Capítulo 2: Camada de aplicação

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 FTP
- 2.4 Correio eletrônico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Aplicações P2P
- 2.7 Programação de sockets com UDP
- 2.8 Programação de sockets com TCP

Programação de sockets

Objetivo: aprender a criar aplicação cliente-servidor que se comunica usando sockets

- criada, usada e liberada explicitamente pelas aplicações
- paradigma cliente-servidor
- dois tipos de serviços de transporte por meio da API socket:
  - UDP
  - TCP

socket

Uma interface *criada pela aplicação e controlada pelo SO* (uma "porta") na qual o processo da aplicação pode enviar e receber mensagens para/de outro processo da aplicação

Fundamentos de programação de socket

- servidor deve estar rodando antes que o cliente possa lhe enviar algo
- servidor deve ter um socket (porta) pelo qual recebe e envia segmentos
- da mesma forma, o cliente precisa de um socket

- socket é identificado localmente com um número de porta

- cliente precisa saber o endereço IP do servidor e o número de porta do socket

Programação de socket com UDP

UDP: sem "conexão" entre cliente e servidor

- sem "handshaking"
- emissor conecta de forma explícita endereço IP e porta do destino a cada segmento
- SO conecta endereço IP e porta do socket emissor a cada segmento
- Servidor pode extrair endereço IP, porta do emissor a partir do segmento recebido

ponto de vista da aplicação

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

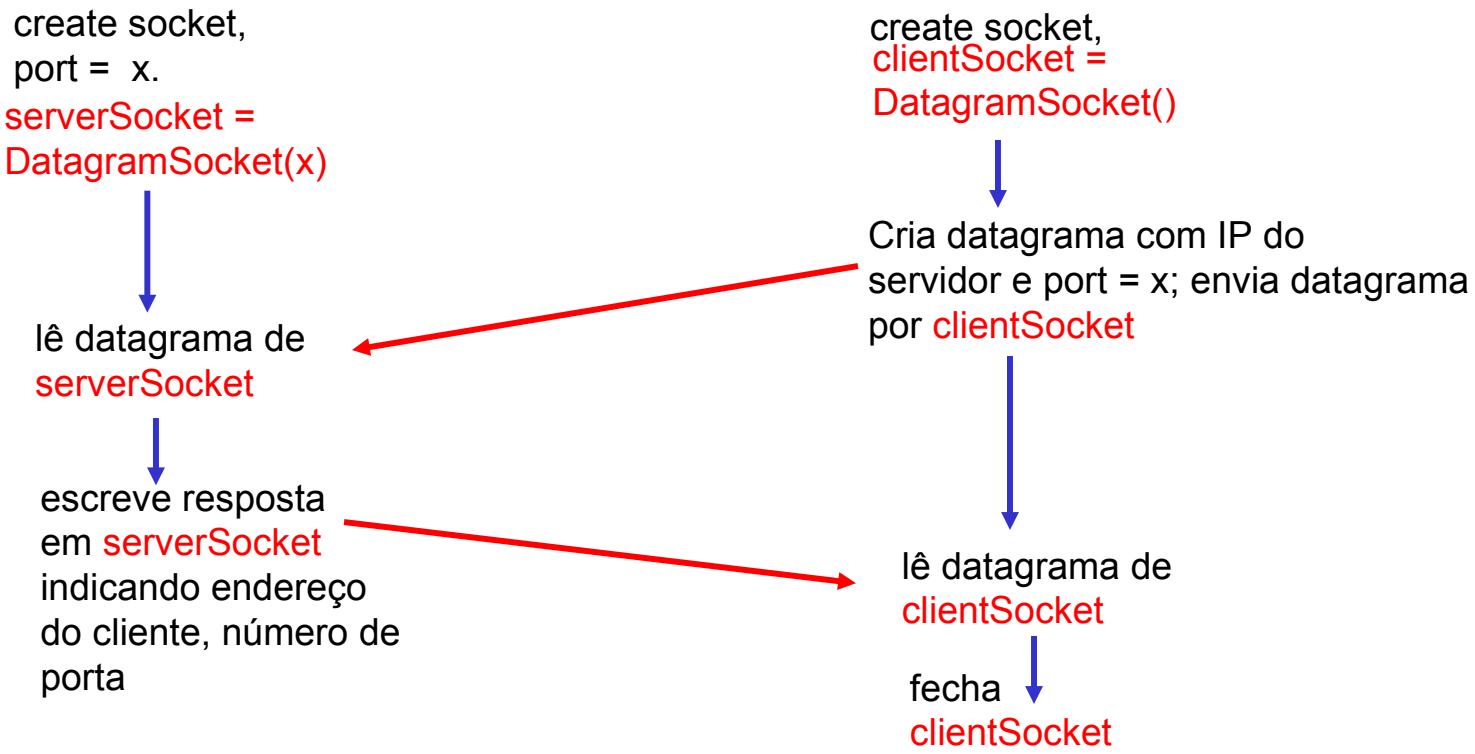
Exemplo em curso

- cliente:
  - usuário digita linha de texto
  - programa cliente envia linha ao servidor
- servidor:
  - servidor recebe linha de texto
  - coloca todas as letras em maiúsculas
  - envia linha modificada ao cliente
- cliente:
  - recebe linha de texto
  - apresenta

