Capítulo 2: Camada de Aplicação

jorgelima@gmail.com

Baseado nos slides de Kurose e Ross

Capítulo 2: Roteiro

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 Transferência de arquivo: FTP
- 2.4 Correio Eletrônico na Internet
- 2.5 DNS: o serviço de diretório da Internet
- 2.6 Aplicações P2P
- 2.7 Programação e desenvolvimento de aplicações com TCP
- 2.8 Programação de *sockets* com UDP

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*
- aprender sobre protocolos através do estudo de protocolos populares da camada de aplicação:
  - HTTP
  - FTP
  - SMTP/ POP3/ IMAP
  - DNS
- Criar aplicações de rede
  - programação usando a API de *sockets*

Algumas aplicações de rede

- Correio eletrônico
- A Web
- Mensagens instantâneas
- Login em computador remoto como Telnet e SSH
- Compartilhamento de arquivos P2P
- Jogos multiusuários em rede
- *Streaming* de vídeos armazenados (YouTube, Hulu, Netflix)
- Telefonia por IP (Skype)
- Videoconferência em tempo real
- Busca
- ...
- ...

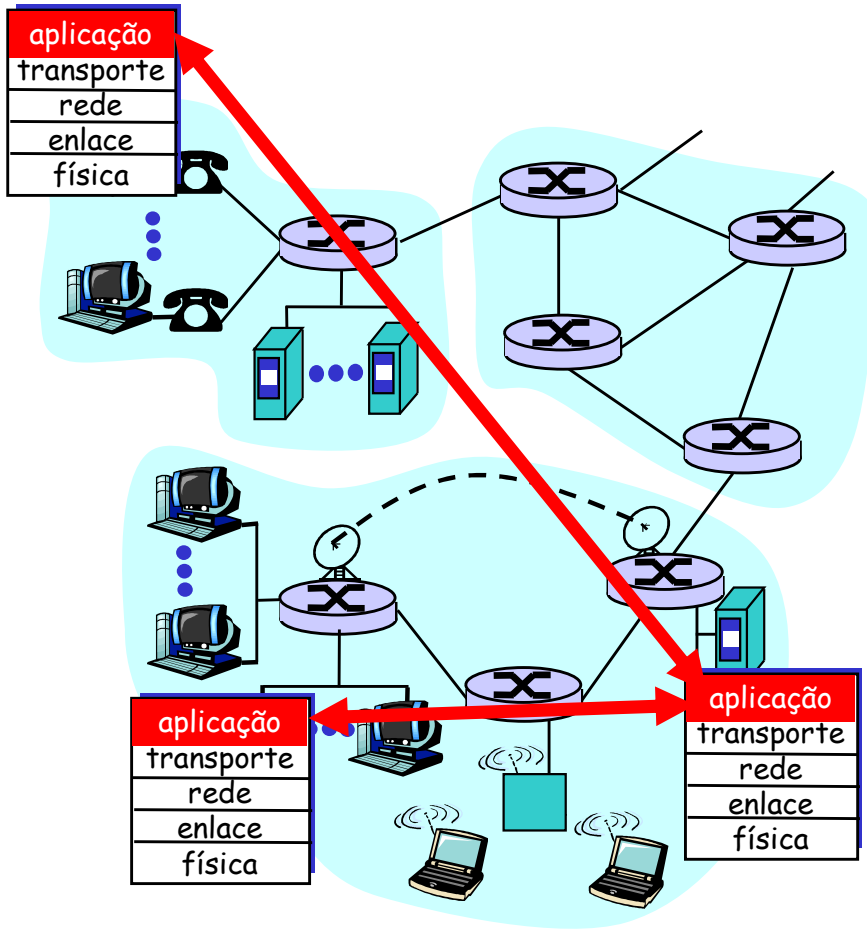
Criando uma aplicação de rede

Programas que

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

Programas não relacionados ao núcleo da rede

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



Arquiteturas das aplicações de rede

- Estruturas possíveis das aplicações:
  - Cliente-servidor
  - Peer-to-peer (P2P)

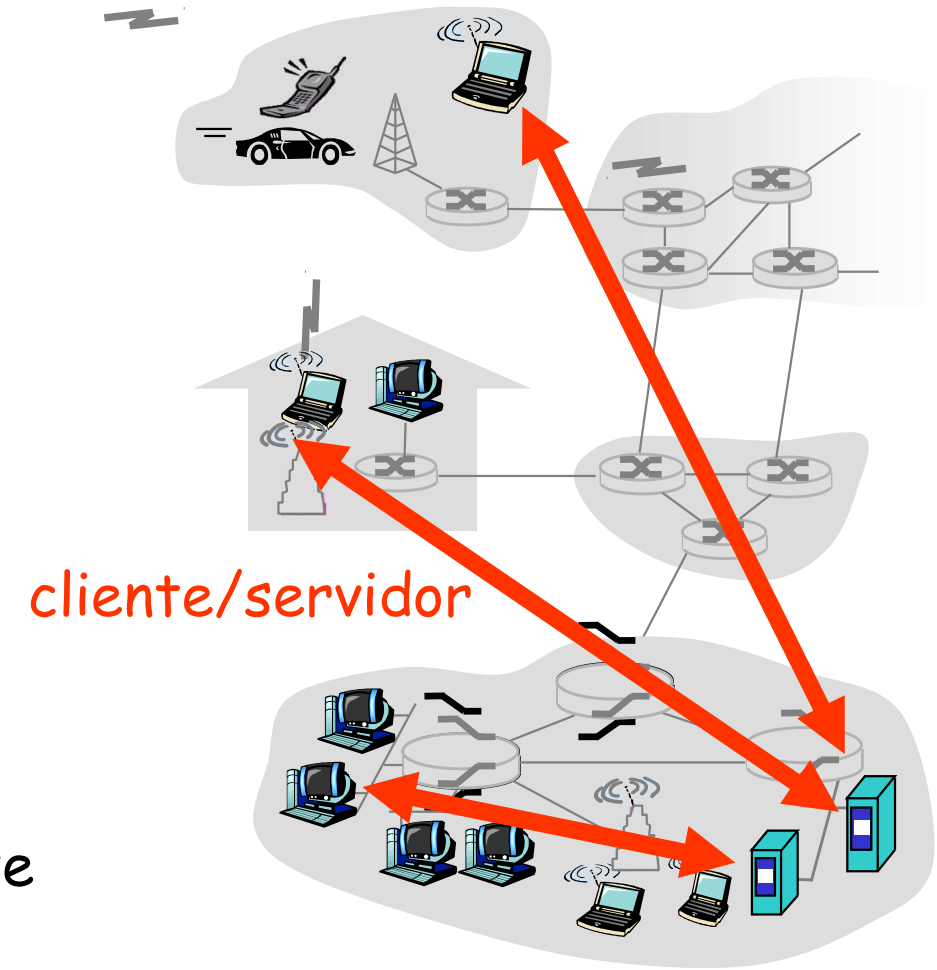
Arquitetura cliente-servidor

Servidor:

- Sempre ligado
- Endereço IP permanente
- Escalabilidade com *data centers*

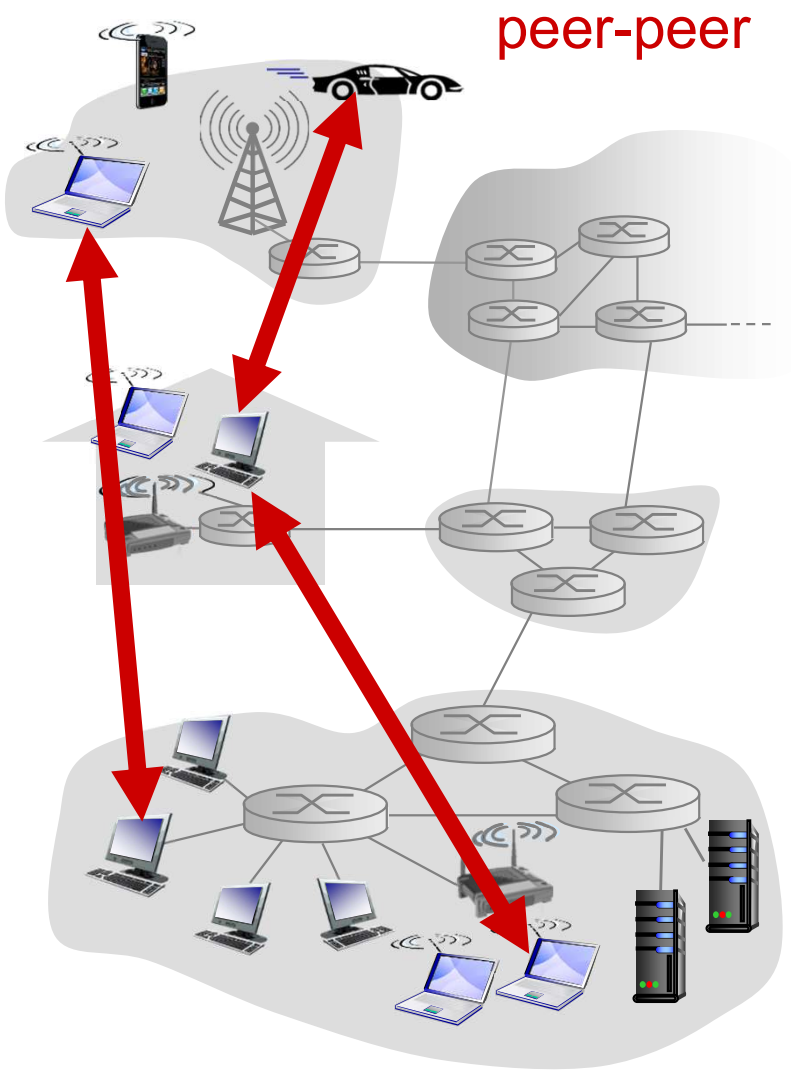
Clientes:

- Comunicam-se com o servidor
- Podem estar conectados intermitentemente
- Podem ter endereços IP dinâmicos
- Não se comunicam diretamente com outros clientes



Arquitetura P2P

- Não há servidor sempre ligado
- Sistemas finais arbitrários se comunicam diretamente
- Pares solicitam serviços de outros pares e em troca proveem serviços para outros parceiros:
  - Autoescalabilidade – novos pares trazem nova capacidade de serviço assim como novas demandas por serviços.
- Pares estão conectados intermitentemente e mudam endereços IP
  - Gerenciamento complexo



Comunicação entre Processos

Processo: programa que executa num sistema final

- processos no mesmo sistema final se comunicam usando comunicação interprocessos (definida pelo sistema operacional)
- processos em sistemas finais distintos se comunicam trocando mensagens através da rede

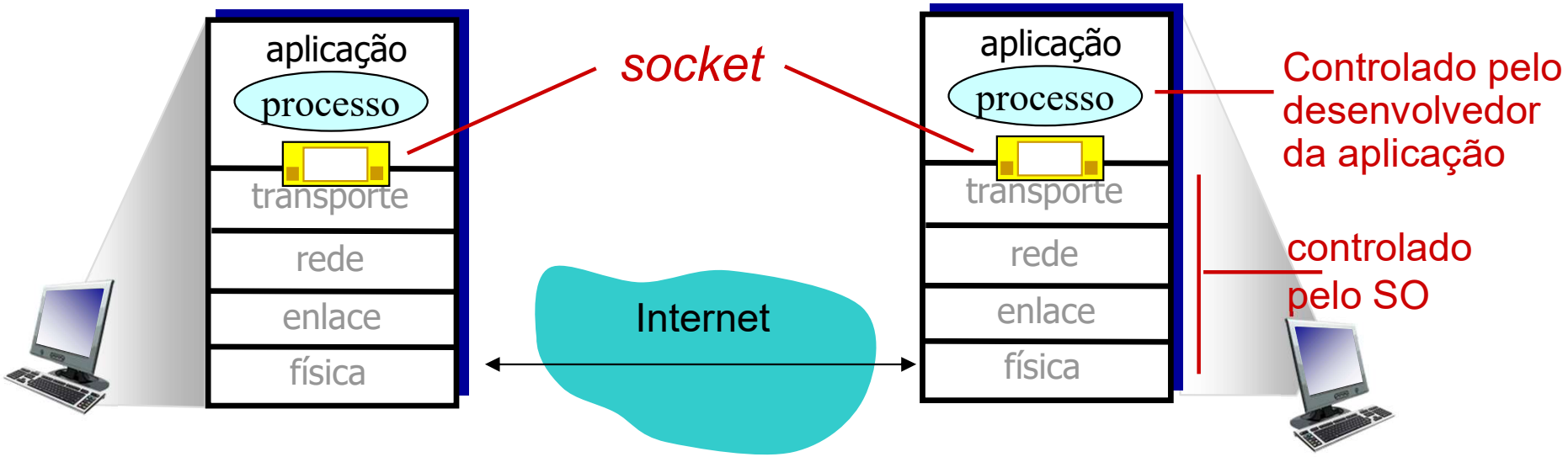
Processo cliente:
processo que inicia a comunicação

Processo servidor:
processo que espera 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



Endereçamento de processos

- Para que um processo receba mensagens, ele deve possuir um identificador
- Cada hospedeiro possui um endereço IP único de 32 bits
- P: o endereço IP do hospedeiro no qual o processo está sendo executado é suficiente para identificar o processo?
- Resposta: Não, muitos processos podem estar executando no mesmo hospedeiro

- O identificador inclui tanto o endereço IP quanto os números das portas associadas com o processo no hospedeiro .
- Exemplo de números de portas:
  - Servidor HTTP: 80
  - Servidor de Correio: 25
- Para enviar uma msg HTTP para o servidor Web gaia.cs.umass.edu
  - Endereço IP: 128.119.245.12
  - Número da porta: 80
- Mais sobre isto posteriormente.

Os protocolos da camada de aplicação definem

- Tipos de mensagens trocadas:
  - ex. mensagens de requisição e resposta
- Sintaxe das mensagens:
  - campos presentes nas mensagens e como são identificados
- Semântica das msgs:
  - significado da informação nos campos
- Regras para quando os processos enviam e respondem às mensagens

Protocolos abertos:

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

Protocolos proprietários:

- Ex., Skype

De que serviços uma aplicação necessita?

Integridade dos dados (sensibilidade a perdas)

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

Temporização (sensibilidade a atrasos)

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

Vazão (*throughput*)

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

Segurança

- Criptografia, integridade dos dados, ...

Requisitos de aplicações de rede selecionadas

Aplicação	Sensib. a Perdas	Vazão	Sensibilidade a atrasos
transferência de arqs	sem perdas	elástica	não
correio	sem perdas	elástica	não
documentos Web	sem perdas	elástica	não
áudio/vídeo em tempo real	tolerante	áudio: 5kbps-1Mbps vídeo:10kbps-5Mbps	sim, 100's mseg
áudio/vídeo gravado	tolerante	Igual acima	sim, alguns segs
jogos interativos	tolerante	Alguns kbps-10Mbps	sim, 100's mseg
mensagem instantânea	sem perdas	elástica	sim e não

Serviços providos pelos protocolos de transporte da Internet

Serviço TCP:

- *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
- *orientado a conexão:* apresentação requerida entre cliente e servidor

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 protocolo como o 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
Web	HTTP [RFC 2616]	TCP
transferência de arquivos	FTP [RFC 959]	TCP
streaming multimídia	HTTP (ex. Youtube) RTP [RFC 1889]	TCP ou UDP
telefonia Internet	SIP, RTP, proprietário (ex., Skype)	TCP ou UDP

Tornando o TCP seguro

TCP & UDP

- Sem criptografia
- Senhas em texto aberto enviadas aos sockets atravessam a Internet em texto aberto

SSL

- Provê conexão TCP criptografada
- Integridade dos dados
- Autenticação do ponto terminal

SSL está na camada de aplicação

- Aplicações usam bibliotecas SSL, que "falam" com o TCP

API do socket SSL

- Senhas em texto aberto enviadas ao socket atravessam a rede criptografadas
- Vide Capítulo 7

Capítulo 2: Roteiro

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 Transferência de arquivo: FTP
- 2.4 Correio Eletrônico na Internet
- 2.5 DNS: o serviço de diretório da Internet
- 2.6 Aplicações P2P
- 2.7 Programação e desenvolvimento de aplicações com TCP
- 2.8 Programação de *sockets* com UDP

A Web e o HTTP

Primeiro uma revisão...

