

Camada de Aplicação

Kalinka Castelo Branco

Slides Baseados nos slides do livro: Redes de Computadores e a Internet -
Kurose e Ross

Capítulo 2: Camada de Aplicação

Metas do capítulo:

- ❑ aspectos conceituais e de implementação de protocolos de aplicação em redes
 - modelos de serviço da camada de transporte
 - paradigma cliente servidor
 - paradigma *peer-to-peer*
- ❑ aprenda sobre protocolos através do estudo de protocolos populares da camada de aplicação:
 - HTTP
 - FTP
 - SMTP/ POP3/ IMAP
 - DNS
- ❑ a programação de aplicações de rede
 - programação usando a API de *sockets*

Algumas aplicações de rede

- ❑ E-mail
- ❑ Web
- ❑ Instant messaging
- ❑ Login remoto
- ❑ Compartilhamento de arquivos P2P
- ❑ Jogos de rede multi-usuários
- ❑ Vídeo-clipes armazenados
- ❑ Voz sobre IP
- ❑ Vídeo conferência em tempo real
- ❑ Computação paralela em larga escala
- ❑ ?
- ❑ ?
- ❑ ?

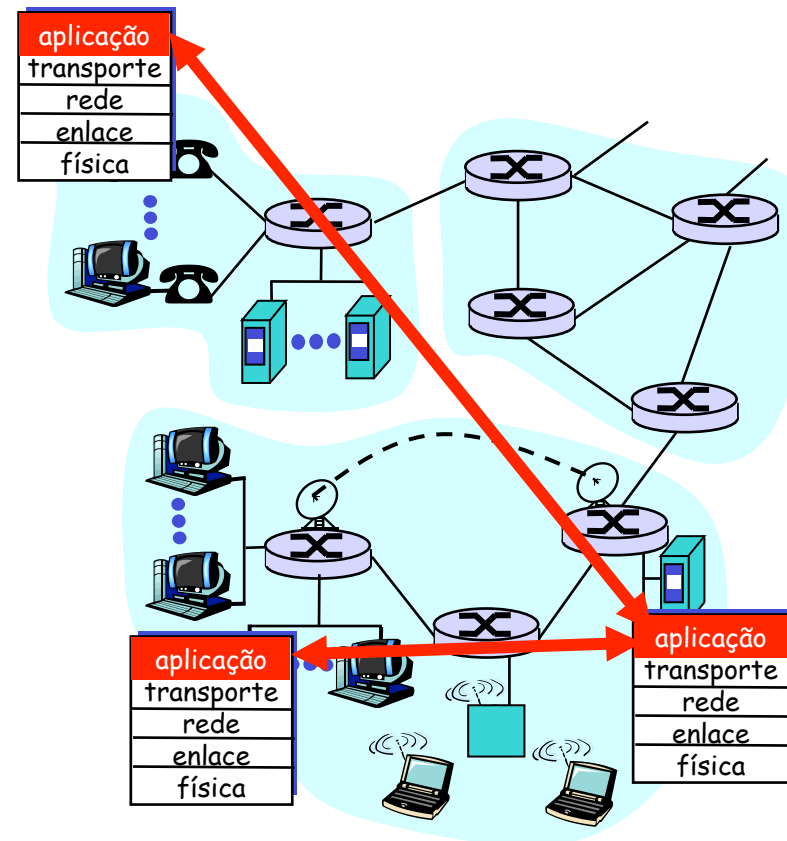
Criando uma aplicação de rede

Programas que

- Executam em diferentes sistemas finais
- Comunicam-se através da rede
- p.ex., Web: servidor Web se comunica com o navegador

Programas não relacionados ao núcleo da rede

- Dispositivos do núcleo da rede não executam aplicações de usuários
- Aplicações nos sistemas finais permite rápido desenvolvimento e disseminação



Arquiteturas das aplicações

- ❑ Cliente-servidor
- ❑ Peer-to-peer (P2P)
- ❑ Híbrido de cliente-servidor e P2P

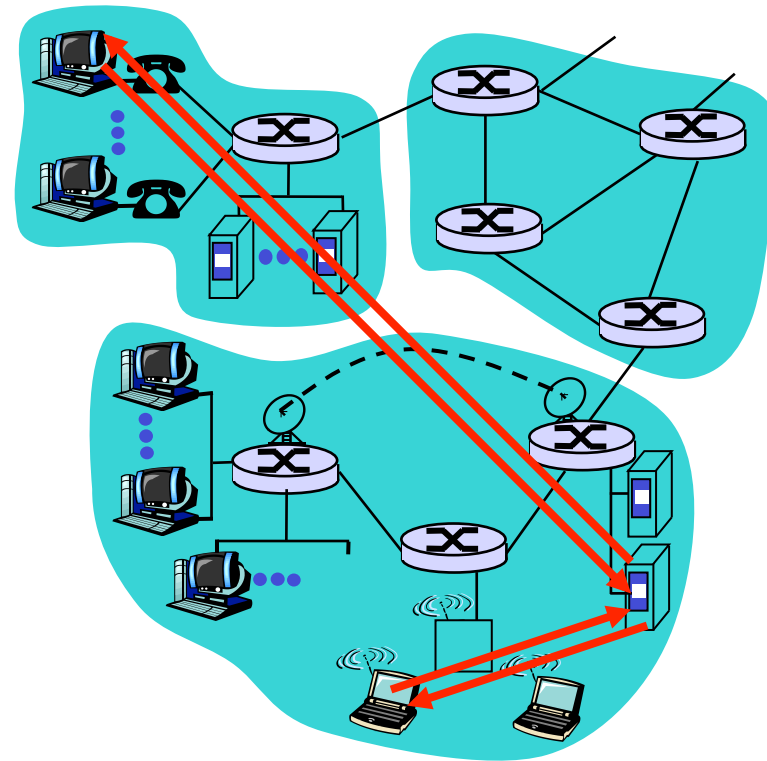
Arquitetura cliente-servidor

Servidor:

- ❑ Sempre ligado
- ❑ Endereço IP permanente
- ❑ Escalabilidade com *server farms*

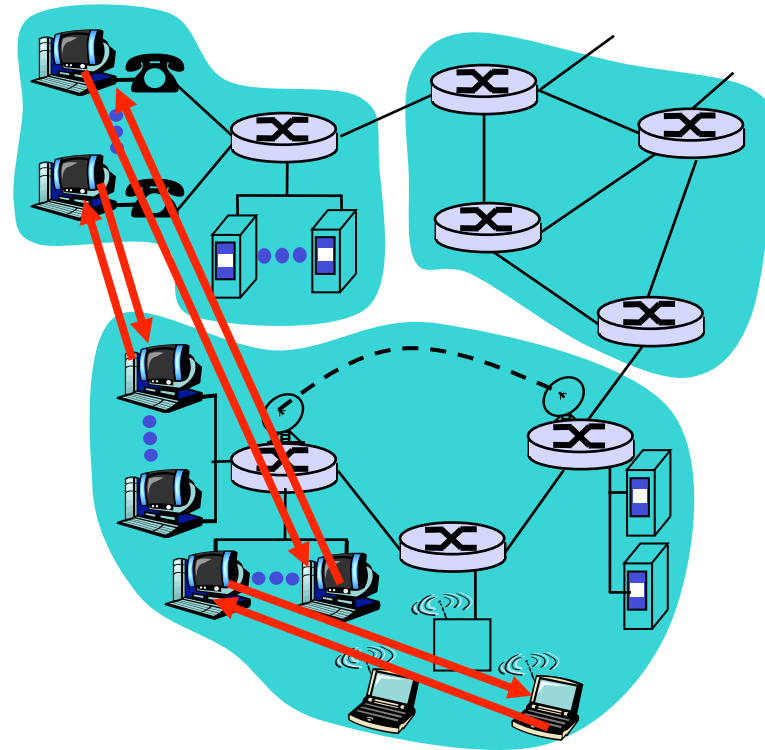
Cliente:

- ❑ Comunica-se com o servidor
- ❑ Pode estar conectado intermitentemente
- ❑ Pode ter endereços IP dinâmicos
- ❑ Não se comunica diretamente com outros clientes



Arquitetura P2P pura

- ❑ Não há servidor sempre ligado
- ❑ Sistemas finais arbitrários se comunicam diretamente
- ❑ Pares estão conectados intermitentemente e mudam endereços IP
- ❑ Exemplo: Gnutella



Altamente escalável
Porém, difícil de gerenciar

Híbrido de cliente-servidor e P2P

Napster

- Transferência de arquivos P2P
- Busca de arquivos centralizada:
 - Pares registram conteúdo no servidor central
 - Pares consultam o mesmo servidor central para localizar conteúdo

Instant messaging

- Conversa entre usuários P2P
- Localização e detecção de presença centralizadas:
 - Usuários registram o seu endereço IP junto ao servidor central quando ficam online
 - Usuários consultam o servidor central para encontrar endereços IP dos contatos

Processos em comunicação

Processo: programa que executa num hospedeiro

- processos no mesmo hospedeiro se comunicam usando **comunicação entre processos** definida pelo sistema operacional (SO)

- processos em hospedeiros distintos se comunicam trocando **mensagens através da rede**

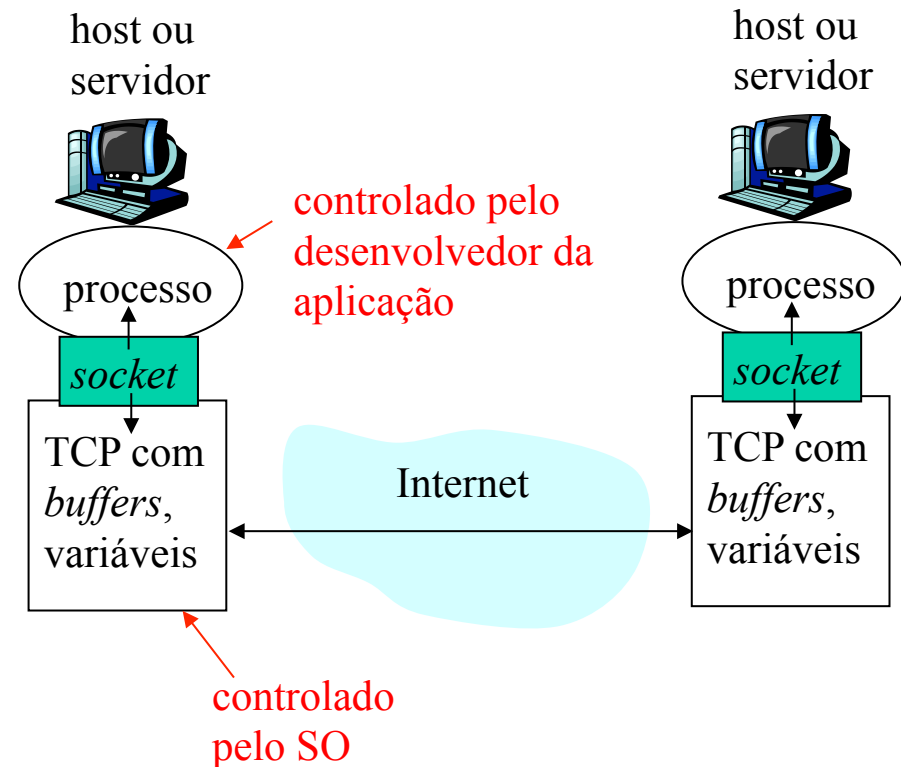
Processo cliente:
processo que inicia a comunicação

Processo servidor:
processo que espera para ser contatado

- Nota: aplicações com arquiteturas P2P possuem processos clientes e processos servidores

Sockets

- ❑ Os processos enviam/recebem mensagens para/dos seus *sockets*
- ❑ Um socket é análogo a uma porta
 - Processo transmissor envia a mensagem através da porta
 - O processo transmissor assume a existência da infraestrutura de transporte no outro lado da porta que faz com que a mensagem chegue ao *socket* do processo receptor



- ❑ API: (1) escolha do protocolo de transporte; (2) habilidade para fixar alguns parâmetros

Endereçando os processos

- ❑ Para que um processo receba mensagens, ele deve possuir um identificador
- ❑ Cada *host* possui um endereço IP único de 32 bits
- ❑ **P:** o endereço IP do *host* no qual o processo está sendo executado é suficiente para identificar o processo?
- ❑ **Resposta:** Não, muitos processos podem estar executando no mesmo *host*
- ❑ O identificador inclui tanto o endereço IP quanto os **números das portas** associadas com o processo no *host*.
- ❑ Exemplo de números de portas:
 - Servidor HTTP: 80
 - Servidor de Correio: 25

Os protocolos da camada de aplicação definem

- ❑ Tipos de mensagens trocadas, ex. mensagens de pedido e resposta
- ❑ Sintaxe dos tipos das mensagens: campos presentes nas mensagens e como são identificados
- ❑ Semântica dos campos, i.e., significado da informação nos campos
- ❑ Regras para quando os processos enviam e respondem às mensagens

Protocolos de domínio público:

- ❑ definidos em RFCs
- ❑ Permitem a interoperação
- ❑ ex, HTTP e SMTP

Protocolos proprietários:

- ❑ Ex., KaZaA

De que serviço de transporte uma aplicação precisa?

Perda de dados

- ❑ algumas apls (p.ex. áudio) podem tolerar algumas perdas
- ❑ outras (p.ex., transf. de arquivos, telnet) requerem transferência 100% confiável

Largura de banda

- ❑ algumas apls (p.ex., multimídia) requerem quantia mínima de banda para serem "viáveis"
- ❑ outras apls ("apls elásticas") conseguem usar qq quantia de banda disponível

Temporização

- ❑ algumas apls (p.ex., telefonia Internet, jogos interativos) requerem baixo retardo para serem "viáveis"

Requisitos do serviço de transporte de apls comuns

Aplicação	Perdas	Banda	Sensibilidade temporal
transferência de arqs	sem perdas	elástica	não
correio	sem perdas	elástica	não
documentos WWW	sem perdas	elástica	não
áudio/vídeo de tempo real	tolerante	áudio: 5Kb-1Mb vídeo: 10Kb-5Mb	sim, 100's mseg
áudio/vídeo gravado	tolerante	como anterior	sim, alguns segs
jogos interativos	tolerante	> alguns Kbps	sim, 100's mseg
apls financeiras	sem perdas	elástica	sim e não

Serviços providos por protocolos de transporte Internet

Serviço TCP:

- ❑ *orientado a conexão:* inicialização requerida entre cliente e servidor
- ❑ *transporte confiável* entre processos remetente e receptor
- ❑ *controle de fluxo:* remetente não vai "afogar" receptor
- ❑ *controle de congestionamento:* estrangular remetente quando a rede estiver carregada
- ❑ *não provê:* garantias temporais ou de banda mínima

Serviço UDP:

- ❑ transferência de dados não confiável entre processos remetente e receptor
 - ❑ não provê: estabelecimento da conexão, confiabilidade, controle de fluxo, controle de congestionamento, garantias temporais ou de banda mínima
- P: Qual é o interesse em ter um UDP?