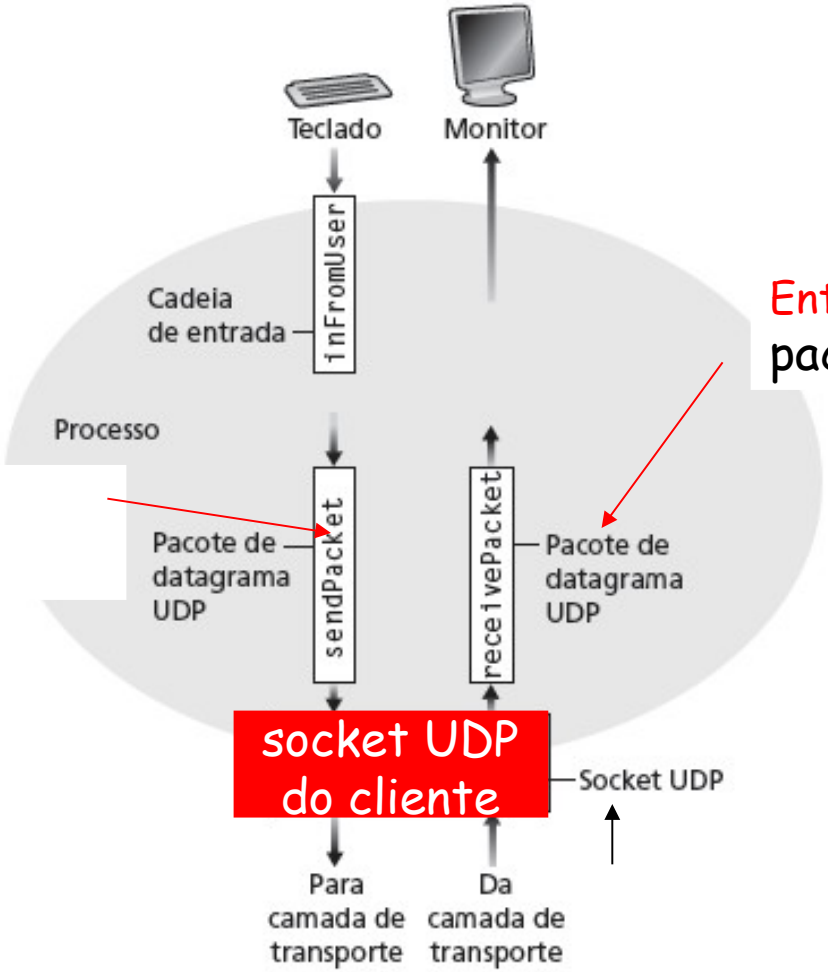
Interação de socket cliente/servidor: UDP

servidor (rodando em `hostid`)

cliente



Exemplo: cliente Java (UDP)



Entrada: recebe pacote

Saída: envia pacote

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

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

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

     DatagramSocket clientSocket = new DatagramSocket();

     InetAddress IPAddress = InetAddress.getByName("hostname");

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

     String sentence = inFromUser.readLine();

     sendData = sentence.getBytes();
```

cria cadeia de entrada

cria socket do cliente

traduz hostname para endereço IP usando DNS

cria datagrama com dados a enviar, tam., end. IP, porta

envia datagrama ao servidor

lê datagrama do servidor

```
DatagramPacket sendPacket =
   new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

clientSocket.send(sendPacket);

DatagramPacket receivePacket =
   new DatagramPacket(receiveData, receiveData.length);

clientSocket.receive(receivePacket);

String modifiedSentence =
    new String(receivePacket.getData());