- Páginas Web consistem de objetos
- um objeto pode ser um arquivo HTML, uma imagem JPEG, um applet Java, um arquivo de áudio,...
- Páginas Web consistem de um arquivo base HTML 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 (usando o protocolo HTTP) e "visualiza" objetos Web
  - *servidor:* servidor Web envia (usando o protocolo HTTP) objetos em resposta a pedidos

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
  - A conexão é então encerrada
- Baixar múltiplos objetos requer o uso de múltiplas conexões

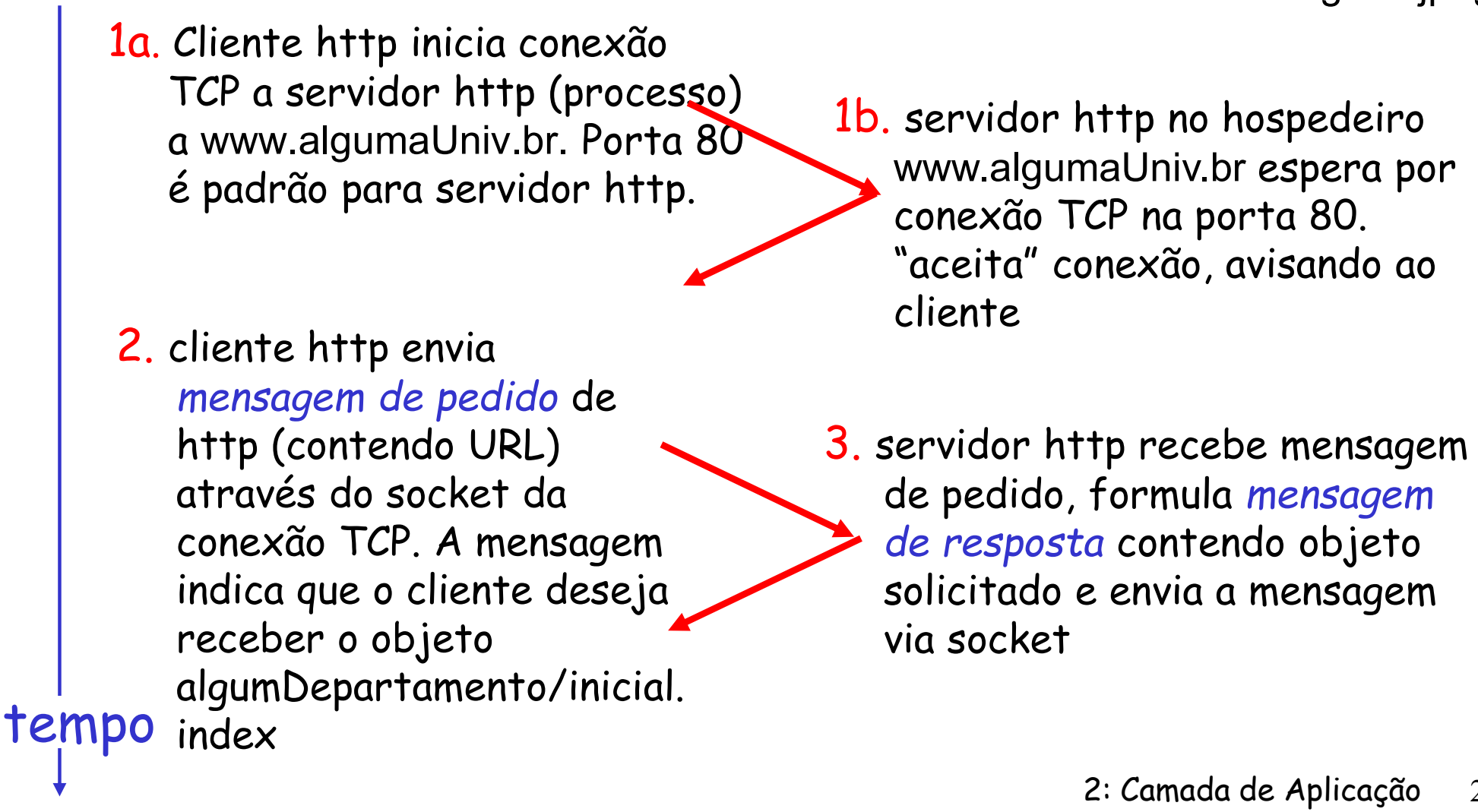
HTTP persistente

- Múltiplos objetos podem ser enviados sobre uma única conexão TCP entre cliente e servidor

Exemplo de HTTP não persistente

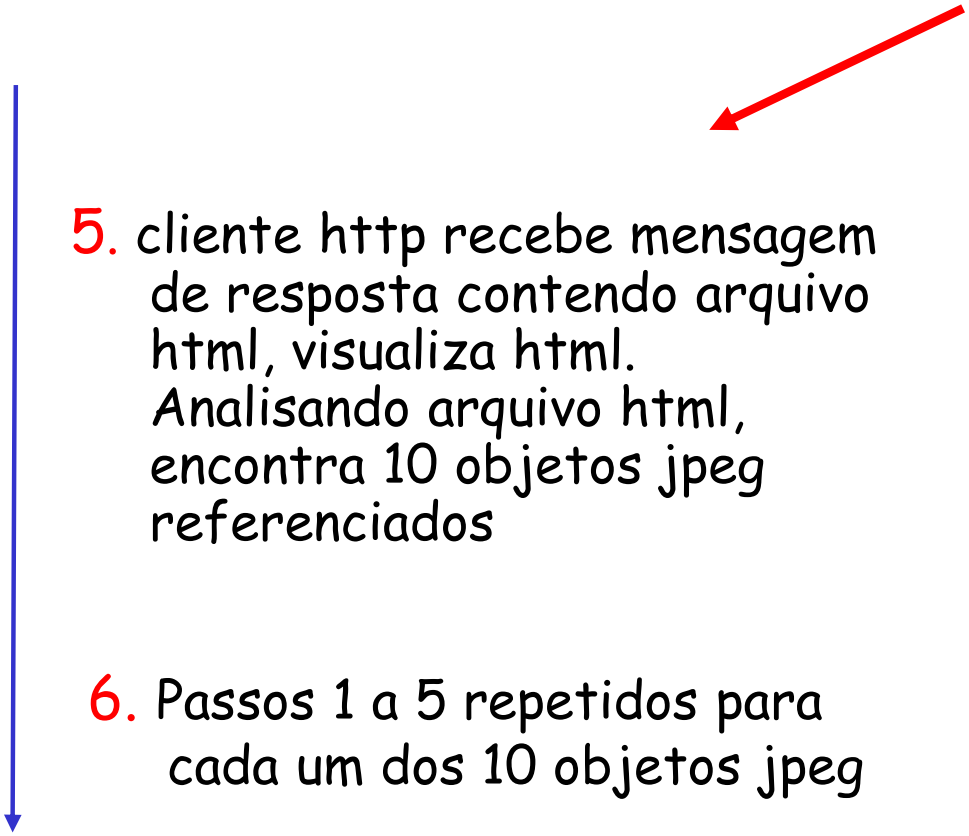
Supomos que usuário digita a URL
www.algumaUniv.br/algumDepartmento/inicial.index

(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

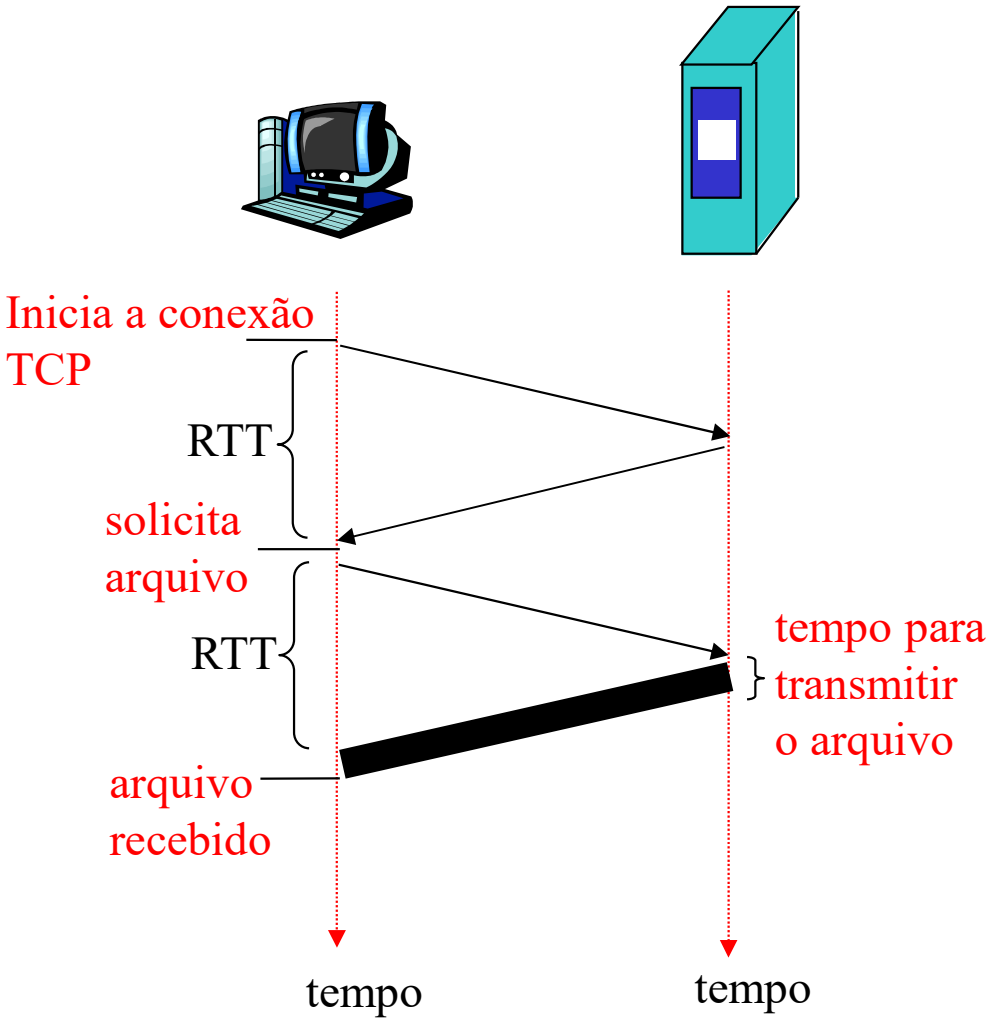
Modelagem do tempo de resposta

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+tempo de transmissão do arquivo



HTTP persistente

Problemas com o HTTP não persistente:

- requer 2 RTTs para cada objeto
- SO aloca recursos do hospedeiro (*overhead*) para cada conexão TCP
- os *browser* 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 aberta
- o cliente envia os pedidos logo que encontra um objeto referenciado
- pode ser necessário apenas um RTT para todos os objetos referenciados

Mensagem de requisição HTTP

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

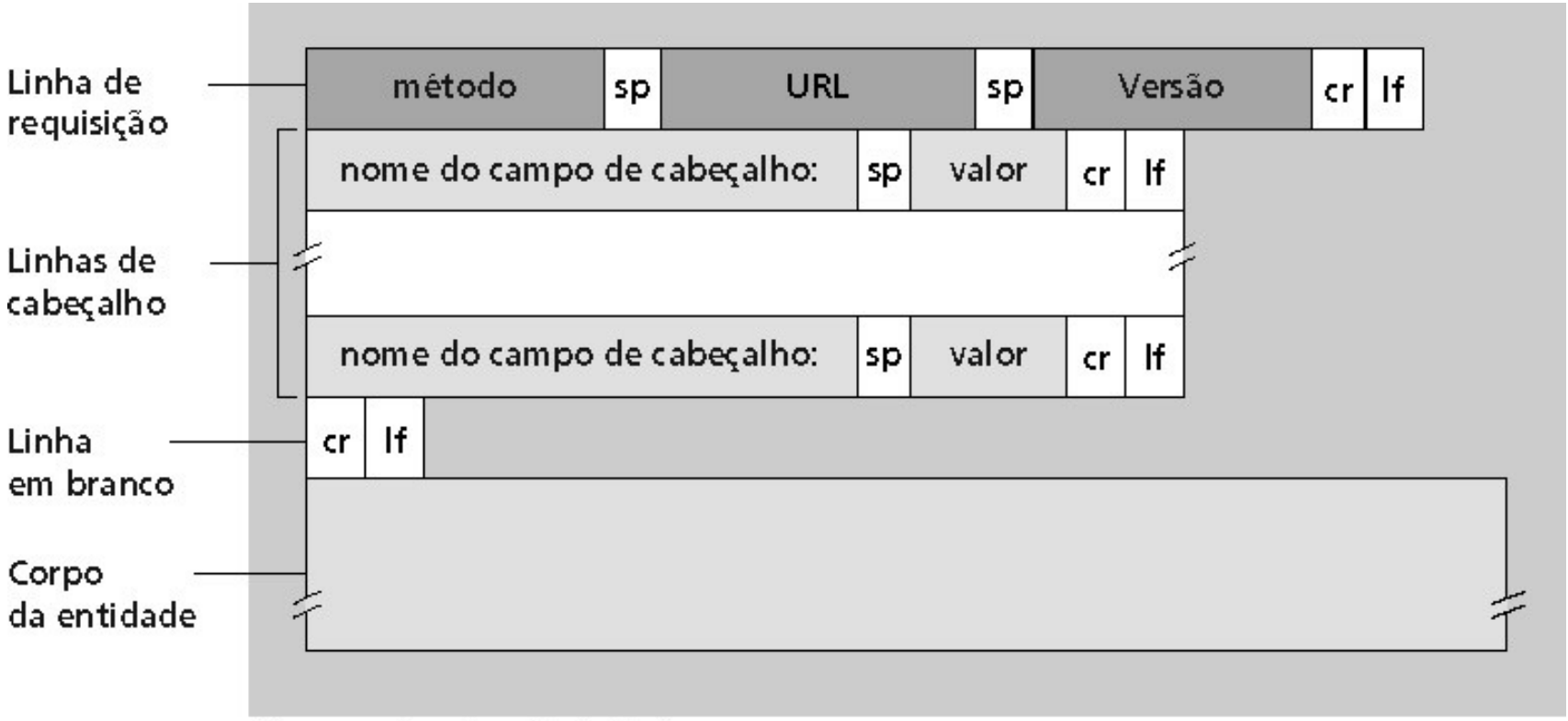
linha da requisição (comandos GET, POST, HEAD)

linhas de cabeçalho

Carriage return, line feed indicam fim de mensagem

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

Mensagem de requisição HTTP: formato geral



Obs.: cr = carriage return; lf = line feed

Enviando conteúdo de formulário

Método POST :

- Páginas Web frequentemente contêm formulário de entrada
- Conteúdo é enviado para o servidor no corpo da mensagem

Método URL:

- Usa o método GET
- Conteúdo é enviado para o servidor no campo URL:

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

Tipos de métodos

HTTP/1.0

- GET
- POST
- HEAD
  - Pede para o servidor não enviar o objeto requerido junto com a resposta

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

Mensagem de resposta HTTP

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

linhas de cabeçalho

dados, p.ex., arquivo html solicitado

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```

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 cis.poly.edu 80
```

Abre conexão TCP para a porta 80 (porta padrão do servidor http) a cis.poly.edu. Qualquer coisa digitada é enviada para a porta 80 do cis.poly.edu

2. Digite um pedido GET HTTP:

```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

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 !