Apls Internet: seus protocolos e seus protocolos de transporte

Aplicação	Protocolo da camada de apl	Protocolo de transporte usado
correio eletrônico	SMTP [RFC 2821]	TCP
acesso terminal remoto	telnet [RFC 854]	TCP
WWW	HTTP [RFC 2616]	TCP
transferência de arquivos	ftp [RFC 959]	TCP
streaming multimídia	proprietário (p.ex. RealNetworks)	TCP ou UDP
telefonia Internet	proprietário (p.ex., Dialpad)	tipicamente UDP

Web e HTTP

Primeiro algum jargão

- ❑ Páginas Web consistem de objetos
- ❑ Objeto pode ser um arquivo HTML, uma imagem JPEG, um applet Java, um arquivo de áudio,...
- ❑ Páginas Web consistem de um arquivo HTML base que inclui vários objetos referenciados
- ❑ Cada objeto é endereçável por uma URL
- ❑ Exemplo de URL:

`www.someschool.edu/someDept/pic.gif`

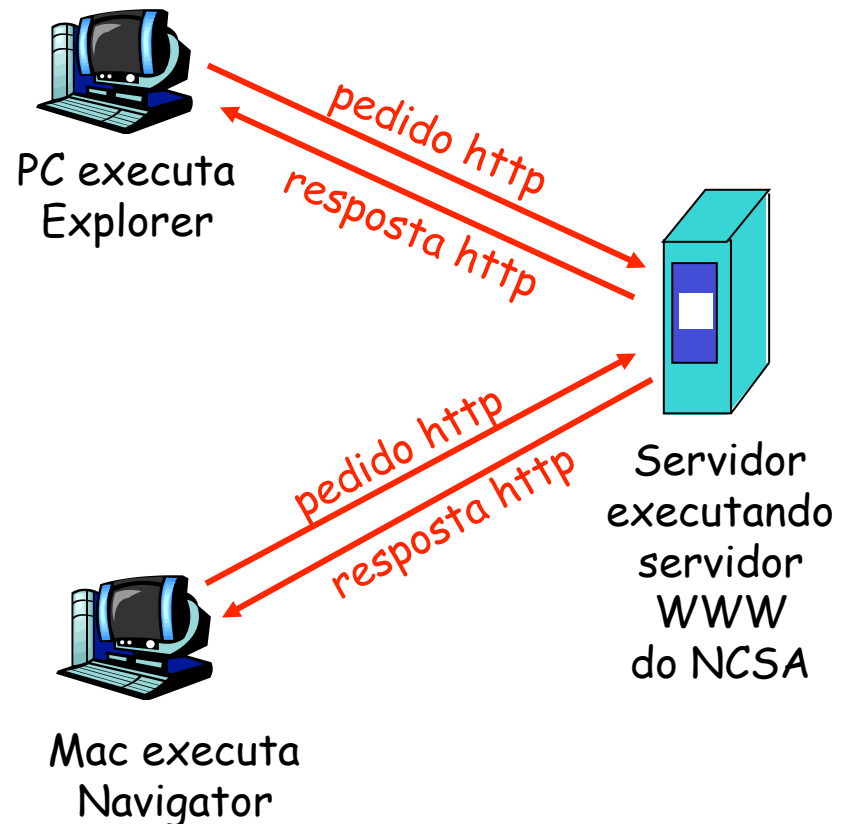
nome do hospedeiro

nome do caminho

Protocolo HTTP

HTTP: *hypertext transfer protocol*

- ❑ protocolo da camada de aplicação da Web
- ❑ modelo cliente/servidor
 - *cliente*: browser que pede, recebe, "visualiza" objetos Web
 - *servidor*: servidor Web envia objetos em resposta a pedidos
- ❑ HTTP 1.0: RFC 1945
- ❑ HTTP 1.1: RFC 2068



Mais sobre o protocolo HTTP

Usa serviço de transporte TCP:

- ❑ cliente inicia conexão TCP (cria *socket*) ao servidor, porta 80
- ❑ servidor aceita conexão TCP do cliente
- ❑ mensagens HTTP (mensagens do protocolo da camada de apl) trocadas entre *browser* (cliente HTTP) e servidor Web (servidor HTTP)
- ❑ encerra conexão TCP

HTTP é "sem estado"

- ❑ servidor não mantém informação sobre pedidos anteriores do cliente

Nota

Protocolos que mantêm "estado" são complexos!

- ❑ história passada (estado) tem que ser guardada
- ❑ Caso caia servidor/cliente, suas visões do "estado" podem ser inconsistentes, devem ser reconciliadas

Conexões HTTP

HTTP não persistente

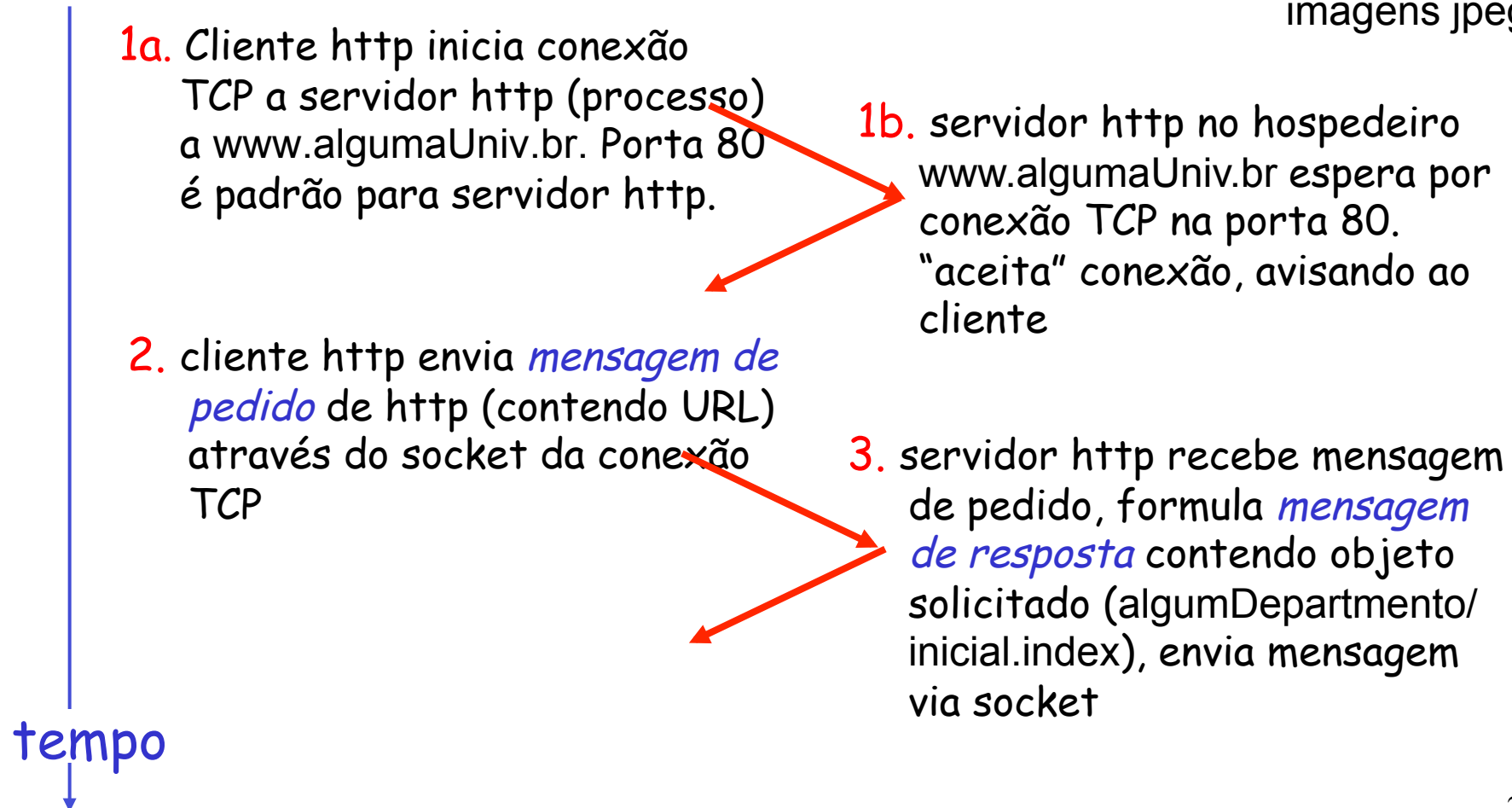
- ❑ No máximo um objeto é enviado numa conexão TCP
- ❑ HTTP/1.0 usa o HTTP não persistente

HTTP persistente

- ❑ Múltiplos objetos podem ser enviados sobre uma única conexão TCP entre cliente e servidor
- ❑ HTTP/1.1 usa conexões persistentes no seu modo *default*


Exemplo de HTTP não persistente

Supomos que usuário digita a URL `www.algumaUniv.br/algumDepartamento/inicial.index` (contém texto, referências a 10 imagens jpeg)



Exemplo de HTTP não persistente (cont.)

4. servidor http encerra conexão TCP .



5. cliente http recebe mensagem de resposta contendo arquivo html, visualiza html.
Analisando arquivo html, encontra 10 objetos jpeg referenciados

6. Passos 1 a 5 repetidos para cada um dos 10 objetos jpeg



tempo

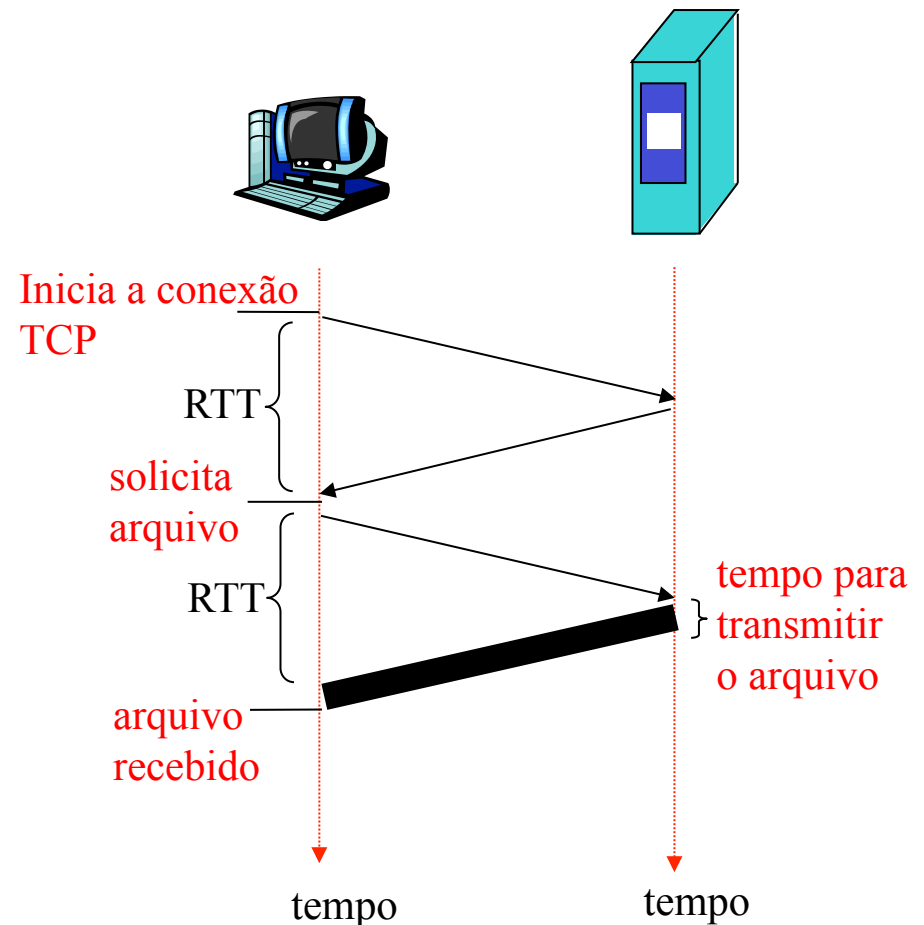
Modelagem do tempo de resposta

Definição de RTT (Round Trip Time): intervalo de tempo entre a ida e a volta de um pequeno pacote entre um cliente e um servidor

Tempo de resposta:

- um RTT para iniciar a conexão TCP
- um RTT para o pedido HTTP e o retorno dos primeiros bytes da resposta HTTP
- tempo de transmissão do arquivo

total = $2RTT + \text{tempo de transmissão}$



HTTP persistente

Problemas com o HTTP não persistente:

- ❑ requer 2 RTTs para cada objeto
- ❑ SO aloca recursos do *host* para cada conexão TCP
- ❑ os *browsers* frequentemente abrem conexões TCP paralelas para recuperar os objetos referenciados

HTTP persistente

- ❑ o servidor deixa a conexão aberta após enviar a resposta
- ❑ mensagens HTTP seguintes entre o mesmo cliente/servidor são enviadas nesta conexão

Persistente sem *pipelining*:

- ❑ o cliente envia um novo pedido apenas quando a resposta anterior tiver sido recebida
- ❑ um RTT para cada objeto referenciado

Persistente com *pipelining*:

- ❑ *default* no HTTP/1.1
- ❑ o cliente envia os pedidos logo que encontra um objeto referenciado
- ❑ pode ser necessário apenas um RTT para todos os objetos referenciados

Formato de mensagem HTTP: pedido

- ❑ Dois tipos de mensagem HTTP: *pedido, resposta*
- ❑ *mensagem de pedido HTTP:*
 - ASCII (formato legível por pessoas)

linha do pedido
(comandos GET,
POST, HEAD)

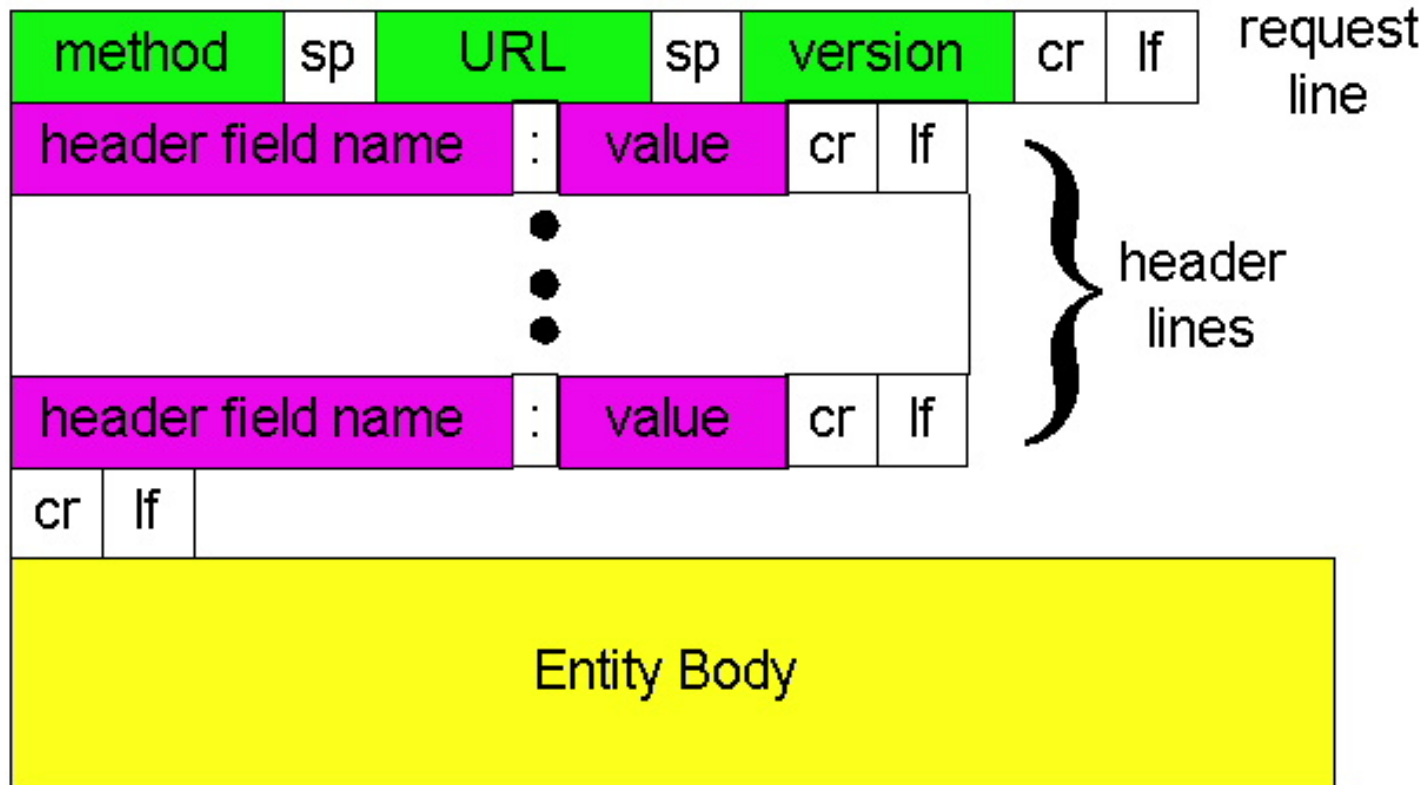
linhas do
cabeçalho

```
GET /somedir/page.html HTTP/1.0
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
```

Carriage return,
line feed
indicam fim
de mensagem

(carriage return (CR), line feed(LF) adicionais)

Mensagem de pedido HTTP: formato geral



Tipos de métodos

HTTP/1.0

- ❑ GET
- ❑ POST
- ❑ HEAD
 - Pede para o servidor não enviar o objeto requerido junto com a resposta (usado p/ debugging)

HTTP/1.1

- ❑ GET, POST, HEAD
- ❑ PUT
 - *Upload* de arquivo contido no corpo da mensagem para o caminho especificado no campo URL
- ❑ DELETE
 - Exclui arquivo especificado no campo URL

Enviando conteúdo de formulário

Método POST :

- ❑ Conteúdo é enviado para o servidor no corpo da mensagem

Método GET:

- ❑ Conteúdo é enviado para o servidor no campo URL:

`www.somesite.com/animalsearch?key=monkeys&max=10`

Formato de mensagem HTTP: resposta

linha de status
(protocolo,
código de status,
frase de status)

linhas de
cabeçalho

dados, p.ex.,
arquivo html
solicitado

```
HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 .....
Content-Length: 6821
Content-Type: text/html
```

```
dados dados dados dados ...
```

códigos de status da resposta HTTP

Na primeira linha da mensagem de resposta servidor->cliente. Alguns códigos típicos:

200 OK

- sucesso, objeto pedido segue mais adiante nesta mensagem

301 Moved Permanently

- objeto pedido mudou de lugar, nova localização especificado mais adiante nesta mensagem (Location:)

400 Bad Request

- mensagem de pedido não entendida pelo servidor

404 Not Found

- documento pedido não se encontra neste servidor

505 HTTP Version Not Supported

- versão de http do pedido não usada por este servidor

Experimente você com HTTP (do lado cliente)

1. Use cliente telnet para seu servidor WWW favorito:

```
telnet www.ic.uff.br 80
```

Abre conexão TCP para a porta 80 (porta padrão do servidor http) a www.ic.uff.br. Qualquer coisa digitada é enviada para a porta 80 do www.ic.uff.br

2. Digite um pedido GET HTTP:

```
GET /~michael/index.html HTTP/1.0
```

Digitando isto (deve teclar ENTER duas vezes), está enviando este pedido GET mínimo (porém completo) ao servidor http

3. Examine a mensagem de resposta enviada pelo servidor HTTP !

Cookies: manutenção do "estado" da conexão

Muitos dos principais sítios Web usam *cookies*

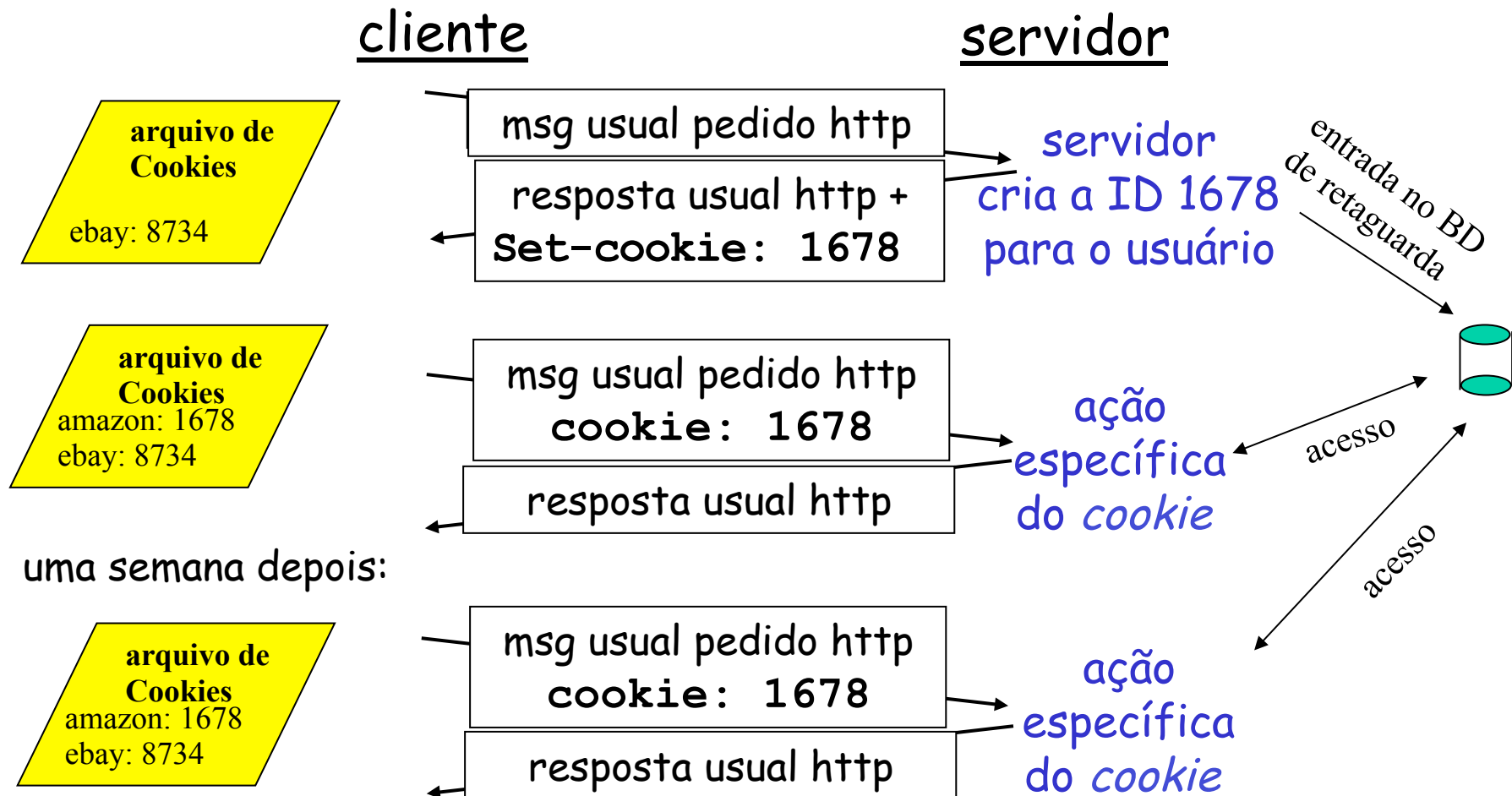
Quatro componentes:

- 1) linha de cabeçalho do *cookie* na mensagem de resposta HTTP
- 2) linha de cabeçalho do *cookie* na mensagem de pedido HTTP
- 3) arquivo do *cookie* mantido no host do usuário e gerenciado pelo browser do usuário
- 4) BD de retaguarda no sítio Web

Exemplo:

- Suzana acessa a Internet sempre do mesmo PC
- Ela visita um sítio específico de comércio eletrônico pela primeira vez
- Quando os pedidos iniciais HTTP chegam no sítio, o sítio cria uma ID única e cria uma entrada para a ID no BD de retaguarda

Cookies: manutenção do "estado" (cont.)



Cookies (continuação)

O que os cookies podem obter:

- ☐ autorização
- ☐ carrinhos de compra
- ☐ sugestões
- ☐ estado da sessão do usuário (*Webmail*)

nota

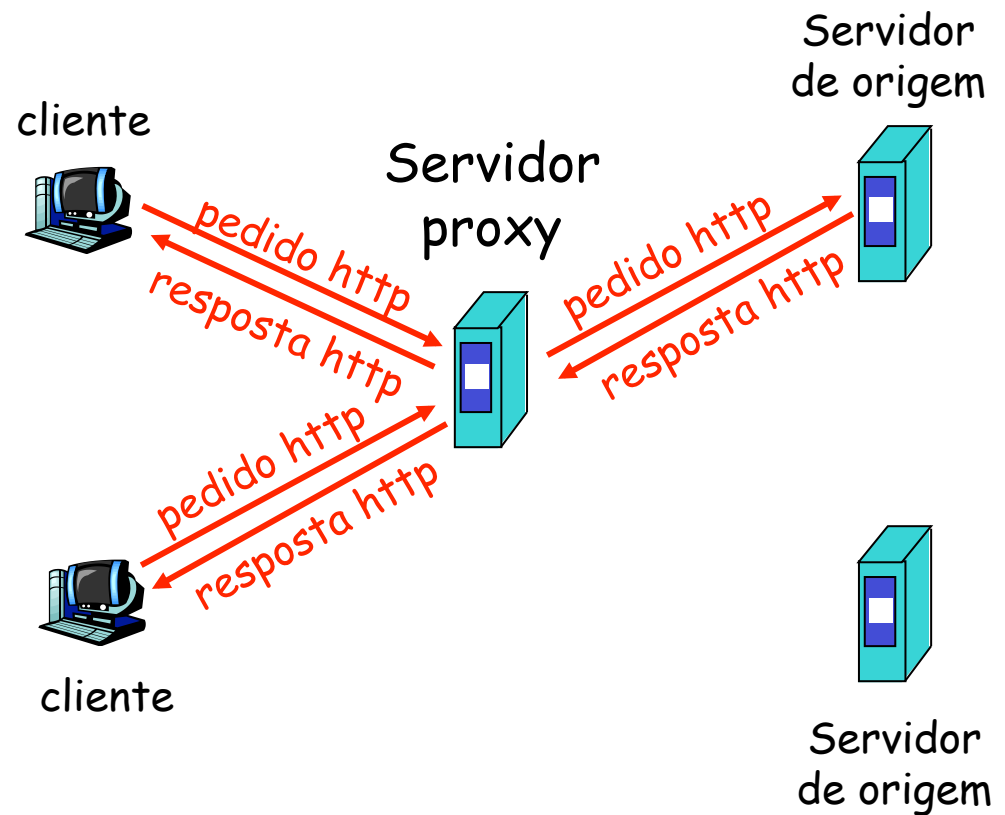
Cookies e privacidade:

- ☐ cookies permitem que os sítios aprendam muito sobre você
- ☐ você pode fornecer nome e e-mail para os sítios
- ☐ mecanismos de busca usam redirecionamento e *cookies* para aprender ainda mais
- ☐ agências de propaganda obtêm perfil a partir dos sítios visitados

Cache Web (servidor proxy)

Meta: atender pedido do cliente sem envolver servidor de origem

- ❑ usuário configura browser: acessos Web via proxy
- ❑ cliente envia todos pedidos HTTP ao proxy
 - se objeto no cache do proxy, este o devolve imediatamente na resposta HTTP
 - senão, solicita objeto do servidor de origem, depois devolve resposta HTTP ao cliente



Mais sobre Caches Web

- ❑ Cache atua tanto como cliente quanto como servidor
- ❑ Tipicamente o cache é instalado por um ISP (universidade, empresa, ISP residencial)

Para que fazer cache Web?

- ❑ Redução do tempo de resposta para os pedidos do cliente
- ❑ Redução do tráfego no canal de acesso de uma instituição
- ❑ A Internet cheia de caches permitem que provedores de conteúdo "pobres" efetivamente forneçam conteúdo!

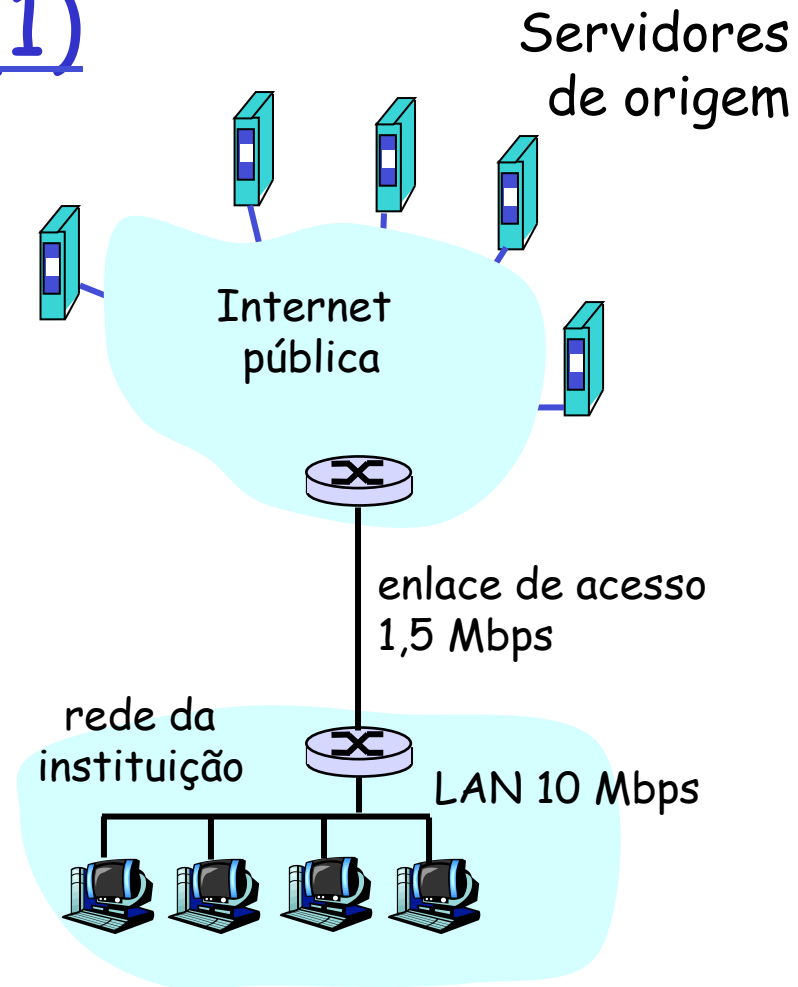
Exemplo de cache (1)

Hipóteses

- ❑ Tamanho médio de um objeto = 100.000 bits
- ❑ Taxa média de solicitações dos *browsers* de uma instituição para os servidores originais = 15/seg
- ❑ Atraso do roteador institucional para qualquer servidor origem e de volta ao roteador = 2seg

Consequências

- ❑ Utilização da LAN = 15%
- ❑ Utilização do canal de acesso = 100%
- ❑ Atraso total = atraso da Internet + atraso de acesso + atraso na LAN = 2 seg + minutos + milisegundos



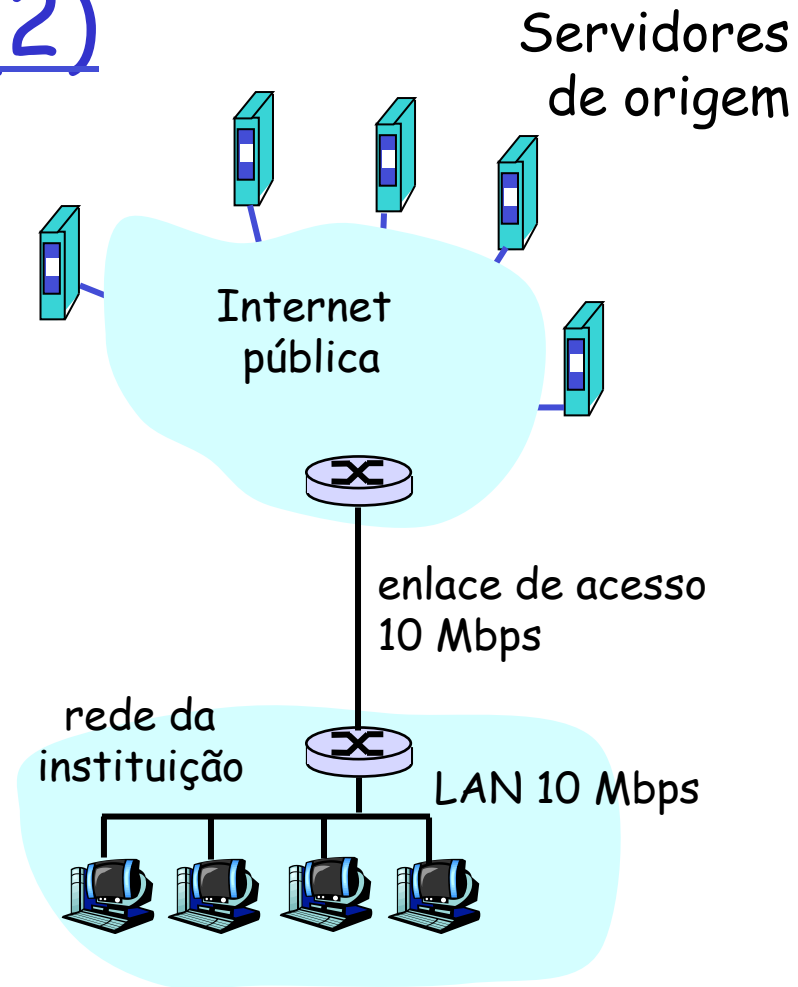
Exemplo de cache (2)

Solução em potencial

- ❑ Aumento da largura de banda do canal de acesso para, por exemplo, 10 Mbps

Consequências

- ❑ Utilização da LAN = 15%
- ❑ Utilização do canal de acesso = 15%
- ❑ Atraso total = atraso da Internet + atraso de acesso + atraso na LAN = 2 seg + msecs + msecs
- ❑ Frequentemente este é uma ampliação cara



