Capítulo 2: Camada de Aplicação

Slides baseados no Livro de
Kurose/Ross

Kelvin Lopes Dias
kld@cin.ufpe.br

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

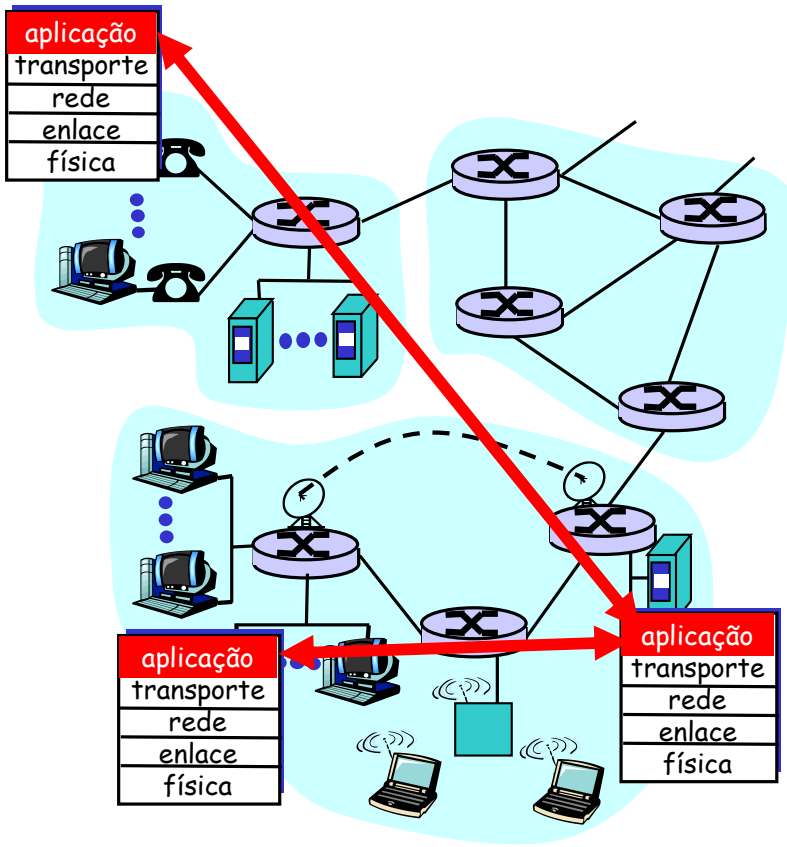
Aplicações e protocolos da camada de aplicação

Aplicação: processos distribuídos em comunicação

- executam em hospedeiros no "espaço de usuário"
- trocam mensagens para implementar aplicação
- p.ex., correio, transf. de arquivo, WWW

Protocolos da camada de apl.

- uma "parte" da aplicação
- define mensagens trocadas por apls e ações tomadas
- usam serviços providos por protocolos de camadas inferiores



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

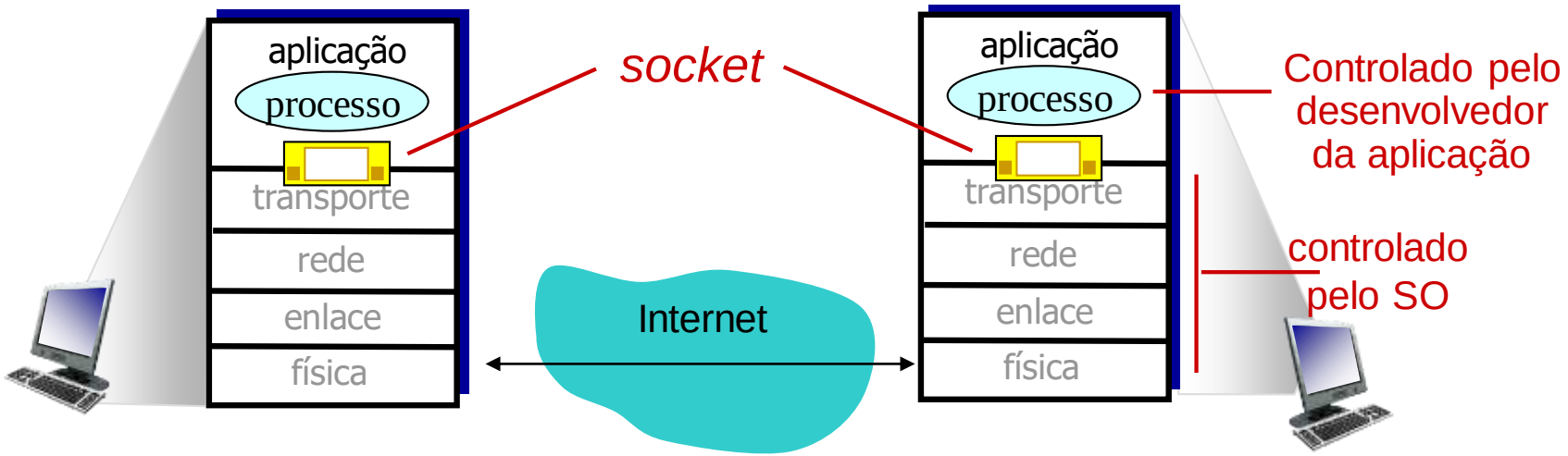
Processo cliente:
processo que inicia a comunicação

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



Paradigma cliente-servidor (C-S)

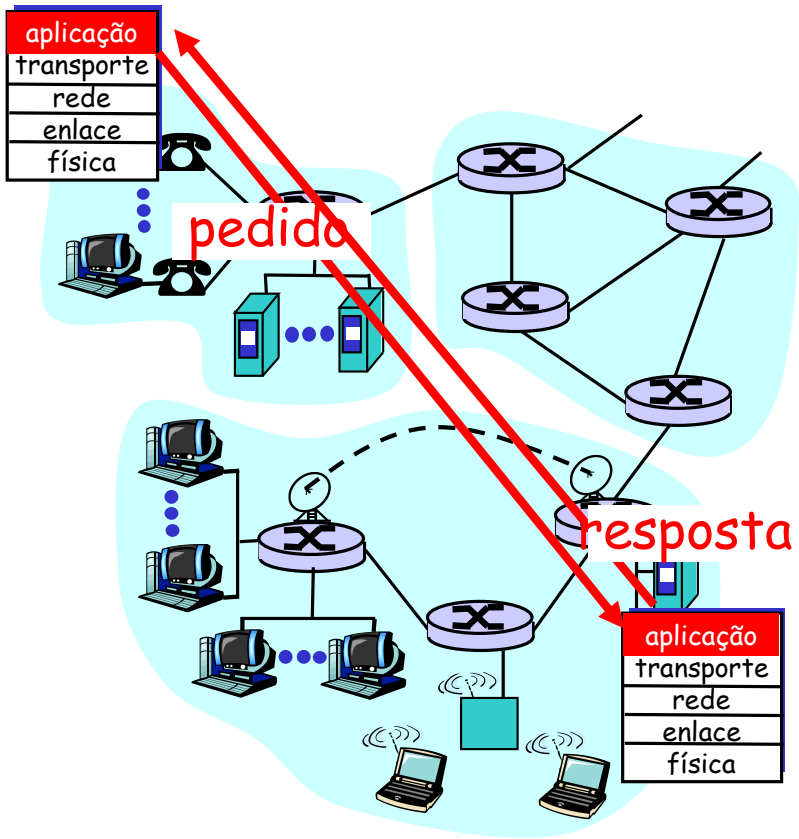
Apl. de rede típica tem duas partes: *cliente* e *servidor*

Cliente:

- inicia contato com o servidor ("fala primeiro")
- tipicamente solicita serviço do servidor
- para WWW, cliente implementado no browser; para correio no leitor de mensagens

Servidor:

- provê ao cliente o serviço requisitado
- p.ex., servidor WWW envia página solicitada; servidor de correio entrega mensagens



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.

Protocolos da camada de aplicação (cont).

API: interface de programação de aplicações

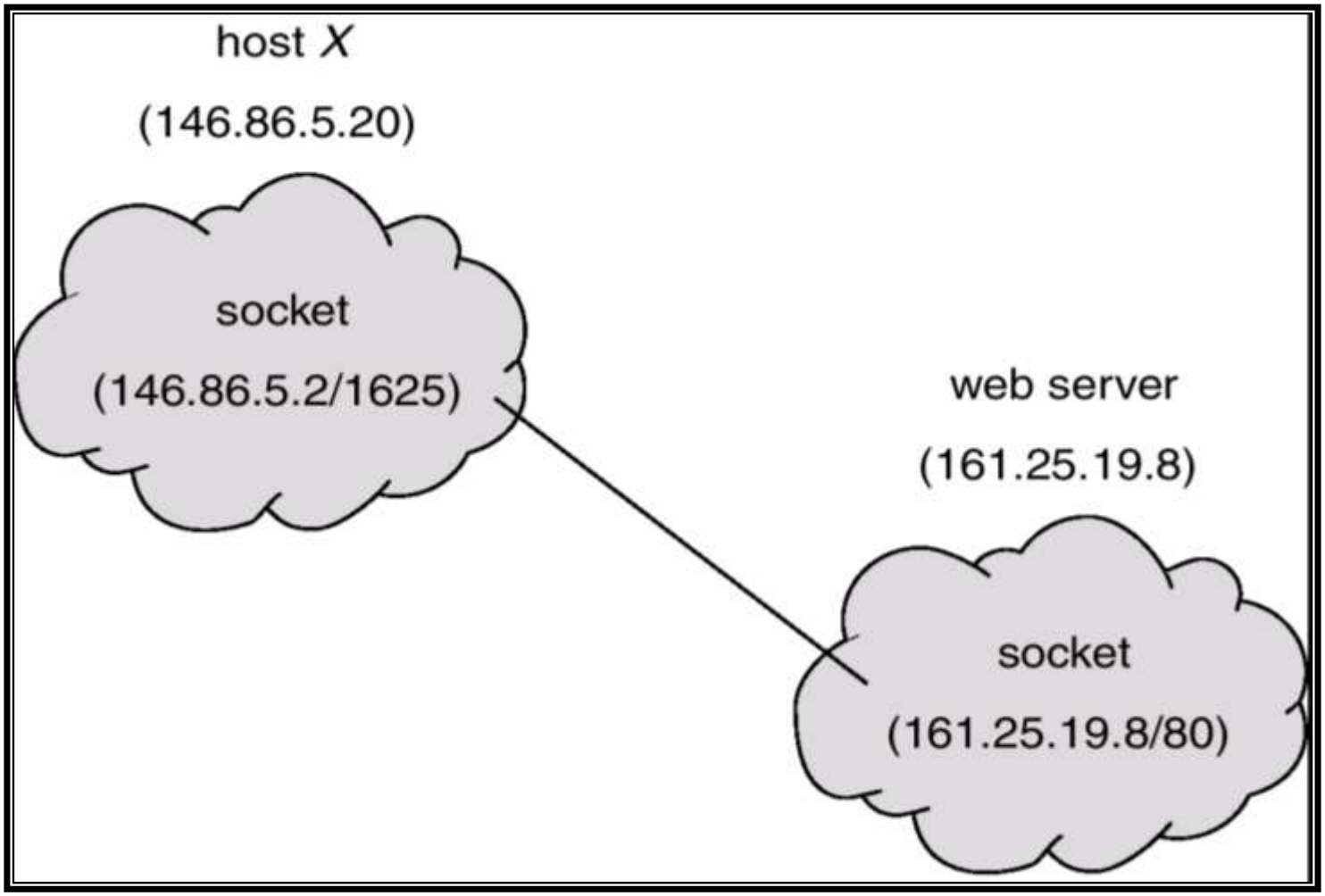
- define interface entre aplicação e camada de transporte
- socket (= tomada) : API da Internet
  - 2 processos se comunicam enviando dados para um socket ou lendo dados de um socket

P: como um processo pode "identificar"o outro processo com o qual quer se comunicar?

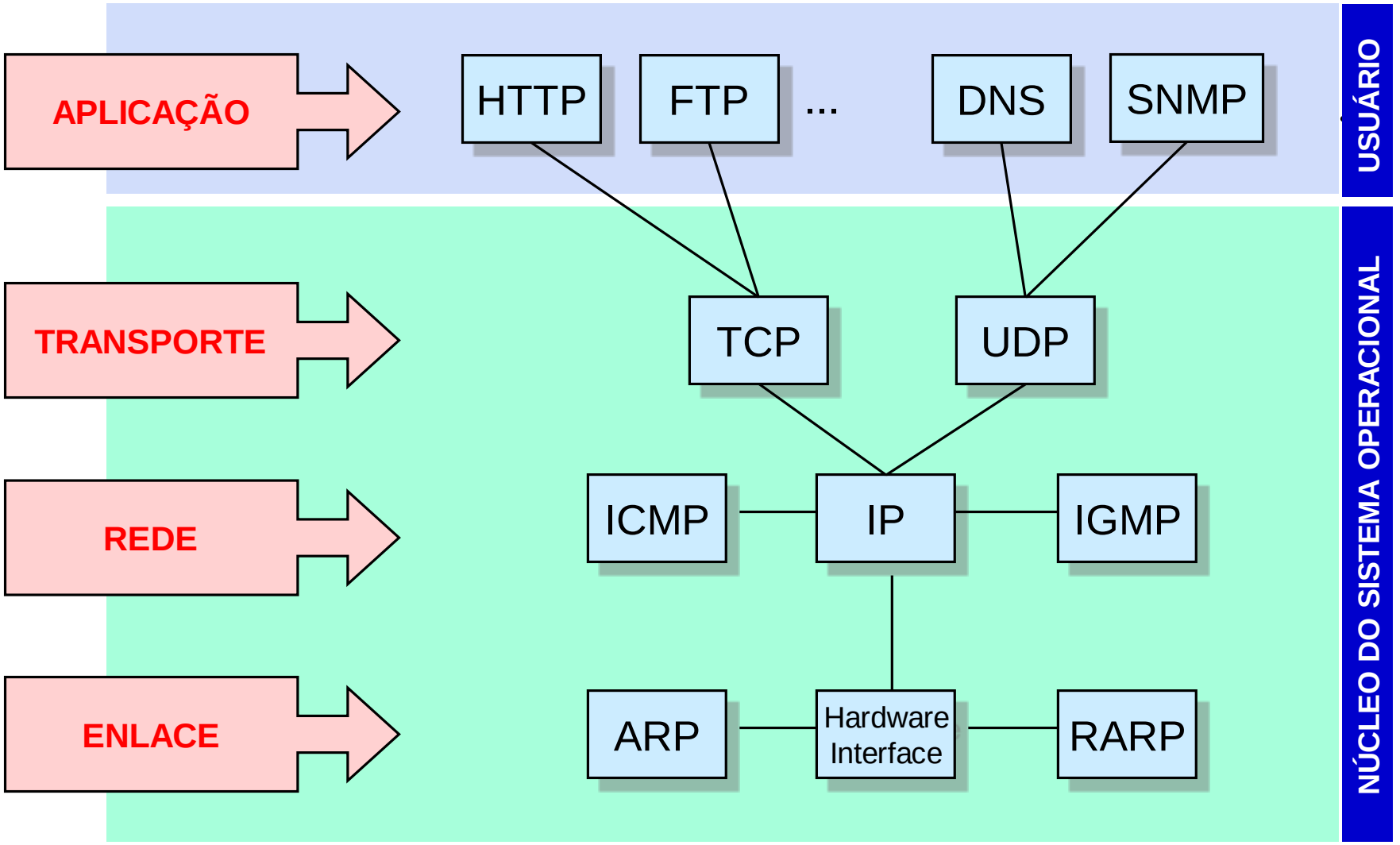
- endereço IP do hospedeiro do outro processo
- "número de porta" - permite que o hospedeiro receptor determine a qual processo deve ser entregue a mensagem

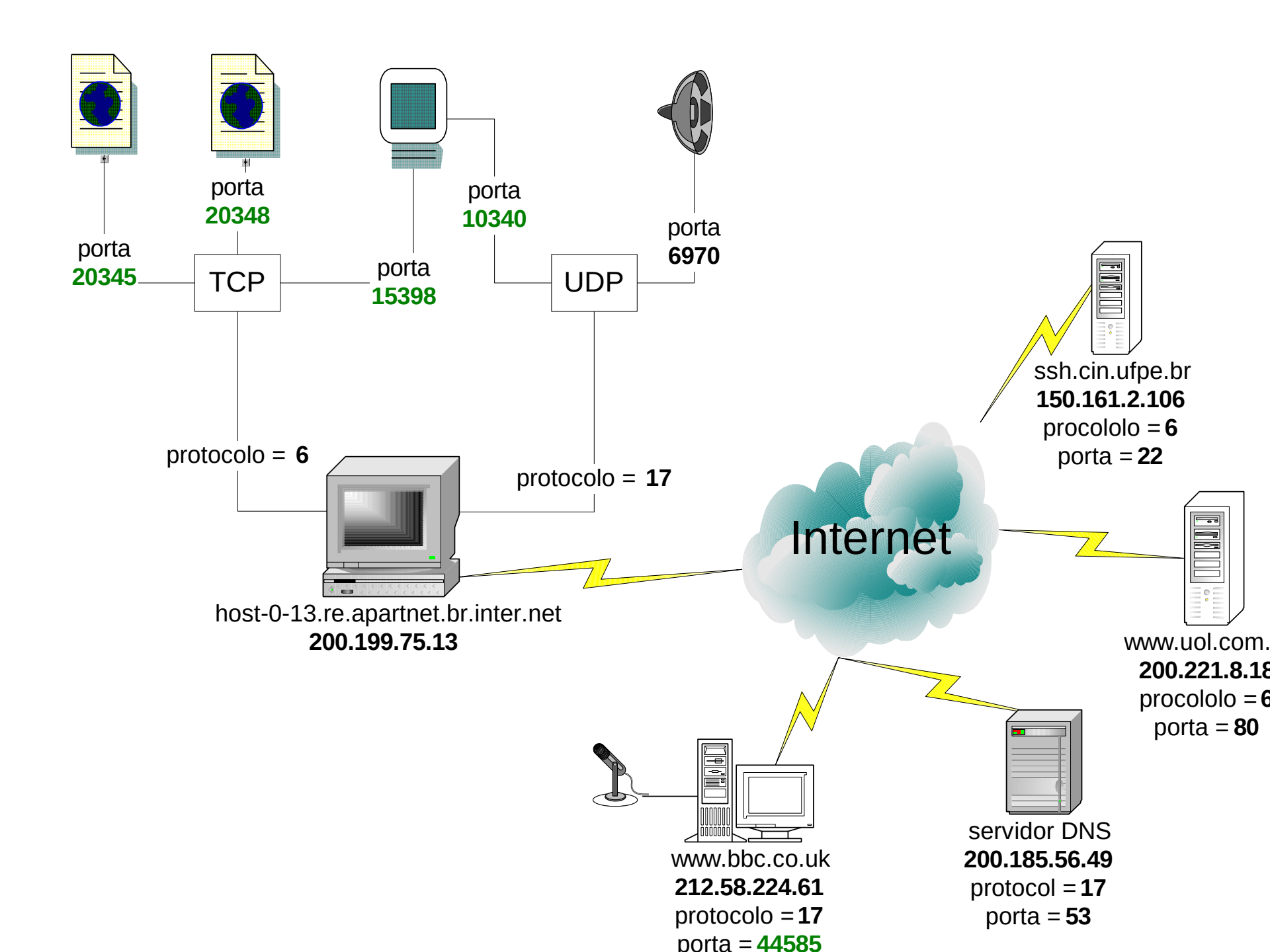
... voltamos mais tarde a este assunto.