(ou use Wireshark para ver as msgs de pedido/resposta HTTP capturadas)

Cookies: manutenção do "estado" da conexão

Muitos dos principais sítios Web usam *cookies*

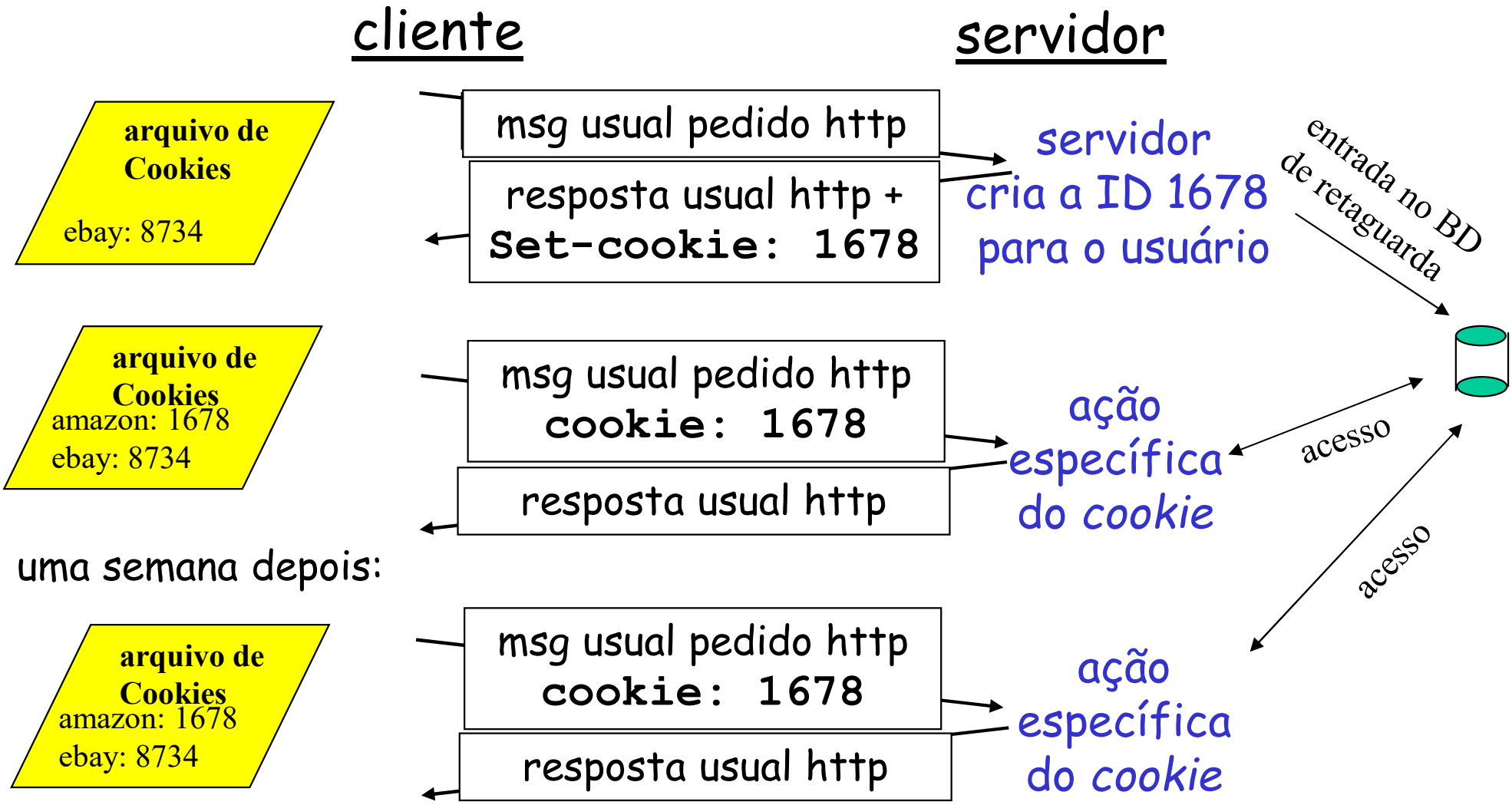
Quatro componentes:

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
  - 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
- recomendações
- 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

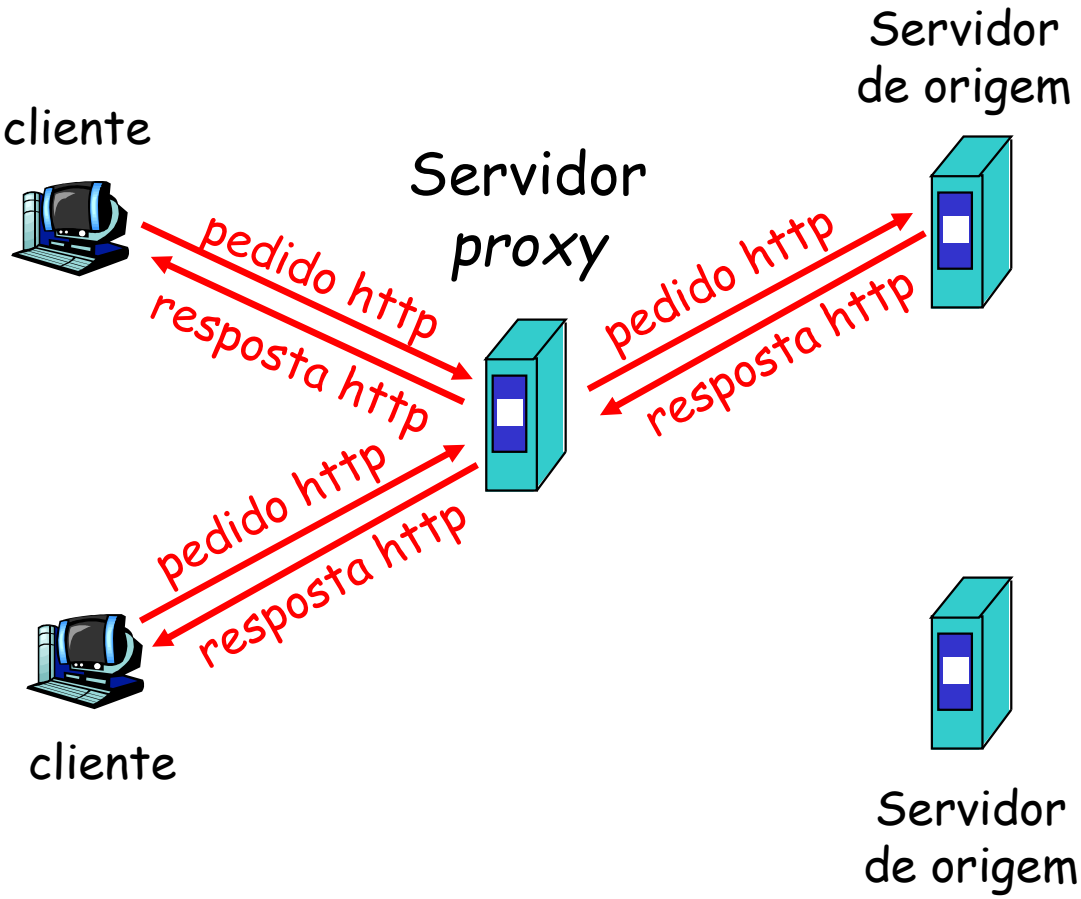
Como manter o "estado":

- Pontos finais do protocolo: mantêm o estado no transmissor/receptor para múltiplas transações
- *Cookies*: mensagens http transportam o estado

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 estiver 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 (mas o compartilhamento de arquivos P2P também!)

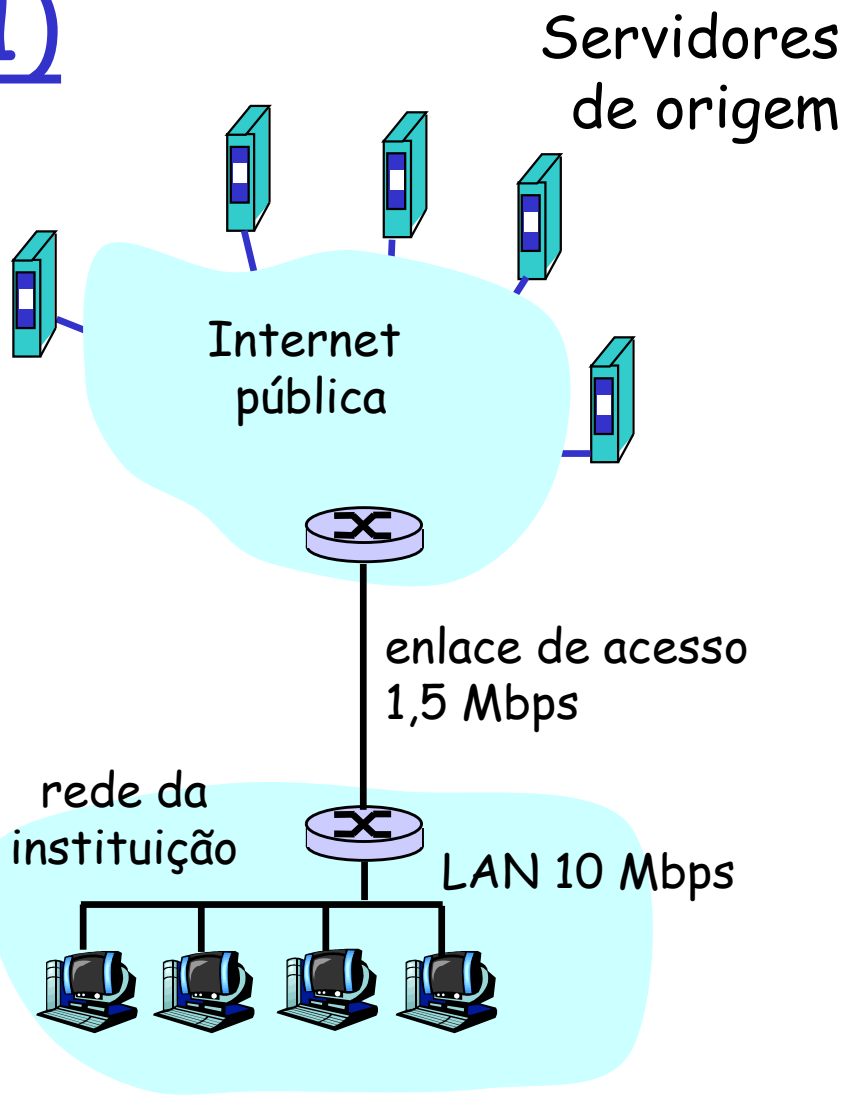
Exemplo de cache (1)

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

Conseqüências

- Utilização da LAN = 15%
- Utilização do canal de acesso = 100% *problema!*
- Atraso total = atraso da Internet + atraso de acesso + atraso na LAN = 2 seg + minutos + microssegundos



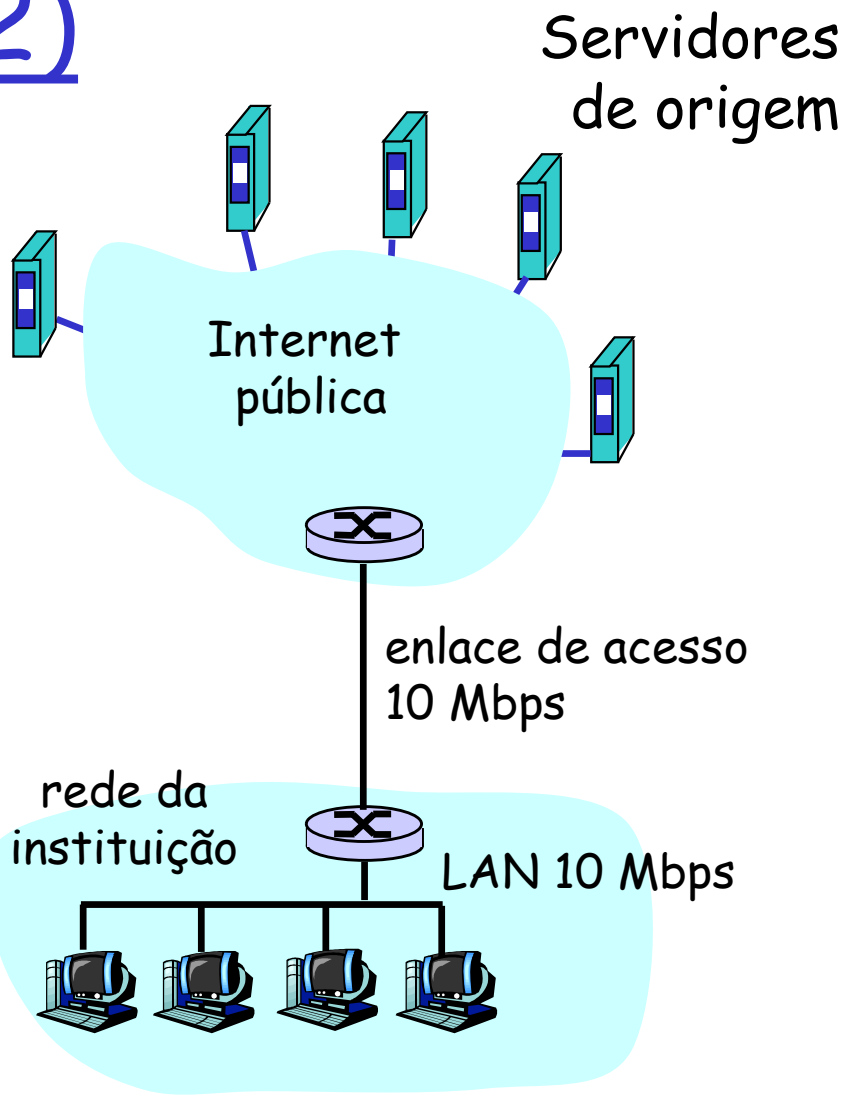
Exemplo de cache (2)

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 + msegs + msegs
- Frequentemente este é uma ampliação cara



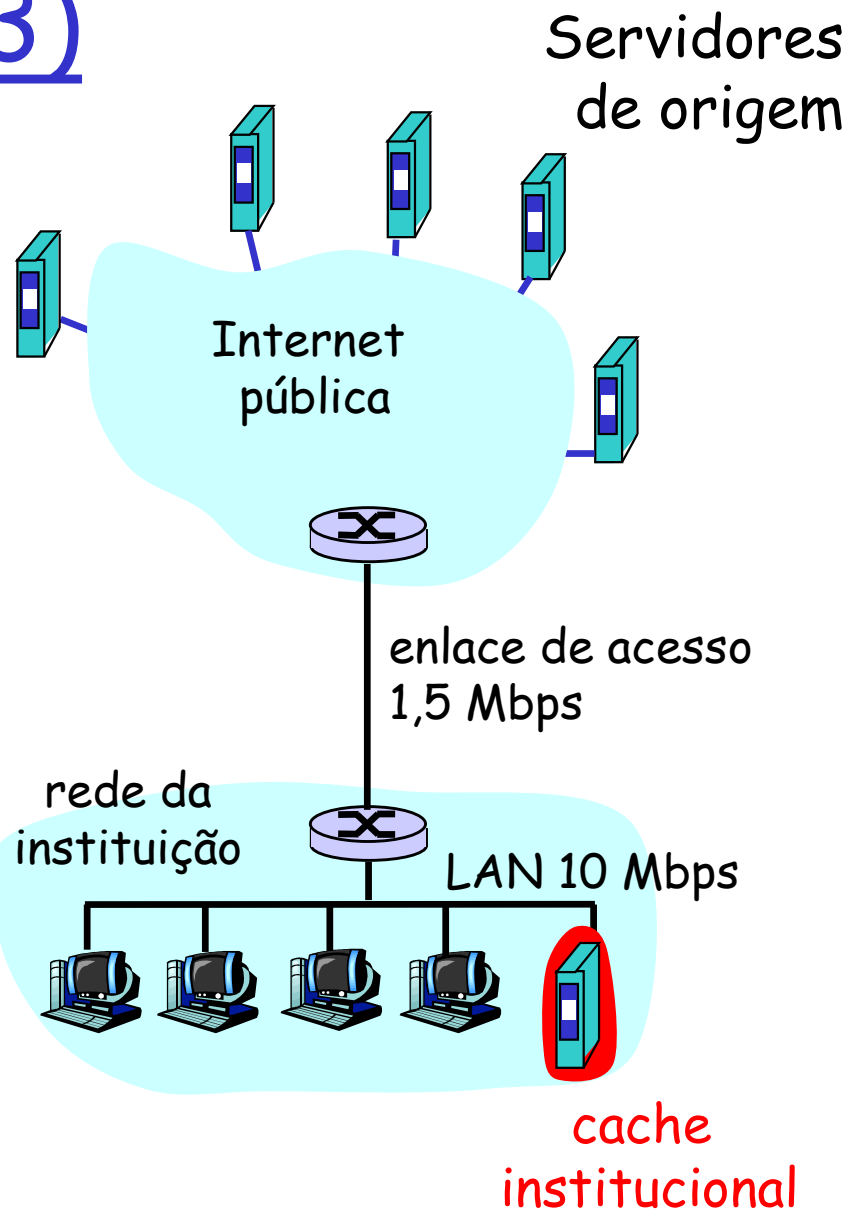
Exemplo de cache (3)

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*2 seg + 0,6*0,01 segs + msegs < 1,3 segs