System.out.println("FROM SERVER:" + modifiedSentence);
clientSocket.close();
}

}
```

Exemplo: servidor Java (UDP)

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

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

     DatagramSocket serverSocket = new DatagramSocket(9876);

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

     while(true)
      {

       DatagramPacket receivePacket =
          new DatagramPacket(receiveData, receiveData.length);

       serverSocket.receive(receivePacket);
```

cria socket de datagrama na porta 9876

cria espaço para datagrama recebido

recebe datagrama

```
String sentence = new String(receivePacket.getData());

InetAddress IPAddress = receivePacket.getAddress();

int port = receivePacket.getPort();

String capitalizedSentence = sentence.toUpperCase();

sendData = capitalizedSentence.getBytes();

DatagramPacket sendPacket =
   new DatagramPacket(sendData, sendData.length, IPAddress,
                      port);

serverSocket.send(sendPacket);
      }
   }
}
```

obtém end. IP # porta do emissor

cria datagrama p/ enviar ao cliente

escreve datagrama no socket

fim do loop while, retorna e espera outro datagrama

Observações sobre UDP

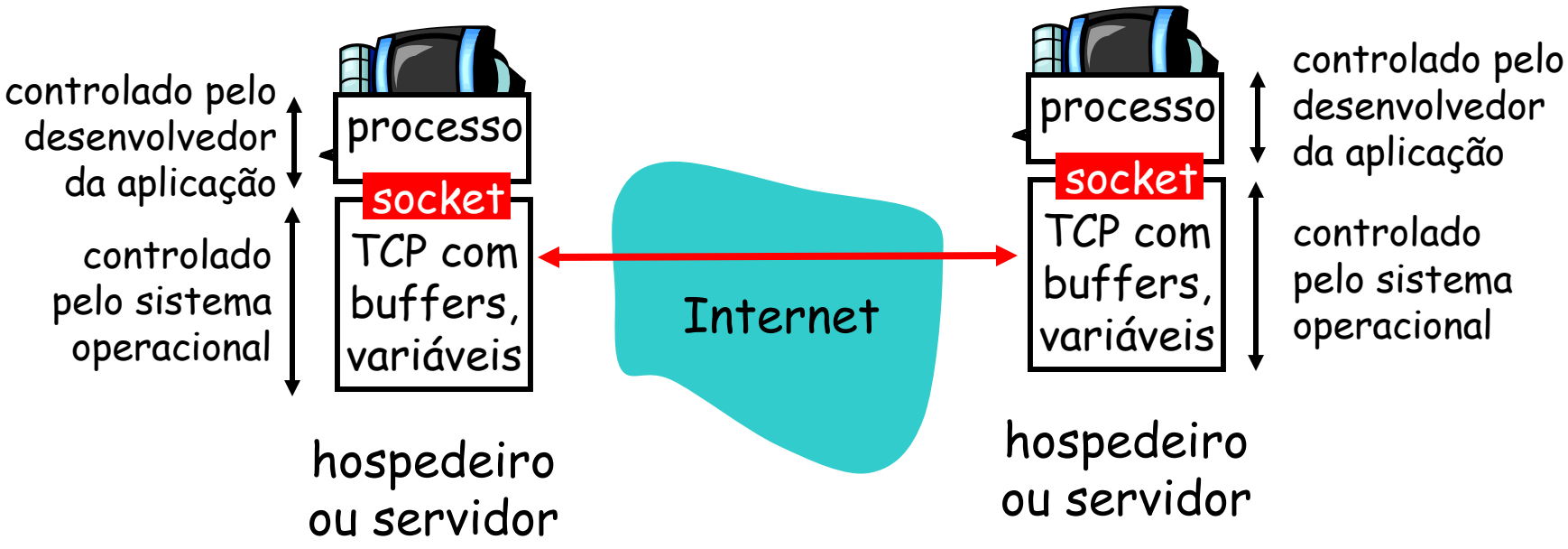
- cliente e servidor usam DatagramSocket
- IP e porta de destino são explicitamente conectados ao segmento
- O cliente não pode enviar um segmento ao servidor sem saber o endereço IP e número de porta do servidor
- Múltiplos clientes podem usar o servidor

Capítulo 2: Camada de aplicação

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 FTP
- 2.4 Correio eletrônico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Aplicações P2P
- 2.7 Programação de sockets com UDP
- 2.8 Programação de sockets com TCP

Programação de socket usando TCP

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



Programação de socket com TCP

cliente deve contactar servidor

- processo servidor primeiro deve estar rodando
- servidor deve ter criado socket (porta) que aceita contato do cliente

cliente contacta servidor:

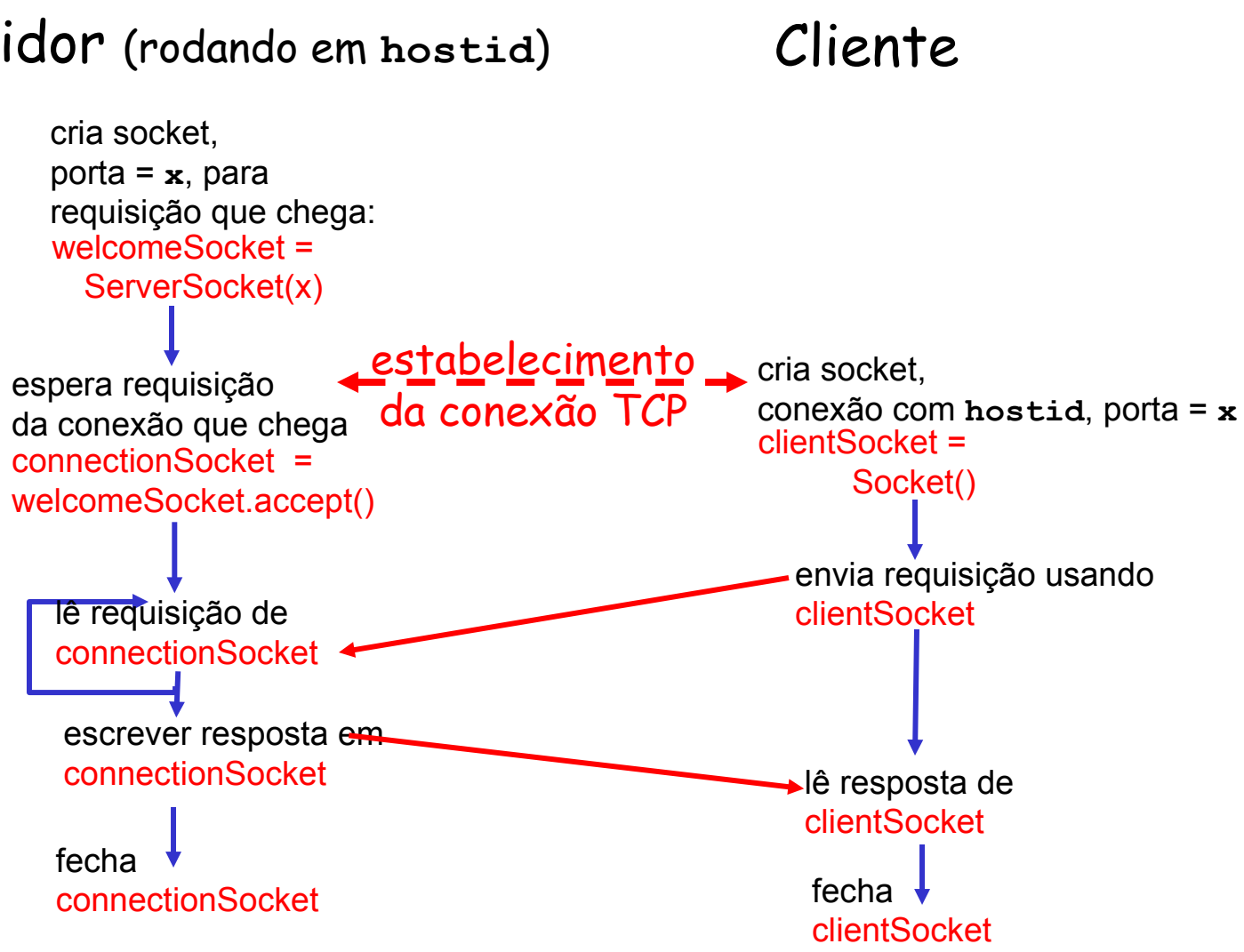
- criando socket TCP local ao cliente
- especificando endereço IP, # porta do processo servidor
- quando cliente cria socket: cliente TCP estabelece conexão com servidor TCP

- quando contactado pelo cliente, servidor TCP cria novo socket para processo servidor se comunicar com cliente
  - permite que servidor fale com múltiplos clientes
  - números de porta de origem usados para distinguir clientes