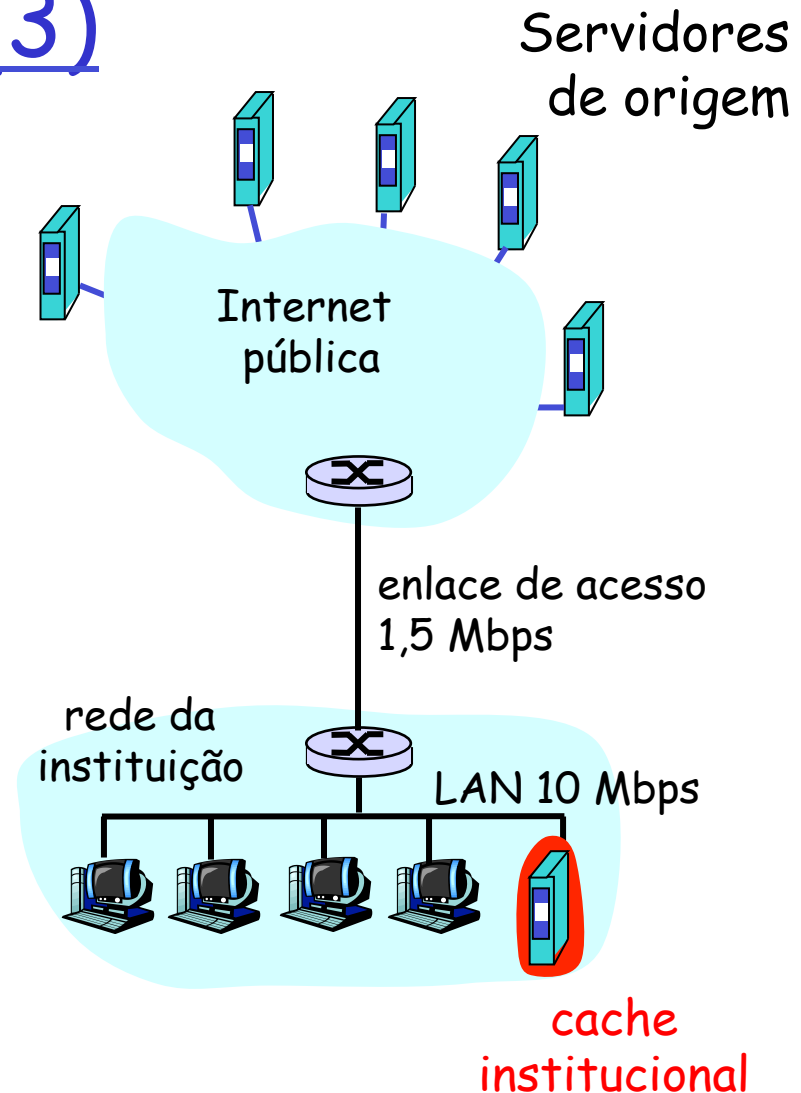
Exemplo de cache (3)

Instale uma cache

- Assuma que a taxa de acerto seja de 0,4

Consequências

- 40% dos pedidos serão atendidos quase que imediatamente
- 60% dos pedidos serão servidos pelos servidores de origem
- Utilização do canal de acesso é reduzido para 60%, resultando em atrasos desprezíveis (ex. 10 mseg)
- Atraso total = atraso da Internet + atraso de acesso + atraso na LAN = $0,6 \times 2 \text{ seg} + 0,6 \times 0,01 \text{ segs} + \text{msecs} < 1,3 \text{ segs}$

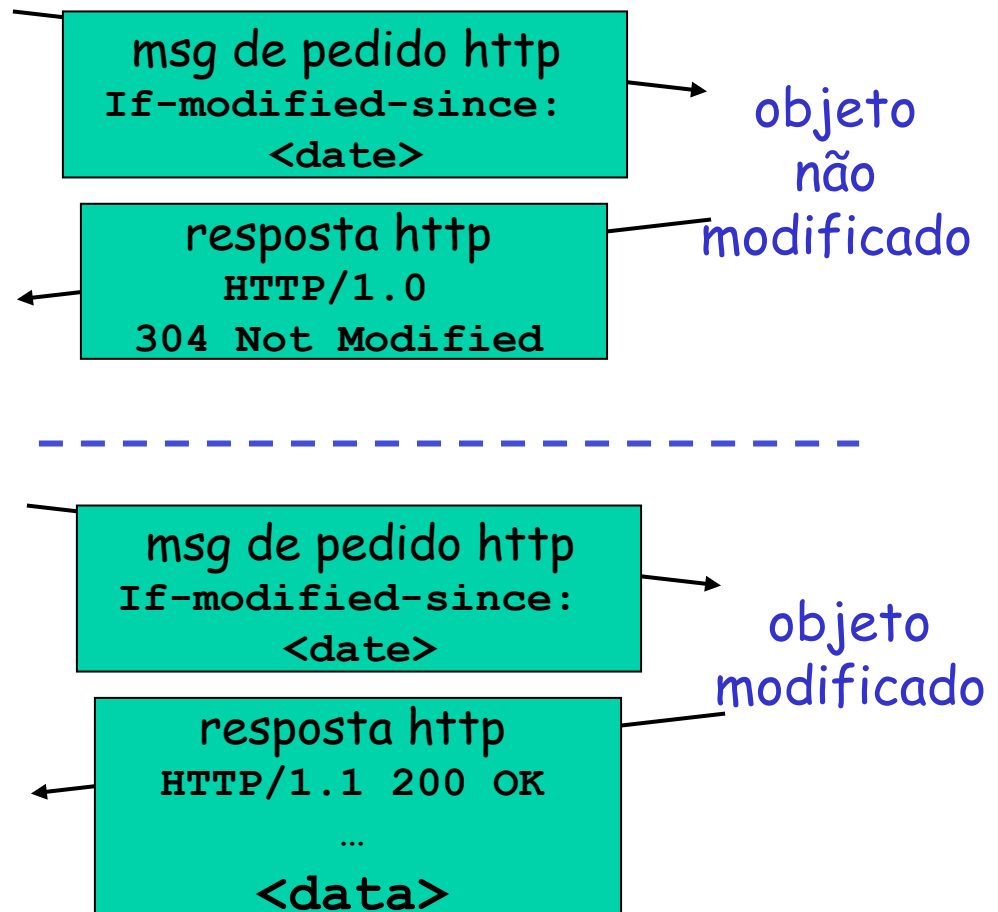


GET condicional

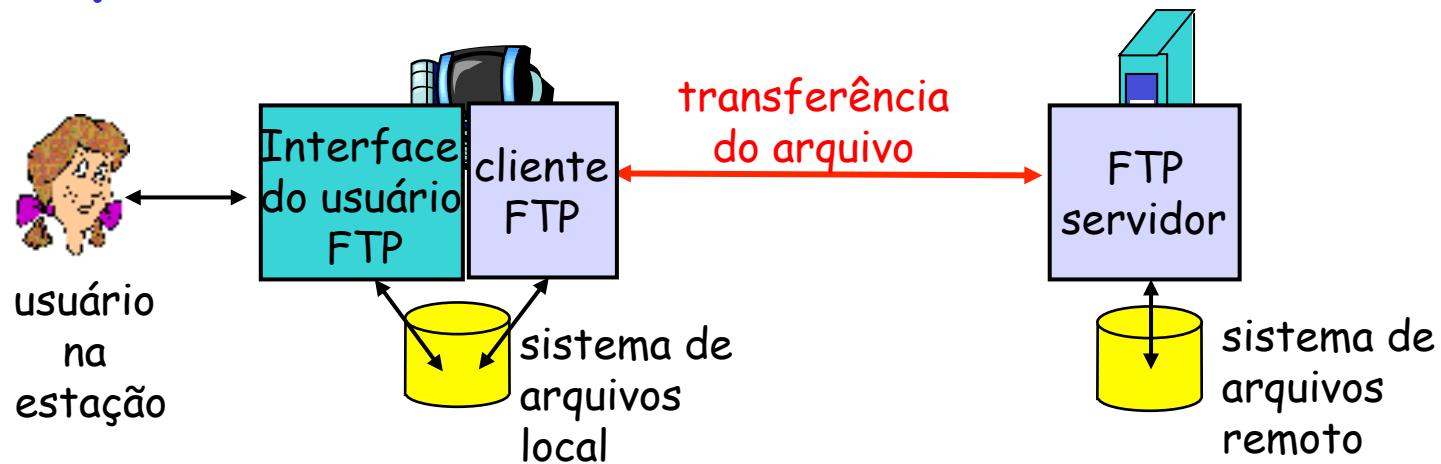
- ❑ **Meta:** não enviar objeto se cliente já tem (no cache) versão atual
- ❑ **cache:** especifica data da cópia no cache no pedido http
If-modified-since:
 <date>
- ❑ **servidor:** resposta não contém objeto se cópia no cache é atual:
HTTP/1.0 304 Not Modified

cache

servidor



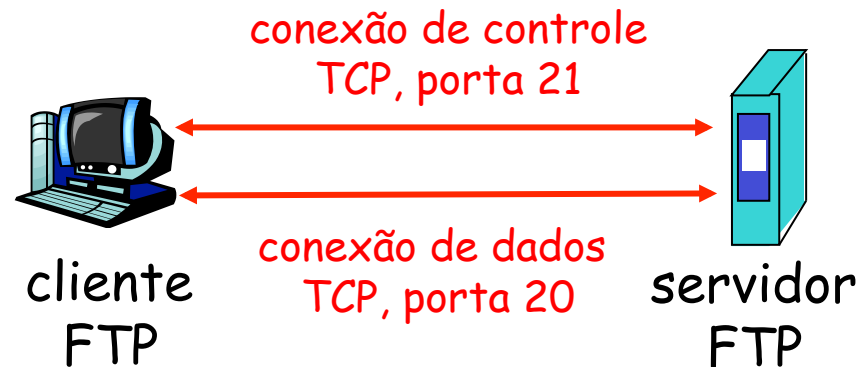
FTP: o protocolo de transferência de arquivos



- ❑ transferir arquivo de/para hospedeiro remoto
- ❑ modelo cliente/servidor
 - *cliente*: lado que inicia transferência (pode ser de ou para o sistema remoto)
 - *servidor*: hospedeiro remoto
- ❑ ftp: RFC 959
- ❑ servidor ftp: porta 21

FTP: conexões separadas p/ controle, dados

- ❑ cliente FTP contata servidor FTP na porta 21, especificando o TCP como protocolo de transporte
- ❑ O cliente obtém autorização através da conexão de controle
- ❑ O cliente consulta o diretório remoto enviando comandos através da conexão de controle
- ❑ Quando o servidor recebe um comando para a transferência de um arquivo, ele abre uma conexão de dados TCP para o cliente
- ❑ Após a transmissão de um arquivo o servidor fecha a conexão



- ❑ O servidor abre uma segunda conexão TCP para transferir outro arquivo
- ❑ Conexão de controle: "fora da faixa"
- ❑ Servidor FTP mantém o "estado": diretório atual, autenticação anterior

FTP: comandos, respostas

Comandos típicos:

- ❑ enviados em texto ASCII pelo canal de controle
- ❑ **USER** *nome*
- ❑ **PASS** *senha*
- ❑ **LIST** devolve lista de arquivos no diretório atual
- ❑ **RETR** arquivo recupera (lê) arquivo remoto
- ❑ **STOR** arquivo armazena (escreve) arquivo no hospedeiro remoto

Códigos de retorno típicos

- ❑ código e frase de status (como para http)
- ❑ 331 Username OK, password required
- ❑ 125 data connection already open; transfer starting
- ❑ 425 Can't open data connection
- ❑ 452 Error writing file

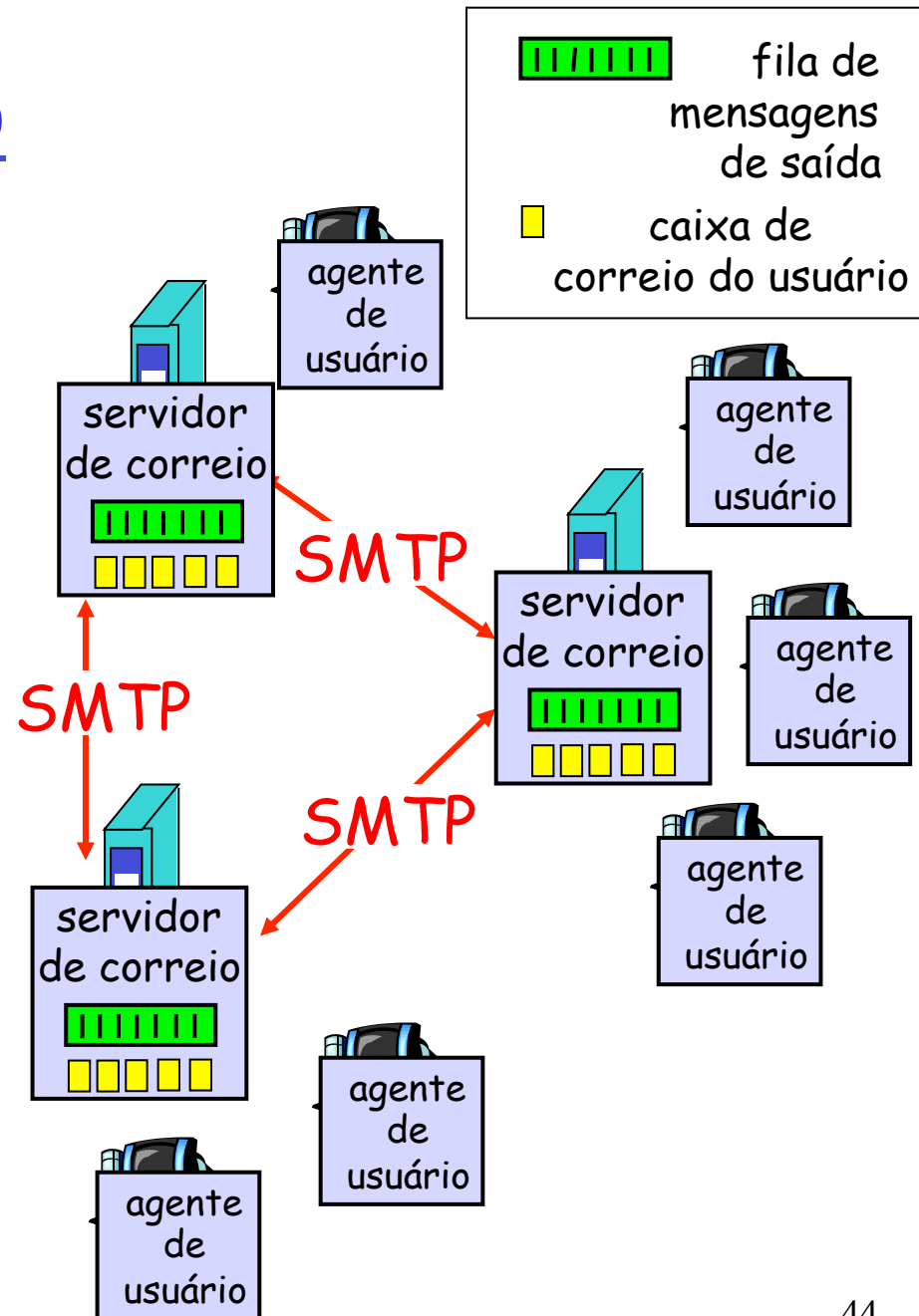
Correio Eletrônico

Três grandes componentes:

- ❑ agentes de usuário (UA)
- ❑ servidores de correio
- ❑ simple mail transfer protocol: SMTP

Agente de Usuário

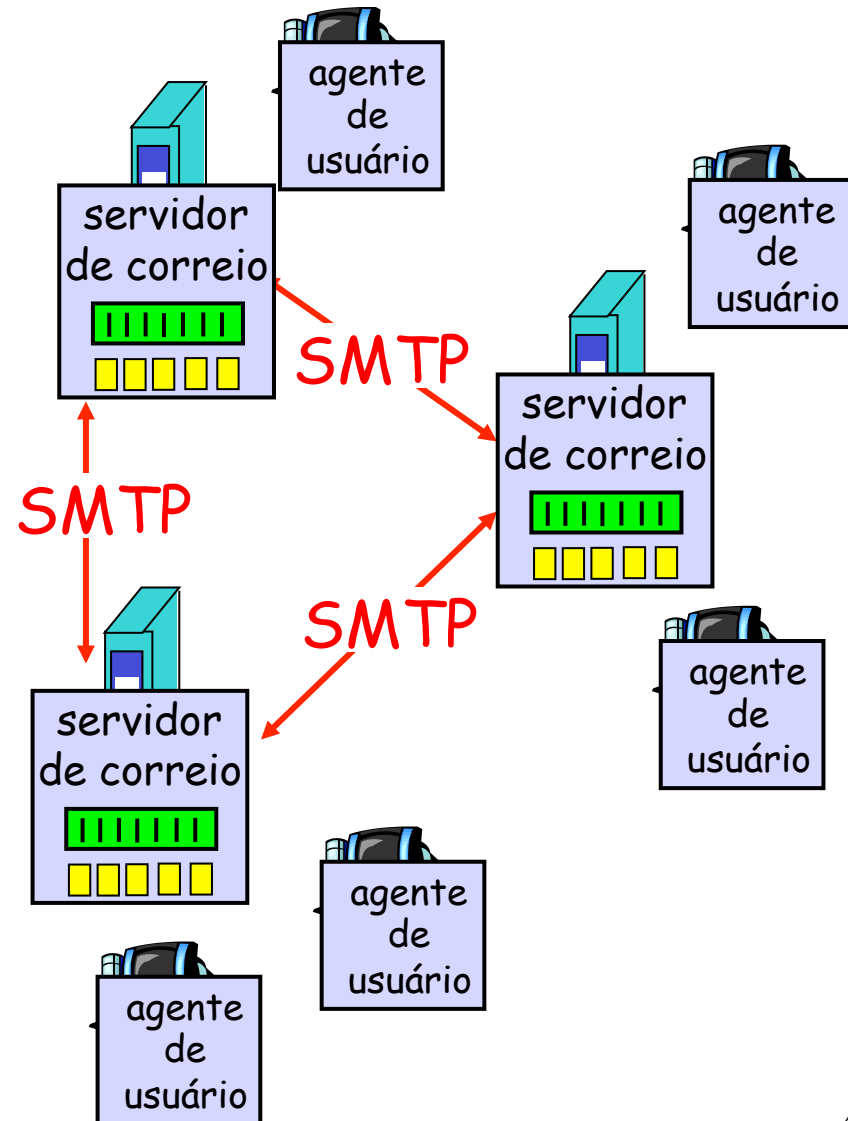
- ❑ a.k.a. "leitor de correio"
- ❑ compor, editar, ler mensagens de correio
- ❑ p.ex., Eudora, Outlook, elm, Netscape Messenger
- ❑ mensagens de saída e chegando são armazenadas no servidor