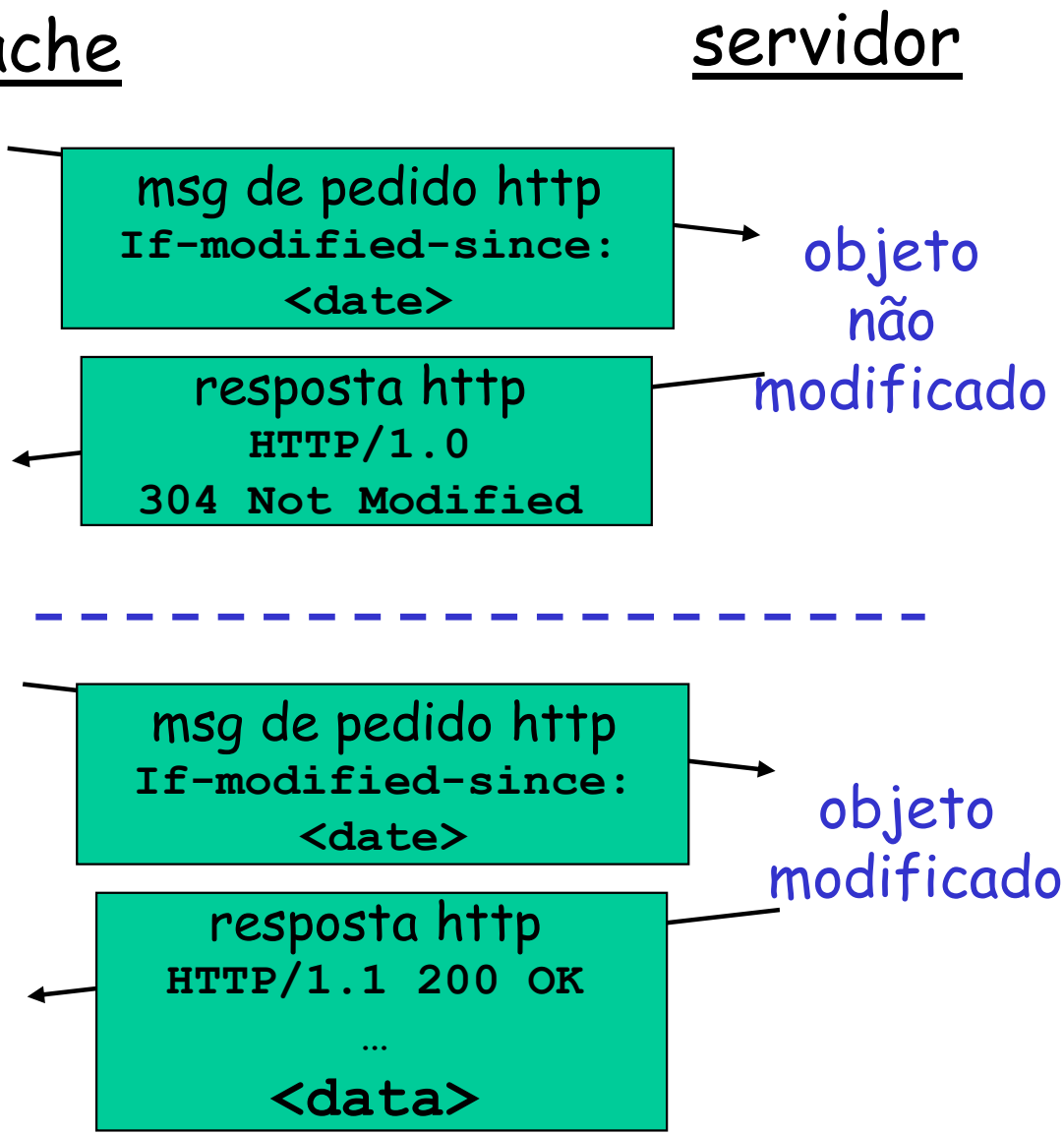
GET condicional

- Meta: não enviar objeto se cliente já tem (no cache) versão atual
  - Sem atraso para transmissão do objeto
  - Diminui a utilização do enlace
- cache: especifica data da cópia no cache no pedido HTTP

```
If-modified-since:
   <date>
```

- servidor: resposta não contém objeto se cópia no cache for atual:

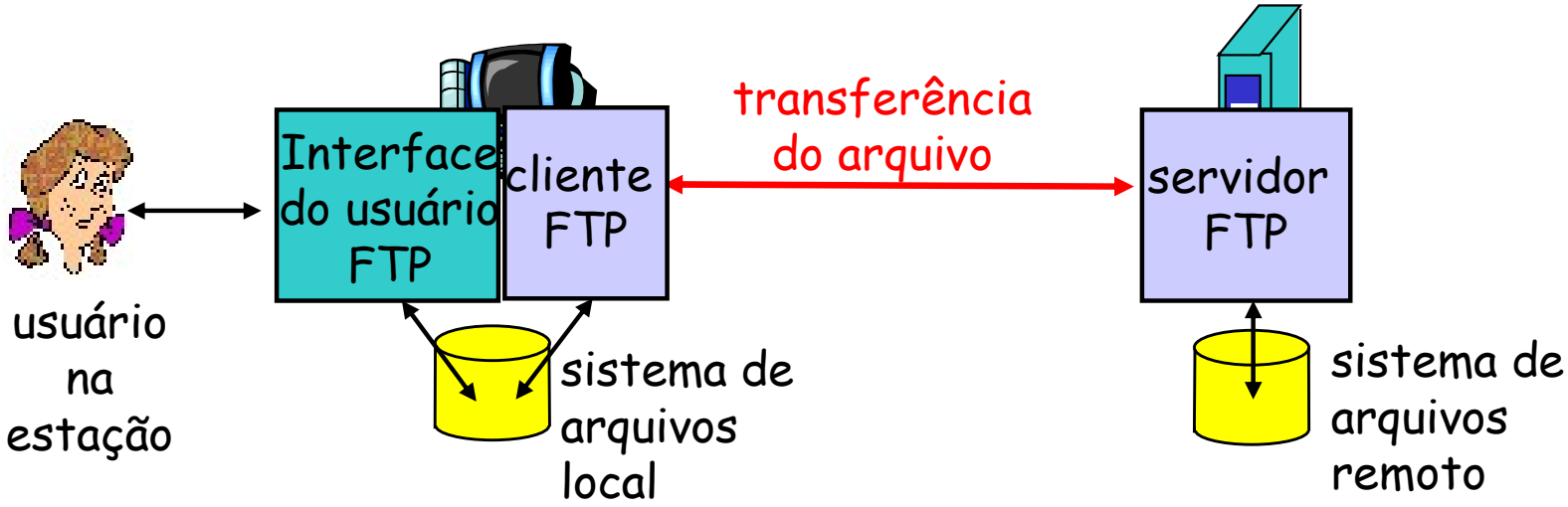
```
HTTP/1.0 304 Not
   Modified
```



Capítulo 2: Roteiro

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 Transferência de arquivo: FTP
- 2.4 Correio Eletrônico na Internet
- 2.5 DNS: o serviço de diretório da Internet
- 2.6 Aplicações P2P
- 2.7 Programação e desenvolvimento de aplicações com TCP
- 2.8 Programação de *sockets* com UDP

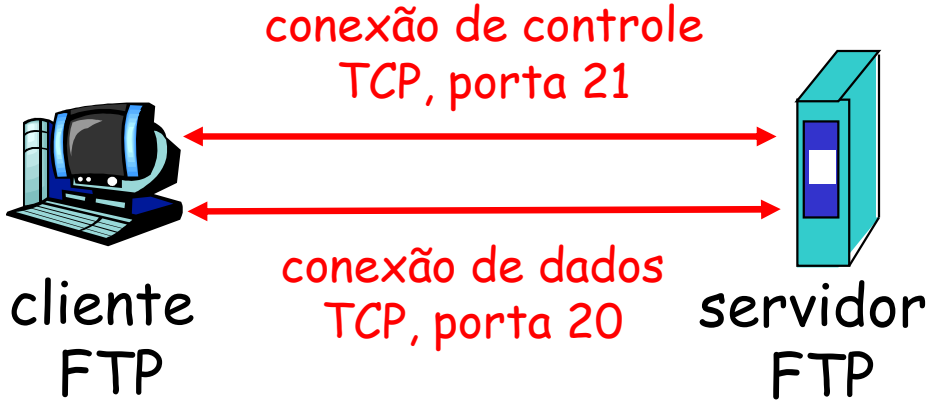
FTP: o protocolo de transferência de arquivos



- 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**

Capítulo 2: Roteiro

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 Transferência de arquivo: FTP
- 2.4 Correio Eletrônico na Internet
- 2.5 DNS: o serviço de diretório da Internet
- 2.6 Aplicações P2P
- 2.7 Programação e desenvolvimento de aplicações com TCP
- 2.8 Programação de *sockets* com UDP

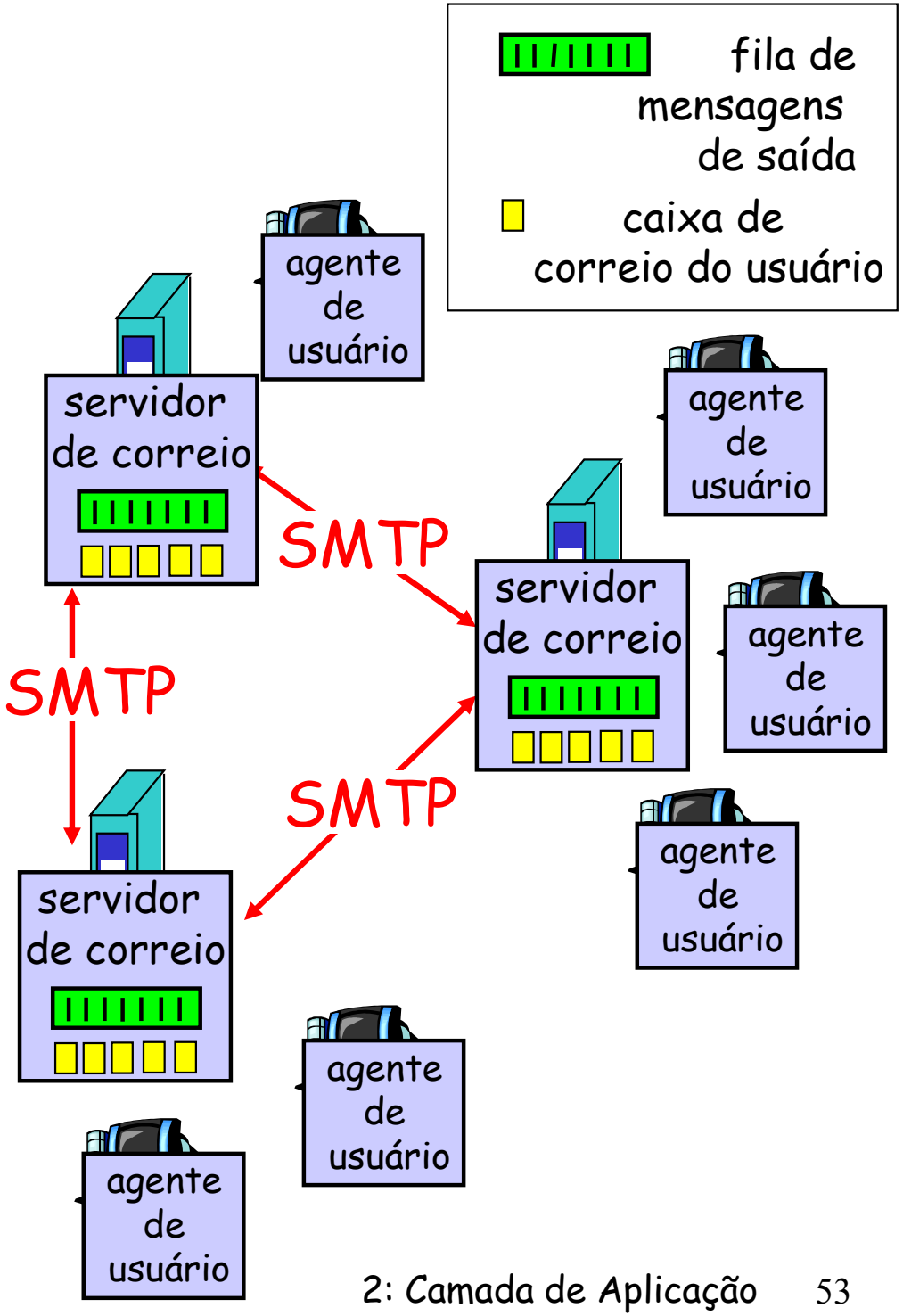
Correio Eletrônico

Três grandes componentes:

- agentes de usuário (UA)
- servidores de correio
- *Simple Mail Transfer Protocol*: SMTP

Agente de Usuário

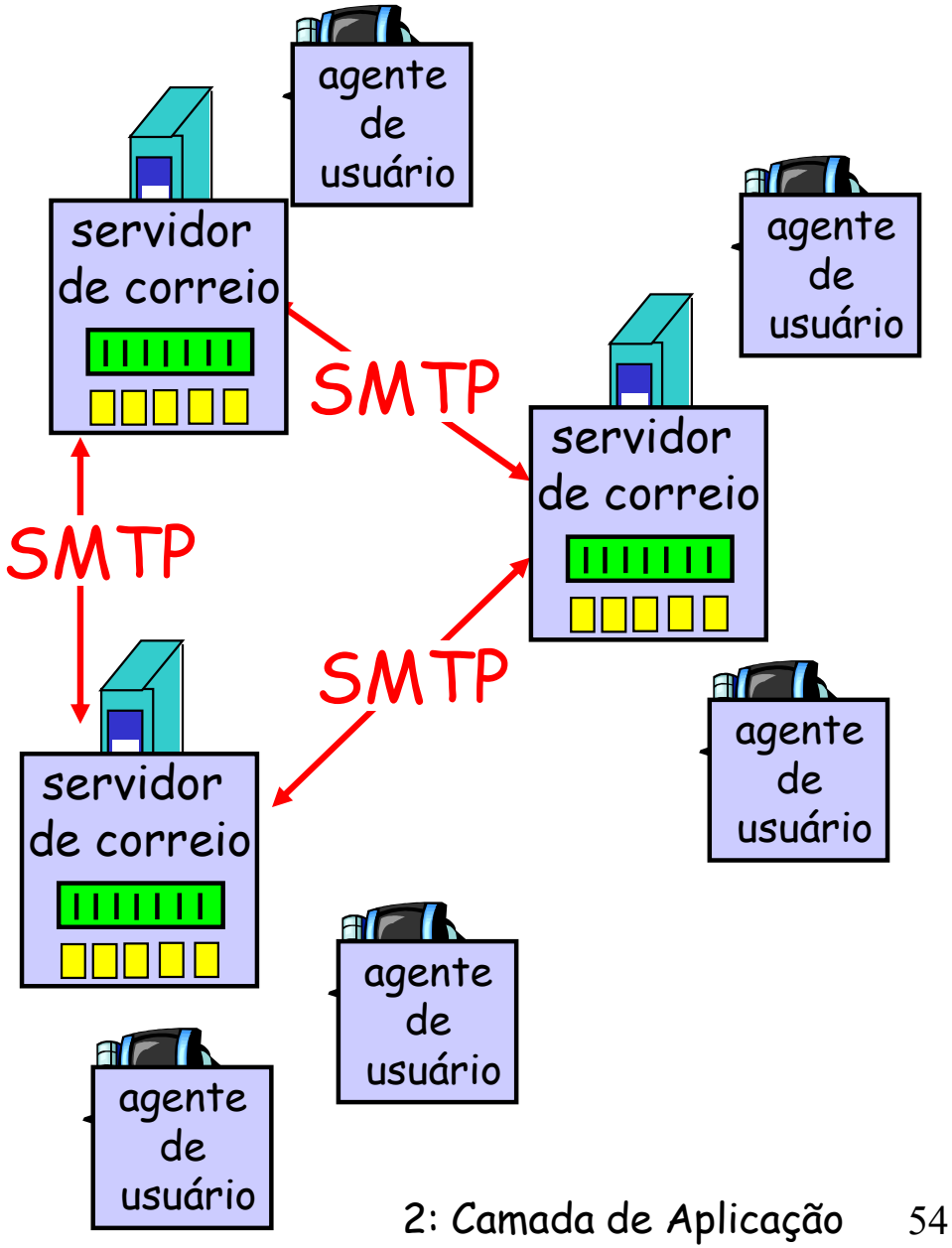
- a.k.a. "leitor de correio"
- compor, editar, ler mensagens de correio
- p.ex., Outlook, Thunderbird, cliente de mail do iPhone
- mensagens de saída e chegando são armazenadas no servidor



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]

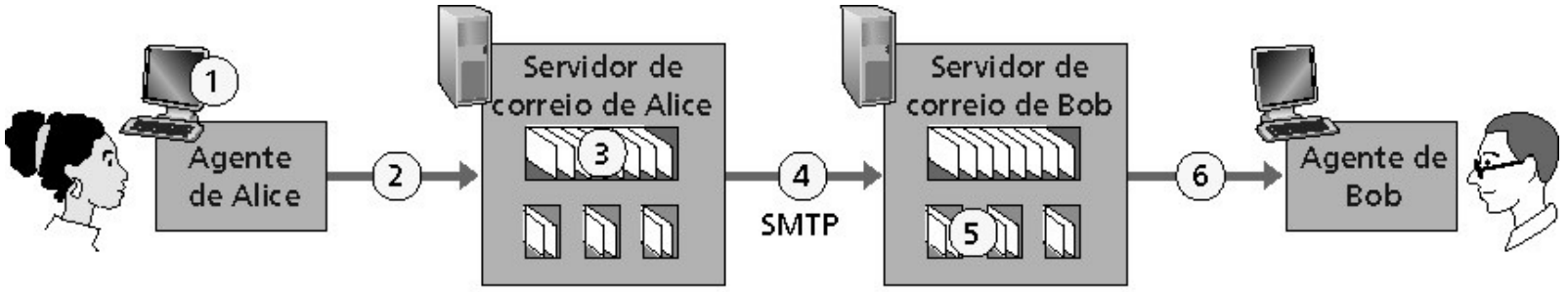
- usa TCP para a transferência confiável de msgs do correio do cliente ao servidor, porta 25
- transferência direta: servidor remetente ao servidor receptor
- três fases da transferência
  - *handshaking* (saudação)
  - transferência das mensagens
  - encerramento
- interação comando/resposta (como o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