Sockets = (endereço IP, porta)



Protocolos da Internet




porta
20345
porta
20348
TCP
porta
15398
porta
10340
UDP
porta
6970
protocolo = 6
protocolo = 17
host-0-13.re.apartnet.br.inter.net
200.199.75.13
Internet
ssh.cin.ufpe.br
150.161.2.106
procololo = 6
porta = 22
www.uol.com.
200.221.8.18
procololo = 6
porta = 80
www.bbc.co.uk
212.58.224.61
protocolo = 17
porta = 44585
servidor DNS
200.185.56.49
protocol = 17
porta = 53

Algumas Portas Bem Conhecidas

- 21 – FTP
- 22 – SSH
- 23 – TELNET
- 25 – SMTP
- 53 – DNS
- 69 – TFTP
- 80 – HTTP
- 88 – KERBEROS
- 101 – POP3
- 161 – SNMP
- 443 – HTTPS
- 995 – POP3 SSL
- 1433 – MS-SQL SERVER
- 3006 – MYSQL

Detalhes em www.iana.org/assignments/port-numbers

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

Temporização

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

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

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 ms
áudio/vídeo gravado	tolerante	como anterior	sim, alguns segs
jogos interativos	tolerante	> alguns kbps	sim, 100’s ms
apls financeiras	sem perdas	elástica	sim e não

Serviços providos por protocolos de transporte Internet

serviço TCP:

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

serviço UDP:

- transferência de dados não confiável entre processos remetente e receptor
- não provê: setup da conexão, confiabilidade, controle de fluxo, controle de congestionamento, garantias temporais ou de banda mínima

P: Qual é o interesse em ter um 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:

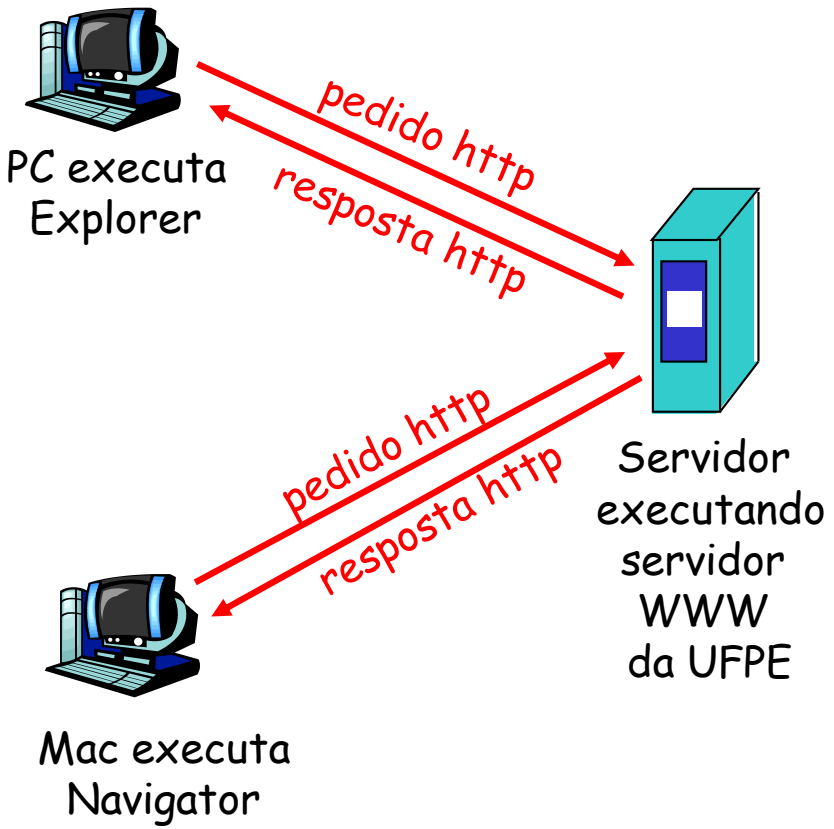
`www.someschool.edu/someDept/pic.gif`

nome do hospedeiro | nome do caminho

WWW: o protocolo http

http: hypertext transfer protocol

- protocolo da camada de aplicação para WWW
- modelo cliente/servidor
  - *cliente:* browser que pede, recebe, "visualiza" objetos WWW
  - *servidor:* servidor WWW envia objetos em resposta a pedidos
- http1.0: RFC 1945
- http1.1: RFC 2068, 2616



Mais sobre o protocolo http

http: 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 WWW (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!

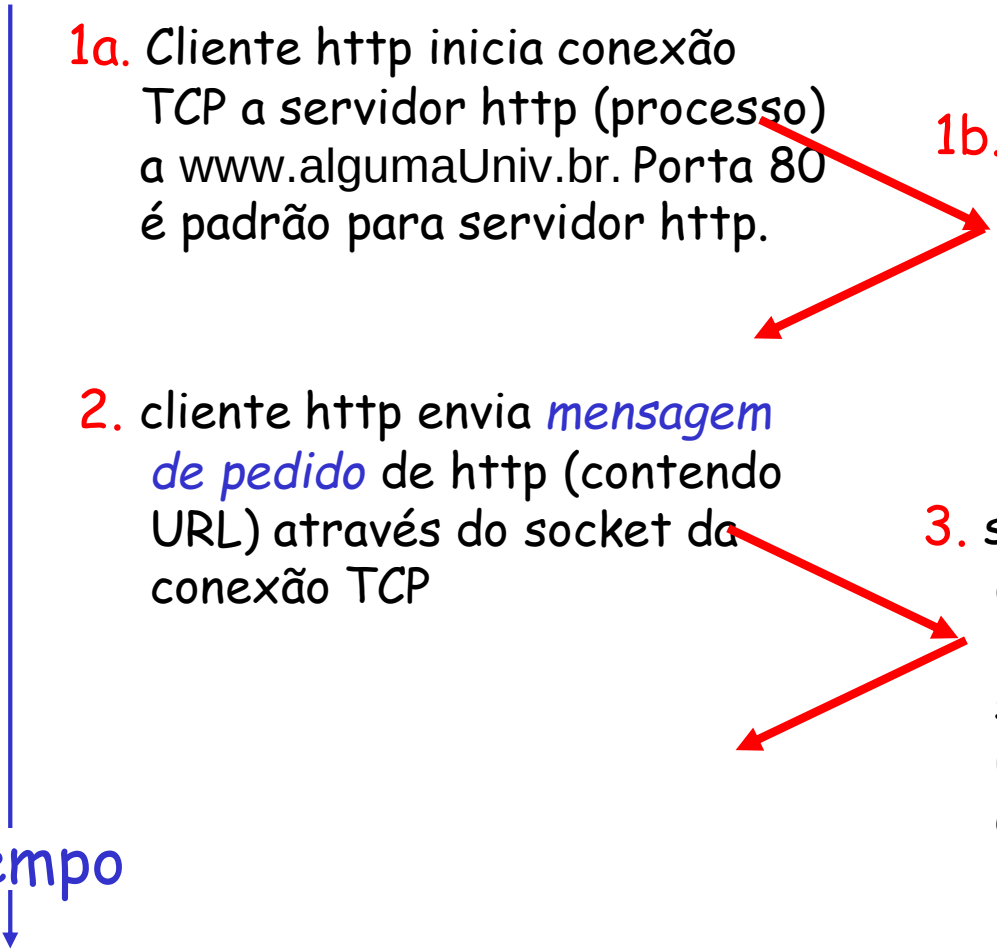
- história passada (estado) tem que ser guardada
- Caso caia servidor/cliente, suas visões do "estado" podem ser inconsistentes, devem ser reconciliadas

Exemplo de http

Supomos que usuário digita a URL

www.algumaUniv.br/algumDepartmento/inicial.index

(contém texto, referências a 10 imagens jpeg)



Exemplo de http (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

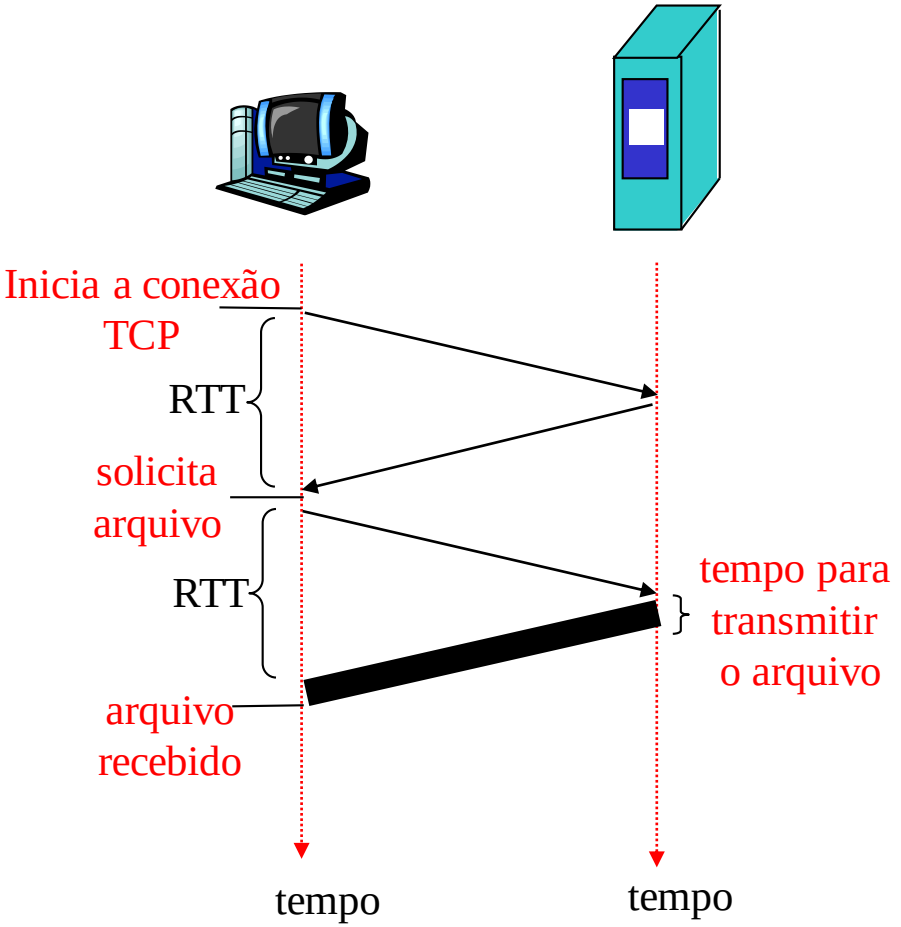
Modelagem do tempo de resposta

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



Conexões não persistentes e persistentes

Não persistente

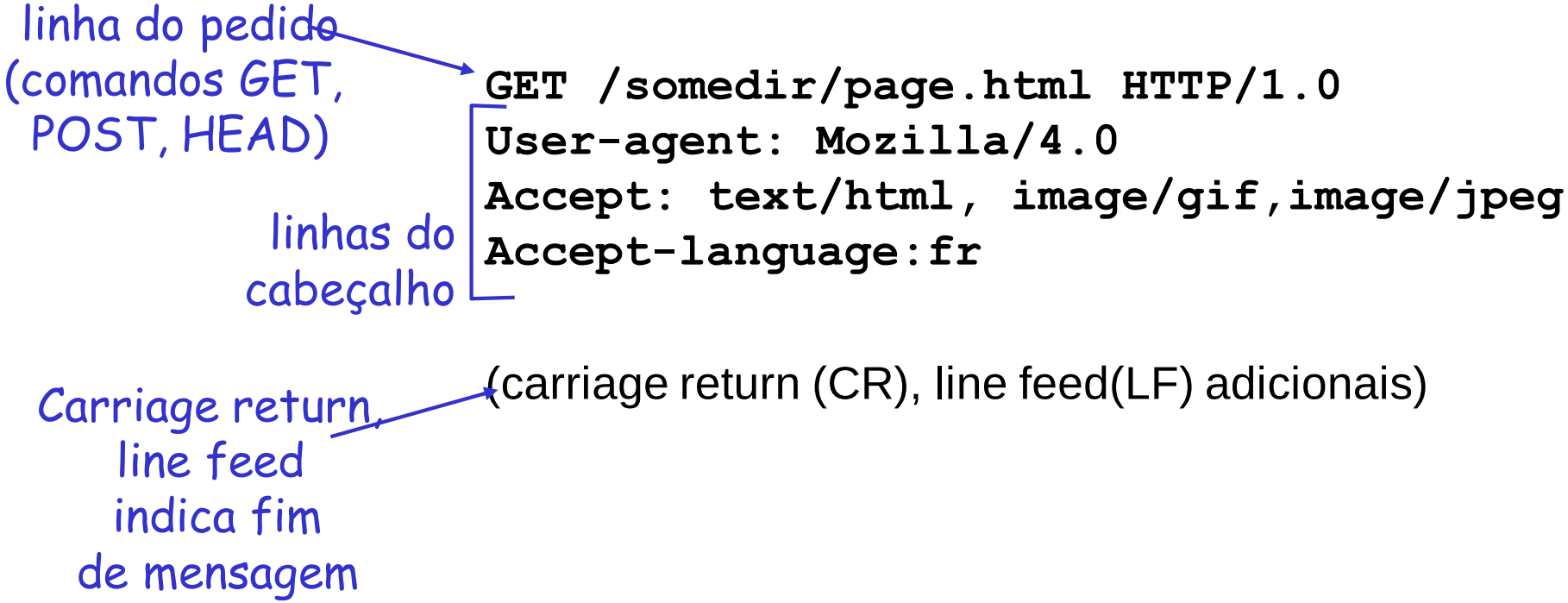
- HTTP/1.0
- servidor analisa pedido, responde, e encerra conexão TCP
- 2 RTTs para trazer cada objeto (RTT=round trip time)
- transferência de cada objeto sofre de partida lenta

Persistente

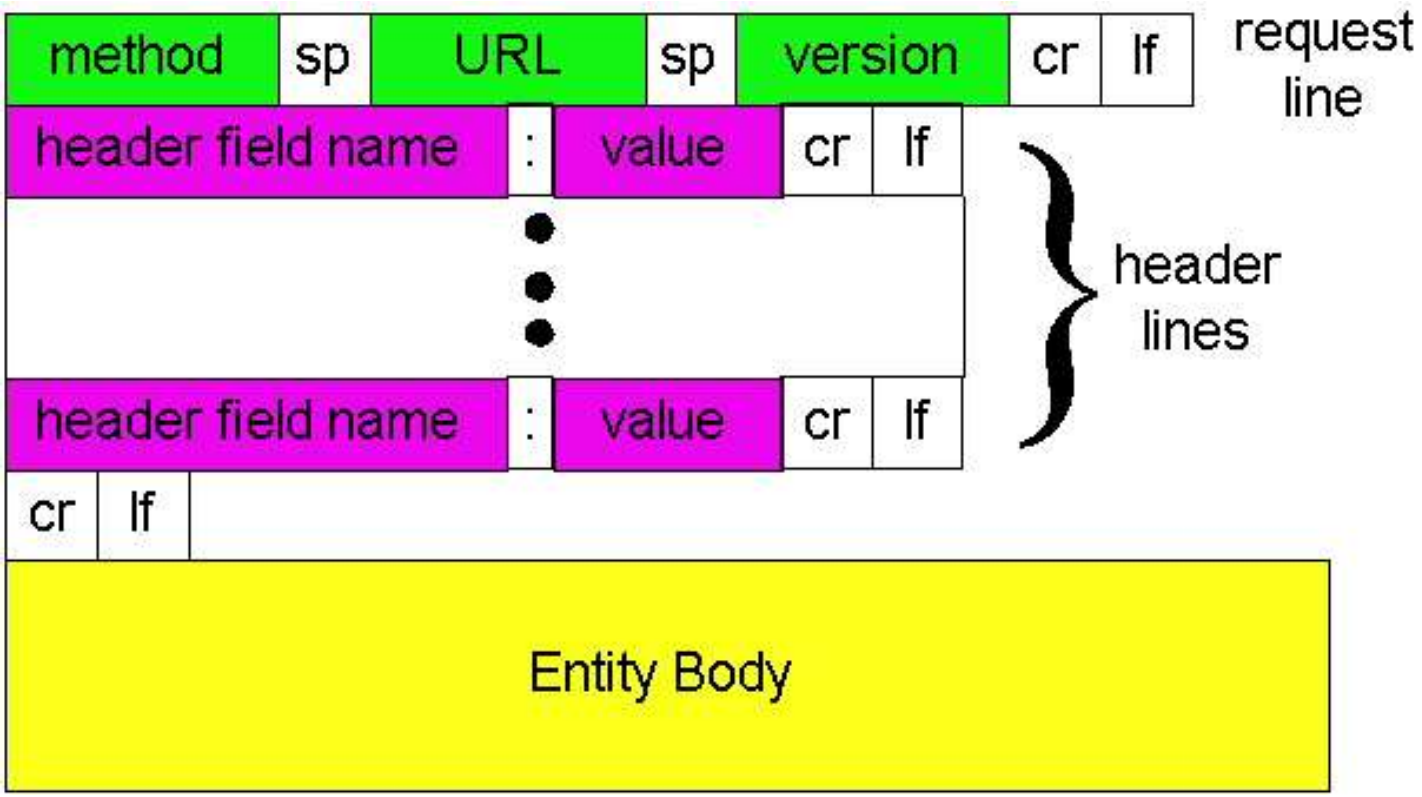
- default para HTTP/1.1
- na mesma conexão TCP: servidor analisa pedido, responde, analisa novo pedido, ...
- Cliente envia pedidos para todos objetos referenciados assim que recebe o HTML base .
- Menos RTTs e menos partida lenta.