ponto de vista da aplicação

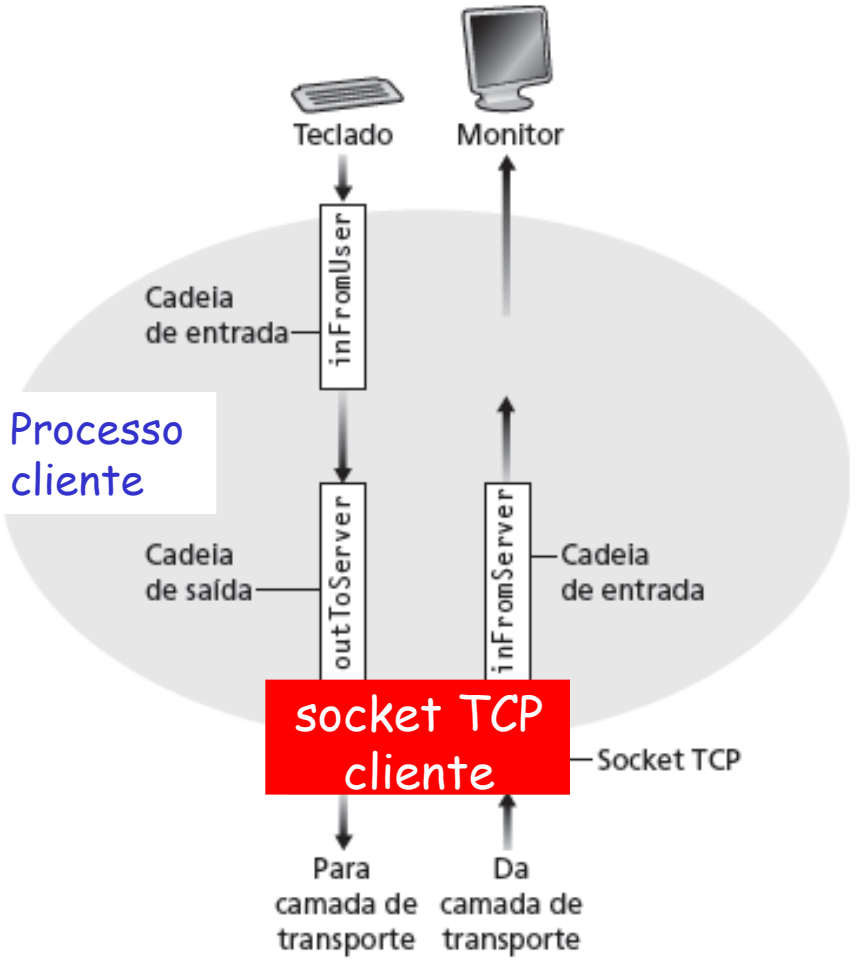
TCP oferece transferência de bytes confiável, em ordem ("pipe") entre cliente e servidor

Interação de socket cliente/servidor: TCP



Jargão de cadeia

- uma cadeia é uma sequência de caracteres que flui para dentro ou fora de um processo.
- uma cadeia de entrada está conectada a uma fonte de entrada para o processo, p. e., teclado ou socket.
- uma cadeia de saída está conectada a uma fonte de saída, p. e., monitor ou socket.



Programação de socket com TCP

Exemplo de apl. cliente-servidor:

1) cliente lê linha da entrada padrão (cadeia **`inFromUser`**), envia ao servidor via socket (cadeia **`outToServer`**)
2) servidor lê linha do socket
3) servidor converte linha para maiúsculas, envia de volta ao cliente
4) cliente lê, imprime linha modificada do socket (cadeia **`inFromServer`**)

Exemplo: cliente Java (TCP)

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

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

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

        Socket clientSocket = new Socket("hostname", 6789);

        BufferedReader inFromServer =
          new BufferedReader(new
          InputStreamReader(clientSocket.getInputStream()));
```

cria cadeia de entrada

cria socket cliente, conexão com servidor

cria cadeia de entrada conectada ao socket

cria cadeia de saída conectada ao socket

envia linha ao servidor

lê linha do servidor

```
DataOutputStream outToServer =
 new DataOutputStream(clientSocket.getOutputStream());

sentence = inFromUser.readLine();

outToServer.writeBytes(sentence + '\n');

modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();