Gerência da Porta 25

http://antispam.br/

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



Legenda:

Fila de mensagens

Caixa postal do usuário

Interação SMTP típica

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr ... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu 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* (recupera)
- SMTP: *push* (envia)
- 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

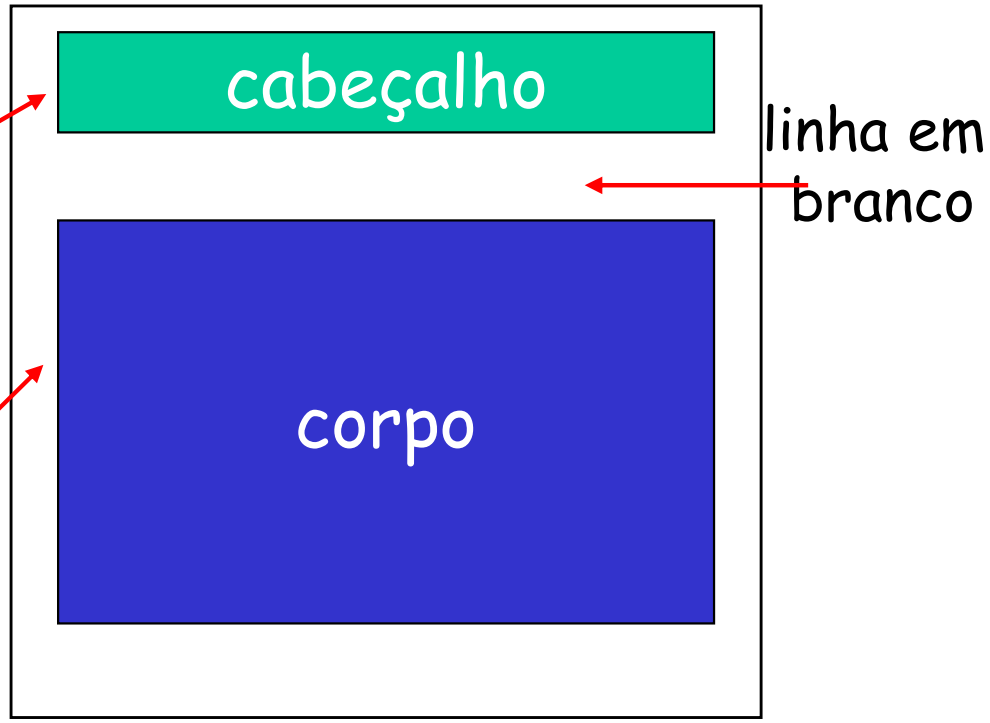
Formato de uma mensagem

SMTP: protocolo para trocar msgs de correio

RFC 822: padrão para formato de mensagem de texto:

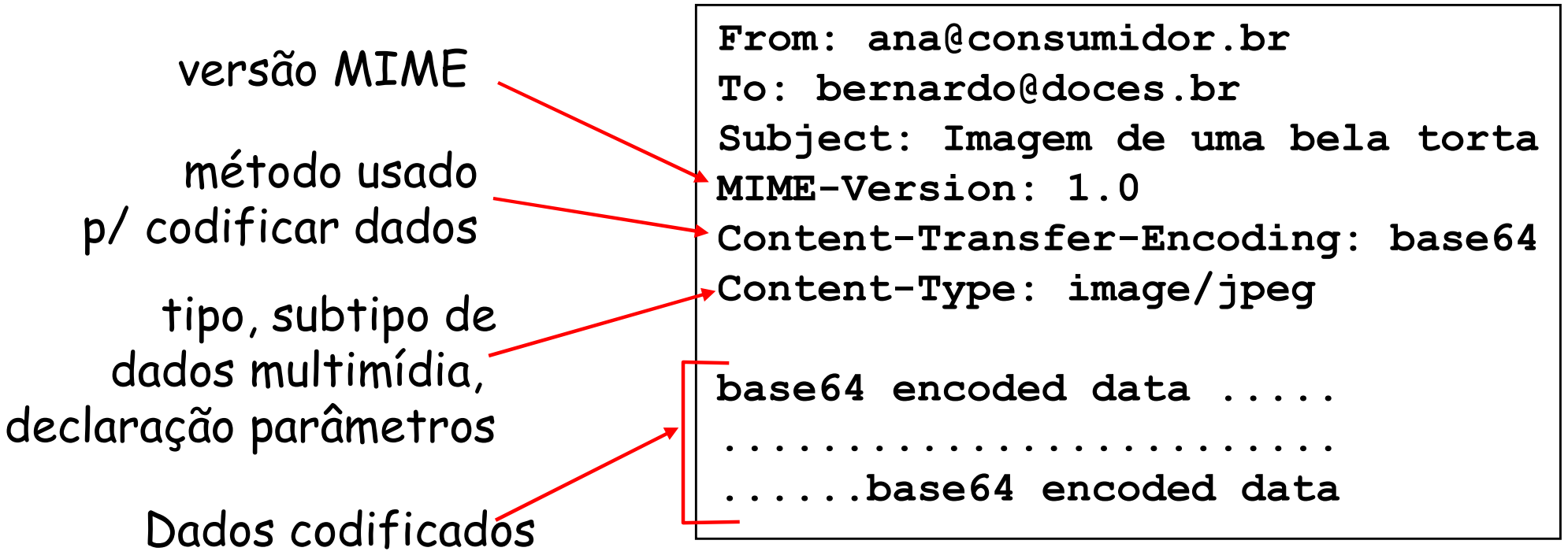
- linhas de cabeçalho, p.ex.,
  - To:
  - From:
  - Subject:

  diferentes dos comandos de smtp FROM, RCPT TO
- 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