formato de mensagem http: pedido

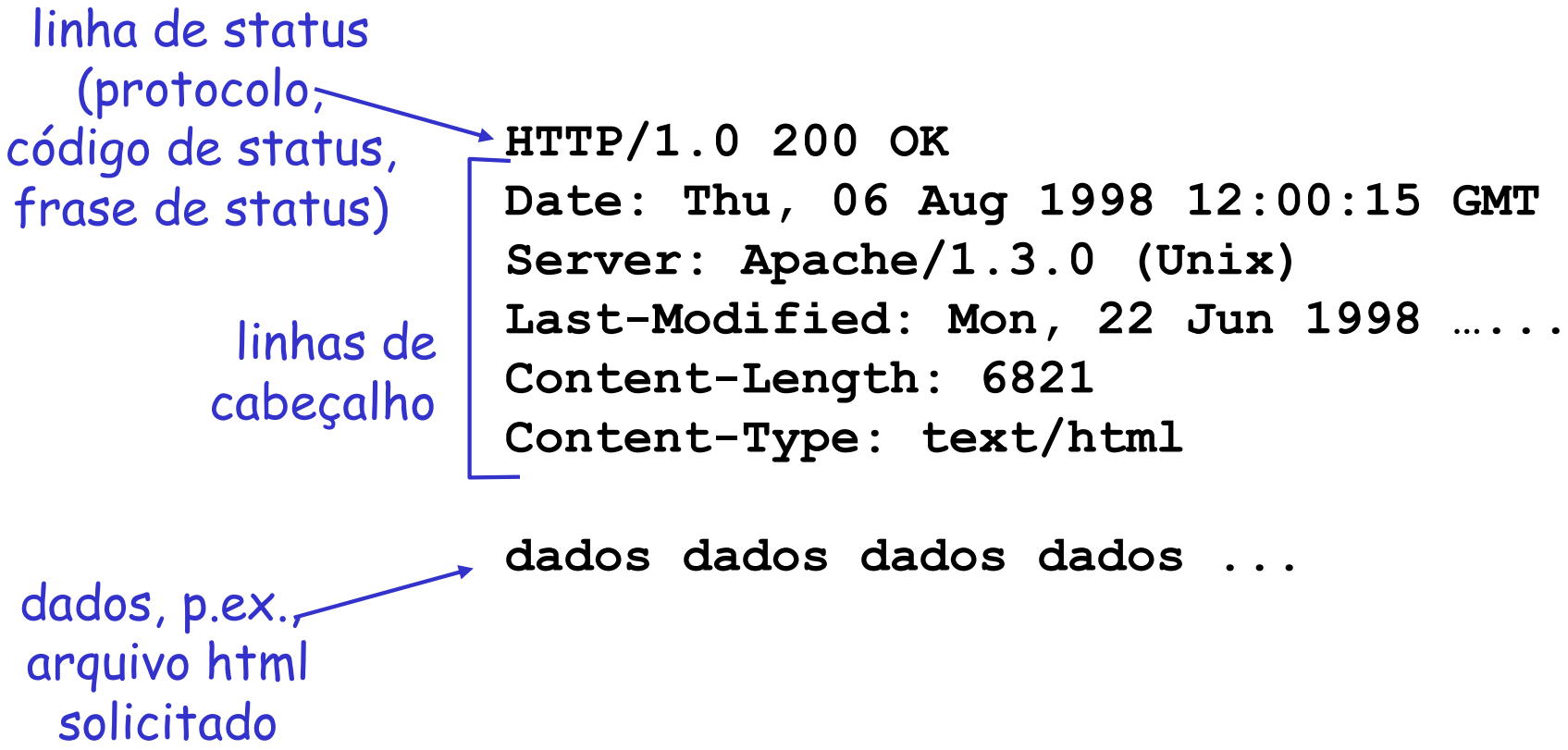
- Dois tipos de mensagem http: *pedido, resposta*
- mensagem de pedido http:
  - ASCII (formato legível por pessoas)



mensagem de pedido http: formato geral



formato de mensagem http: resposta



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

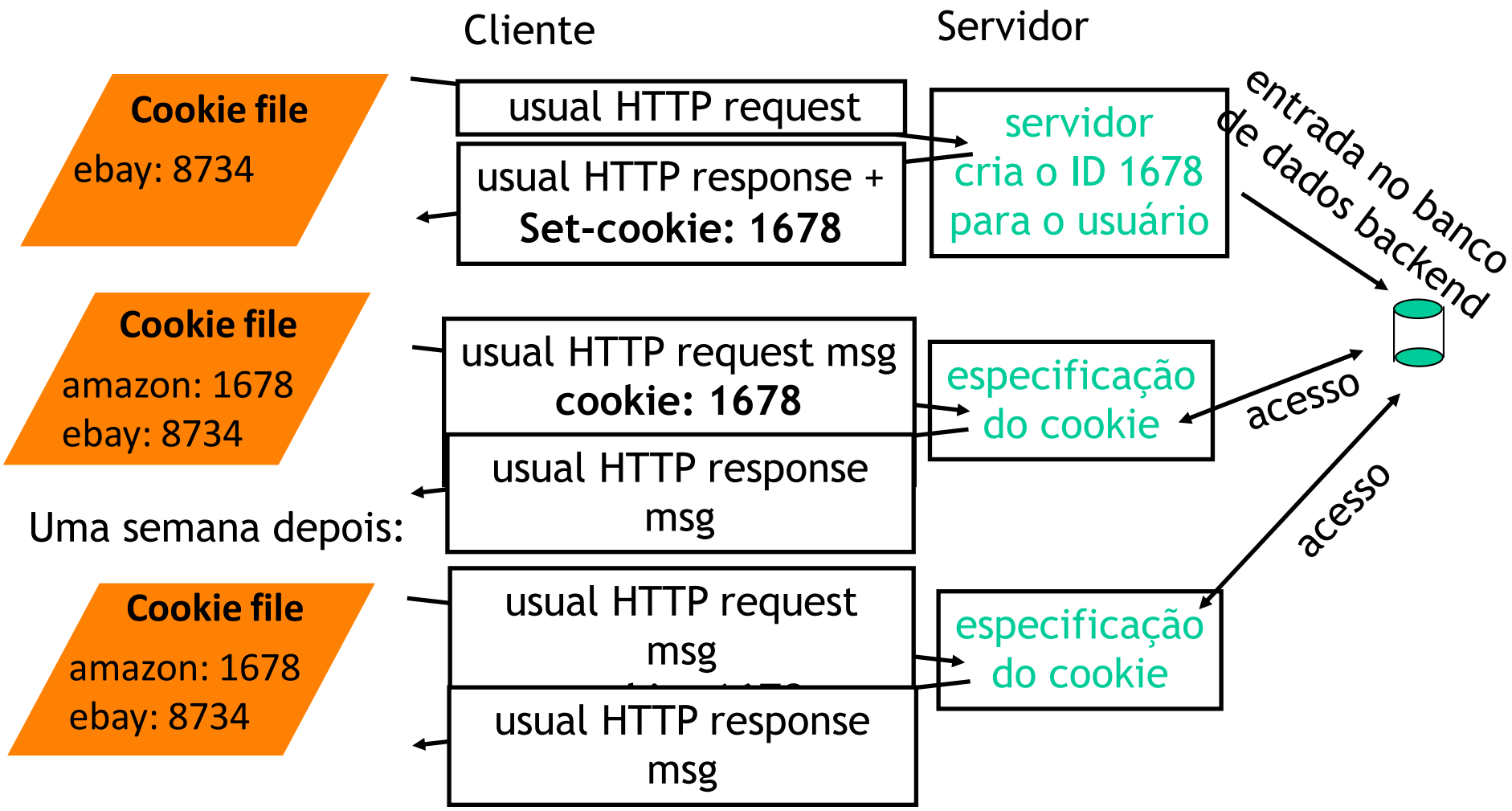
404 Not Found

- documento pedido não se encontra neste servidor

505 HTTP Version Not Supported

- versão de http do pedido não usada por este servidor

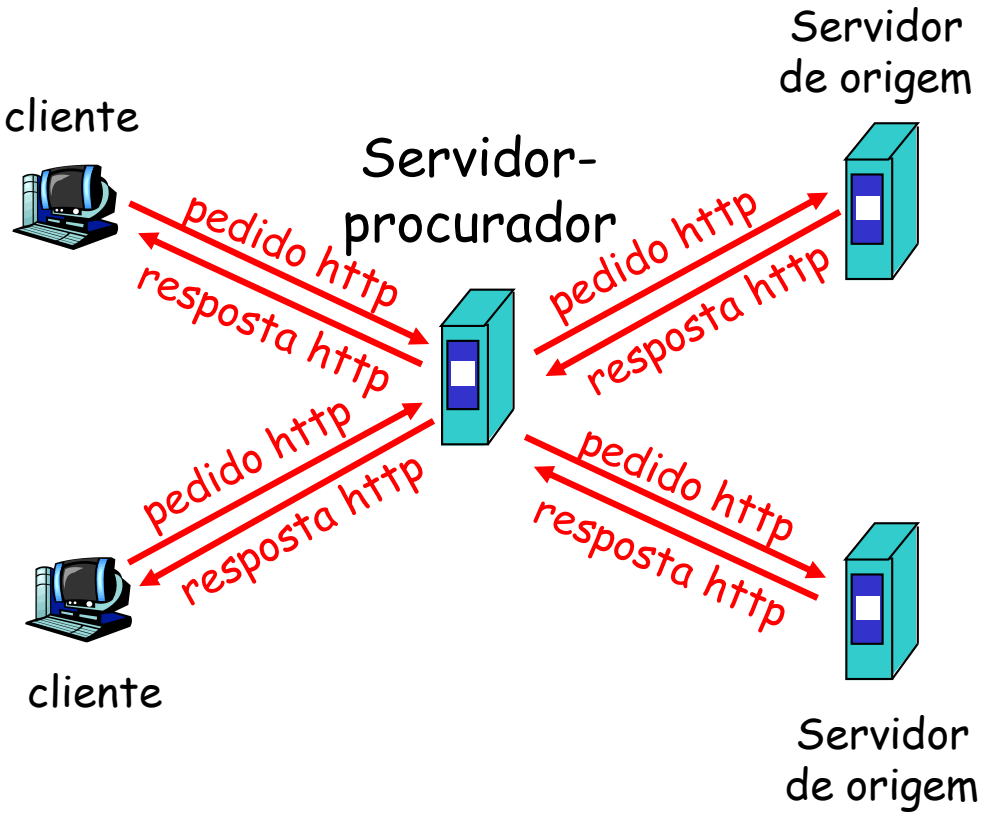
Cookies: mantendo “estado”



Cache WWW (servidor-proxy)

Meta: atender pedido do cliente sem envolver servidor de origem

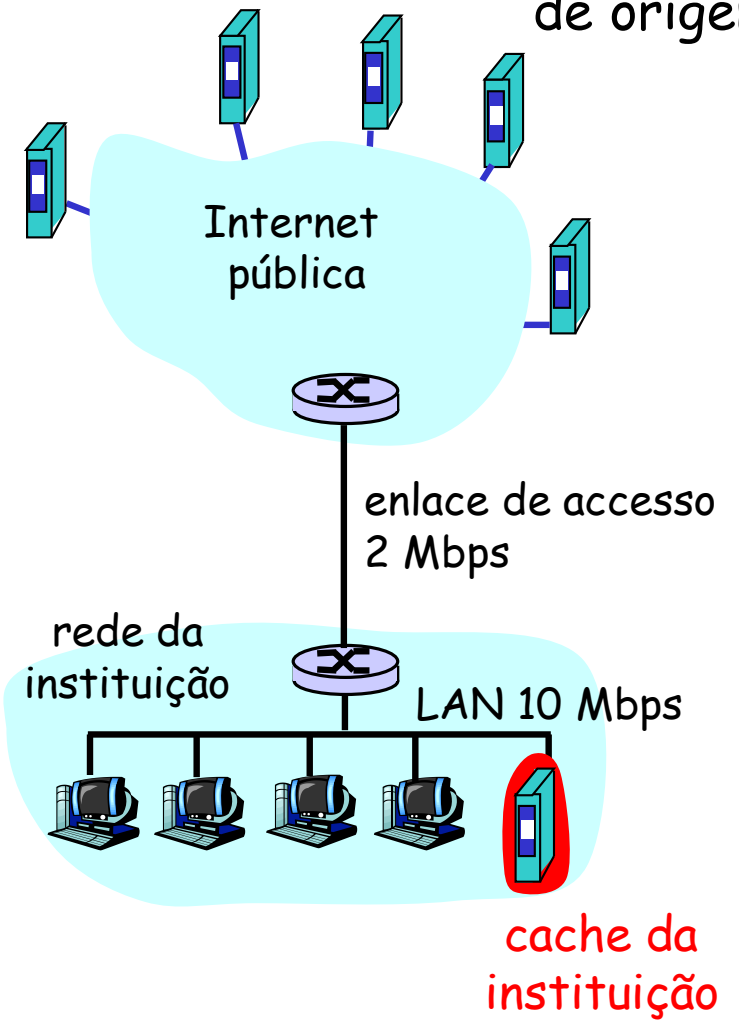
- usuário configura browser: acessos WWW via procurador/proxy
- cliente envia todos pedidos http ao procurador
  - se objeto estiver no cache do procurador, este o devolve imediatamente na resposta http
  - senão, solicita objeto do servidor de origem, depois devolve resposta http ao cliente



Por que usar cache WWW?

Suposição: cache está "próximo" do cliente (p.ex., na mesma rede)

- tempo de resposta menor: cache "mais próximo" do cliente
- diminui tráfego aos servidores distantes
  - muitas vezes o enlace que liga a rede da instituição ou do provedor à Internet é um gargalo



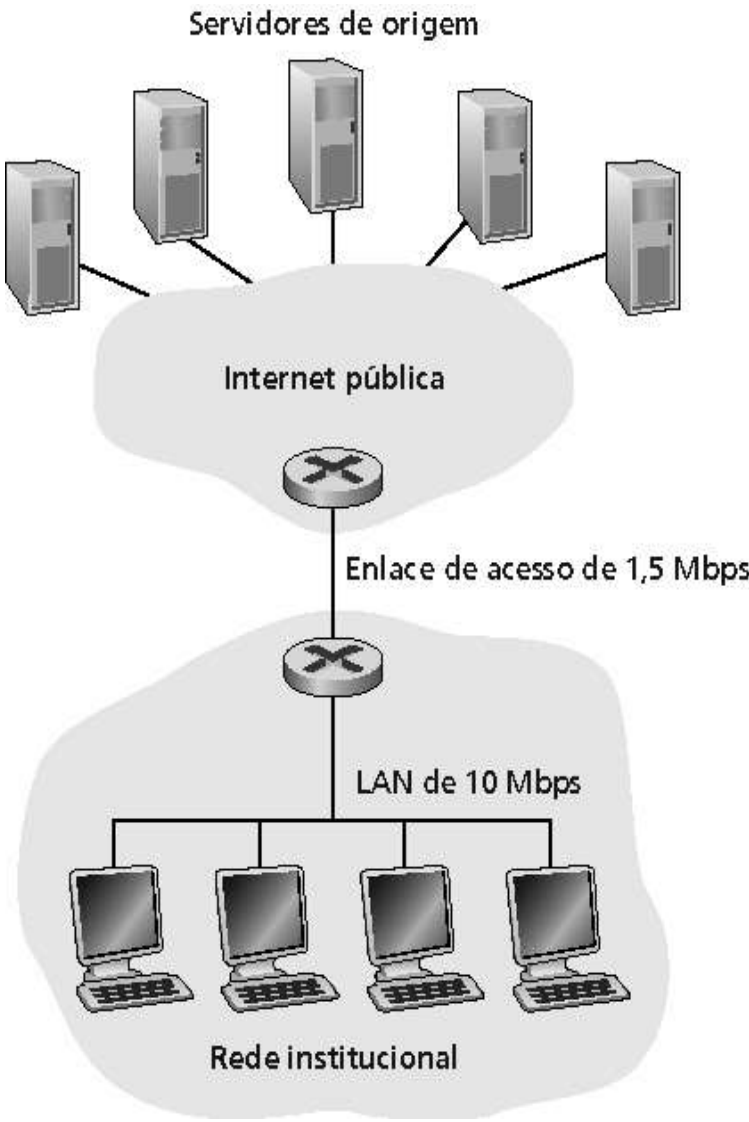
Exemplo de caching

Suponha:

- Tamanho médio objeto = 100.000 bits
- Taxa média de requisições dos browsers da instituição para os servidores de origem = 15 req/s
- Atraso da Internet = 2 s

Conseqüências:

- Utilização da LAN = 15%
  - (15req/s x 100kb/req)/10Mbps
- Utilização do link de acesso = 100%
  - (15 req/s x 100kb/req)/1,5Mbps
- Atraso total = atraso da Internet + atraso de acesso + atraso da LAN = 2 segundos + minutos + milissegundos



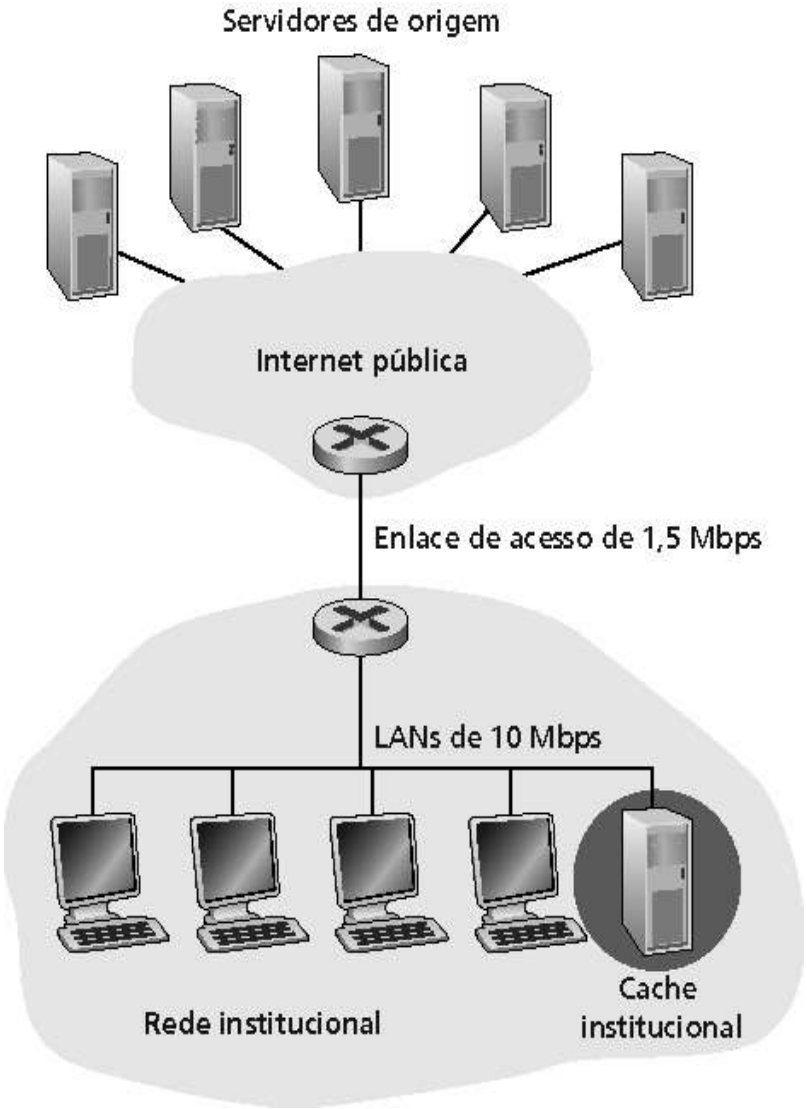
Exemplo de caching

Solução possível

- Aumentar a largura de banda do enlace de acesso, como, 10 Mbps

Conseqüências

- Utilização da LAN = 15%
- Utilização do enlace de acesso = 15%
- Atraso total = atraso da Internet + atraso de acesso + atraso da LAN = 2 segundos + msegs + msegs
- Freqüentemente é um **upgrade** caro