    }
}
```

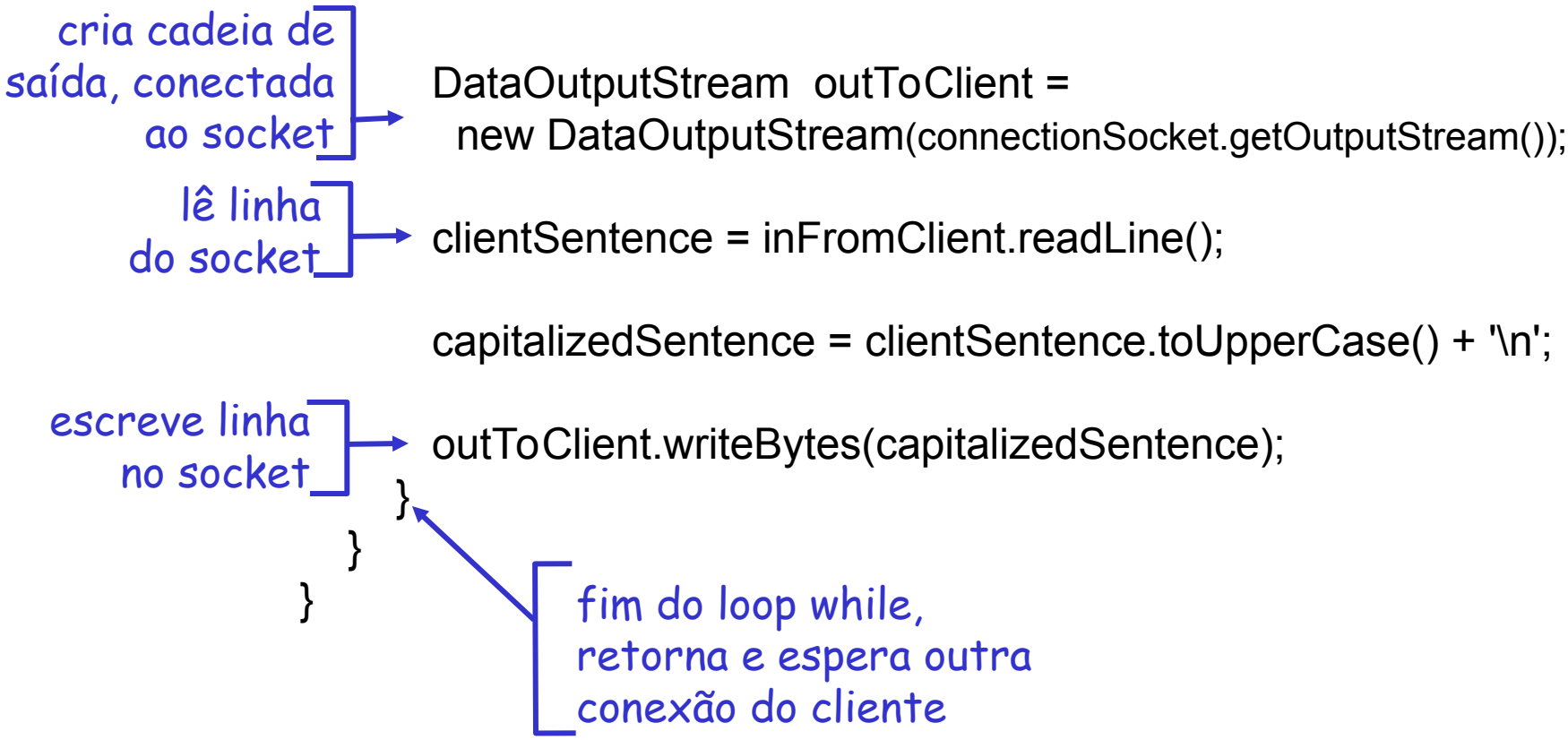
Exemplo: servidor Java (TCP)

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

class TCPServer {

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

     ServerSocket welcomeSocket = new ServerSocket(6789);

     while(true) {

       Socket connectionSocket = welcomeSocket.accept();

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

cria socket de apresentação na porta 6789

espera no socket de apresentação pelo contato do cliente

cria cadeia de entrada, conectada ao socket



Observações sobre TCP

- servidor tem dois tipos de sockets:
  - ServerSocket e connectionSocket
- quando o cliente bate na "porta" de serverSocket, servidor cria connectionSocket e completa conexão TCP.
- IP de destino e porta não são explicitamente conectados ao segmento.
- Múltiplos clientes podem usar o servidor.

Capítulo 2:
Camada de aplicação

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 FTP