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

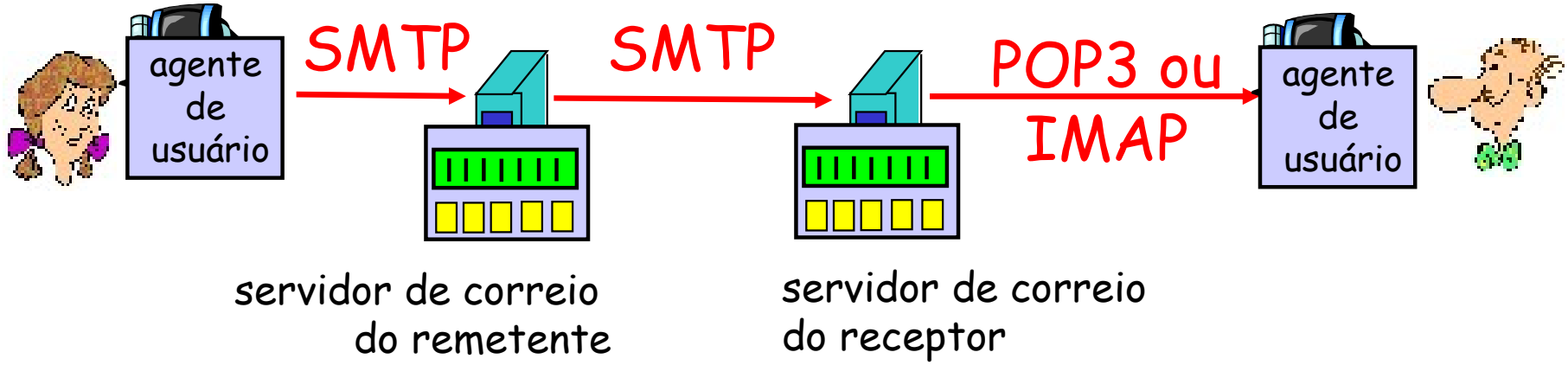
```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=98766789

--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: gmail, Hotmail , Yahoo! Mail, 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

Capítulo 2: Roteiro

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 Transferência de arquivo: FTP
- 2.4 Correio Eletrônico na Internet
- 2.5 DNS: o serviço de diretório da Internet
- 2.6 Aplicações P2P
- 2.7 Programação e desenvolvimento de aplicações com TCP
- 2.8 Programação de *sockets* com UDP

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., www.yahoo.com - 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)
  - nota: função imprescindível da Internet implementada como protocolo de camada de aplicação
  - complexidade na borda da rede

DNS (cont.)

Serviços DNS

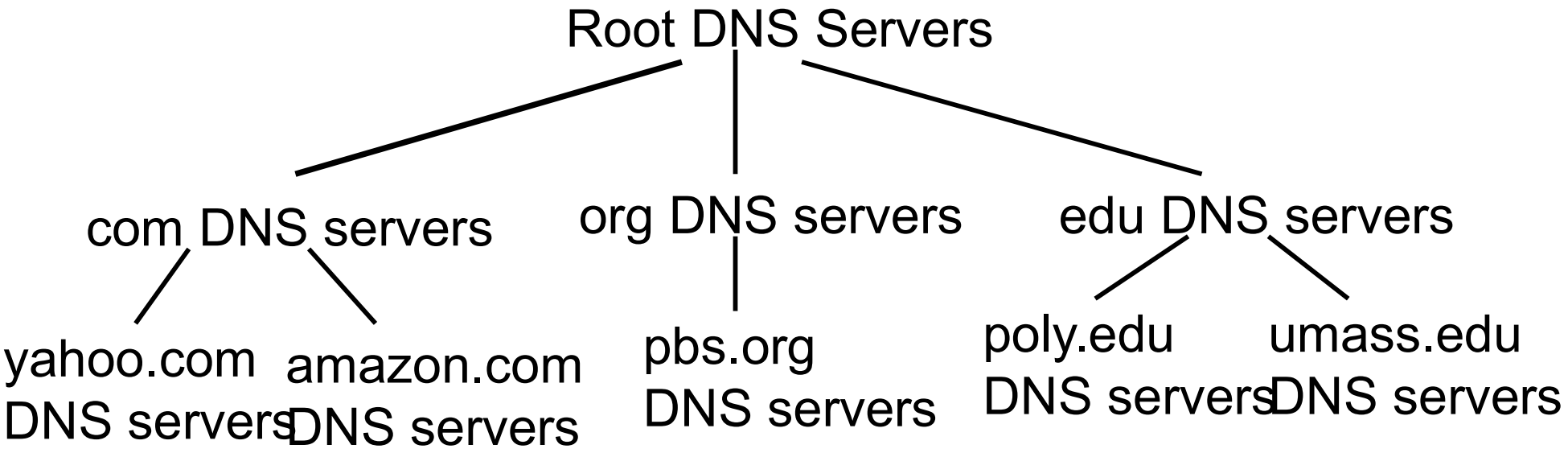
- Tradução de nome de hospedeiro para IP
- Apelidos para hospedeiros (*aliasing*)
  - Nomes canônicos e apelidos
- Apelidos para servidores de e-mail
- Distribuição de carga
  - Servidores Web replicados: conjunto de endereços IP para um mesmo nome

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!

Base de Dados Hierárquica e Distribuída

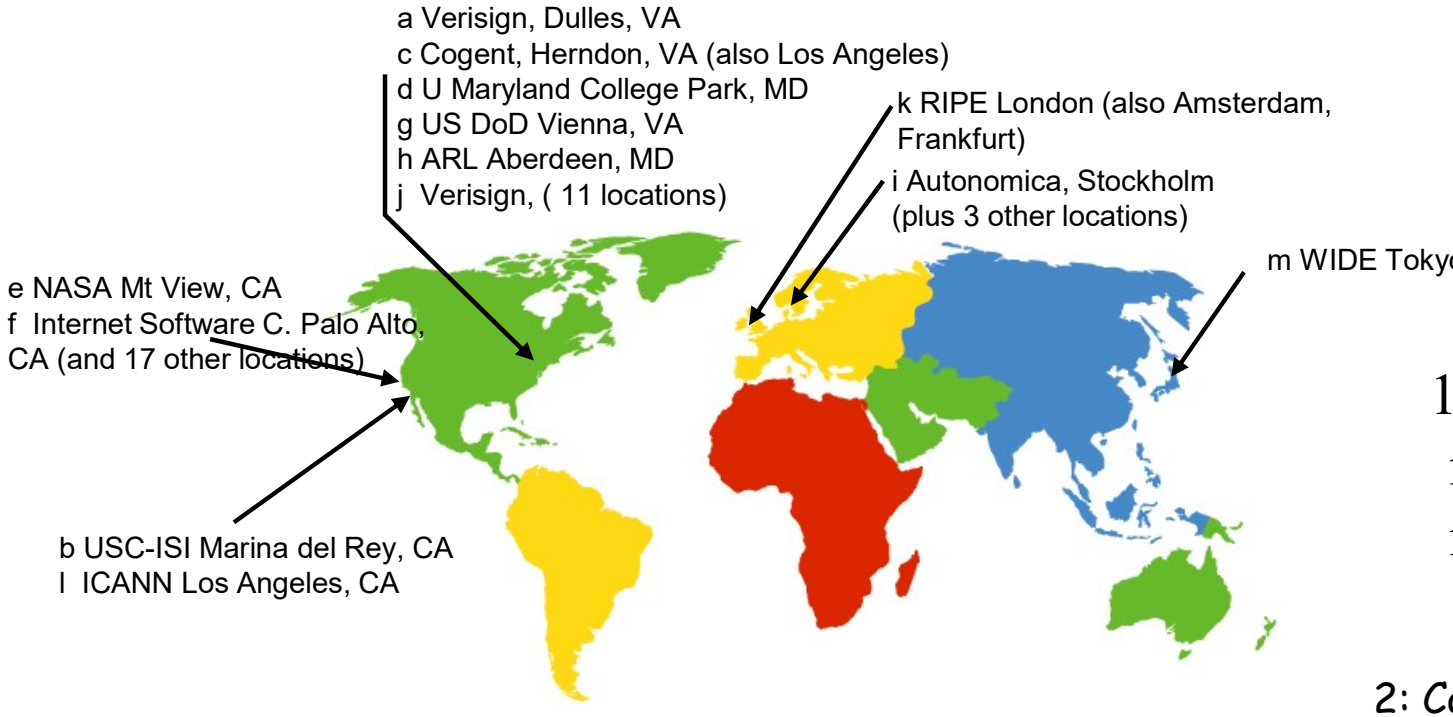


Cliente quer IP para www.amazon.com; 1ª aprox:

- Cliente consulta um servidor raiz para encontrar um servidor DNS .com
- Cliente consulta servidor DNS .com para obter o servidor DNS para o domínio amazon.com
- Cliente consulta servidor DNS do domínio amazon.com para obter endereço IP de www.amazon.com

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



13 servidores de nome raiz em todo o mundo

DNS: Servidores raiz

Hostname	IP Addresses	Manager
a.root-servers.net	198.41.0.4, 2001:503:ba3e::2:30	VeriSign, Inc.
b.root-servers.net	192.228.79.201, 2001:500:84::b	University of Southern California (ISI)
c.root-servers.net	192.33.4.12, 2001:500:2::c	Cogent Communications
d.root-servers.net	199.7.91.13, 2001:500:2d::d	University of Maryland
e.root-servers.net	192.203.230.10, 2001:500:a8::e	NASA (Ames Research Center)
f.root-servers.net	192.5.5.241, 2001:500:2f::f	Internet Systems Consortium, Inc.
g.root-servers.net	192.112.36.4	US Department of Defense (NIC)
h.root-servers.net	198.97.190.53, 2001:500:1::53	US Army (Research Lab)
i.root-servers.net	192.36.148.17, 2001:7fe::53	Netnod
j.root-servers.net	192.58.128.30, 2001:503:c27::2:30	VeriSign, Inc.
k.root-servers.net	193.0.14.129, 2001:7fd::1	RIPE NCC
l.root-servers.net	199.7.83.42, 2001:500:9f::42	ICANN
m.root-servers.net	202.12.27.33, 2001:dc3::35	WIDE Project

Servidores TLD e Oficiais

- Servidores de nomes de Domínio de Alto Nível (TLD):
  - servidores DNS responsáveis por domínios com, org, net, edu, etc, e todos os domínios de países como br, uk, fr, ca, jp.
  - Domínios genéricos: book, globo, rio
  - Lista completa em: https://www.iana.org/domains/root/db
  - NIC.br (Registro .br) para domínio .br (https://registro.br/)
- Servidores de nomes com autoridade:
  - servidores DNS das organizações, provendo mapeamentos oficiais entre nomes de hospedeiros e endereços IP para os servidores da organização (e.x., Web e correio).
  - Podem ser mantidos pelas organizações ou pelo provedor de acesso

Domínios registrados por categorias

13/09/2016

Genéricos

CATEGORIAS	QUANTIDADE	%
COM.BR	3.586.342	91,92
ECO.BR	11.367	0,29
EMP.BR	997	0,03
NET.BR	94.081	2,41

» Ver evolução - total genéricos

Pessoas Físicas

CATEGORIAS	QUANTIDADE	%
BLOG.BR	9.740	0,25
FLOG.BR	155	0,00
NOM.BR	1.609	0,04
VLOG.BR	298	0,01
WIKI.BR	558	0,01

» Ver evolução - total de pessoas físicas

Universidades

CATEGORIAS	QUANTIDADE	%
BR	1.206	0,03
EDU.BR	2.686	0,07

» Ver evolução - total de universidades

Pessoas Jurídicas

CATEGORIAS	QUANTIDADE	%
SEM RESTRIÇÃO		
AGR.BR	2.108	0,05
ART.BR	8.703	0,22

Servidor DNS Local

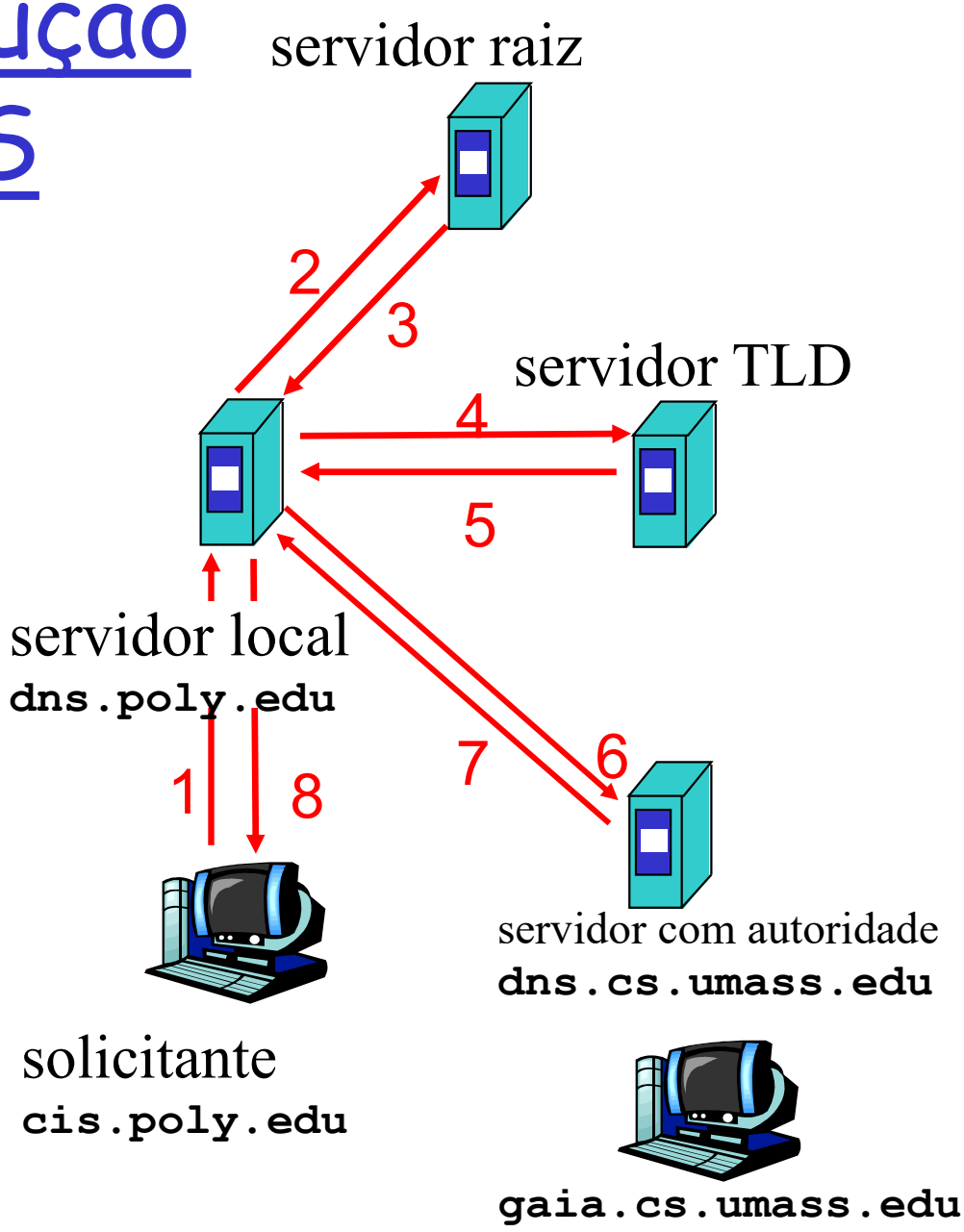
- Não pertence necessariamente à hierarquia
- Cada ISP (ISP residencial, companhia, universidade) possui um.
  - Também chamada do "servidor de nomes default"
- Quanto um hospedeiro faz uma consulta DNS, a mesma é enviada para o seu servidor DNS local
  - Possui uma cache local com pares de tradução nome/endereço recentes (mas podem estar desatualizados!)
  - Atua como um intermediário, enviando consultas para a hierarquia.

Exemplo de resolução de nome pelo DNS

- Hospedeiro em cis.poly.edu quer endereço IP para gaia.cs.umass.edu

consulta interativa:

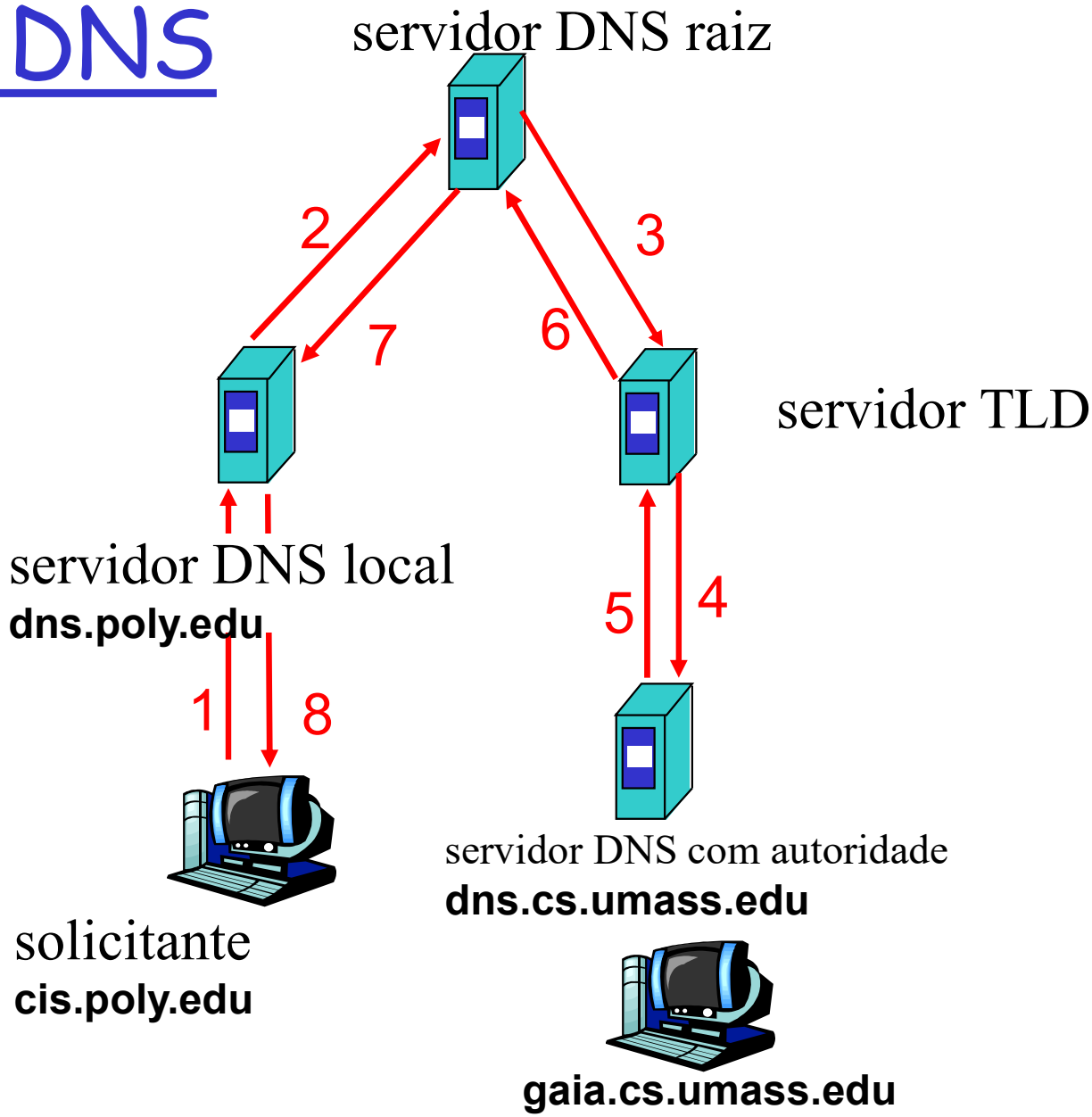
- servidor consultado responde com o nome de um servidor de contato
- "Não conheço este nome, mas pergunte para esse servidor"



Exemplo de resolução de nome pelo DNS

consulta recursiva:

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



DNS: uso de cache, atualização de dados

- uma vez que um servidor qualquer aprende um mapeamento, ele o coloca numa *cache* local
  - entradas na cache são sujeitas a temporização (desaparecem) depois de um certo tempo (TTL)
- Entradas na cache podem estar desatualizadas (tradução nome/endereço do tipo melhor esforço!)
  - Se o endereço IP de um nome de host for alterado, pode não ser conhecido em toda a Internet até que todos os TTLs expirem

Registros DNS

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

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

- Tipo=A
  - **nome** é nome de hospedeiro
  - **valor** é o seu endereço IPv4
  - Tipo=**AAAA** para IPv6
- 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

Inserindo registros no DNS

- Exemplo: acabou de criar a empresa "Network Utopia"
- Registra o nome netutopia.com.br em uma entidade registradora (e.x., Registro.br)
  - Tem de prover para a registradora os nomes e endereços IP dos servidores DNS oficiais (primário e secundário)
  - Registradora insere dois RRs no servidor TLD .br:

```
(netutopia.com.br, dns1.netutopia.com.br, NS)
(dns1.netutopia.com.br, 212.212.212.1, A)
```

- Põe no servidor oficial um registro do tipo A para www.netutopia.com.br e um registro do tipo MX para netutopia.com.br

Ataques ao DNS

Ataques DDoS

- Bombardeia os servidores raiz com tráfego
  - Até o momento não tiveram sucesso
  - Filtragem do tráfego
  - Servidores DNS locais cacheiam os IPs dos servidores TLD, permitindo que os servidores raízes não sejam consultados
- Bombardeio aos servidores TLD
  - Potencialmente mais perigoso

Ataques de redirecionamento

- Pessoa no meio
  - Intercepta as consultas
- Envenenamento do DNS
  - Envia respostas falsas para o servidor DNS que as coloca em cache

Exploração do DNS para DDoS

- Envia consultas com endereço origem falsificado: IP alvo
- Requer amplificação

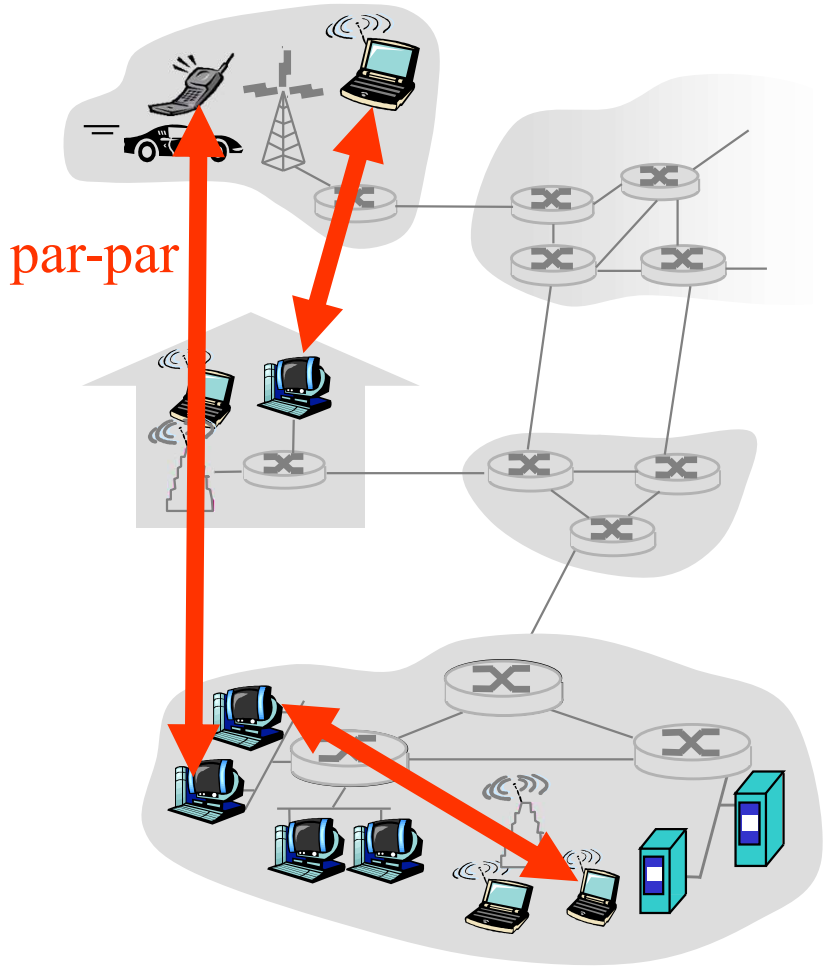
Capítulo 2: Roteiro

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 Transferência de arquivo: FTP
- 2.4 Correio Eletrônico na Internet
- 2.5 DNS: o serviço de diretório da Internet
- 2.6 Aplicações P2P
- 2.7 Programação e desenvolvimento de aplicações com TCP
- 2.8 Programação de *sockets* com UDP

Arquitetura P2P pura

- *sem* servidor sempre ligado
- sistemas finais arbitrários se comunicam diretamente
- pares estão conectados de forma intermitente e mudam seus endereços IP

- Exemplos:
  - Distribuição de arquivos (BitTorrent)
  - *Streaming* (KanKan)
  - VoIP (Skype)

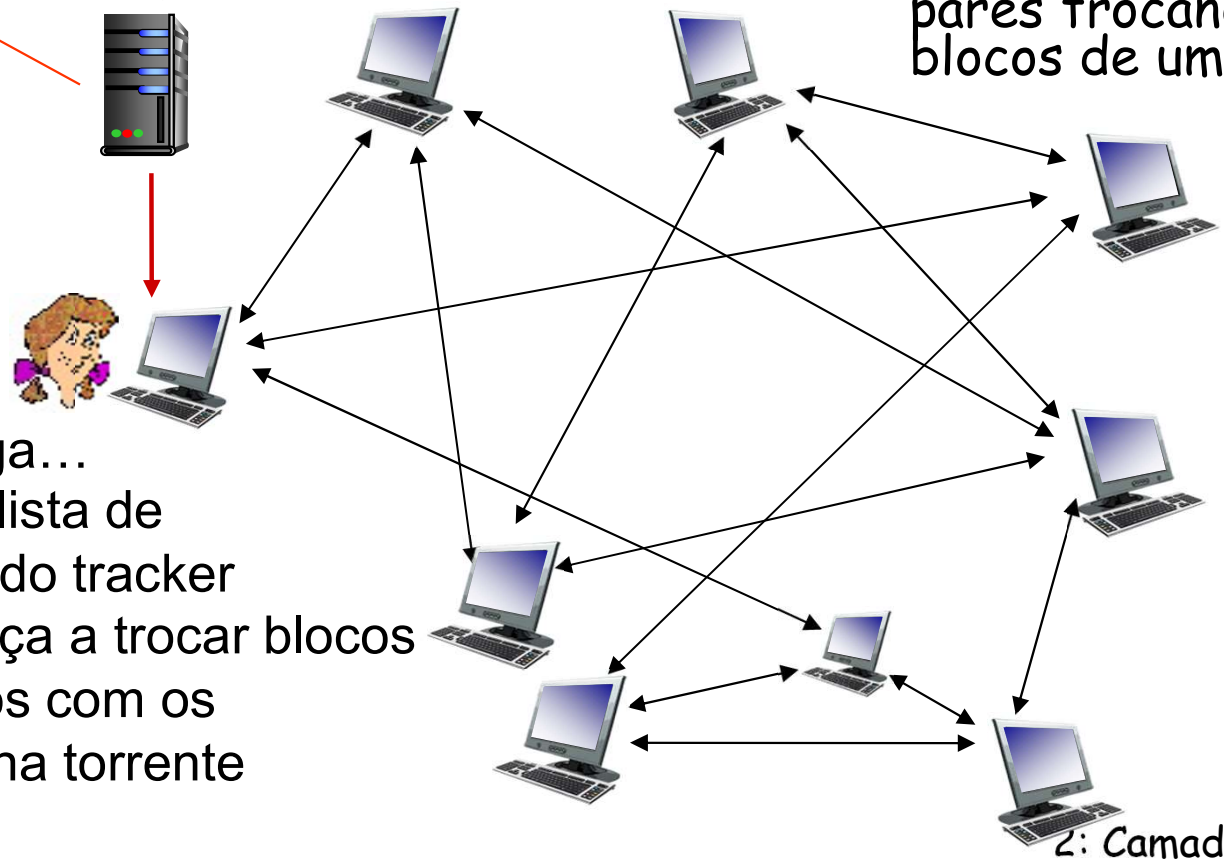


Distribuição de arquivo P2P: BitTorrent

- arquivos divididos em blocos de 256kb
- Pares numa torrente enviam/recebem blocos do arquivo

tracker: registra pares participantes de uma torrente

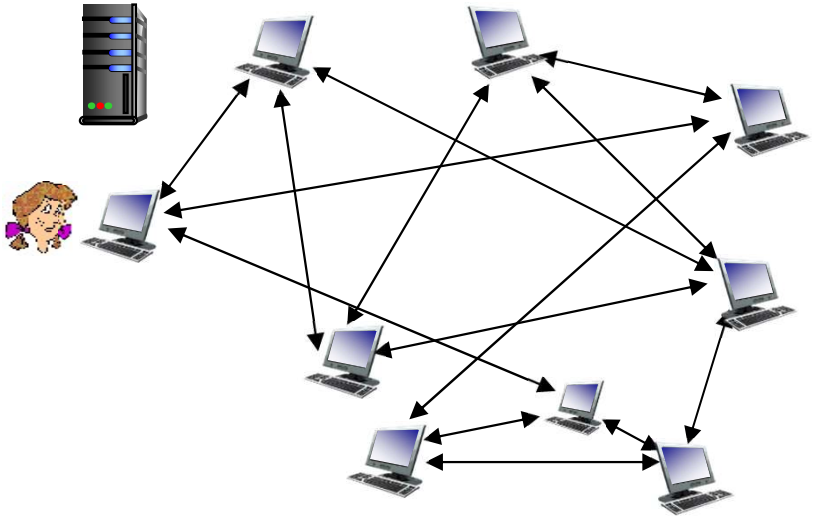
torrente: grupo de pares trocando blocos de um arquivo



Alice chega…
… obtém lista de parceiros do tracker
… e começa a trocar blocos de arquivos com os parceiros na torrente

Distribuição de arquivo P2P: BitTorrent

- par que se une à torrente:
  - não tem nenhum bloco, mas irá acumulá-los com o tempo
  - registra com o *tracker* para obter lista dos pares, conecta a um subconjunto de pares ("vizinhos")



- enquanto faz o download, par carrega blocos para outros pares
- par pode mudar os parceiros com os quais troca os blocos
- pares podem entrar e sair
- quando o par obtiver todo o arquivo, ele pode (egoisticamente) sair ou permanecer (altruisticamente) na torrente

BitTorrent: pedindo, enviando blocos de arquivos

obtendo blocos:

- num determinado instante, pares distintos possuem diferentes subconjuntos de blocos do arquivo
- periodicamente, um par (Alice) pede a cada vizinho a lista de blocos que eles possuem
- Alice envia pedidos para os pedaços que ainda não tem
  - Primeiro os mais raros

Enviando blocos: toma lá, dá cá!

- Alice envia blocos para os quatro vizinhos que estejam lhe enviando blocos *na taxa mais elevada*
  - outros pares foram sufocados por Alice
  - Reavalia os 4 mais a cada 10 segs
- a cada 30 segs: seleciona aleatoriamente outro par, começa a enviar blocos
  - "*optimistically unchoked*"
  - o par recém escolhido pode se unir aos 4 mais

Capítulo 2: Roteiro

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 Transferência de arquivo: FTP
- 2.4 Correio Eletrônico na Internet
- 2.5 DNS: o serviço de diretório da Internet
- 2.6 Aplicações P2P
- 2.7 Programação e desenvolvimento de aplicações com TCP
- 2.8 Programação de *sockets* com UDP

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

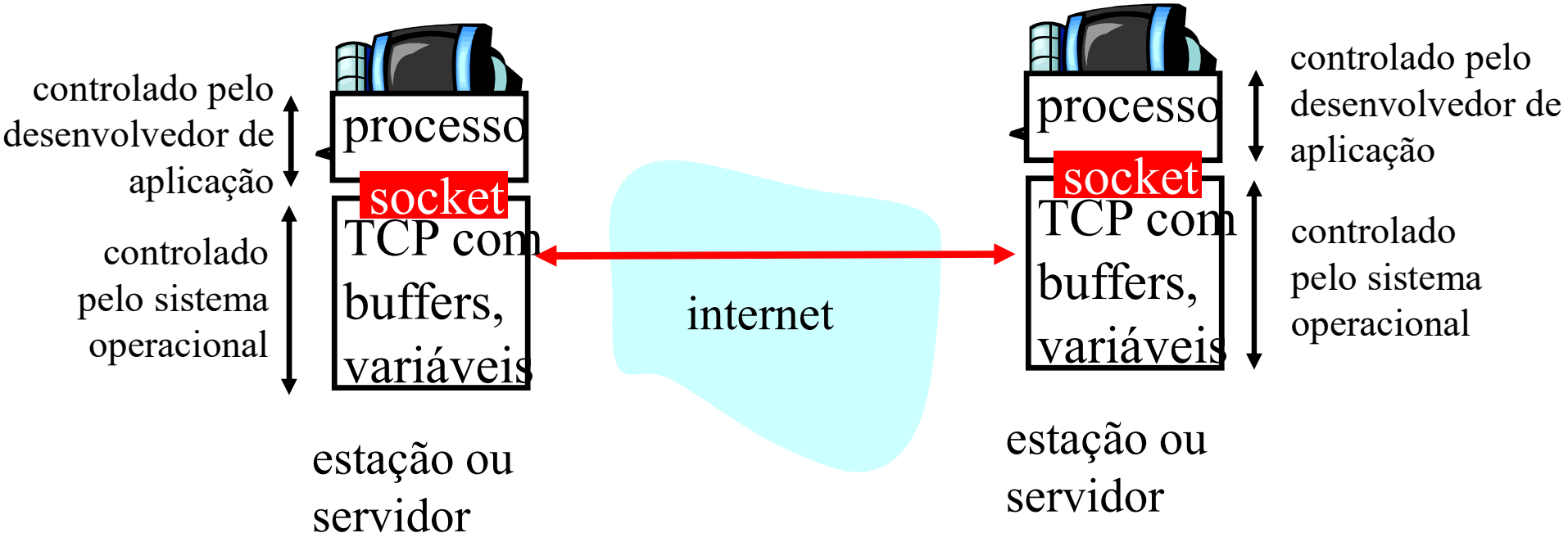
socket

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:

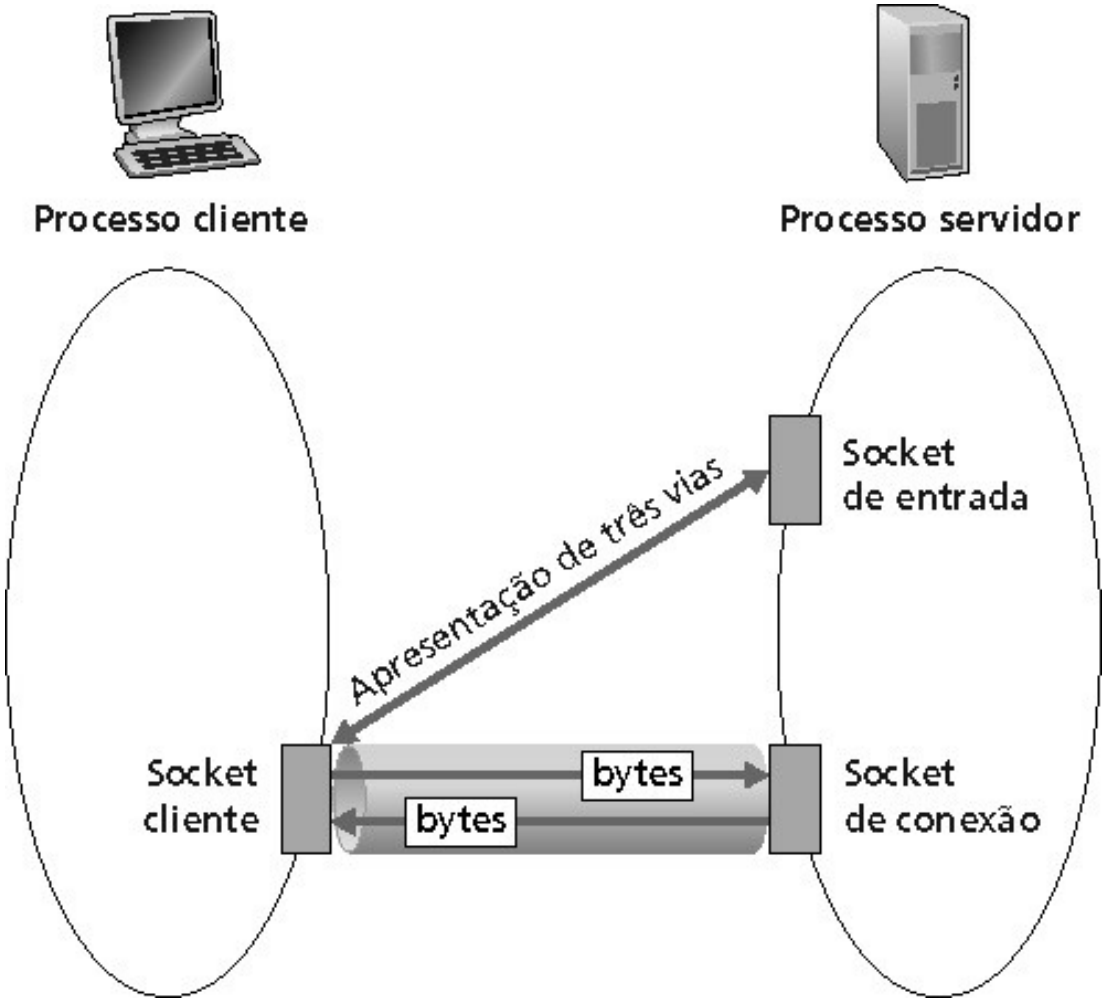
- criar socket TCP local ao cliente
- especificar endereço IP, número de porta do processo servidor
- Quando cliente cria socket: TCP cliente cria conexão com TCP do servidor

- 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
  - Endereço IP e porta origem são usados para distinguir os clientes (mais no cap. 3)

ponto de vista da aplicação

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

Comunicação entre sockets



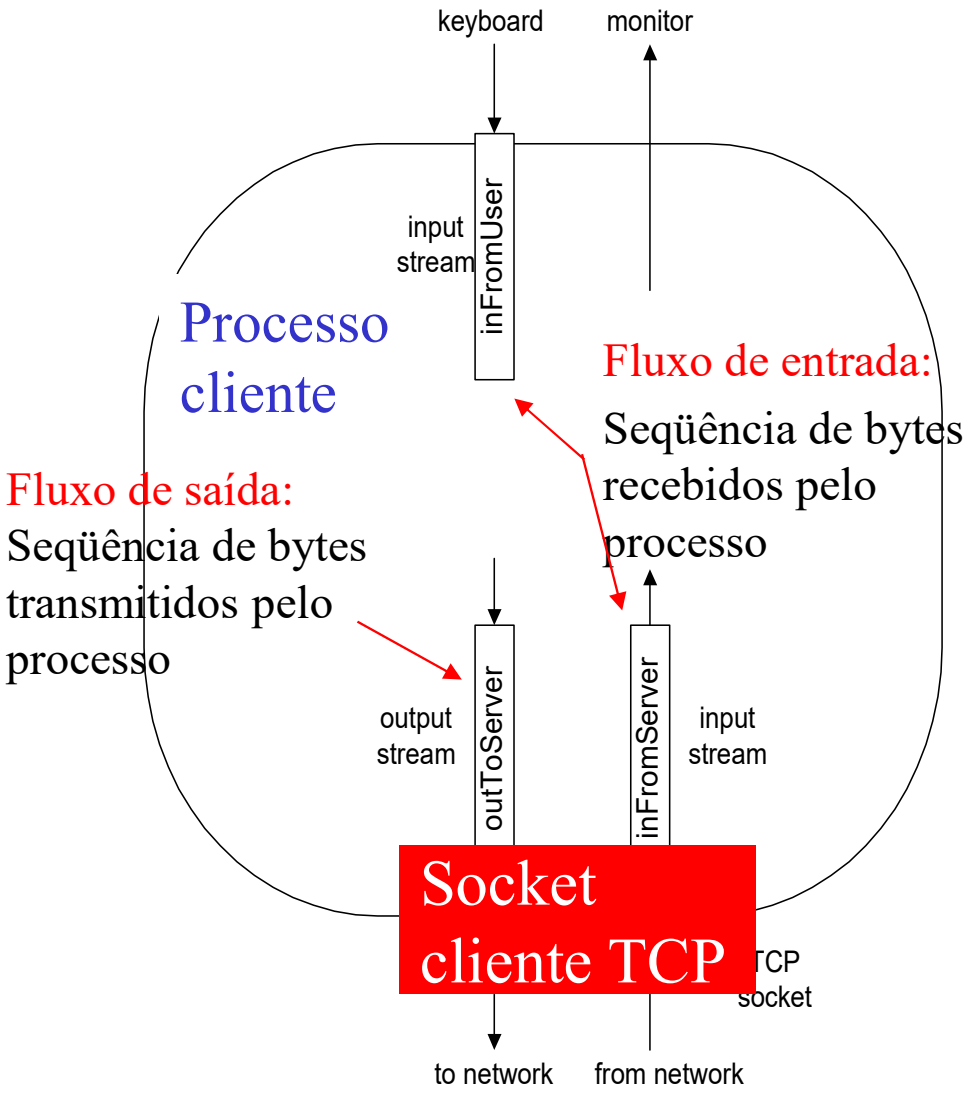
Jargão para Fluxo (*Stream*)

- Um fluxo (*stream*) é uma seqüência de caracteres que fluem de ou para um processo.
- Um fluxo de entrada é conectado a alguma fonte de entrada para o processo, por exemplo, teclado ou *socket*.
- Um fluxo de saída é conectado a uma fonte de saída, por exemplo, um monitor ou um *socket*.

Programação com sockets usando TCP

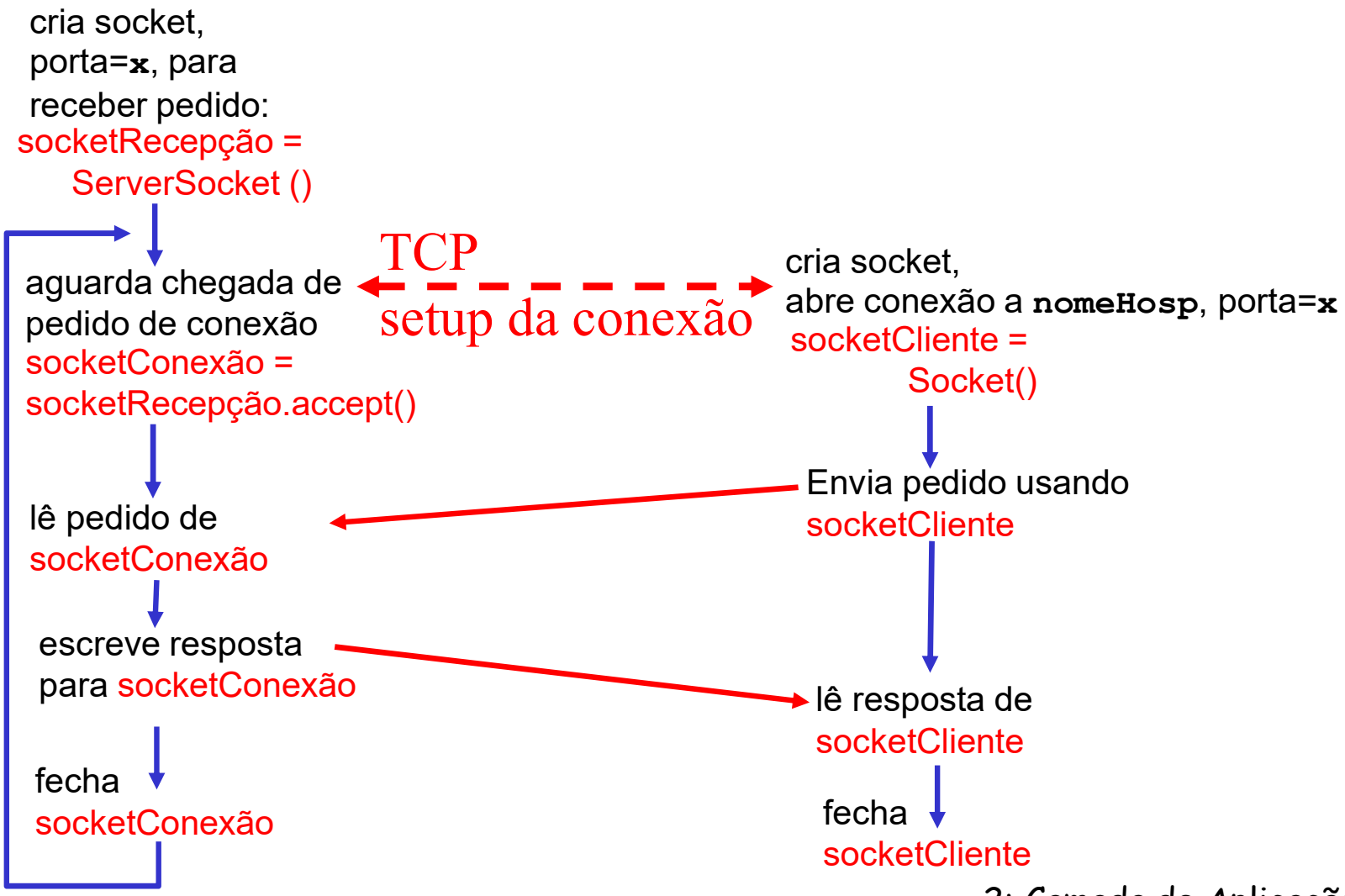
Exemplo de apl. cliente-servidor:

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



Interações cliente/servidor usando o TCP

Servidor (executa em **`nomeHosp`**) Cliente



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();

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

fraseModificada = doServidor.readLine();

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

socketCliente.close();

    }
}
```