Exemplo de caching

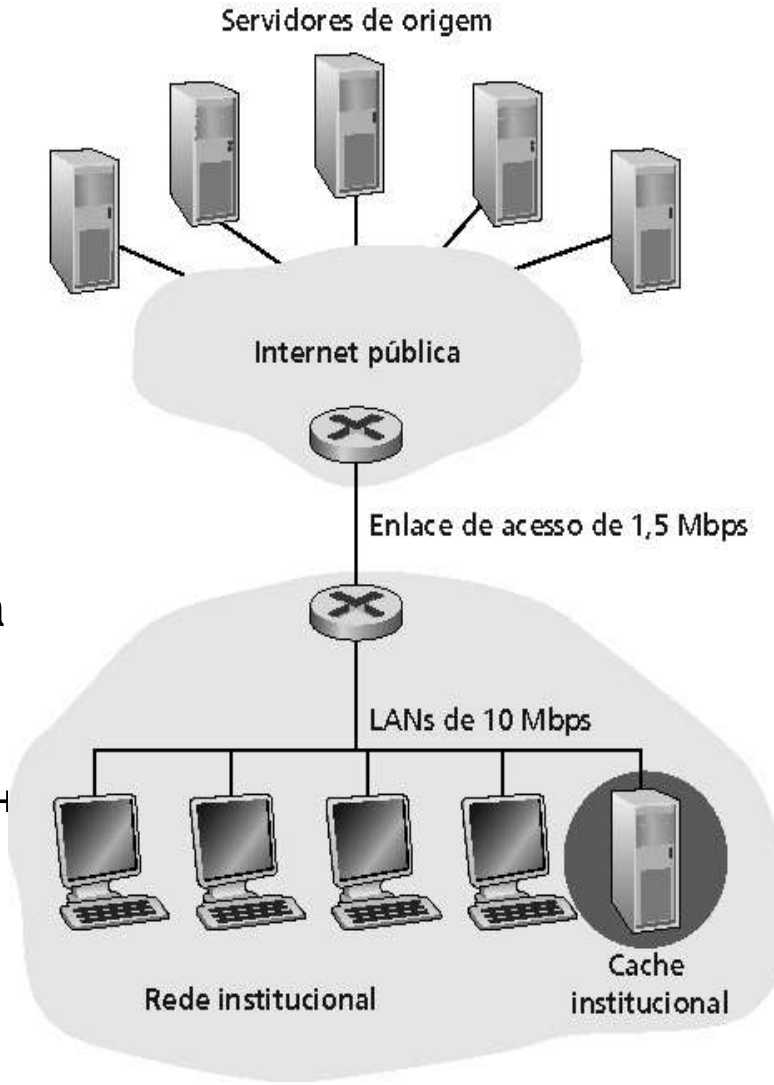
Instalação do cache

- Suponha que a taxa de acertos seja .4

Conseqüência

- 40% das requisições serão satisfeitas quase que imediatamente
- 60% das requisições serão satisfeitas pelo servidor de origem
- Utilização do enlace de acesso reduzida pa 60%, resultando em atrasos insignificantes (como 10 mseg)
- Média de atraso total = atraso da Internet + atraso de acesso + atraso da LAN =

.6*(2.01) s + .4 *(0,01) s < 1,4 s



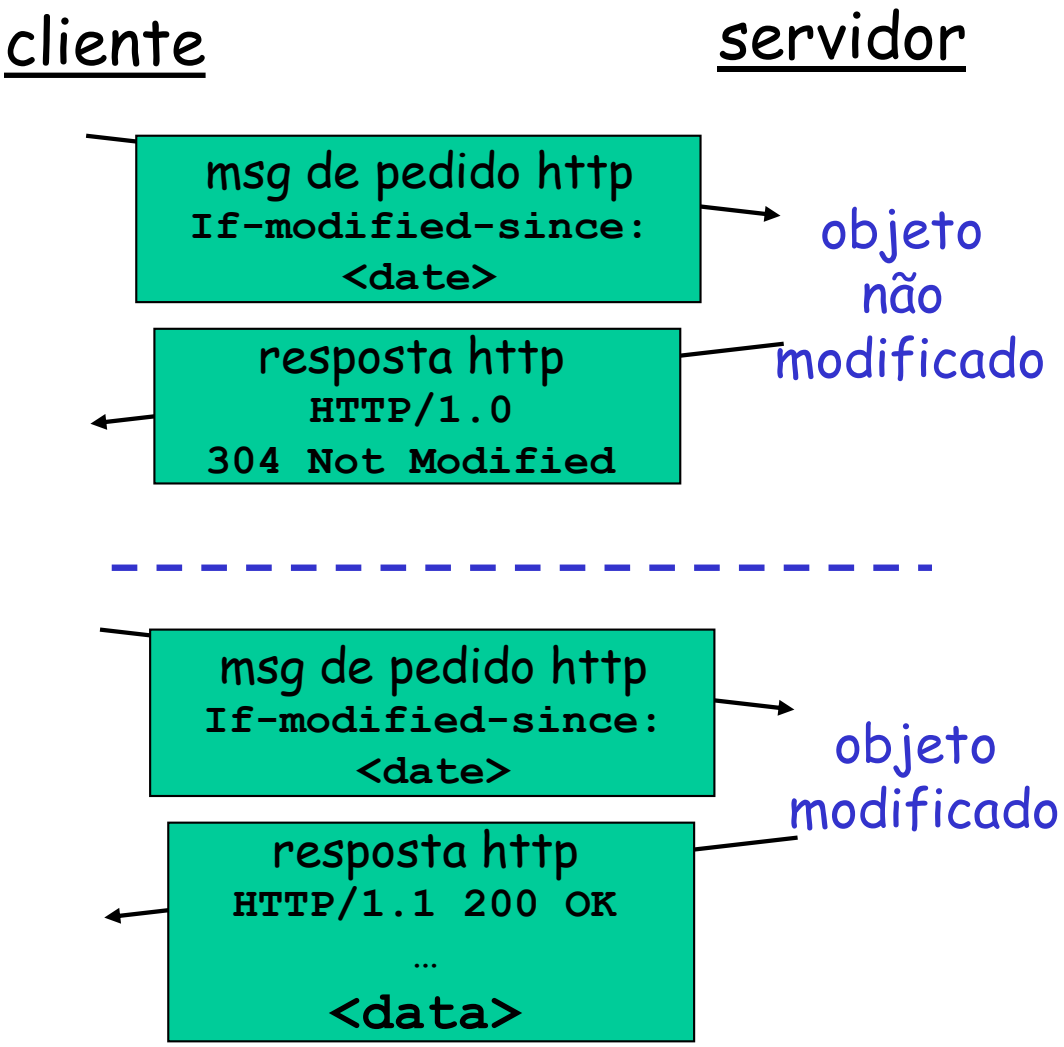
Interação usuário-servidor: GET condicional

- Enviado pelo proxy ao servidor Web
- Meta: não enviar objeto se cliente já tem (no cache) versão atual
- cliente: 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
```



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:

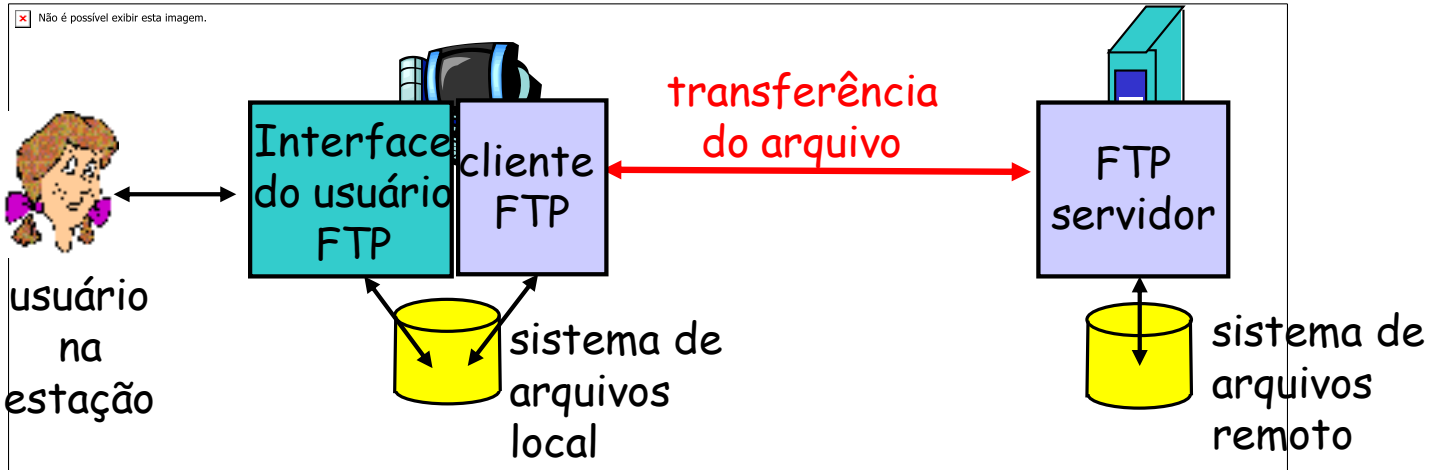
```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

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)

ftp: o protocolo de transferência de arquivos

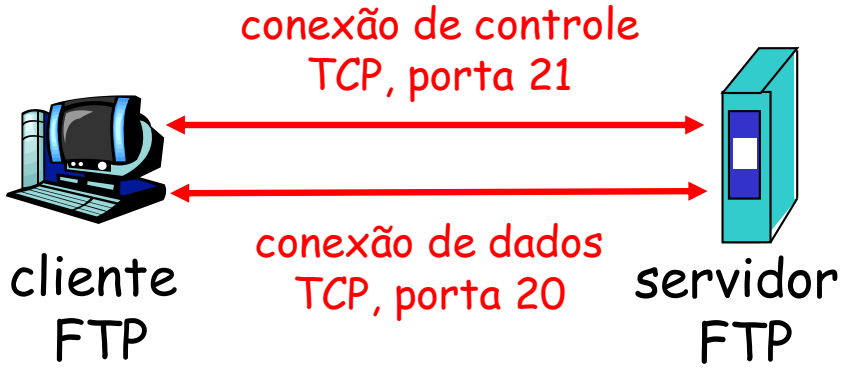


- transferir arquivo de/para hospedeiro remoto
- modelo cliente/servidor
  - *cliente:* lado que inicia transferência (pode ser de ou para o sistema remoto)
  - *servidor:* hospedeiro remoto
- ftp: RFC 959
- servidor ftp: porta 21

ftp: conexões separadas p/ controle, dados

- cliente ftp contata servidor ftp na porta 21, especificando TCP como protocolo de transporte
- são abertas duas conexões TCP paralelas:
  - controle: troca comandos, respostas entre cliente, servidor.

    "controle fora da banda"
  - dados: dados de arquivo de/para servidor
- servidor ftp mantém "estado": diretório corrente, autenticação realizada



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**

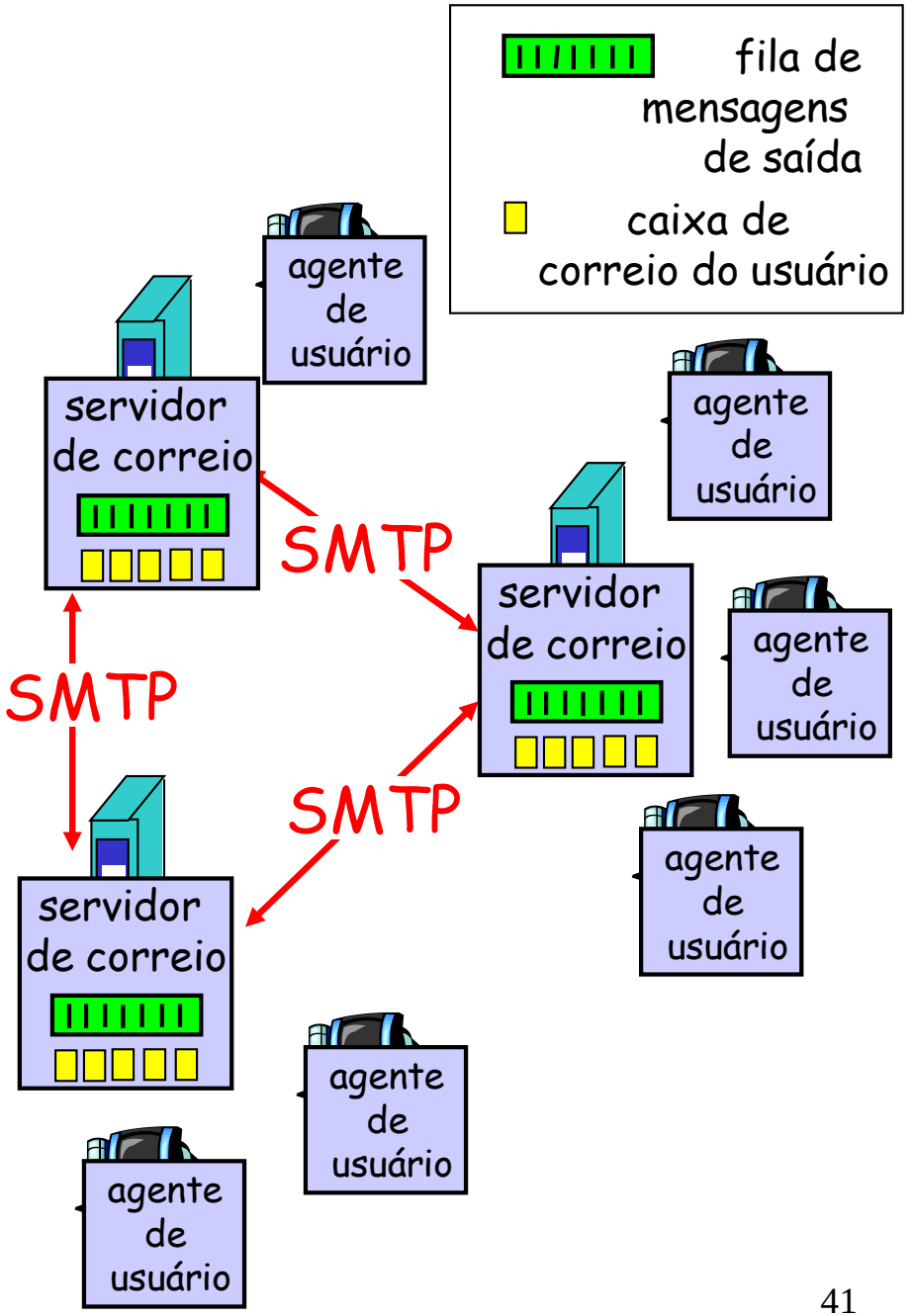
Correio Eletrônico

Três grandes componentes:

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

Agente de Usuário

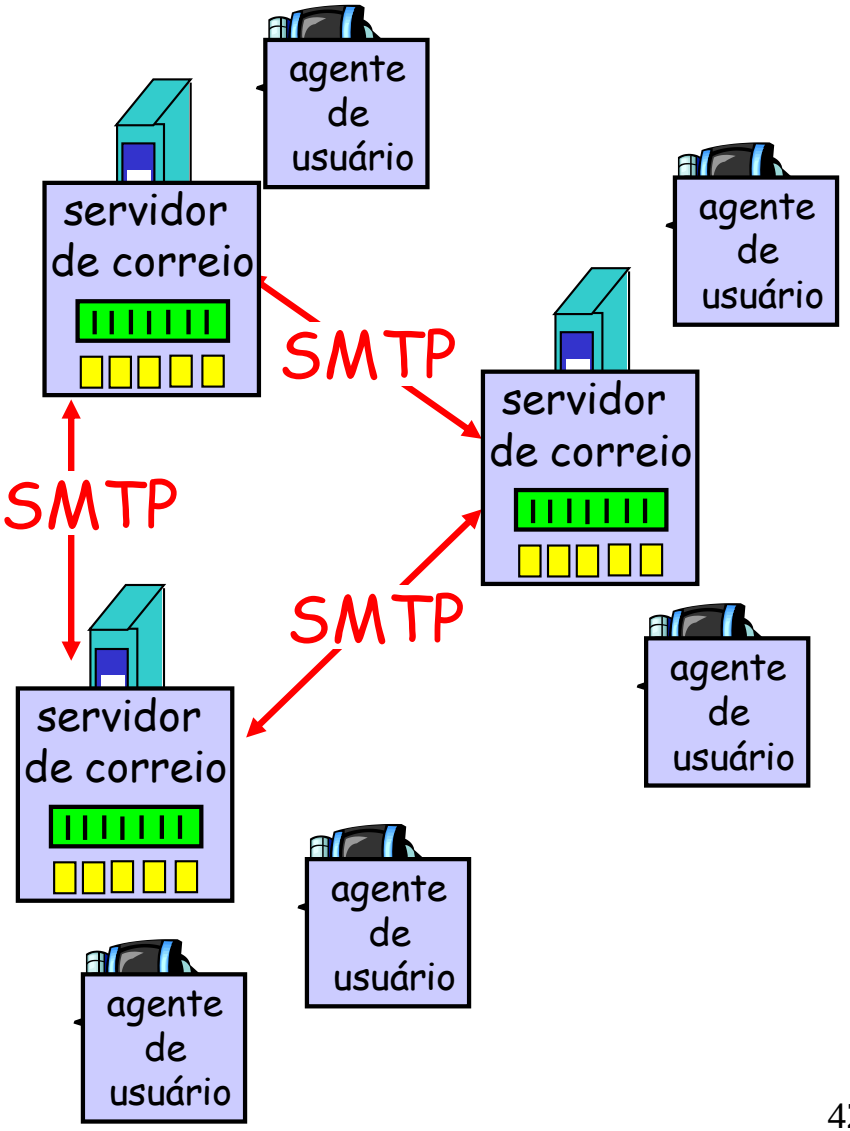
- a.k.a. "leitor de correio"
- compor, editar, ler mensagens de correio
- p.ex., Eudora, Outlook, elm, Netscape Messenger
- mensagens de saída e chegando são armazenadas no servidor



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

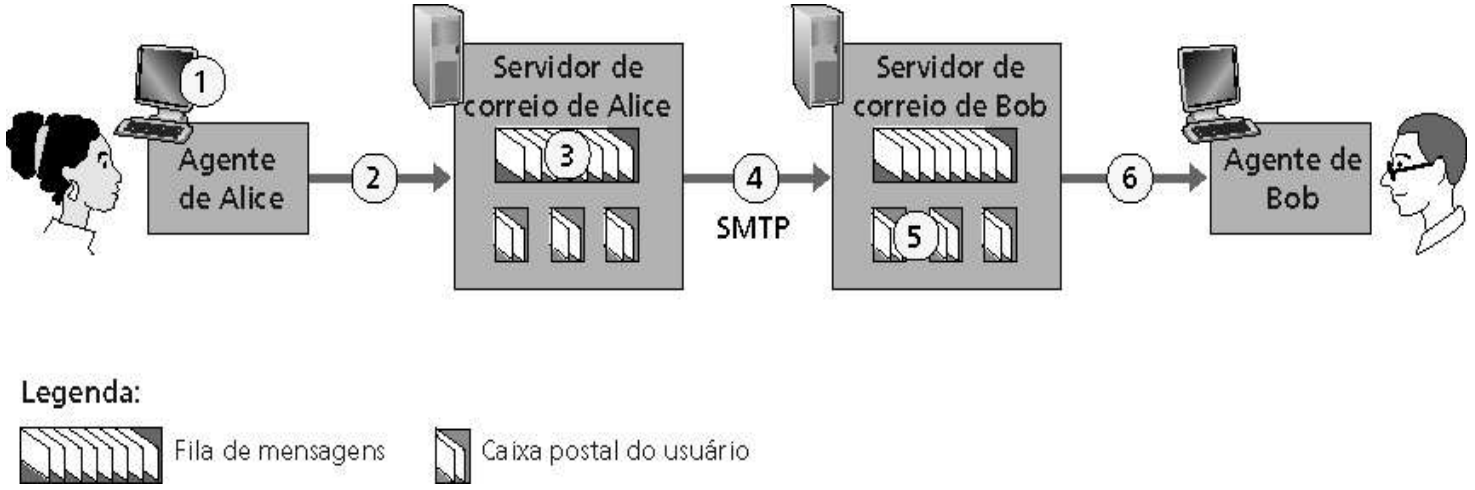


Correio Eletrônico: smtp [RFC 821]

