName

```
   FileInputStream inFile = new FileInputStream (
         fileName);
```

Determina o tamanho do arquivo e constrói um vetor de bytes do mesmo tamanho

```
   byte[] fileInBytes = new byte[];
   inFile.read(fileInBytes);
```

Servidor Web Simples, cont

Inicia a construção da mensagem de resposta

Transmissão do cabeçalho da resposta HTTP.

```
            outToClient.writeBytes(
                    "HTTP/1.0 200 Document Follows\r\n");

            if (fileName.endsWith(".jpg"))
                outToClient.writeBytes("Content-Type: image/jpeg\r\n");
            if (fileName.endsWith(".gif"))
                outToClient.writeBytes("Content-Type:
                        image/gif\r\n");
            outToClient.writeBytes("Content-Length: " + numOfBytes +
                    "\r\n");
            outToClient.writeBytes("\r\n");

            outToClient.write(fileInBytes, 0, numOfBytes);
            connectionSocket.close();
            }

        else System.out.println("Bad Request Message");
        }
    }
```

Capítulo 2: Resumo

Terminamos nosso estudo de aplicações de rede!

- Requisitos do serviço de aplicação:
  - confiabilidade, banda, retardo
- paradigma cliente-servidor
- modelo de serviço do transporte orientado a conexão, confiável da Internet: TCP
  - não confiável, datagramas: UDP
- Protocolos específicos:
  - http
  - ftp
  - smtp, pop3
  - dns
- programação c/ sockets
  - implementação cliente/servidor
  - usando sockets tcp, udp

Capítulo 2: Resumo

Mais importante: aprendemos sobre *protocolos*

- troca típica de mensagens pedido/resposta:
  - ✓ cliente solicita info ou serviço
  - ✓ servidor responde com dados, código de *status*
- formatos de mensagens:
  - ✓ cabeçalhos: campos com info sobre dados (metadados)
  - ✓ dados: info sendo comunicada
- msgs de controle X dados
  - ✓ na banda, fora da banda
- centralizado X descentralizado
- s/ estado X c/ estado
- transferência de msgs confiável X não confiável
- "complexidade na borda da rede"
- segurança: autenticação