Correio Eletrônico: servidores de correio

Servidores de correio

- ❑ **caixa de correio** contém mensagens de chegada (ainda não lidas) p/ usuário
- ❑ **fila de mensagens** contém mensagens de saída (a serem enviadas)
- ❑ **protocolo SMTP** entre servidores de correio para transferir mensagens de correio
 - cliente: servidor de correio que envia
 - "servidor": servidor de correio que recebe

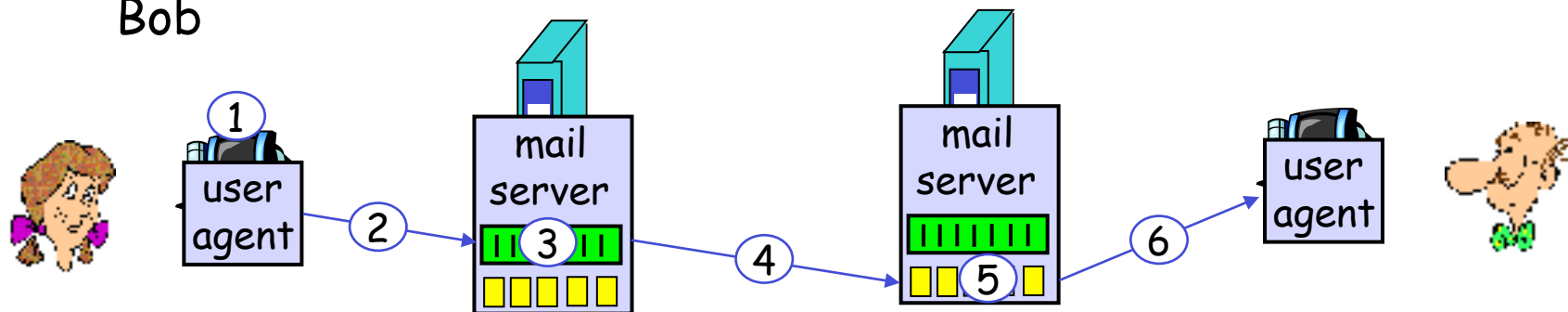


Correio Eletrônico: SMTP [RFC 2821]

- ❑ usa TCP para a transferência confiável de msgs do correio do cliente ao servidor, porta 25
- ❑ transferência direta: servidor remetente ao servidor receptor
- ❑ três fases da transferência
 - *handshaking* (cumprimento)
 - transferência das mensagens
 - encerramento
- ❑ interação comando/resposta
 - **comandos**: texto ASCII
 - **resposta**: código e frase de status
- ❑ mensagens precisam ser em ASCII de 7-bits

Cenário: Alice envia uma msg para Bob

- 1) Alice usa o UA para compor uma mensagem "para" bob@someschool.edu
- 2) O UA de Alice envia a mensagem para o seu servidor de correio; a mensagem é colocada na fila de mensagens
- 3) O lado cliente do SMTP abre uma conexão TCP com o servidor de correio de Bob
- 4) O cliente SMTP envia a mensagem de Alice através da conexão TCP
- 5) O servidor de correio de Bob coloca a mensagem na caixa de entrada de Bob
- 6) Bob chama o seu UA para ler a mensagem



Interação SMTP típica

```
S: 220 doces.br
C: HELO consumidor.br
S: 250 Hello consumidor.br, pleased to meet you
C: MAIL FROM: <ana@consumidor.br>
S: 250 ana@consumidor.br... Sender ok
C: RCPT TO: <bernardo@doces.br>
S: 250 bernardo@doces.br ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Voce gosta de chocolate?
C: Que tal sorvete?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 doces.br closing connection
```


Experimente uma interação SMTP:

- ❑ `telnet nomedoservidor 25`
- ❑ veja resposta 220 do servidor
- ❑ entre comandos `HELO`, `MAIL FROM`, `RCPT TO`, `DATA`, `QUIT`

estes comandos permitem que você envie correio sem usar um cliente (leitor de correio)

SMTP: últimas palavras

- ❑ SMTP usa conexões persistentes
- ❑ SMTP requer que a mensagem (cabeçalho e corpo) sejam em ASCII de 7-bits
- ❑ servidor SMTP usa CRLF.CRLF para reconhecer o final da mensagem

Comparação com HTTP

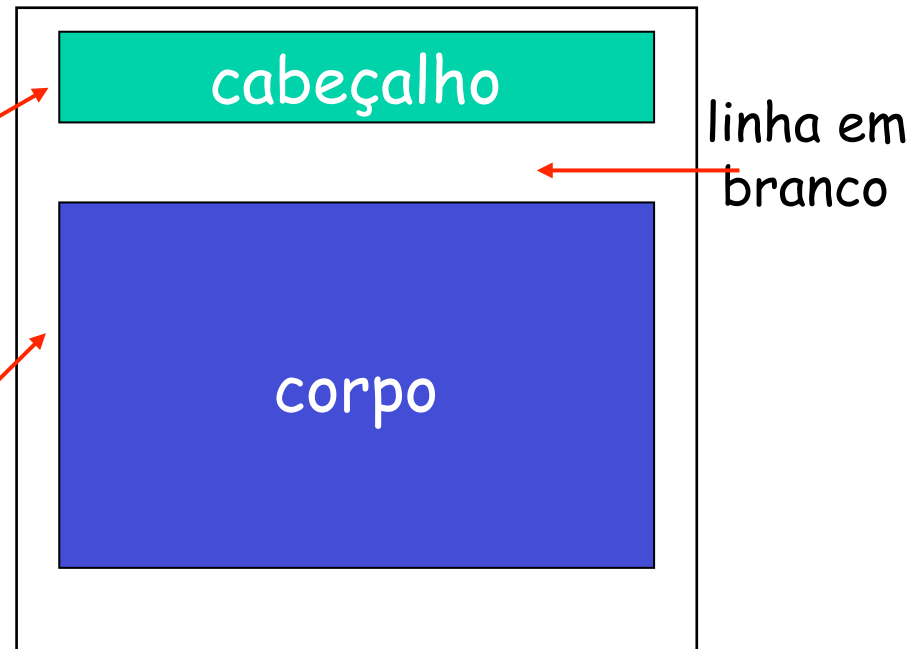
- ❑ HTTP: *pull* (puxar)
- ❑ SMTP: *push* (empurrar)
- ❑ ambos têm interação comando/resposta, códigos de status em ASCII
- ❑ HTTP: cada objeto é encapsulado em sua própria mensagem de resposta
- ❑ SMTP: múltiplos objetos de mensagem enviados numa mensagem de múltiplas partes

Formato de uma mensagem

SMTP: protocolo para trocar msgs de correio

RFC 822: padrão para formato de mensagem de texto:

- ❑ linhas de cabeçalho, p.ex.,
 - To:
 - From:
 - Subject:*diferentes dos comandos de smtp!*
- ❑ corpo
 - a "mensagem", somente de caracteres ASCII



Formato de uma mensagem: extensões para multimídia

- ❑ MIME: *multimedia mail extension*, RFC 2045, 2056
- ❑ linhas adicionais no cabeçalho da msg declaram tipo do conteúdo MIME

versão MIME

método usado
p/ codificar dados

tipo, subtipo de
dados multimídia,
declaração parâmetros

Dados codificados

```
From: ana@consumidor.br
To: bernardo@doces.br
Subject: Imagem de uma bela torta
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.....
.....base64 encoded data
```

Tipos MIME

Content-Type: tipo/subtipo; parâmetros

Text

- ❑ subtipos exemplos: plain, html
- ❑ charset="iso-8859-1", ascii

Image

- ❑ subtipos exemplos : jpeg, gif

Video

- ❑ subtipos exemplos : mpeg, quicktime

Audio

- ❑ subtipos exemplos : basic (8-bit codificado mu-law), 32kadpcm (codificação 32 kbps)

Application

- ❑ outros dados que precisam ser processados por um leitor para serem "visualizados"
- ❑ subtipos exemplos : msword, octet-stream

Tipo Multipart

From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=98766789

--98766789

Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain

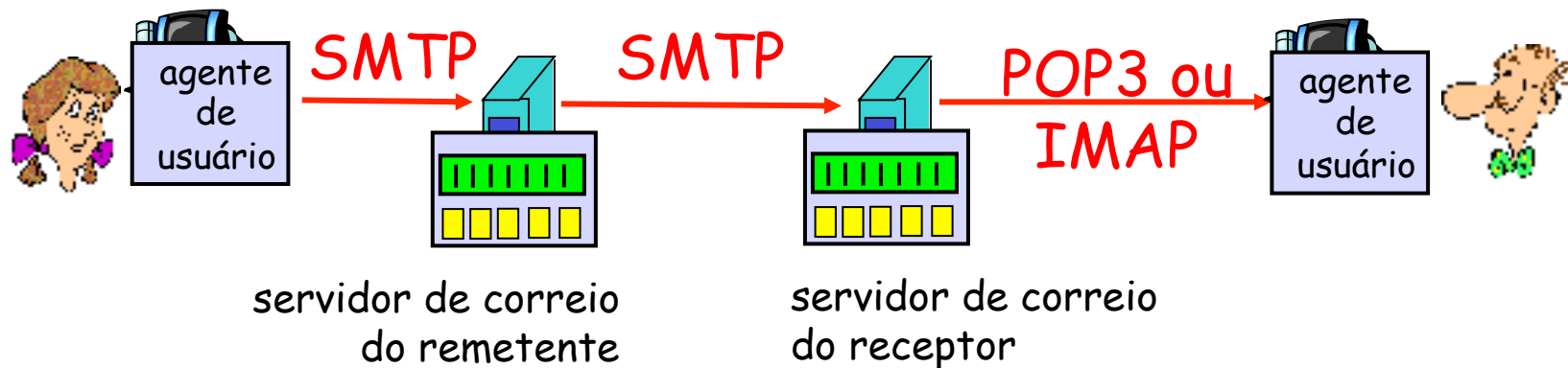
Dear Bob,
Please find a picture of a crepe.

--98766789

Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data
.....
.....base64 encoded data
--98766789--

Protocolos de acesso ao correio



- ❑ SMTP: entrega/armazenamento no servidor do receptor
- ❑ protocolo de acesso ao correio: recupera do servidor
 - POP: Post Office Protocol [RFC 1939]
 - autorização (agente <-->servidor) e transferência
 - IMAP: Internet Mail Access Protocol [RFC 1730]
 - mais comandos (mais complexo)
 - manuseio de msgs armazenadas no servidor
 - HTTP: Hotmail , Yahoo! Mail, Webmail, etc.

Protocolo POP3

fase de autorização

- ❑ comandos do cliente:
 - user: declara nome
 - pass: senha
- ❑ servidor responde
 - +OK
 - -ERR

fase de transação, cliente:

- ❑ list: lista números das msgs
- ❑ retr: recupera msg por número
- ❑ dele: apaga msg
- ❑ quit

```
S: +OK POP3 server ready
C: user ana
S: +OK
C: pass faminta
S: +OK user successfully logged on

C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```


POP3 (mais) e IMAP

Mais sobre o POP3

- ❑ O exemplo anterior usa o modo "*download e delete*".
- ❑ Bob não pode reler as mensagens se mudar de cliente
- ❑ "*Download-e-mantenha*": copia as mensagens em clientes diferentes
- ❑ POP3 não mantém estado entre conexões

IMAP

- ❑ Mantém todas as mensagens num único lugar: o servidor
- ❑ Permite ao usuário organizar as mensagens em pastas
- ❑ O IMAP mantém o estado do usuário entre sessões:
 - nomes das pastas e mapeamentos entre as IDs das mensagens e o nome da pasta

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:

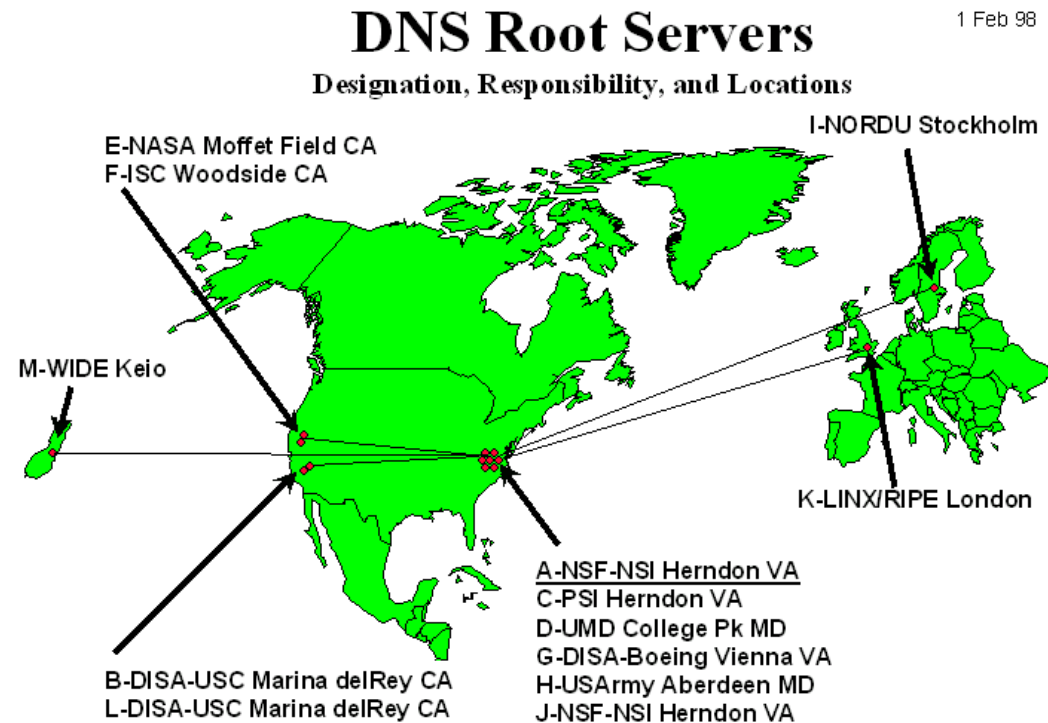
- cada provedor, empresa tem *servidor de nomes local (default)*
- pedido DNS de hospedeiro vai primeiro ao servidor de nomes local

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

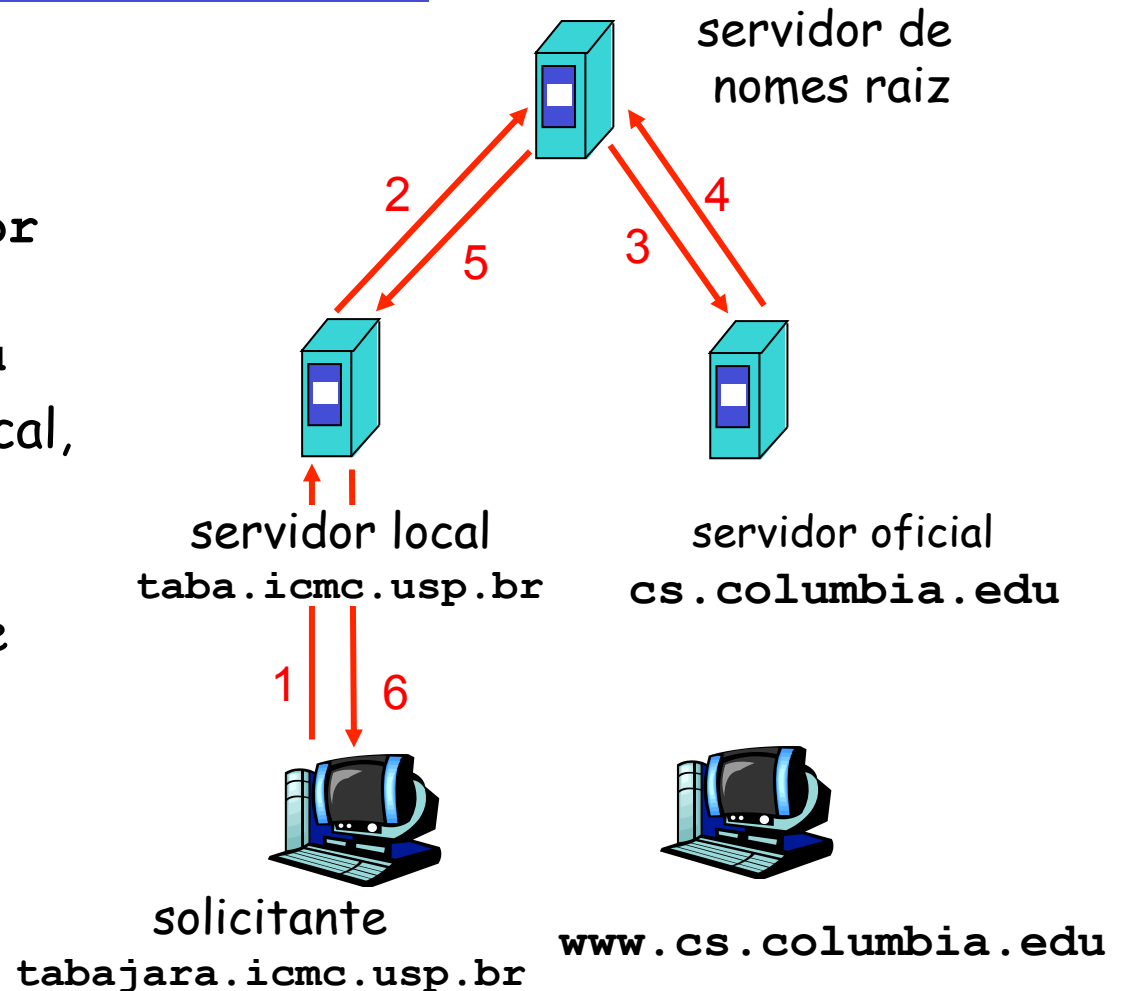


Exemplo simples do DNS

hospedeiro

`tabajara.icmc.usp.br`
requer endereço IP de
`www.cs.columbia.edu`

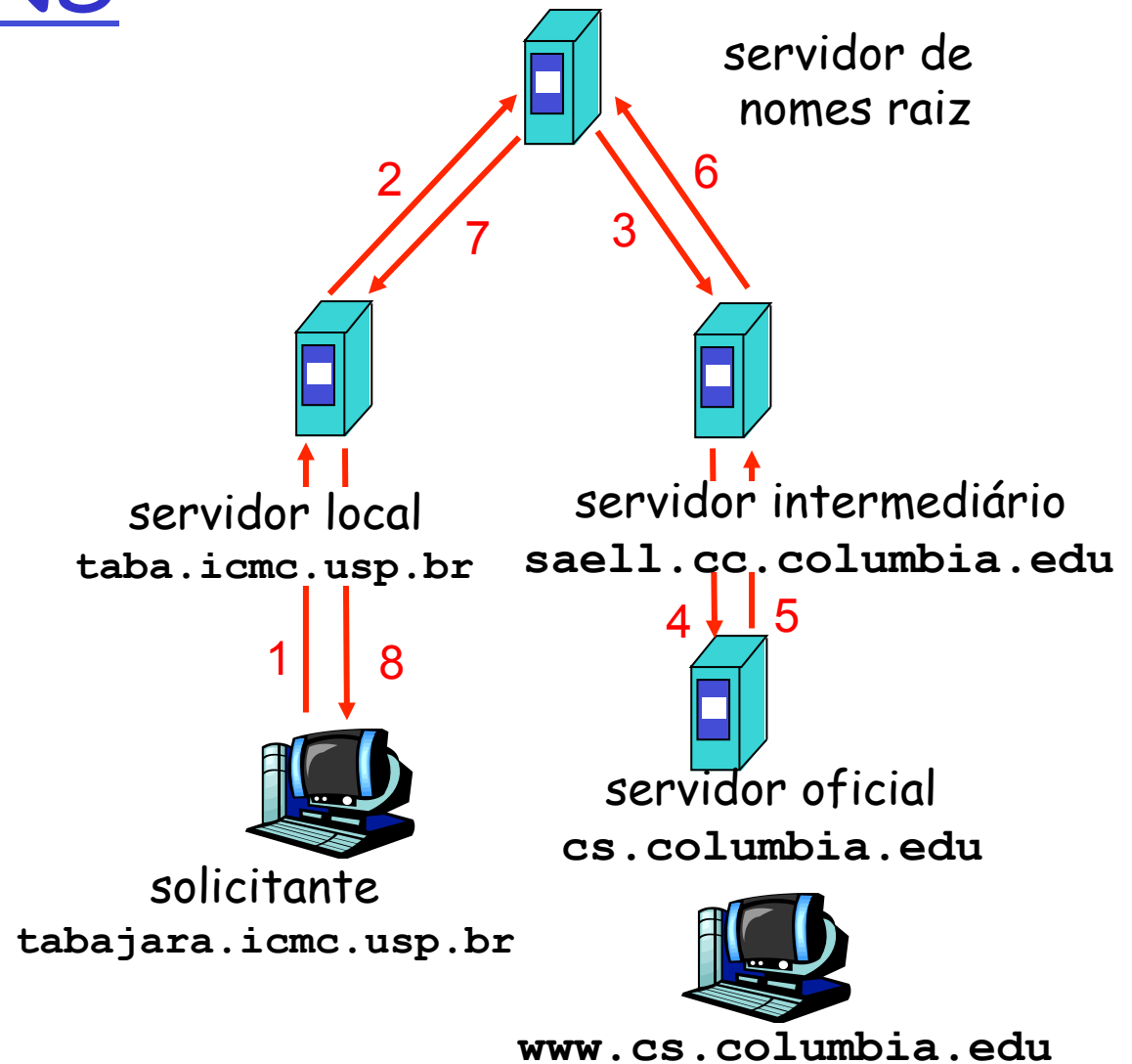
1. Contata servidor DNS local,
`taba.icmc.usp.br`
2. `taba.icmc.usp.br`
contata servidor raiz, se
necessário
3. Servidor raiz contata
servidor oficial
`cs.columbia.edu`, se
necessário



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



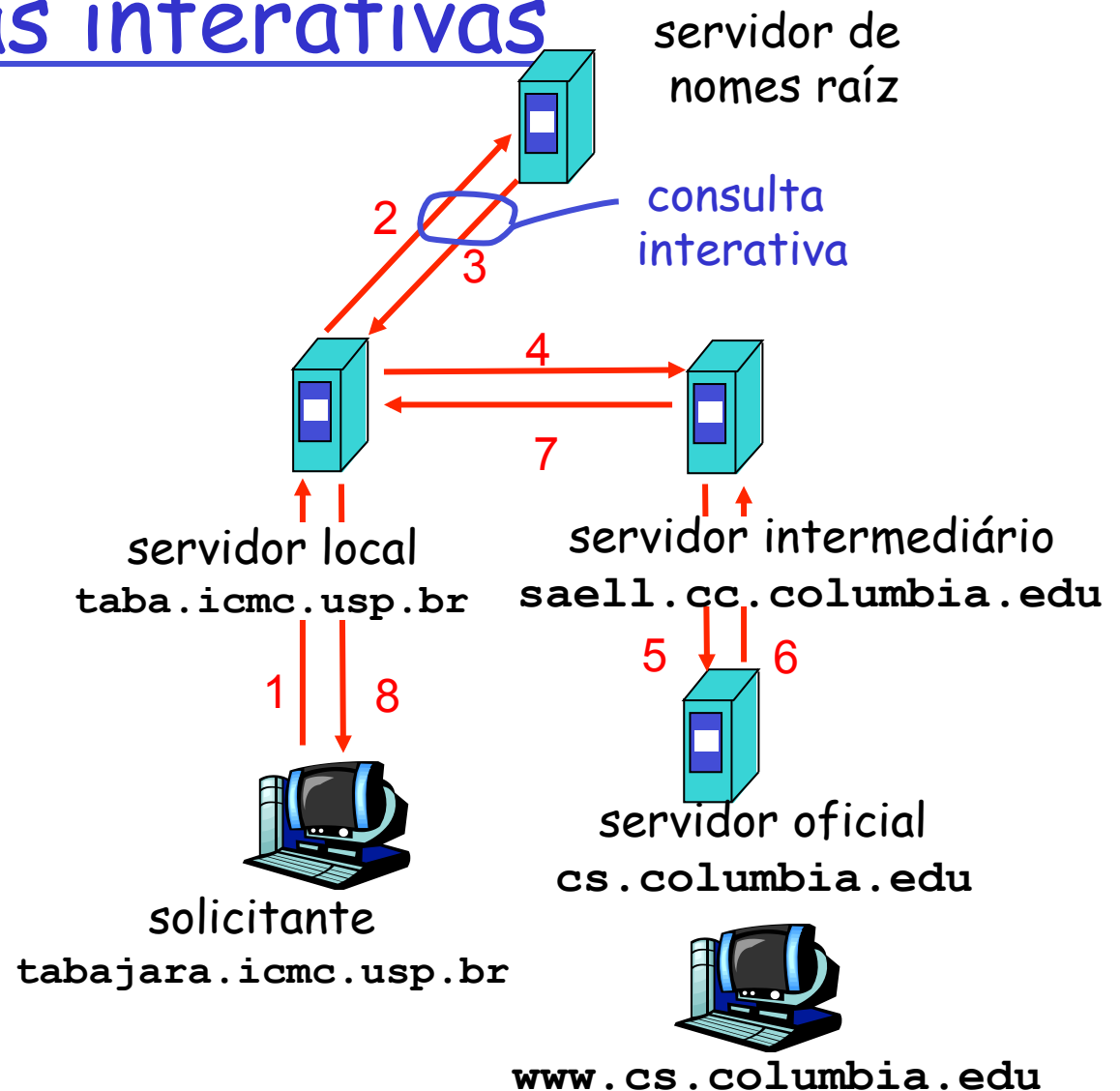
DNS: consultas interativas

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

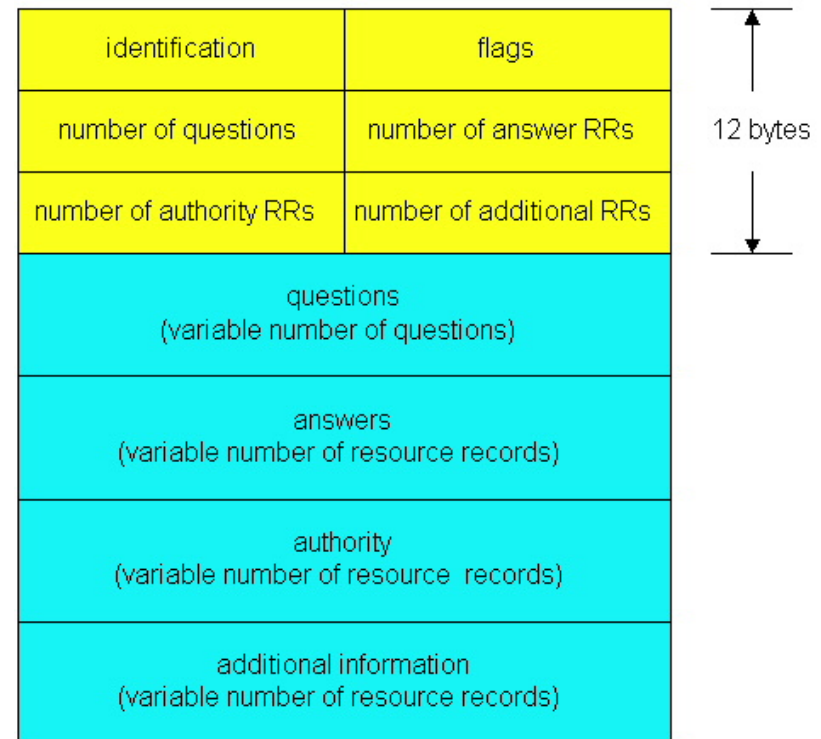
- nome é domínio
- valor é nome do servidor de correio para este domínio

DNS: protocolo e mensagens

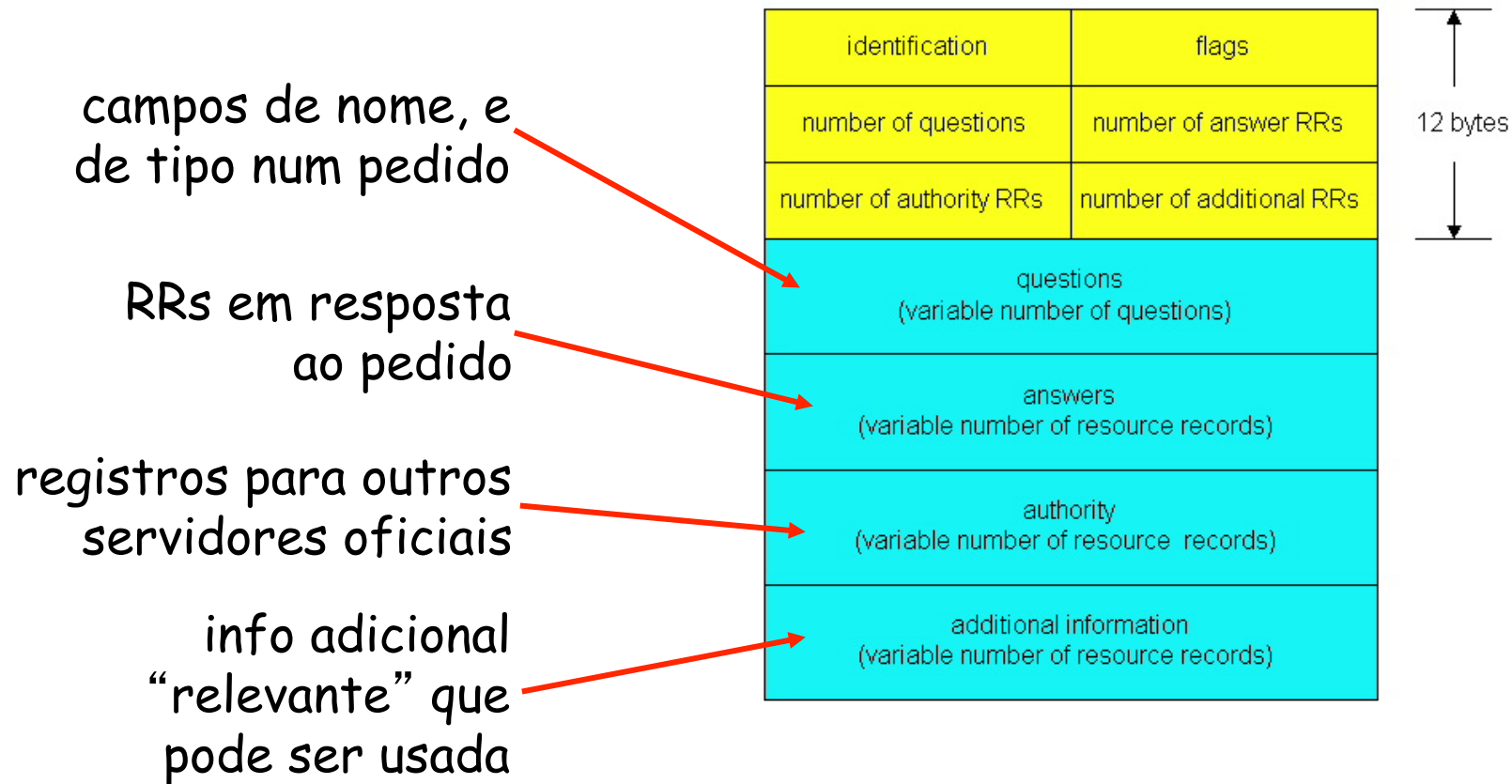
protocolo DNS: mensagens de *pedido* e *resposta*,
ambas com o mesmo *formato de mensagem*

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



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 aplicações
- ❑ paradigma cliente/servidor
- ❑ dois tipos de serviço de transporte via API Sockets
 - datagrama não confiável
 - fluxo de bytes, confiável

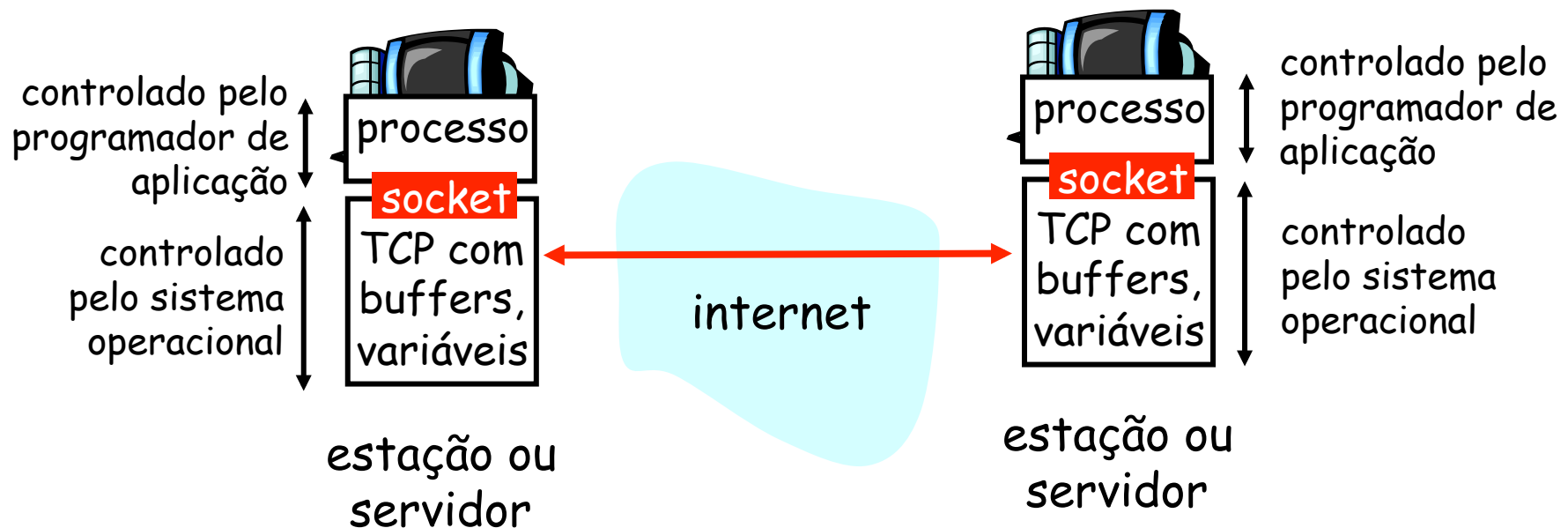
socket

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:

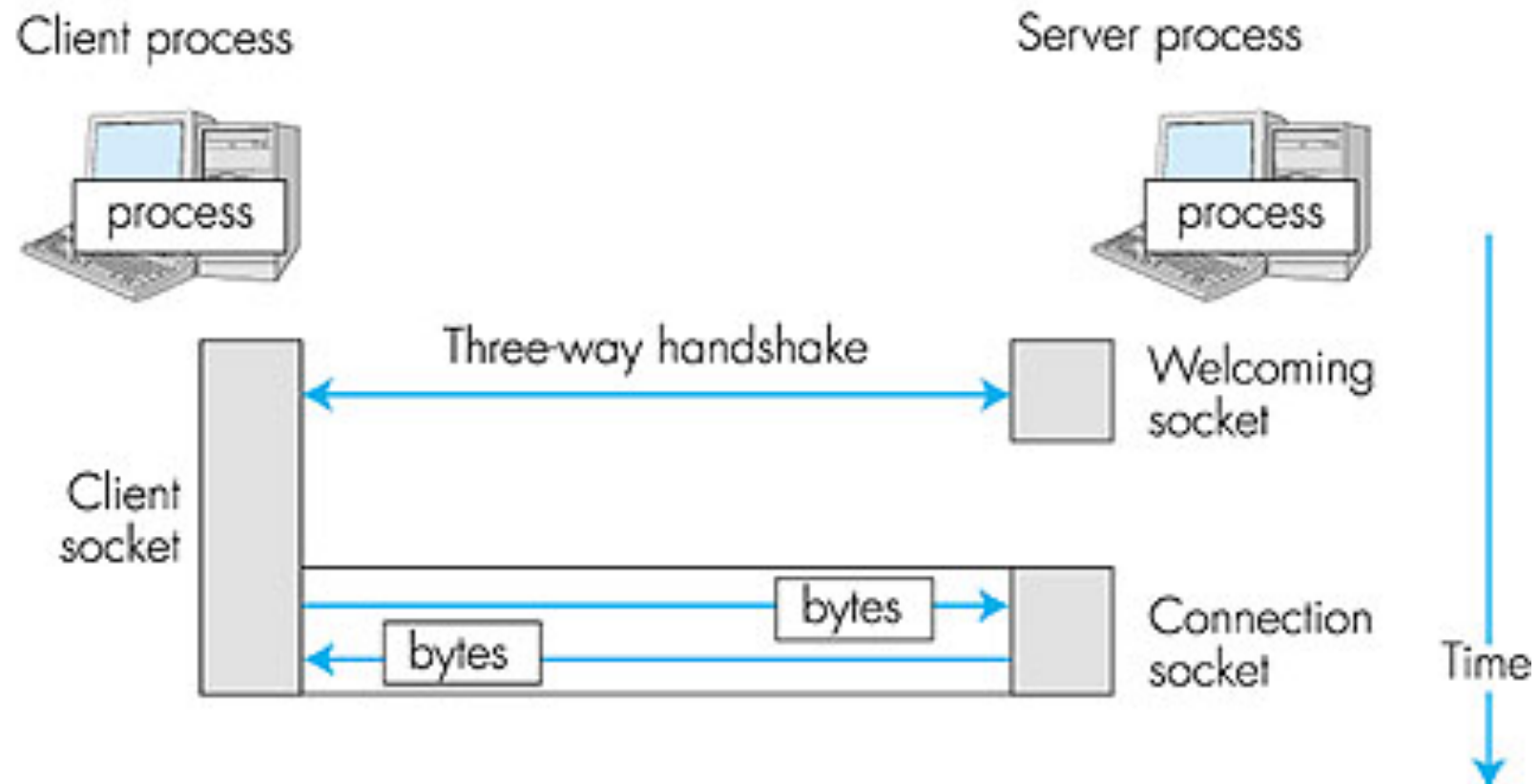
- ❑ criar socket TCP local ao cliente
- ❑ especificar endereço IP, número de porta do processo servidor

- ❑ Quando **cliente cria socket**: TCP do cliente estabelece conexão com TCP do servidor
- ❑ Quando contactado 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

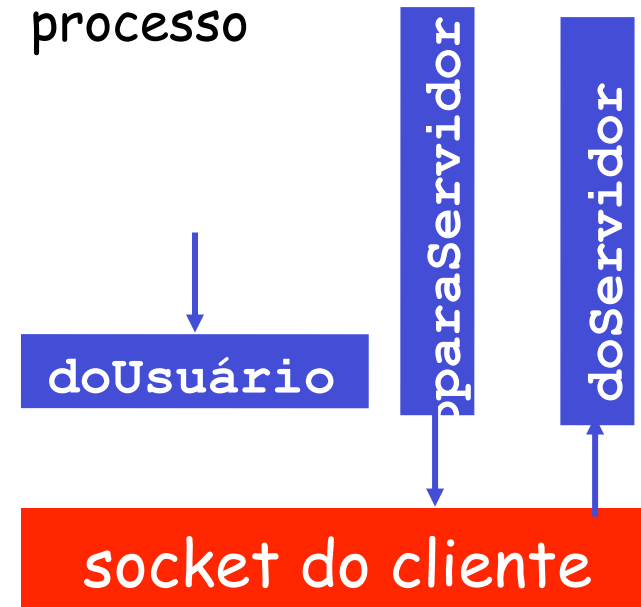


Exemplo de aplicação cliente-servidor

- ❑ cliente lê linha da entrada padrão (fluxo `doUsuario`), 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: sequência de bytes recebida pelo processo

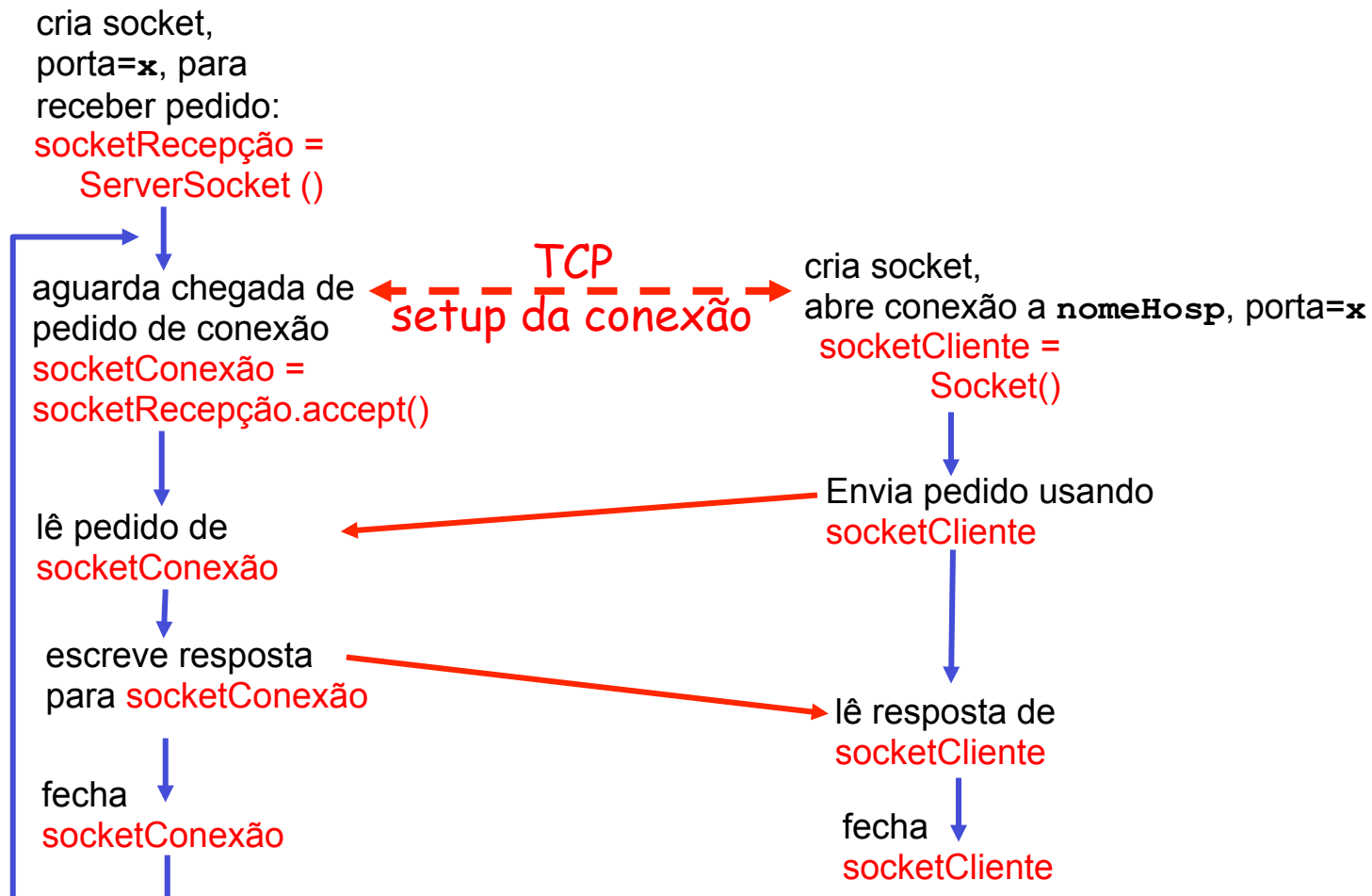
Fluxo de saída: sequência de bytes transmitida pelo processo



Interações cliente/servidor usando o TCP

Servidor (executa em `nomeHosp`)

Cliente



Exemplo: cliente Java (TCP)

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

Contém classe
para Streams IO

Contém classe para
suporte a rede

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