- 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 (apresentação)
  - transferência das mensagens
  - encerramento
- interação comando/resposta
  - comandos: texto ASCII
  - resposta: código e frase de status

Cenário: Alice envia mensagem para Bob

1) Alice usa o agente de usuário (UA) para compor a mensagem "para" bob@someschool.edu
2) O agente de usuário dela 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 do Bob.
4) O cliente SMTP envia a mensagem de Alice pela conexão TCP.
5) O servidor de correio de Bob coloca a mensagem na caixa de correio de Bob.
6) Bob invoca seu agente de usuário 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
```

smtp: últimas palavras

- smtp usa conexões persistentes
- smtp requer que a mensagem (cabeçalho e corpo) sejam em ascii de 7-bits
- algumas cadeias de caracteres não são permitidas numa mensagem (p.ex., `CRLF.CRLF`). Logo a mensagem pode ter que ser codificada (normalmente em base-64 ou "quoted printable")
- servidor smtp usa `CRLF.CRLF` para reconhecer o final da mensagem

Comparação com http

- http: pull (puxar, recuperar)
- email: push (empurrar, enviar)
- 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

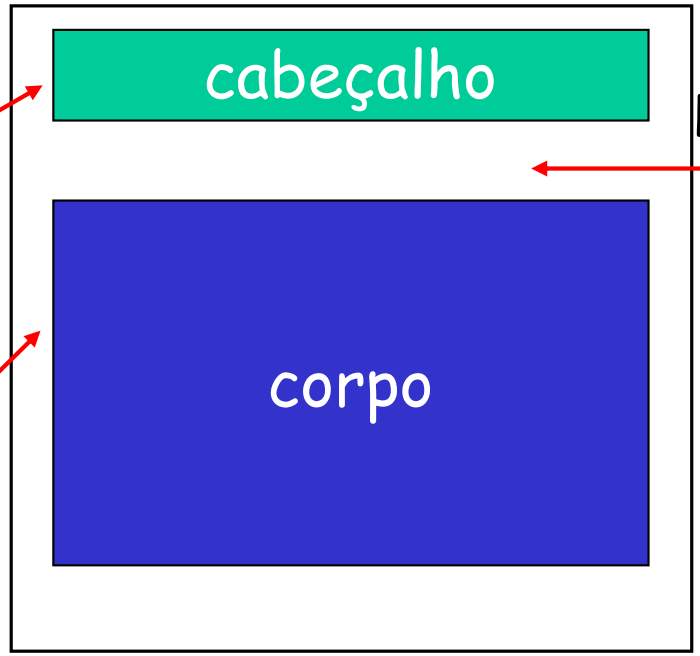
Formato de uma mensagem

smtp: protocolo para trocar msgs de correio

RFC 822: padrão para formato de mensagem de texto:

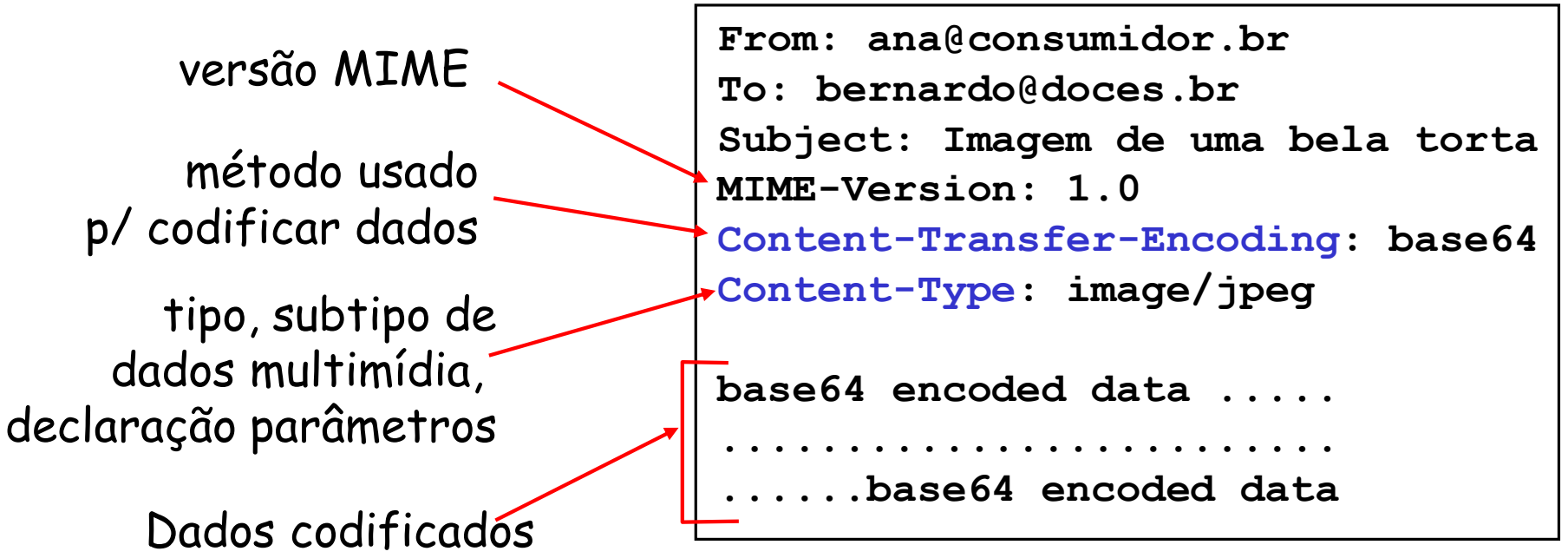
- linhas de cabeçalho, p.ex.,
  - To:
  - From:
  - Subject:

  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



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)

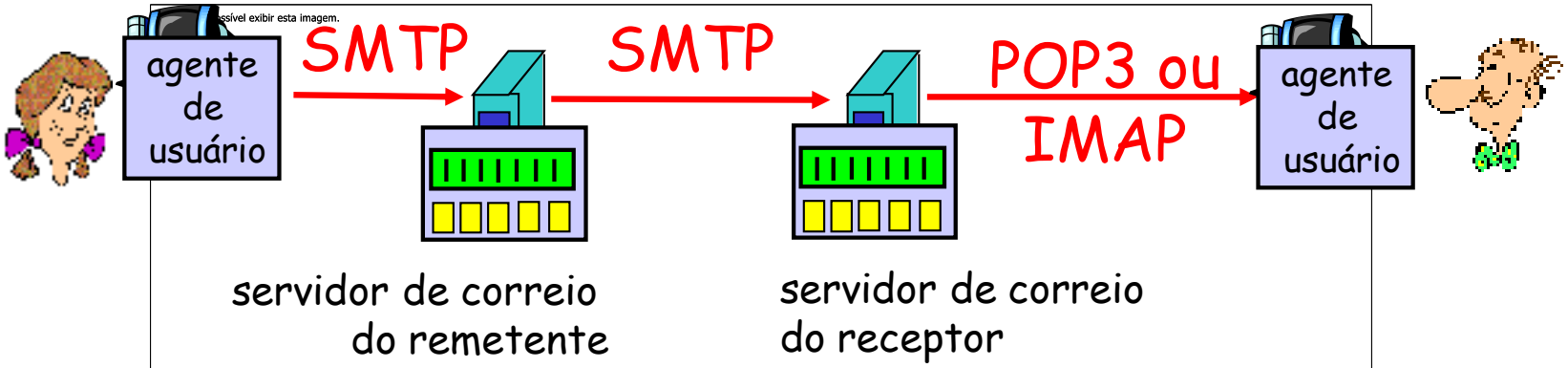
Application

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

Tipo Multipart

```
From: ana@consumidor.br
To: bernardo@doces.br
Subject: Imagem de uma bela torta
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=98766789

--98766789
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain

caro Bernardo,
Anexa a imagem de uma torta deliciosa.
--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 accesso 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. de Passaporte

hospedeiros, roteadores Internet :
- endereço IP (32 bit) - usado p/ endereçar datagramas.
- "nome", e.g., cin.ufpe.br - usado por gente.

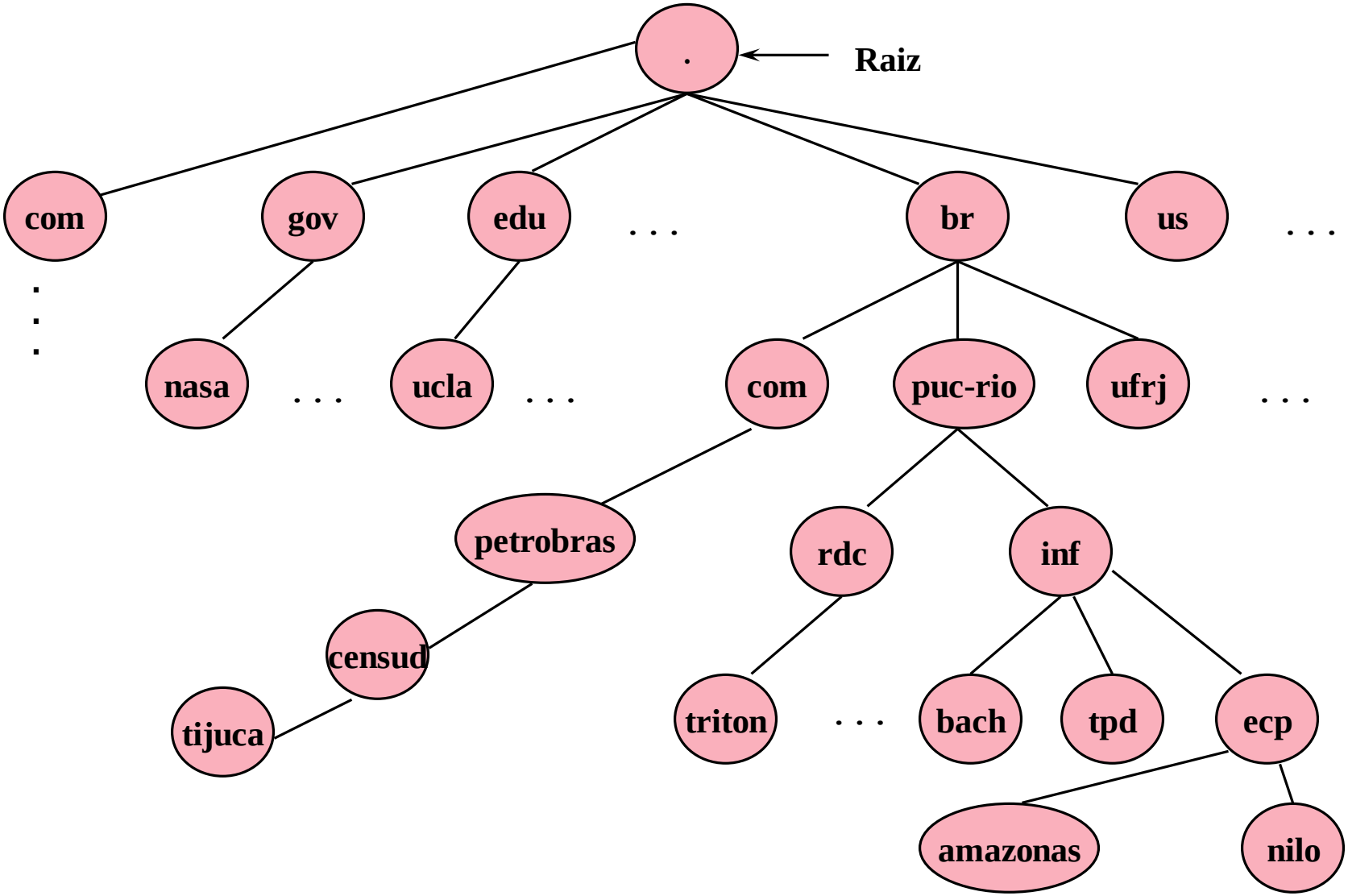
P: como mapear entre nome e endereço IP?

Domain Name System:
- *base de dados distribuída* implementada através de uma hierarquia *servidores de nomes.*
- *protocolo de camada de aplicação* permite que hospedeiros, roteadores e 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

Nomes DNS

- Um nome de domínio é uma concatenação de nomes:
  - nome-n. nome-2.nome-1
- Conceitualmente, o nível mais alto (nome-1) permite diferentes formas diferentes de nomeação:
  - Organizacional
    - com, edu, gov, mil, net e org
  - Geográfica
    - Código dos países:
- Exemplos:
  - inf.puc-rio.br jb.com.br microsoft.com purdue.edu

DNS - Estrutura Hierárquica



DNS (cont.)

Serviços DNS

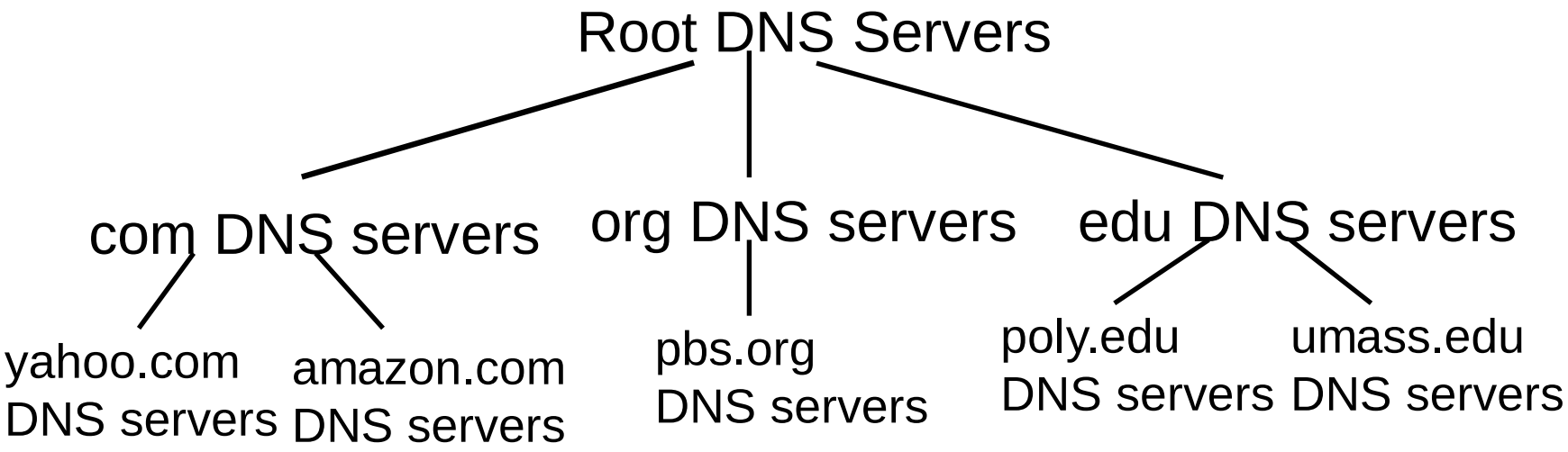
- Tradução de nome de hospedeiro para IP
- Apelidos para hospedeiros (*aliasing*)
  - Nomes canônicos e apelidos
- Apelidos para servidores de e-mail
- Distribuição de carga
  - Servidores Web replicados: conjunto de endereços IP para um mesmo nome

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 de nomes raiz

- São contatados pelos servidores de nomes locais que não podem resolver um nome
- Servidores de nomes raiz:
  - Buscam servidores de nomes autorizados se o mapeamento do nome não for conhecido
  - Conseguem o mapeamento
  - Retornam o mapeamento para o servidor de nomes local

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.
  - 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

Servidores de nomes DNS

- 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

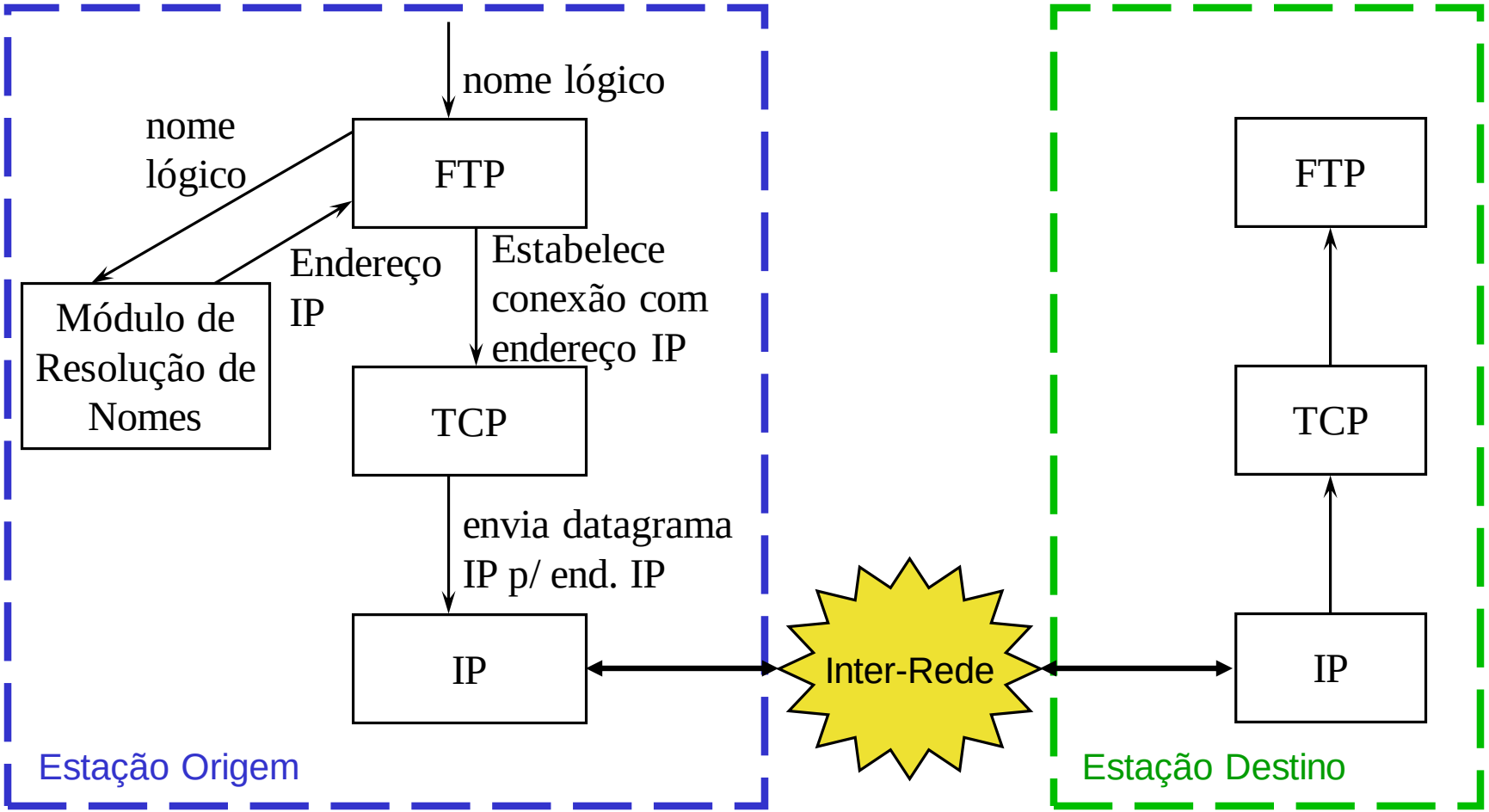
servidor de nomes com autoridade:

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

Implementação do DNS

- A estrutura hierárquica é global e distribuída entre servidores de nomes
- resolução de nomes ⇒ uma pesquisa distribuída
- Tipo da pesquisa:
  - recursiva: fornece resultado
  - iterativa: fornece uma dica
- Uso de cache
  - guardar respostas localmente
  - dados marcados com TTL (Time To Live)

Resolução de Nomes

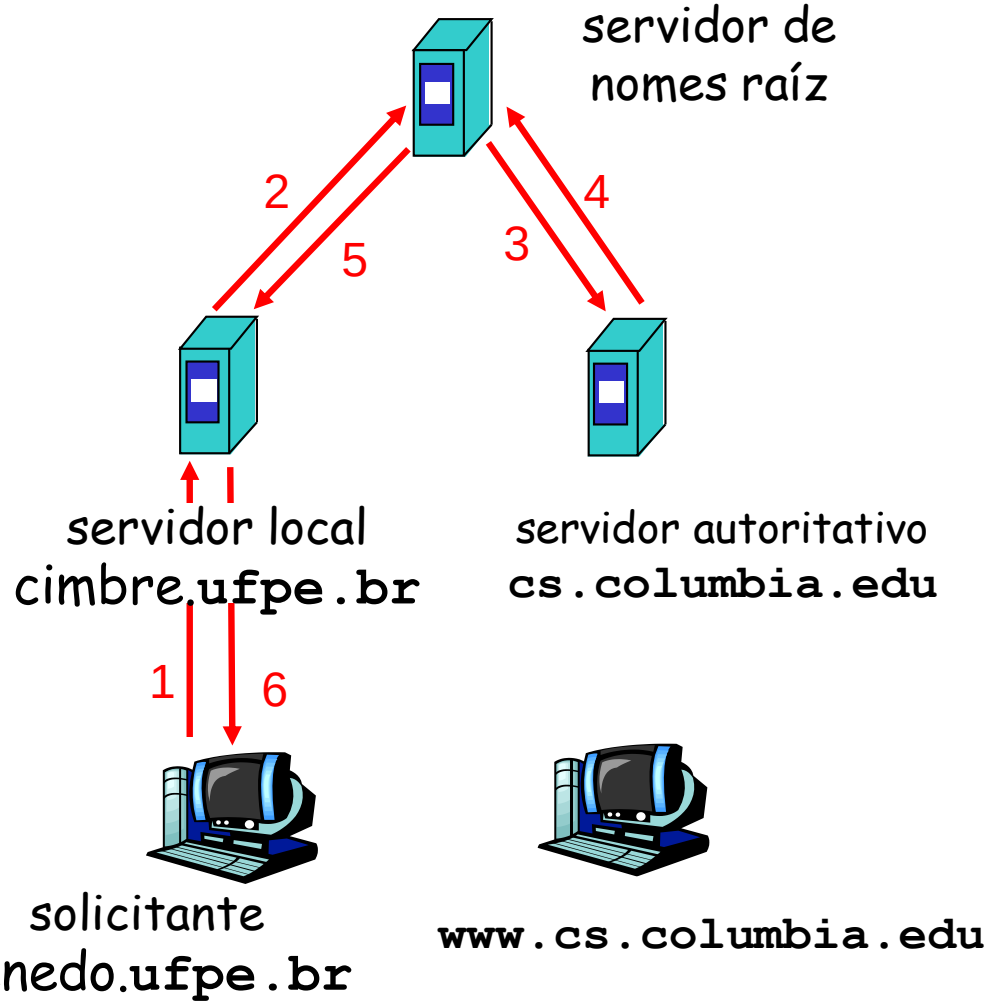


- Módulo Resolução de Nomes: consulta arquivos locais ou um serviço de resolução de nomes

Exemplo simples do DNS

hospedeiro penedo.**ufpe.br** requer endereço IP de **www.cs.columbia.edu**

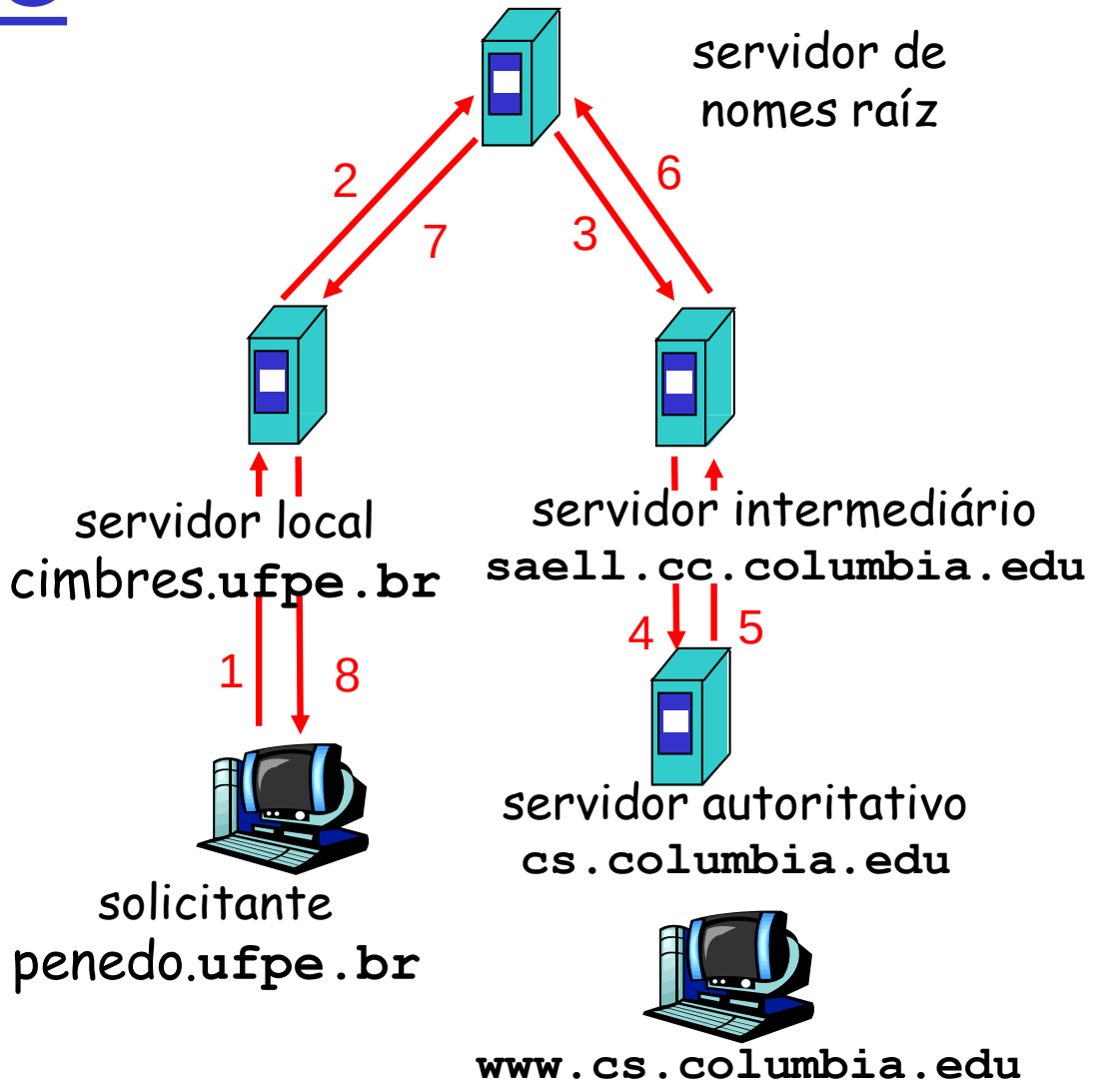
1. Contata servidor DNS local, cimbres.**ufpe.br**
2. cimbres.**ufpe.br** contata servidor raiz, se necessário
3. Servidor raiz contata servidor com autoridade **cs.columbia.edu**, se necessário



Exemplo de DNS

Servidor raiz:

- pode não conhecer o servidor de nomes com autoridade
- pode conhecer *servidor de nomes intermediário:* a quem contacta para descobrir o servidor de nomes autoritativo



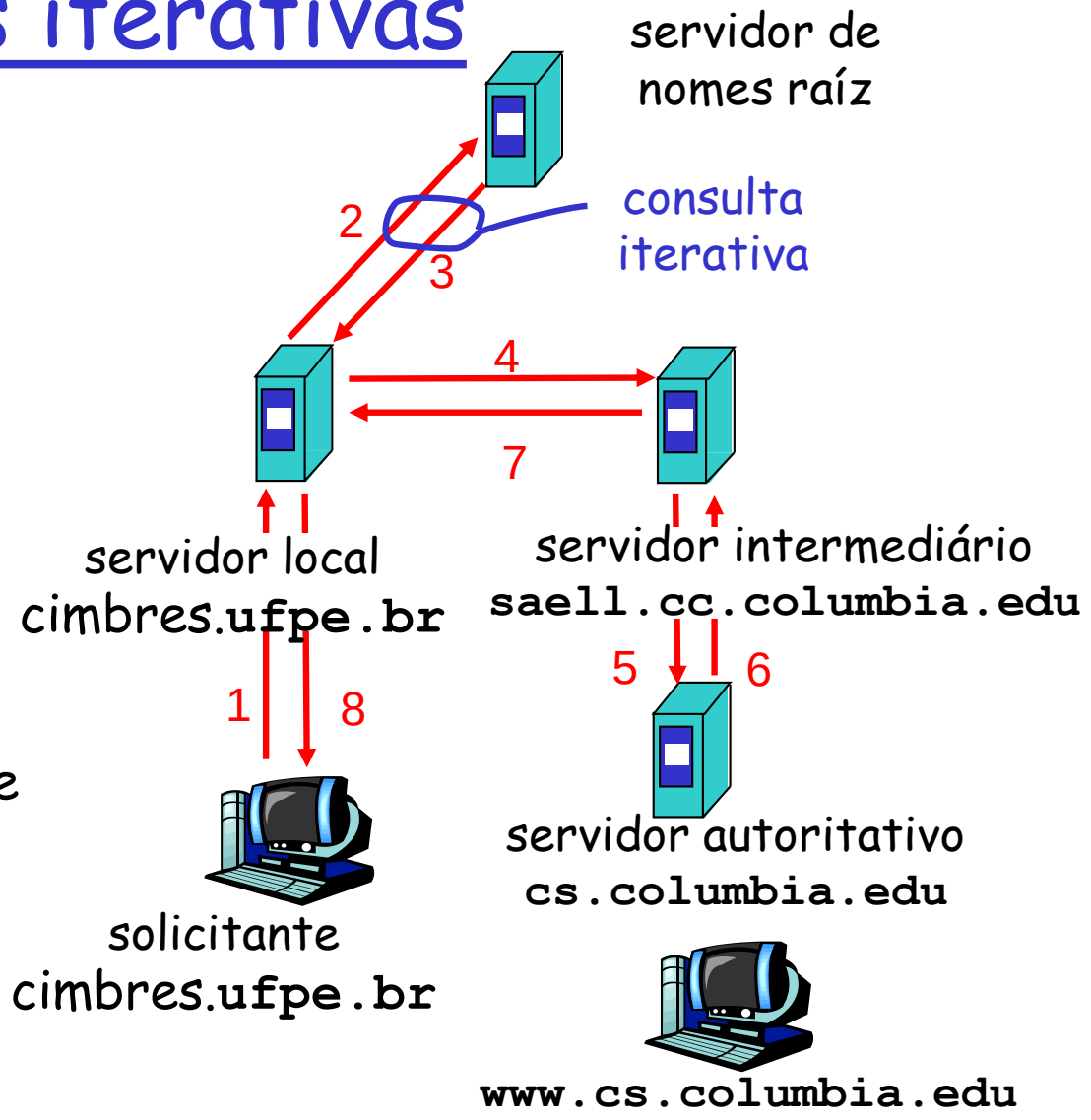
DNS: consultas iterativas

consulta recursiva:

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

consulta iterativa:

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



Registros do DNS

DNS: base de dados distribuída que armazena registros de recursos (RR)

formato dos RR: **(name, value, type,ttl)**

- Type = A
  - **name** é o nome do computador
  - **value** é o endereço IP

- Type = NS
  - **name** é um domínio (ex.: foo.com)
  - **value** é o endereço IP do servidor de nomes autorizados para este domínio

- Type = CNAME
  - **name** é um "apelido" para algum nome "canônico" (o nome real) www.ibm.com é realmente servereast.backup2.ibm.com
  - **value** é o nome canônico

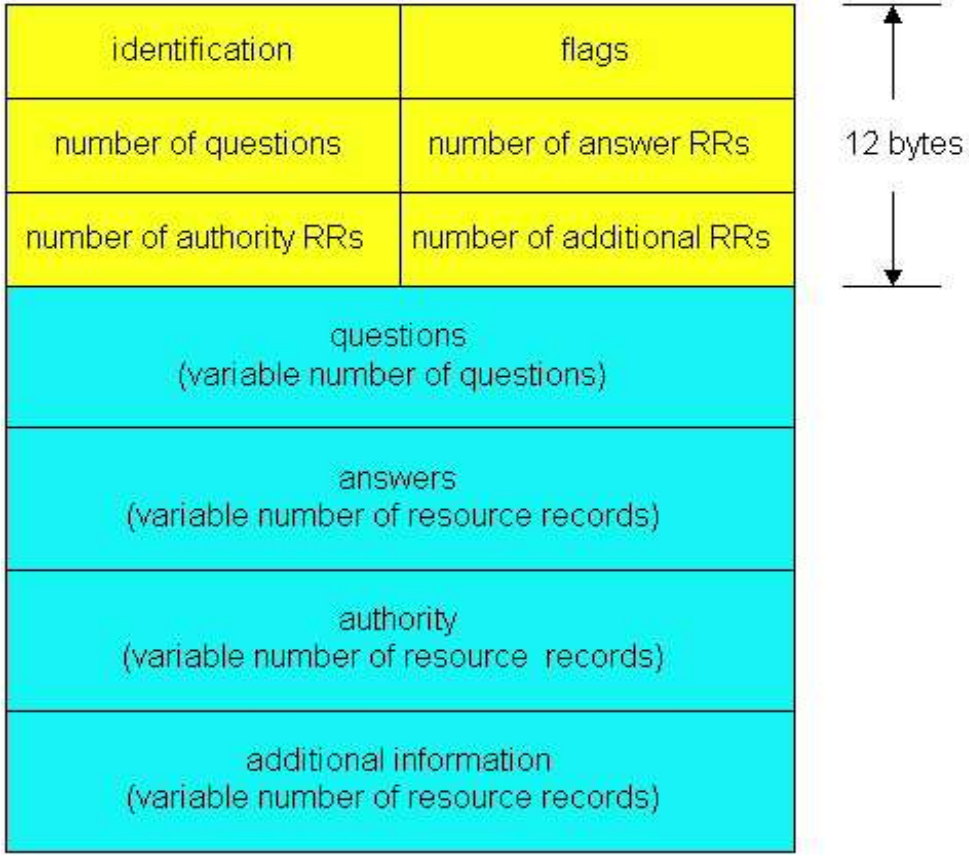
- Type = MX
  - **value** é o nome do servidor de correio associado com **name**