Envia linha ao servidor

Lê linha do servidor

Exemplo: servidor Java (TCP)

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

class servidorTCP {

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

     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

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

Lê linha do socket

```
fraseCliente= doCliente.readLine();

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

Escreve linha ao socket

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

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

Exemplo: cliente Python (TCP)

inclui a biblioteca de sockets do Python

cria socket TCP socket para o servidor, porta remota 12000

não há necessidade de especificar nem o nome do servidor nem a porta

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
```

Exemplo: servidor Python (TCP)

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while 1:
    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024)
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
```

cria socket TCP de recepção

servidor inicia a escuta por solicitações TCP

loop infinito

servidor espera no accept() por solicitações, um novo socket é criado no retorno

lê bytes do socket (mas não precisa ler endereço como no UDP)

fecha conexão para este cliente (mas *não* o socket de recepção)

Capítulo 2: Roteiro

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 Transferência de arquivo: FTP
- 2.4 Correio Eletrônico na Internet
- 2.5 DNS: o serviço de diretório da Internet
- 2.6 Aplicações P2P
- 2.7 Programação e desenvolvimento de aplicações com TCP
- 2.8 Programação de *sockets* com UDP

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

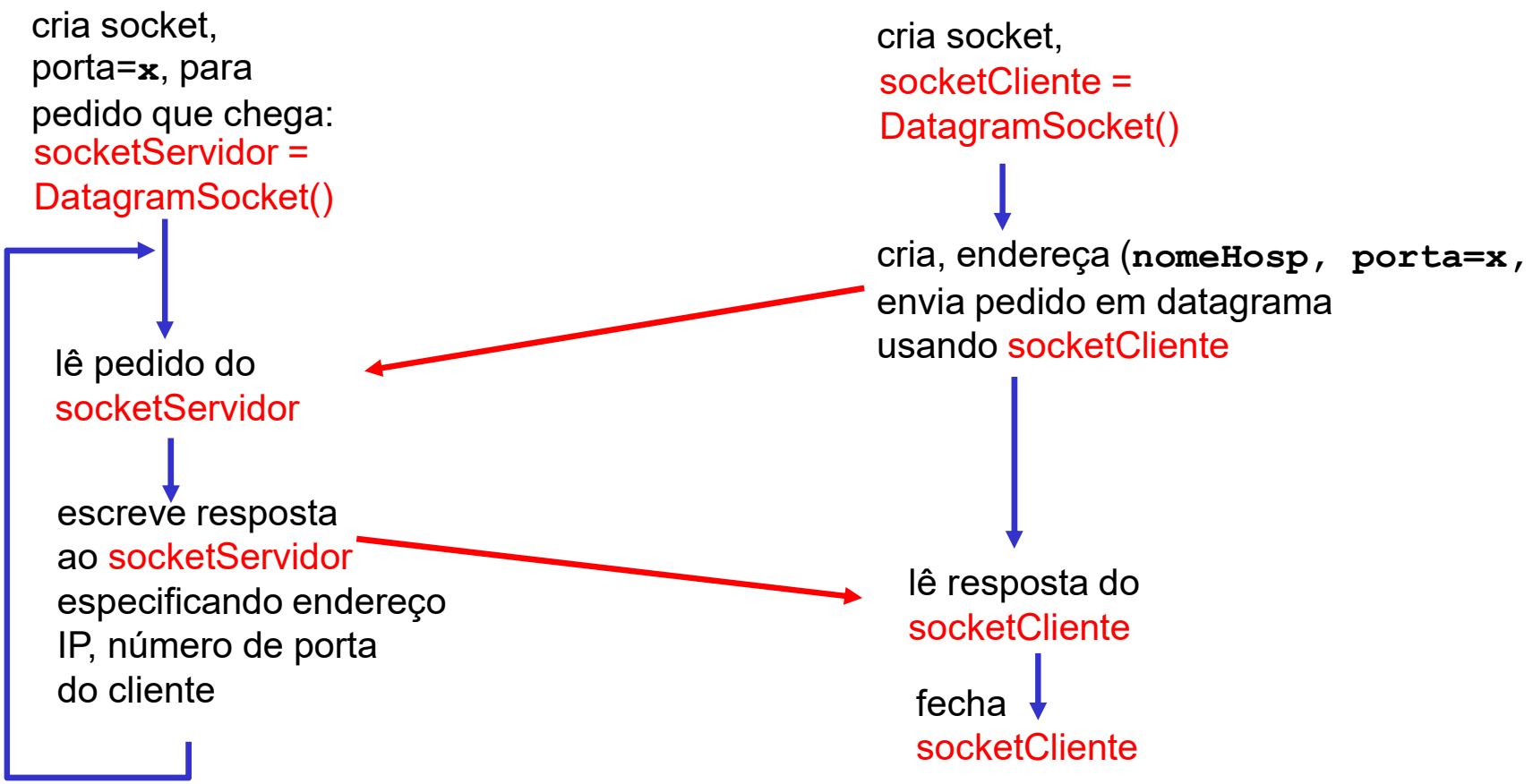
ponto de vista da aplicação

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

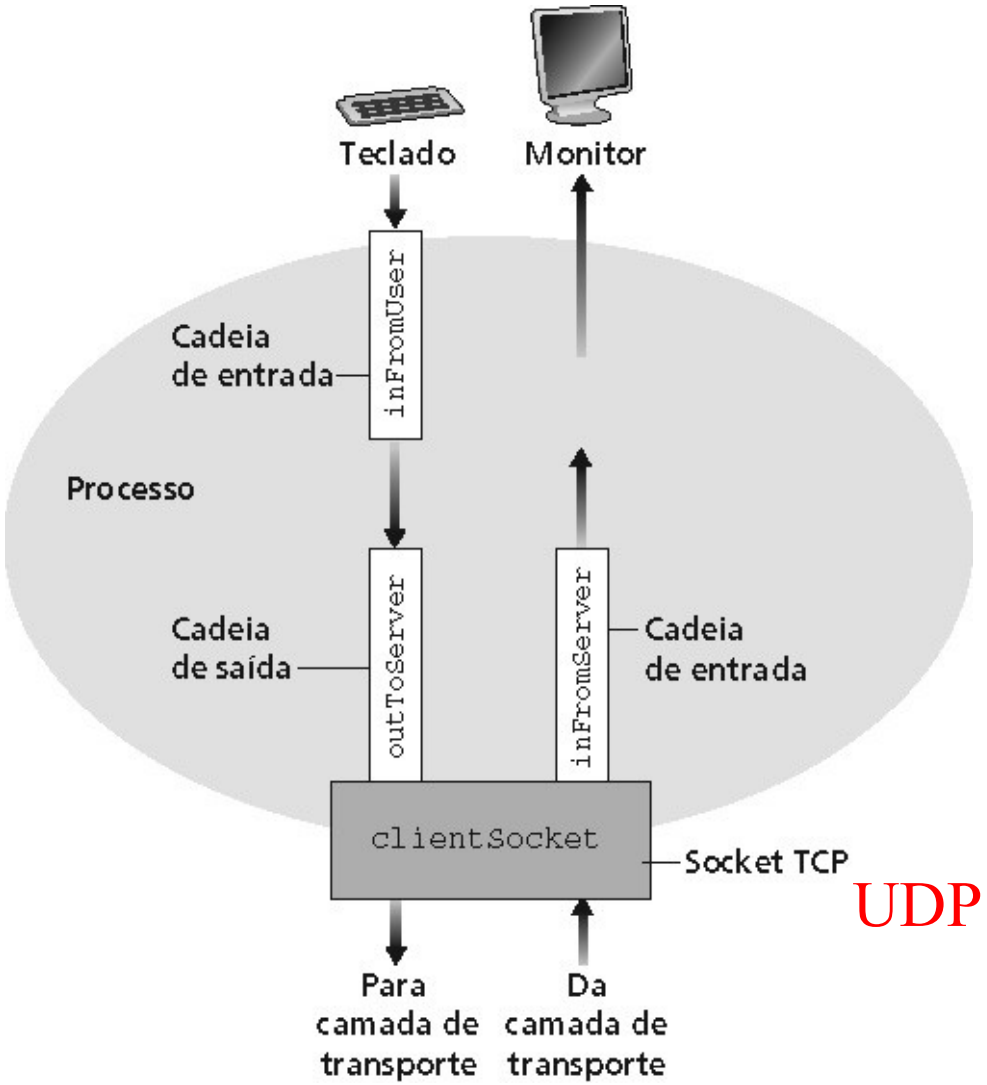
Interações cliente/servidor usando o UDP

Servidor (executa em **`nomeHosp`**)

Cliente



Exemplo: Cliente Java (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 doUsuario=
        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[] dadosEnvio = new byte[1024];
      byte[] dadosRecebidos = new byte[1024];

      String frase = doUsuario.readLine();

      dadosEnvio = 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();
}

}
```

Exemplo: cliente Python (UDP)

inclui a biblioteca de sockets do Python

cria socket UDP para servidor

obtém entrada do teclado do usuário

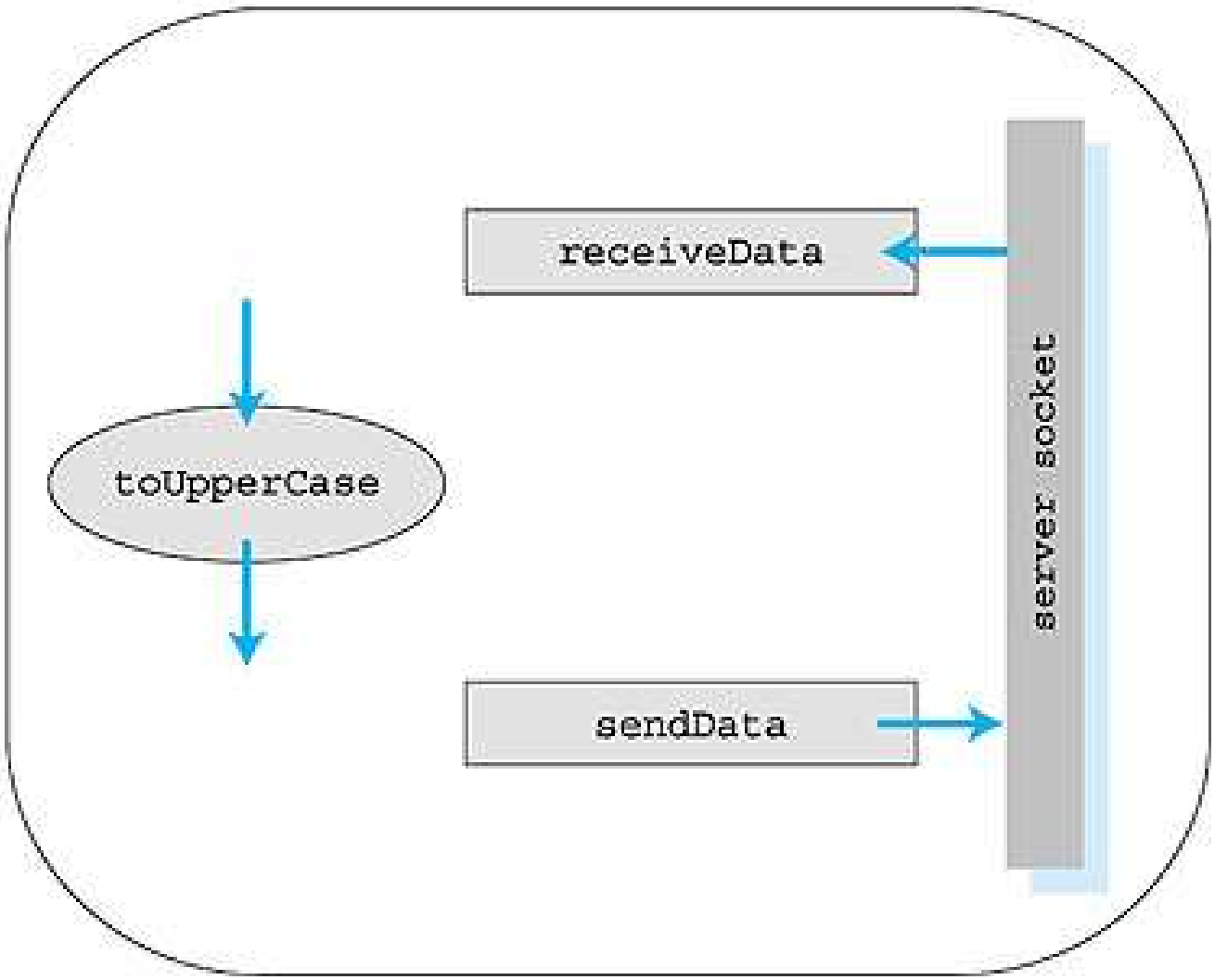
acrescenta o nome do servidor e número da porta à mensagem; envia pelo socket

lê caracteres de resposta do socket e converte em *string*

imprime string recebido e fecha socket

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(socket.AF_INET,
                      socket.SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message,(serverName, serverPort))
modifiedMessage, serverAddress =
                      clientSocket.recvfrom(2048)
print modifiedMessage
clientSocket.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 laço while, volta ao início e aguarda chegar outro datagrama

Exemplo: servidor Python (UDP)

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print "The server is ready to receive"
while 1:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.upper()
    serverSocket.sendto(modifiedMessage, clientAddress)
```

cria socket UDP

liga socket à porta local número 12000

loop infinito

lê mensagem do socket UDP, obtendo endereço do cliente (IP e porta do cliente)

retorna *string* em maiúsculas para este cliente

Capítulo 2: Resumo

Nosso estudo sobre aplicações de rede está agora completo!

- Arquiteturas de aplicações
  - cliente-servidor
  - P2P
- Requisitos de serviço das aplicações:
  - confiabilidade, banda, atraso
- Modelos de serviço de transporte da Internet
  - orientado à conexão, confiável: TCP
  - não confiável, datagramas: UDP
- Protocolos específicos:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent, DHT
- Programação de sockets

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

Temas importantes:

- msgs de controle vs. dados
  - na banda, fora da banda
- centralizado vs. descentralizado
- s/ estado vs. c/ estado
- transferência de msgs confiável vs. não confiável
- "complexidade na borda da rede"