```
        String frase;  
        String fraseModificada;
```

Cria
fluxo de entrada

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

Cria
socket de cliente,
conexão ao servidor

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

Cria
fluxo de saída
ligado ao socket

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

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.*;
```

Contém classe
para Streams IO

```
class servidorTCP {
```

Contém classe para
suporte a rede

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

```
        String fraseCliente;  
        StringfFraseMaiusculas;
```

Cria socket
para recepção
na porta 6789

```
        ServerSocket socketRecepcao = new ServerSocket(6789);
```

Aguarda, no socket
para recepção, o
contato do cliente

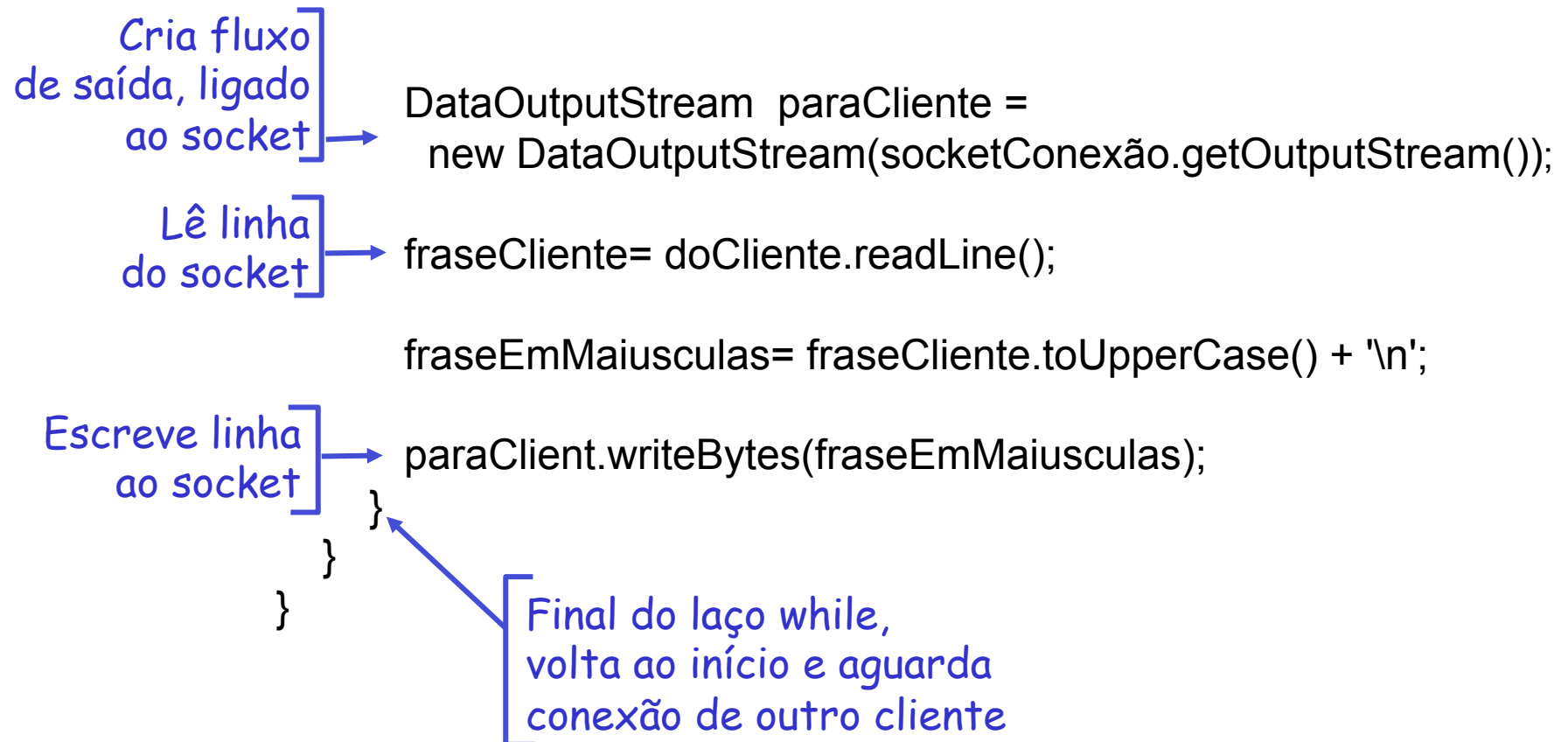
```
        while(true) {
```

```
            Socket socketConexao = socketRecepcao.accept();
```