DNS: protocolo, mensagens

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

cabeçalho de msg

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



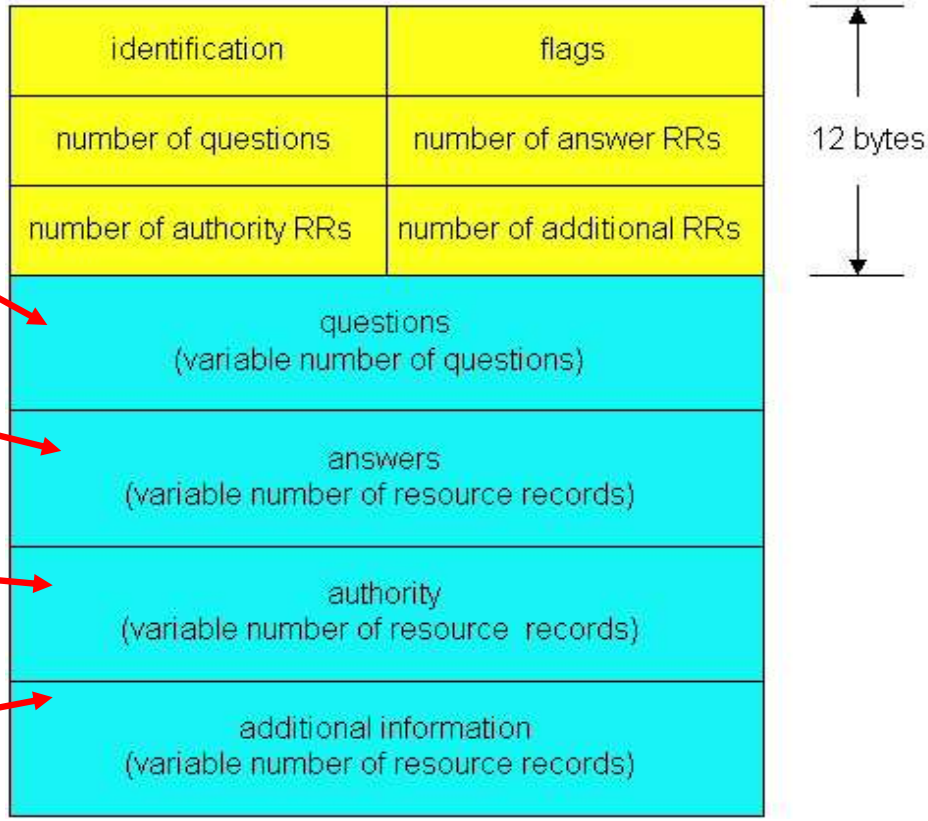
DNS: protocolo, mensagens

campos nome, tipo num pedido

RRs em resposta ao pedido

registros para servidores autoritativos

info adicional "relevante" que pode ser usada



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)
```

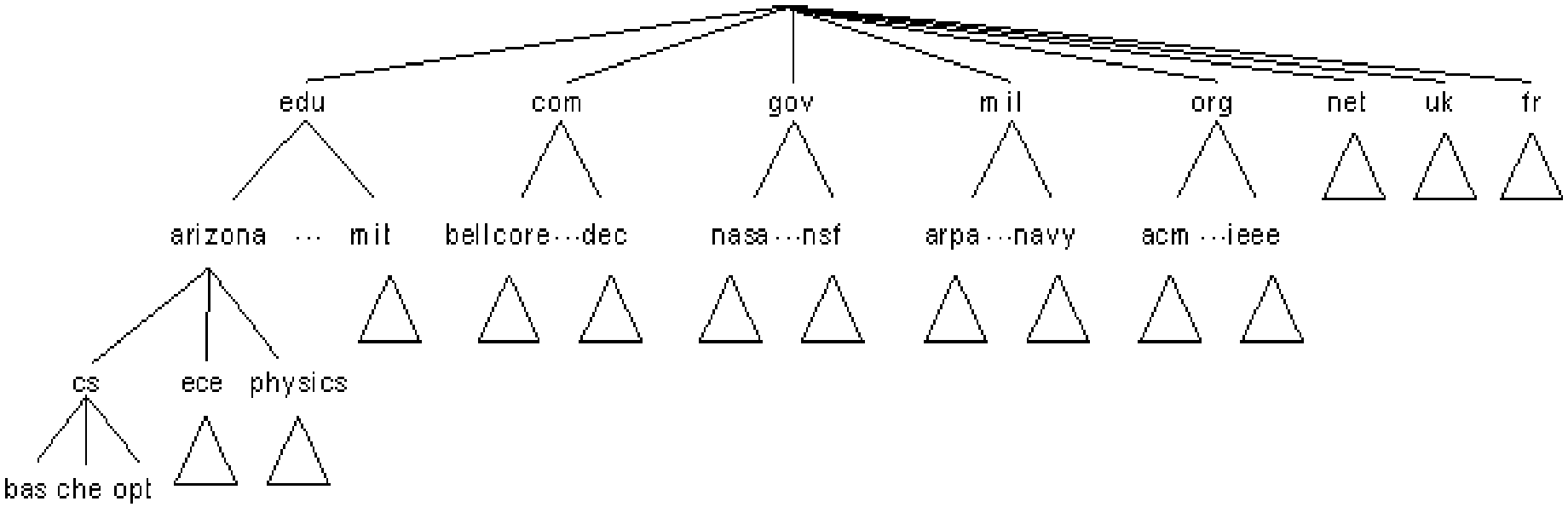
- Põe no servidor oficial um registro do tipo A para www.netutopia.com.br e um registro do tipo MX para netutopia.com.br

Exemplo

Servidor Raiz:

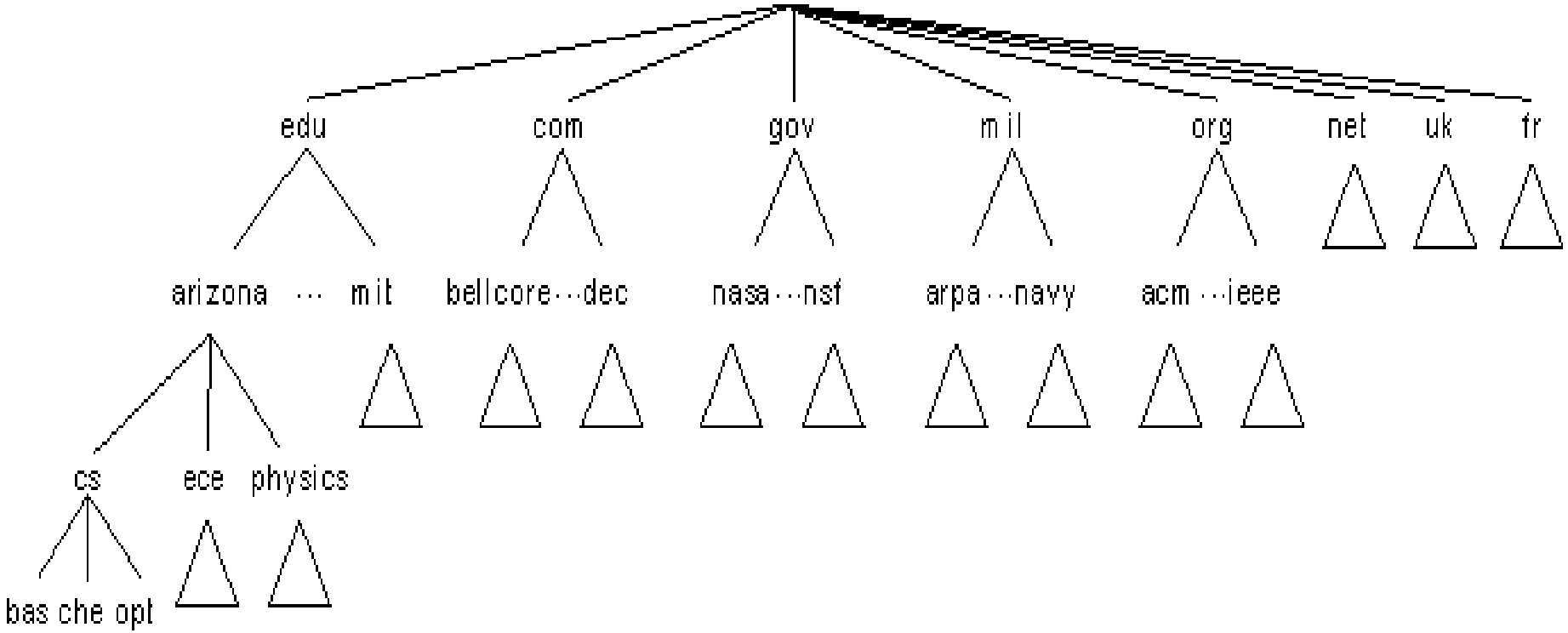
<arizona.edu, telcom.arizona.edu, NS, IN>
<telcom.arizona.edu, 128.196.128.233, A, IN>

<bellcore.com, thumper.bellcore.com, NS, IN>
<thumper.bellcore.com, 128.96.32.20, A, IN>



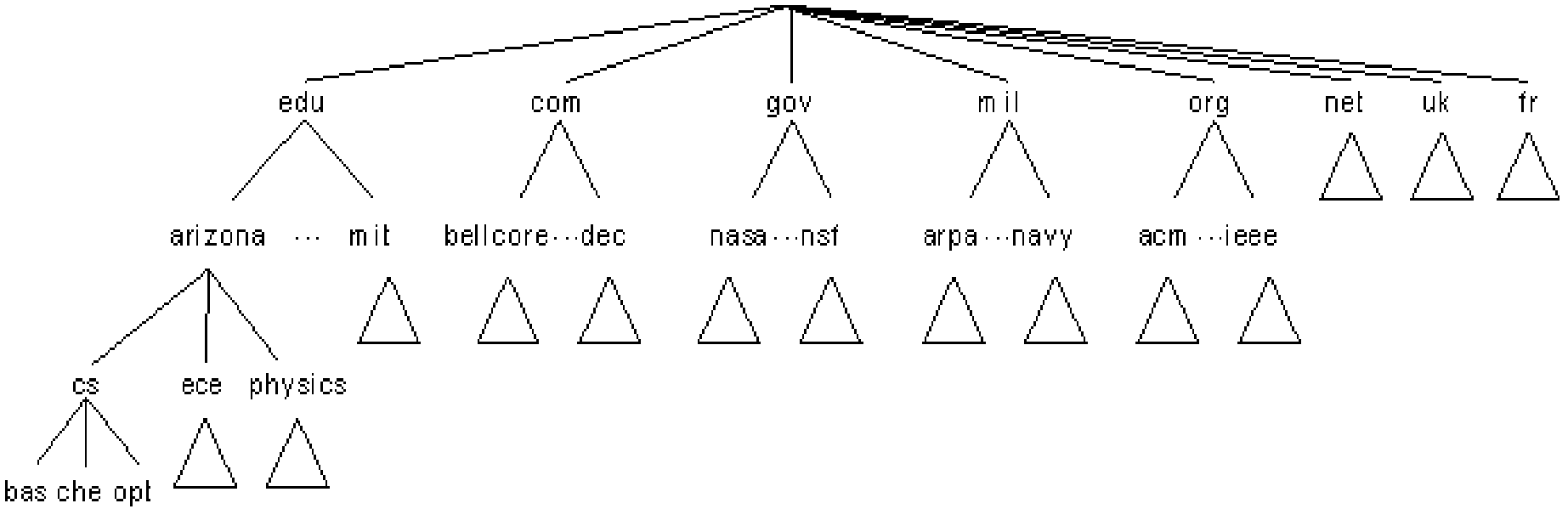
Servidor Arizona:

```
<cs.arizona.edu, optima.cs.arizona.edu, NS, IN>
<optima.cs.arizona.edu, 192.12.69.5, A, IN>

<ece.arizona.edu, helios.ece.arizona.edu, NS, IN>
<helios.ece.arizona.edu, 128.196.28.166, A, IN>

<jupiter.physics.arizona.edu, 128.196.4.1, A, IN>
<saturn.physics.arizona.edu, 128.196.4.2, A, IN>
<mars.physics.arizona.edu, 128.196.4.3, A, IN>
<venus.physics.arizona.edu, 128.196.4.4, A, IN>
```

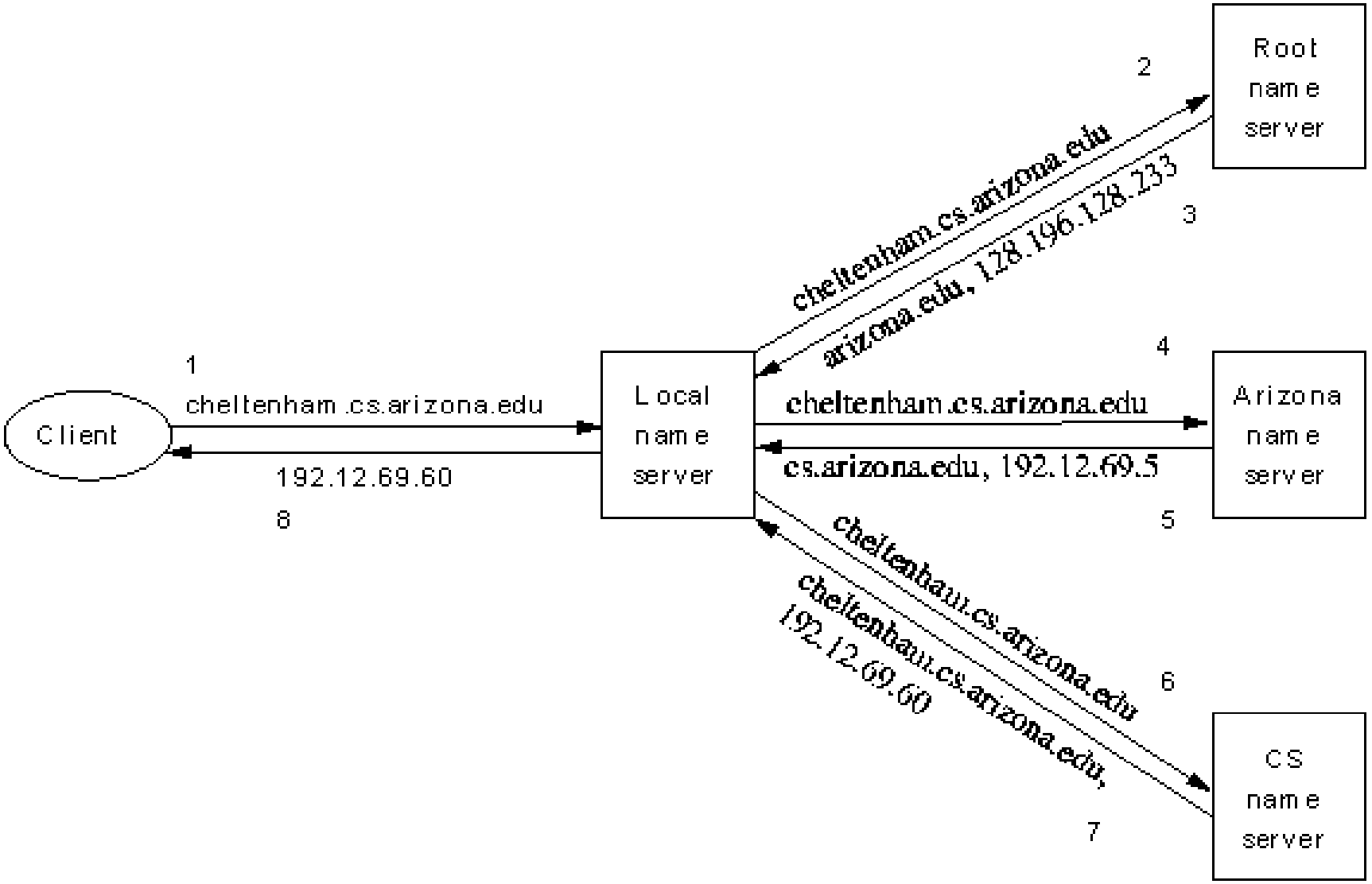


Servidor CS:

```
<cs.arizona.edu, optima.cs.arizona.edu, MX, IN>

<cheltenham.cs.arizona.edu, 192.12.69.60, A, IN>
<che.cs.arizona.edu, cheltenham.cs.arizona.edu,
  CNAME, IN>

<optima.cs.arizona.edu, 192.12.69.5, A, IN>
<opt.cs.arizona.edu, optima.cs.arizona.edu,
  CNAME, IN>