- 2.4 Correio eletrônico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Aplicações P2P
- 2.7 Programação de sockets com UDP
- 2.8 Programação de sockets com TCP

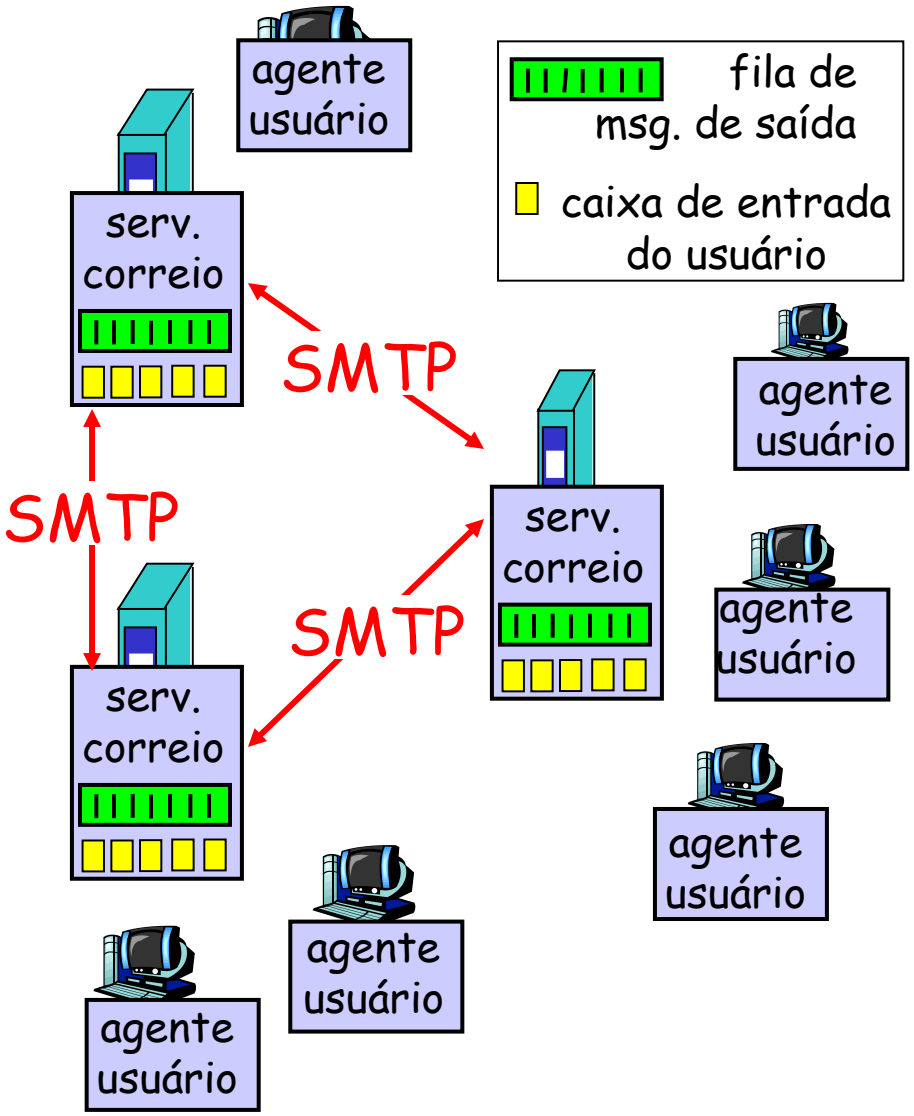
Correio eletrônico

Três componentes principais:

- agentes do usuário
- servidores de correio
- Simple Mail Transfer Protocol: SMTP

Agente do usuário

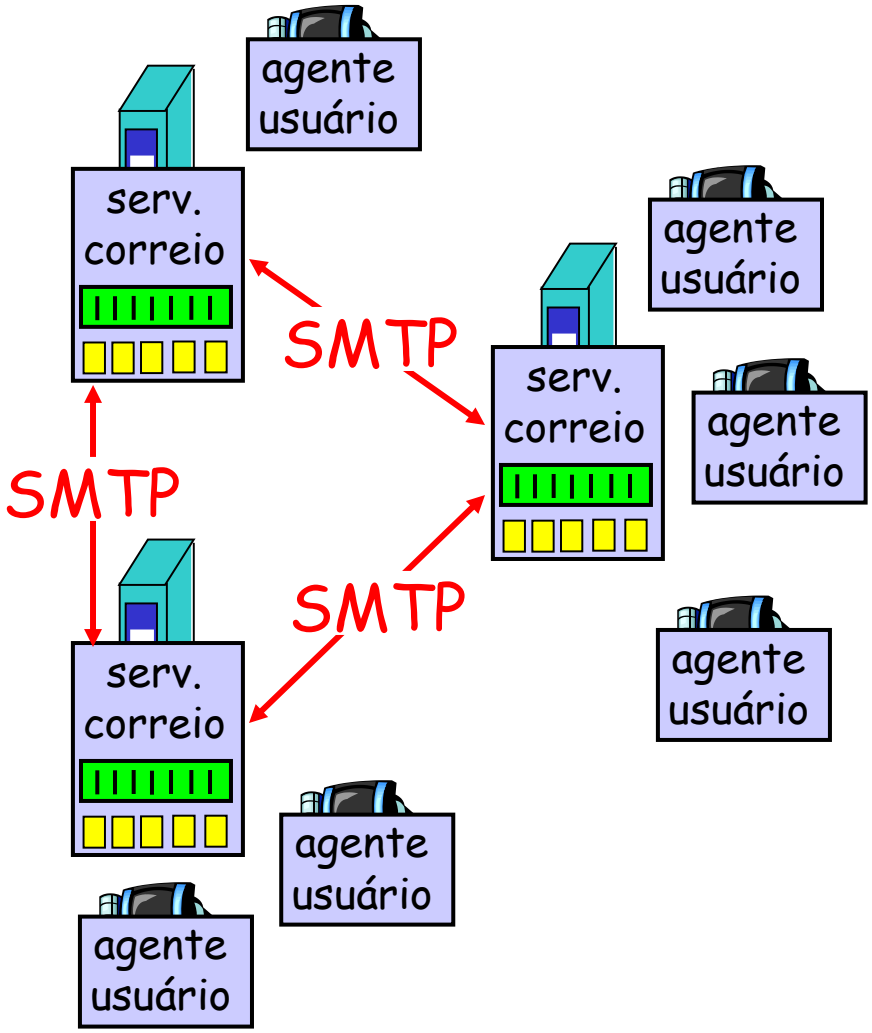
- também chamado "leitor de correio"
- redigir, editar, ler mensagens de correio eletrônico
- p. e., Outlook, Mozilla Thunderbird
- mensagens entrando e saindo armazenadas no servidor



Correio eletrônico

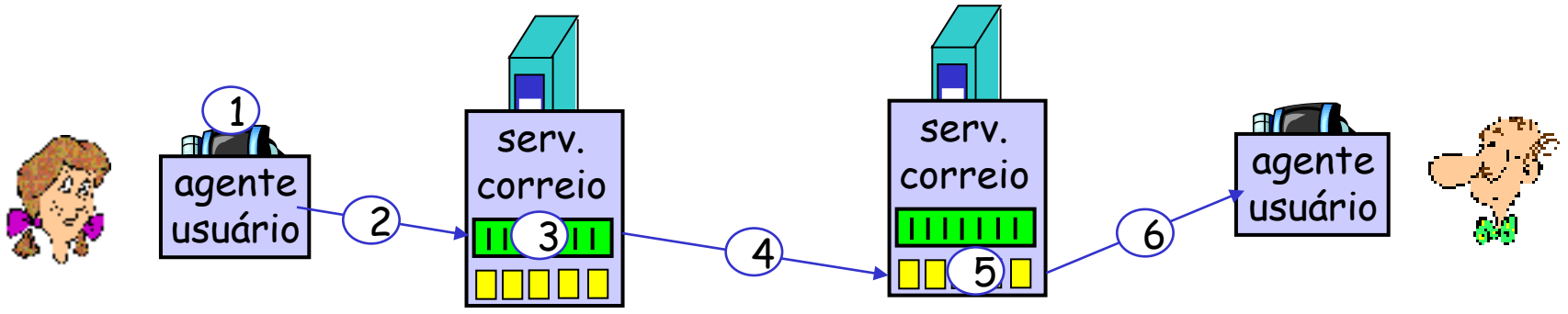
servidores de correio

- caixa de correio contém mensagens que chegam para o usuário
- fila de mensagens com mensagens de correio a serem enviadas
- protocolo SMTP entre servidores de correio para enviar mensagens de e-mail
  - cliente: agente usuário servidor: servidor de envio de correio