Cria fluxo de
entrada, ligado
ao socket

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

Exemplo: servidor Java (TCP), cont



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

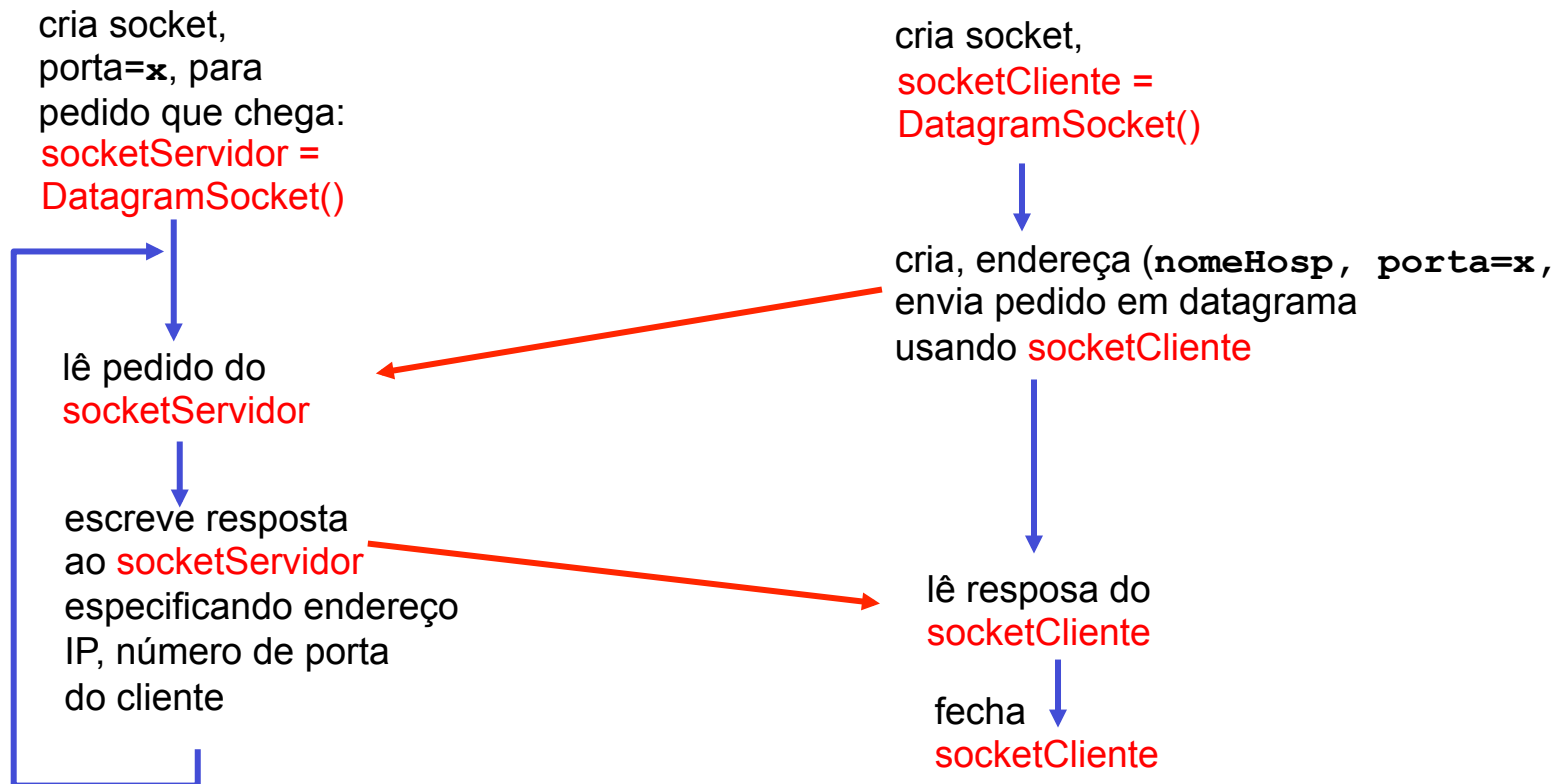
ponto de vista da aplicação

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

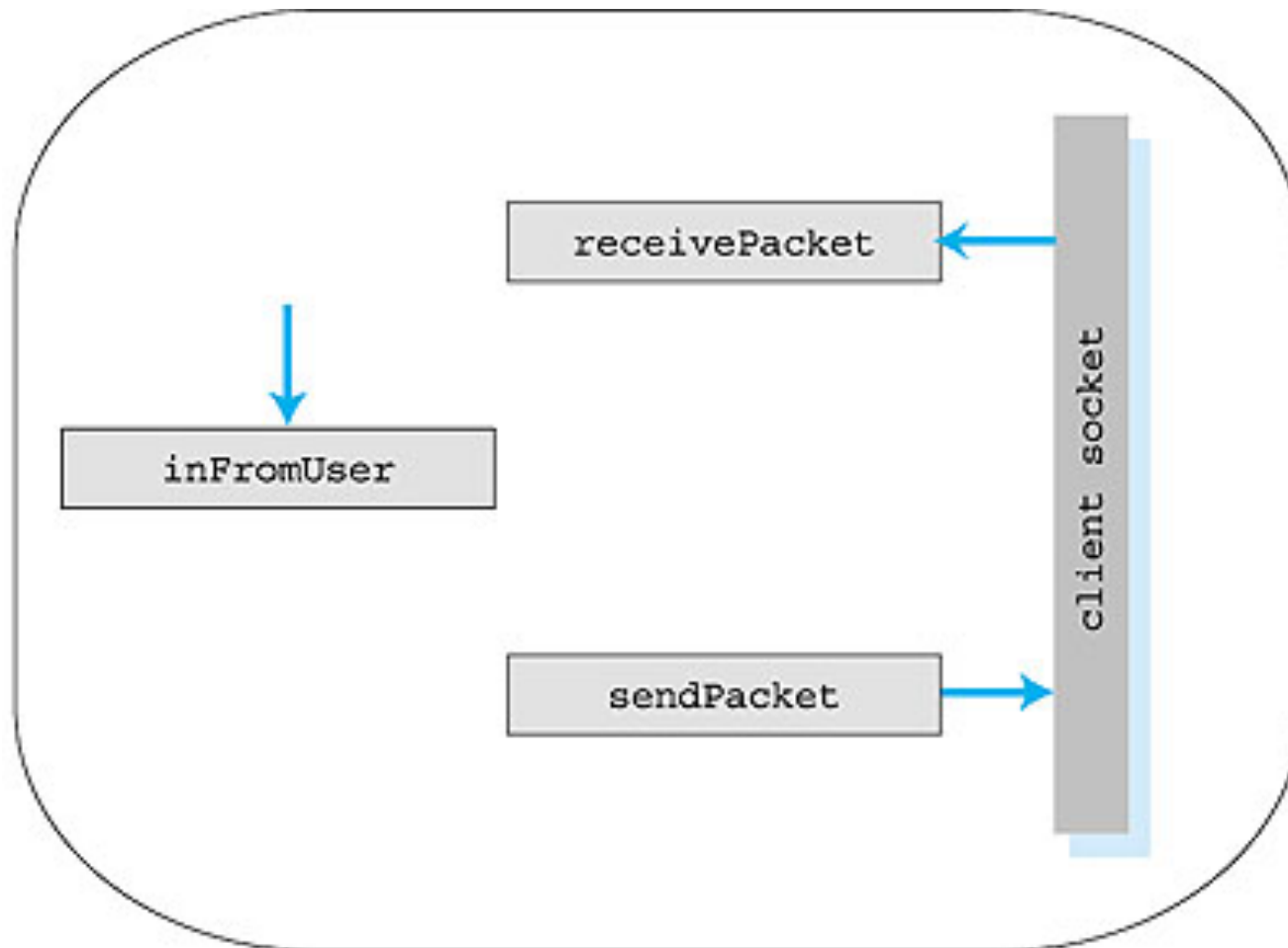
Interações cliente/servidor usando o UDP

Servidor (executa em `nomeHosp`)

Cliente



Cliente UDP



Exemplo: cliente Java (UDP)

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

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

Cria
fluxo de entrada

```
        BufferedReader do Usuario=  
            new BufferedReader(new InputStreamReader(System.in));
```

Cria
socket de cliente

```
        DatagramSocket socketCliente = new DatagramSocket();
```

Traduz nome de
hospedeiro ao
endereço IP
usando DNS

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

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

```
        String frase = doUsuario.readLine();  
        sendData = frase.getBytes();
```

Exemplo: cliente Java (UDP) cont.

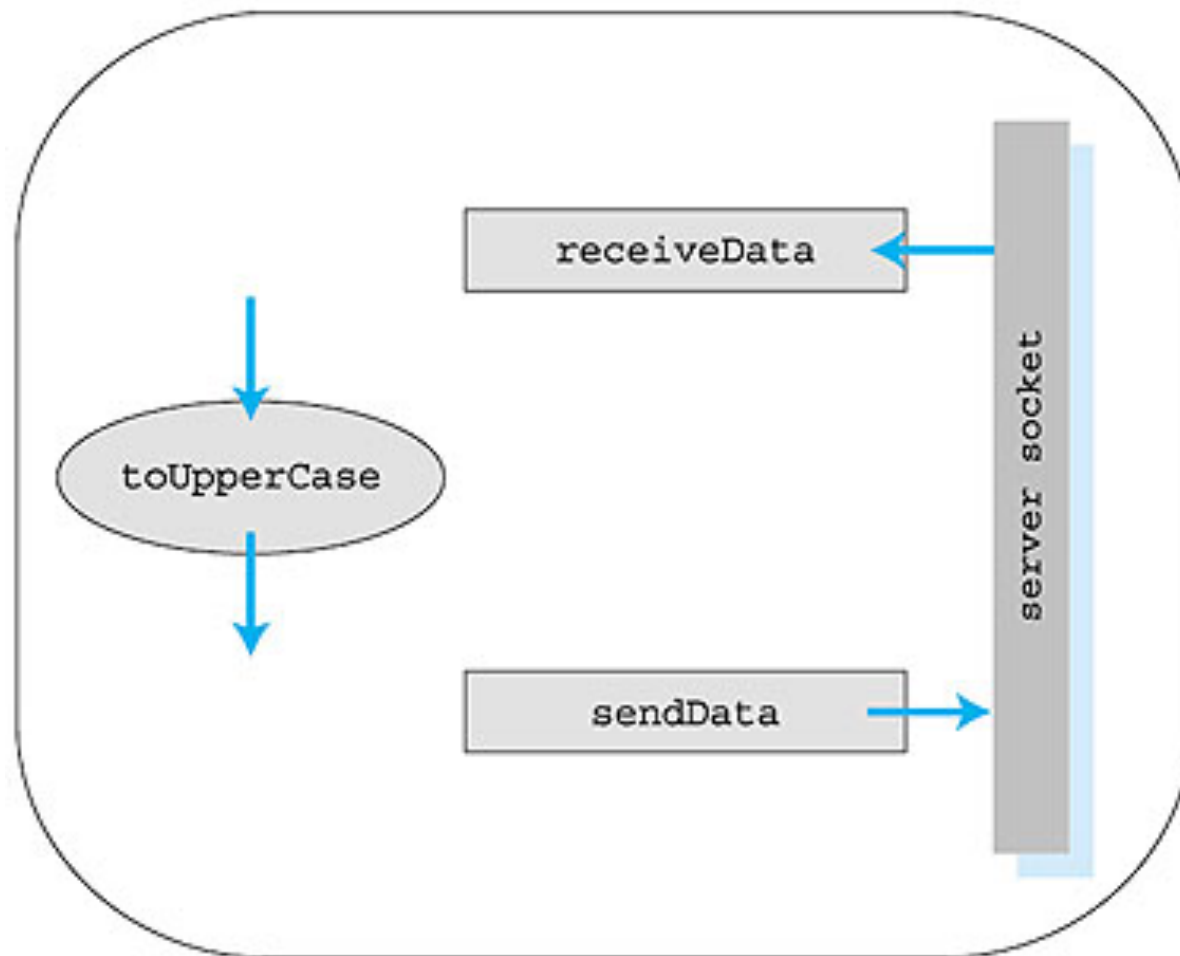
Cria datagrama com dados para enviar, comprimento, endereço IP, porta

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  
    {
```

Cria socket
para datagramas
na porta 9876

```
        DatagramSocket socketServidor = new DatagramSocket(9876);
```

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

```
        while(true)  
        {
```

Aloca memória para
receber datagrama

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

Recebe
datagrama

```
            socketServidor.receive(pacoteRecebido);
```

Exemplo: servidor Java (UDP), cont

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

Obtém endereço  
IP, no. de porta  
do remetente → InetAddress IPAddress = pacoteRecebido.getAddress();
               → int porta = pacoteRecebido.getPort();

String fraseEmMaiusculas = frase.toUpperCase();

dadosEnviados = fraseEmMaiusculas.getBytes();

Cria datagrama p/  
enviar ao cliente → DatagramPacket pacoteEnviado =
                  new DatagramPacket(dadosEnviados,
                                     dadosEnviados.length, IPAddress, porta);

Escreve  
datagrama  
para o socket → socketServidor.send(pacoteEnviado);
               }
               }
               }

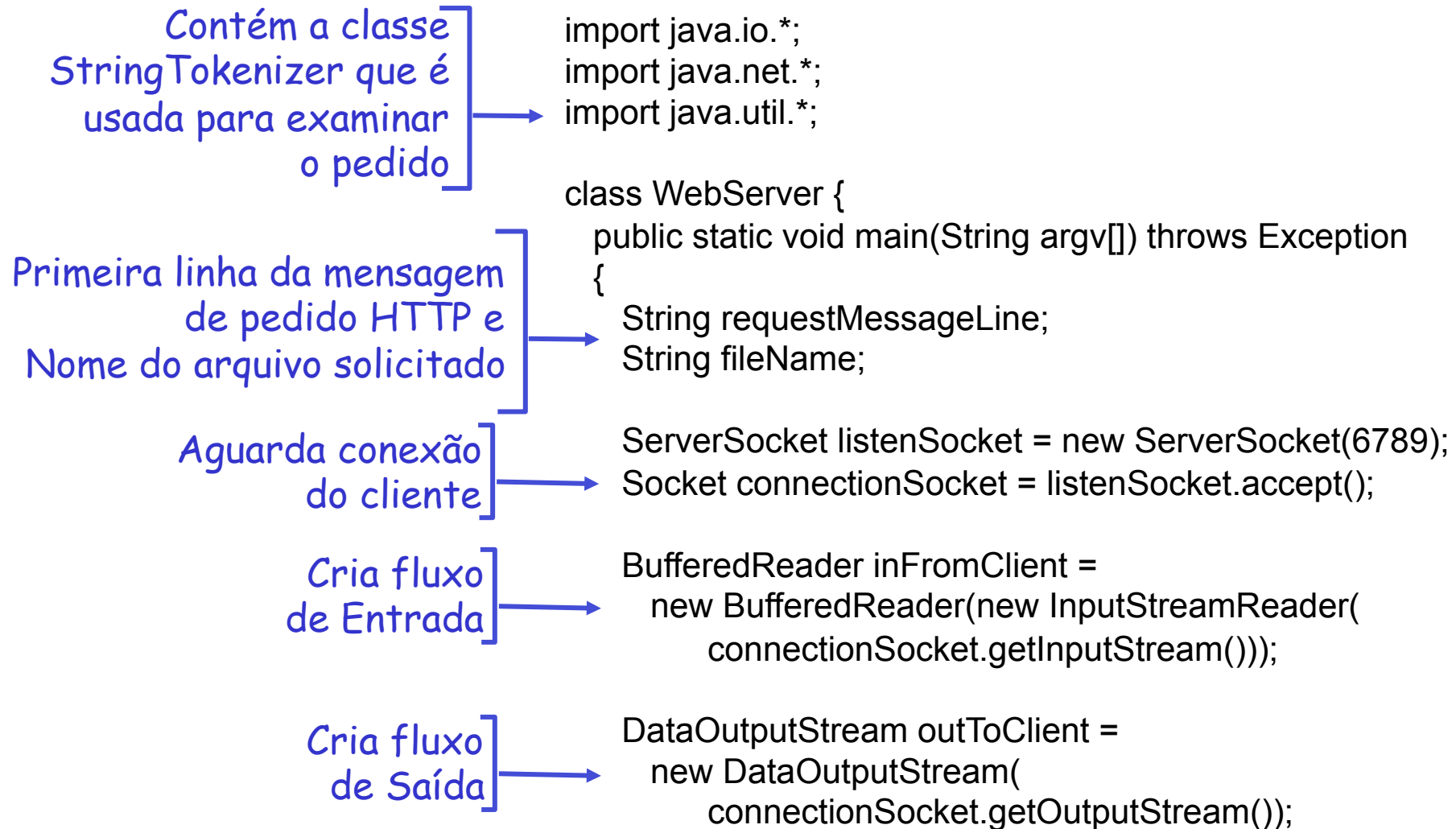
Fim do laço while,  
volta ao início e aguarda  
chegar outro datagrama
```

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



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);
```

Associa o fluxo inFile ao arquivo fileName

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

```
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

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

Transmissão do
cabeçalho da resposta
HTTP.

```
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