  - cliente: servidor de envio de correio servidor: servidor de recepção de correio

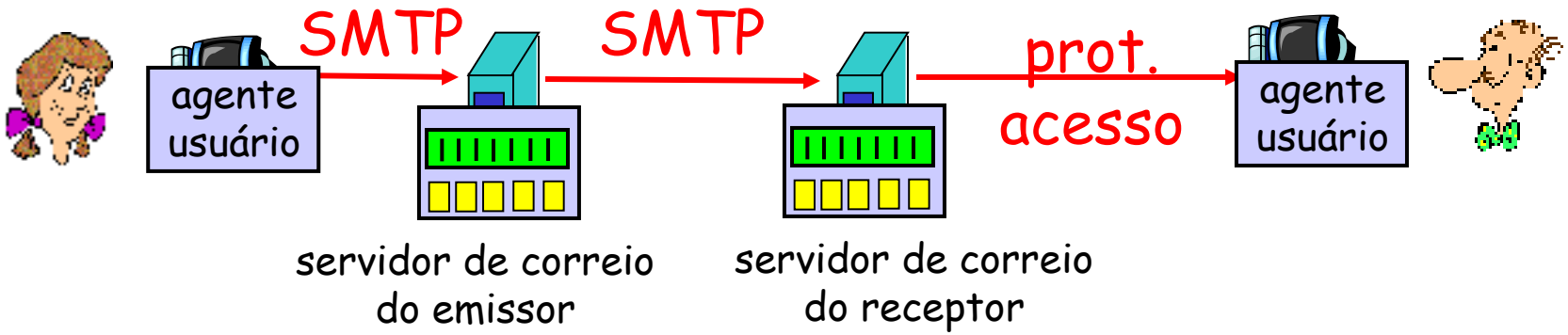


Cenário: Alice envia mensagem a Bob

1) Alice usa AU para redigir mensagem "para" `bob@algumaescola.edu`
2) O AU de Alice envia mensagem ao seu servidor de correio, que é colocada na fila de mensagens
3) Lado cliente do SMTP abre conexão TCP com servidor de correio de Bob
4) Cliente SMTP envia mensagem de Alice pela conexão TCP
5) Servidor de correio de Bob coloca mensagem na caixa de correio de Bob
6) Bob chama seu agente do usuário para ler mensagem



Protocolos de acesso de correio



- SMTP: remessa/armazenamento no servidor do receptor
- protocolo de acesso ao correio: recuperação do servidor
  - POP: Post Office Protocol [RFC 1939]
    - autorização (agente <--> servidor) e download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - mais recursos (mais complexo)
    - manipulação de msgs armazenadas no servidor
  - HTTP: gmail, Hotmail, Yahoo! Mail etc.