<baskerville.cs.arizona.edu, 192.12.69.35, A, IN>
<bas.cs.arizona.edu, baskerville.cs.arizona.edu,CNAME,
 IN>
```



Resolução de nome



DNS - Ferramentas de Diagnóstico

- nslookup
  - Permite acesso às informações de DNS de um domínio
    - Estação responsável pela zona e e-mail do administrador da zona
    - Servidor de Mail da zona
    - Mapeamento de nomes em endereços IP e vice-versa
    - Informações sobre estações (HINFO)
    - ...

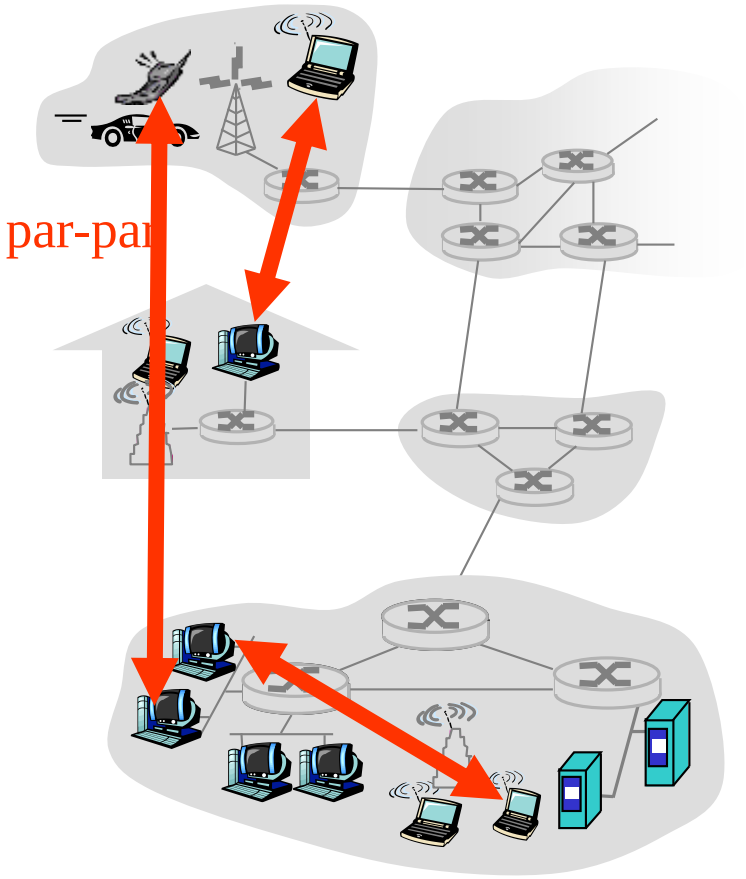
Capítulo 2: Roteiro

- 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

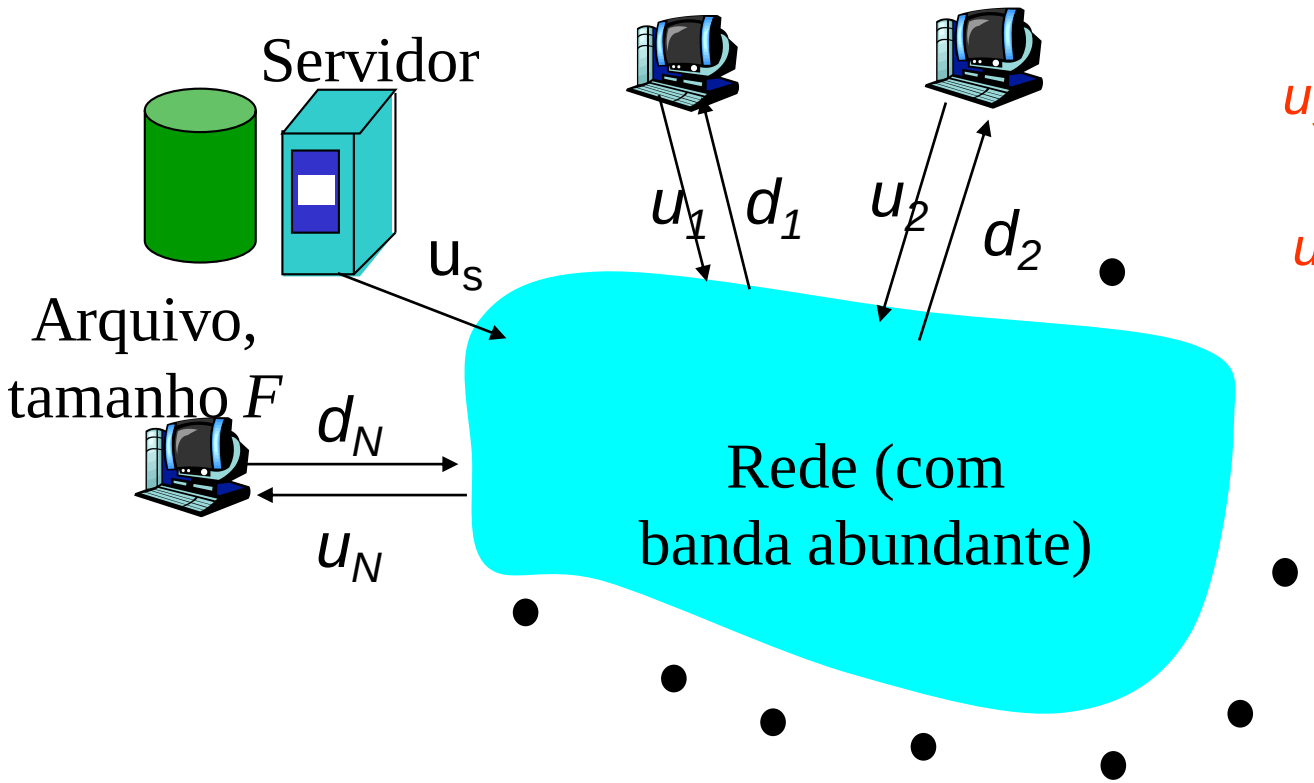
- Exemplos:
  - Distribuição de arquivos (BitTorrent)
  - *Streaming* (KanKan)
  - VoIP (Skype)



Distribuição de Arquivo: C/S x P2P

Pergunta: Quanto tempo leva para distribuir um arquivo de um servidor para N pares?

- Capacide de *upload/download* de um par é um recurso limitado



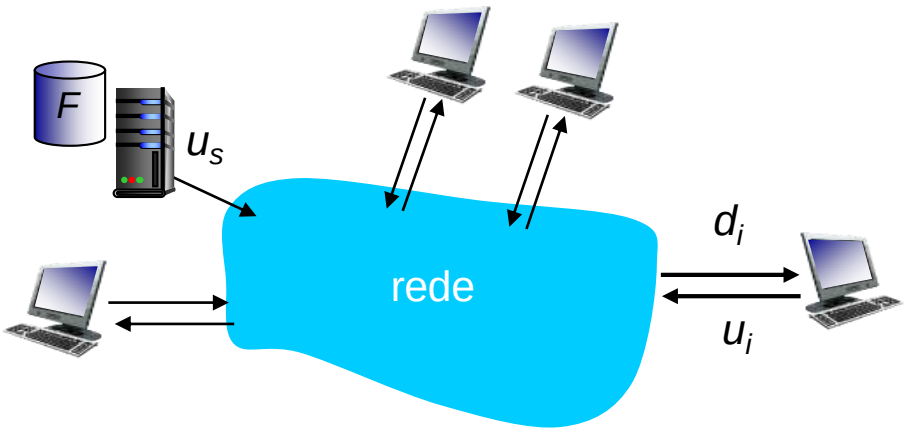
u_s: banda de upload do servidor

u_i: banda de upload do par i

d_i: banda de download do par i

Tempo de distribuição do arquivo: C/S

- transmissão do servidor: deve enviar sequencialmente N cópias do arquivo:
  - Tempo para enviar uma cópia = F/u_s
  - Tempo para enviar N cópias = NF/u_s
- cliente: cada cliente deve fazer o *download* de uma cópia do arquivo
  - d_{min} = taxa mínima de *download*
  - Tempo de *download* para usuário com menor taxa: F/d_{min}



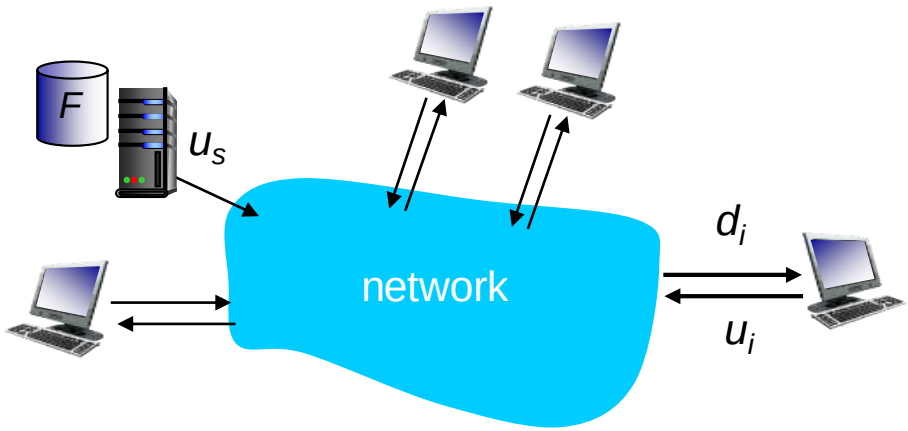
Tempo para distribuir F para N clientes usando abordagem cliente/servidor

$$D_{cs} \geq \max \{ NF/u_s, F/d_{min} \}$$

cresce linearmente com N

Tempo de distribuição do arquivo: P2P

- transmissão do servidor: deve enviar pelo menos uma cópia:
  - tempo para enviar uma cópia: F/u_s
- cliente: cada cliente deve baixar uma cópia do arquivo
  - Tempo de *download* para usuário com menor taxa: F/d_{min}
- clientes: no total devem baixar NF bits
  - Taxa máxima de *upload* : $u_s + \sum u_i$



tempo para distribuir F para N clientes usando abordagem P2P

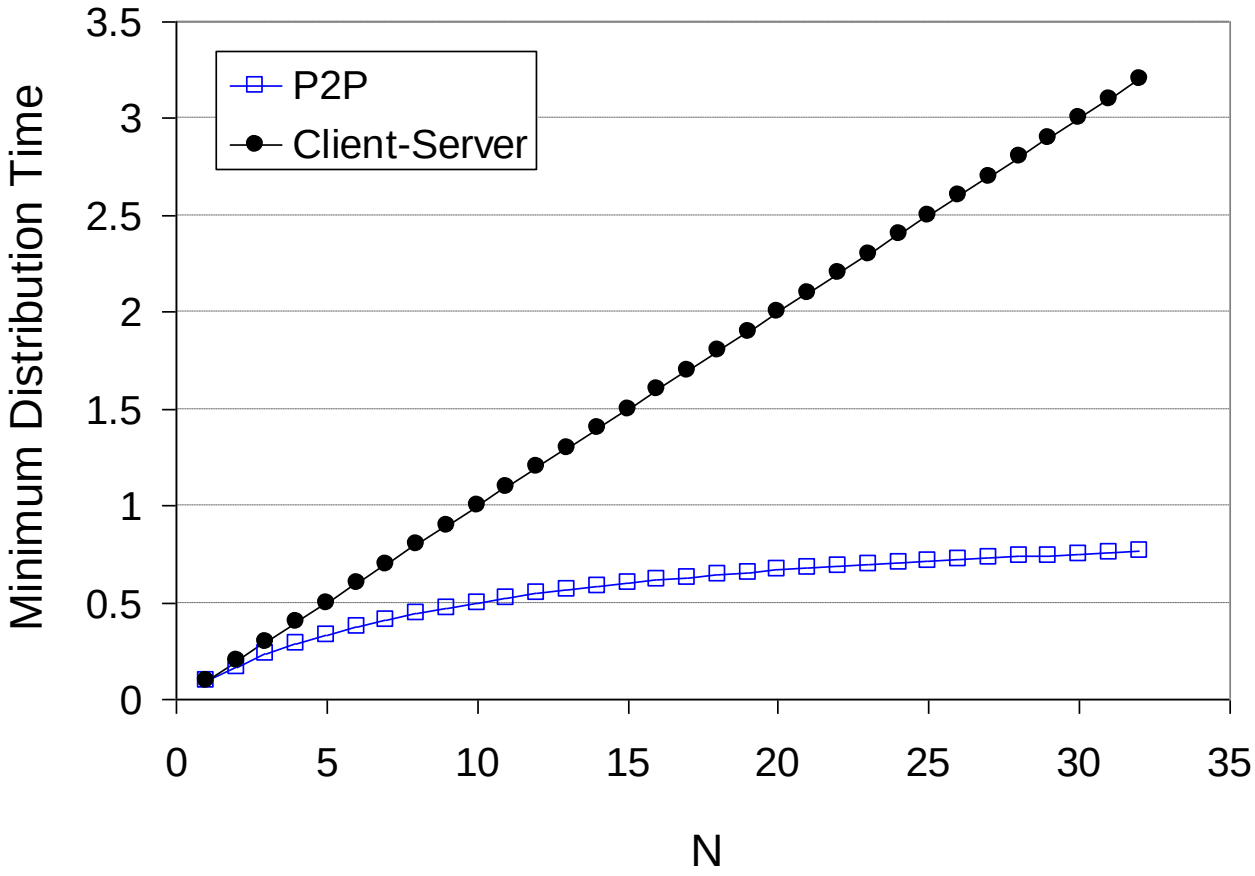
$$D_{P2P} > max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

cresce linearmente com *N* …

… assim como este, cada par traz capacidade de serviço

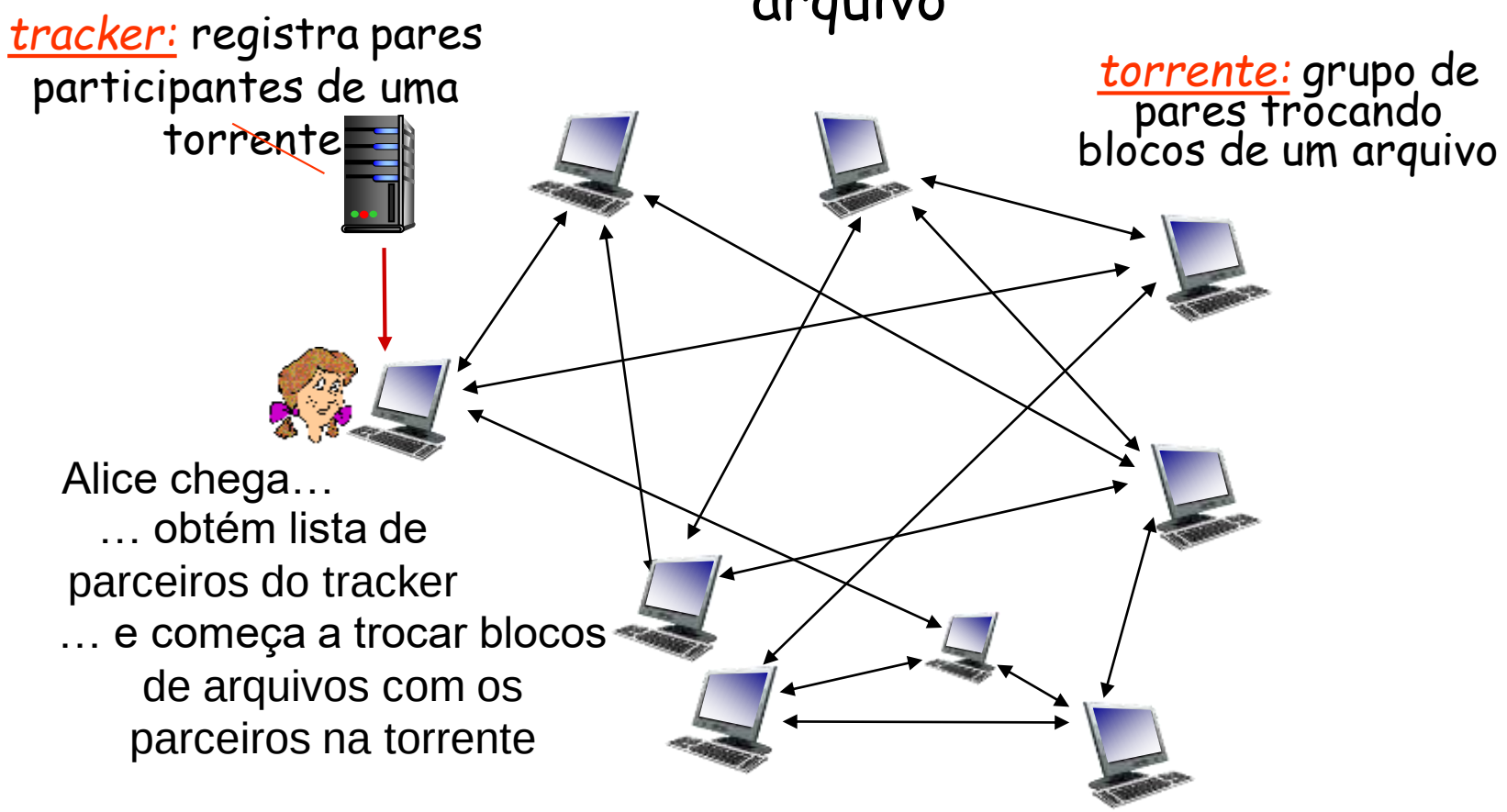
Cliente-servidor x P2P: Exemplo

Taxa de *upload* do cliente= u, F/u = 1 hora, u_s = 10u, $d_{min} \geq u_s$



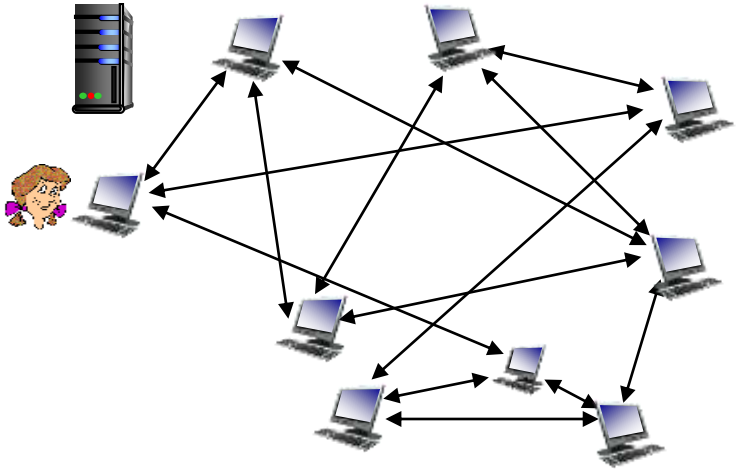
Distribuição de arquivo P2P: BitTorrent

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



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!

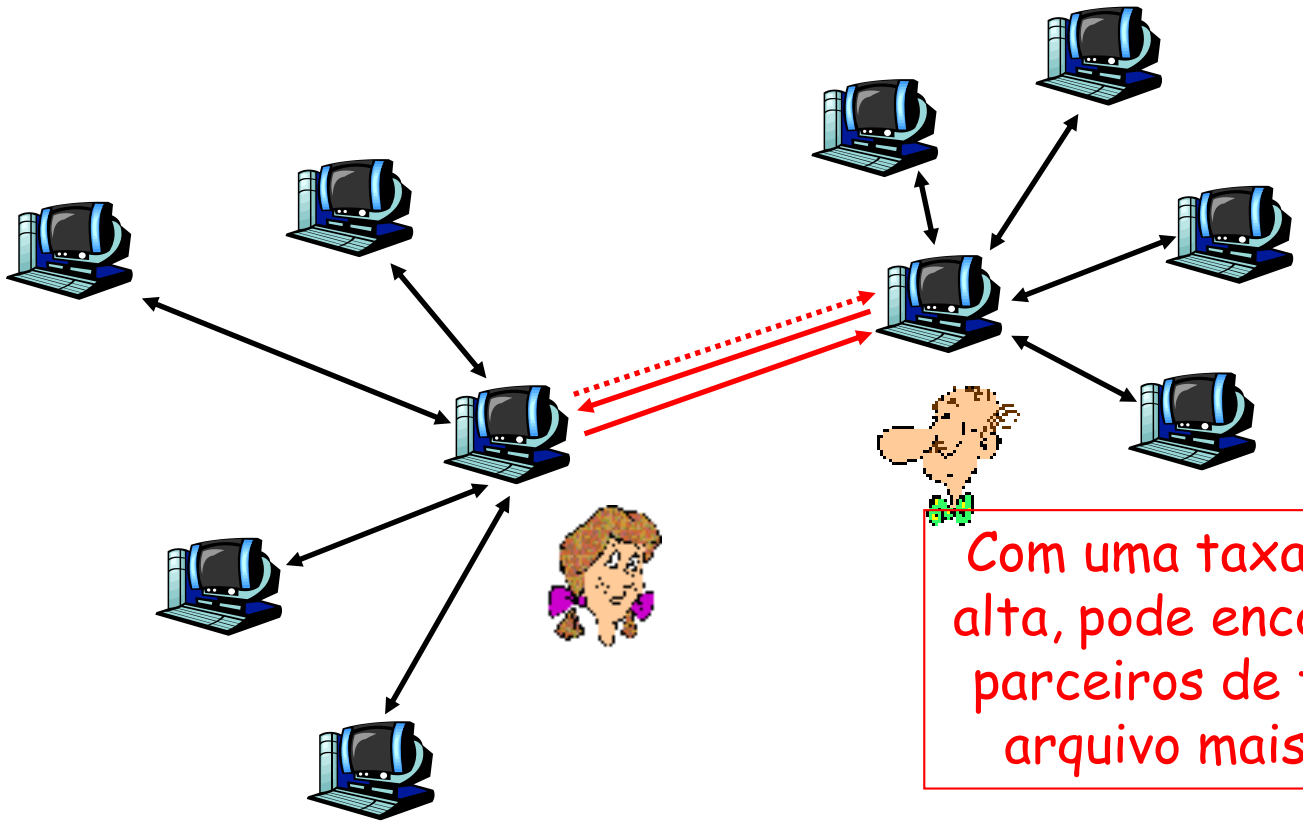
- Alice envia blocos para os quatro vizinhos que estejam lhe enviando blocos *na taxa mais elevada*
  - outros pares foram sufocados por Alice
  - Reavalia os 4 mais a cada 10 segs
- a cada 30 segs: seleciona aleatoriamente outro par, começa a enviar blocos
  - "*optimistically unchoked*"
  - o par recém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

BitTorrent: toma lá, dá cá!

(1) Alice "optimistically unchokes" Bob

(2) Alice se torna um dos quatro melhores provedores de Bob; Bob age da mesma forma

(3) Bob se torna um dos quatro melhores provedores de Alice



Com uma taxa de *upload* mais alta, pode encontrar melhores parceiros de troca e obter o arquivo mais rapidamente!

Distributed Hash Table (DHT)

- DHT: uma base de dados P2P distribuída
- base de dados possui duplas (chave, valor); exemplos:
  - chave: cpf; valor: nome da pessoa
  - chave: título do filme; valor: endereço IP
- Distribui as duplas (chave, valor) entre os milhões de pares
- um par consulta a DHT com a chave
  - a DHT retorna valores que casam com a chave
- pares podem também inserir duplas (chave, valor)

P: como atribuir chaves aos pares?

- questão central:
  - atribuição duplas (chave, valor) aos pares.
- ideia básica:
  - converter cada chave para um inteiro
  - atribuir inteiros para cada par
  - colocar a dupla (chave, valor) no par que esteja mais próximo da chave

Identificadores DHT

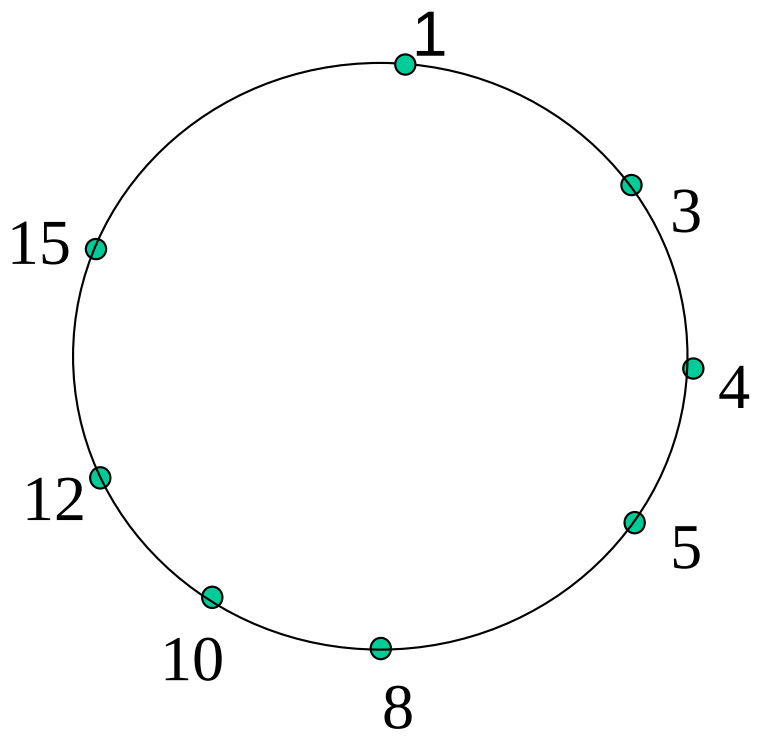
- designa um identificador inteiro a cada par na faixa [0, 2^n-1] de algum n fixo.
  - cada identificador é representado por n bits.

- requer que cada chave seja um inteiro na mesma faixa
- para encontrar a chave inteira, aplica a função de hash à chave original.
  - ex., chave = hash("Led Zeppelin IV")
  - é por isto que é chamada de tabela de "hash" distribuída.

Alocação de chaves aos pares

- regra: atribui a chave ao par que tiver a ID mais próxima.
- convenção de leitura: o mais próximo é o sucessor imediato da chave.
- Ex., n=4; pares: 1,3,4,5,8,10,12,15
  - chave = 13, então par sucessor = 14
  - chave = 15, então par sucessor = 1

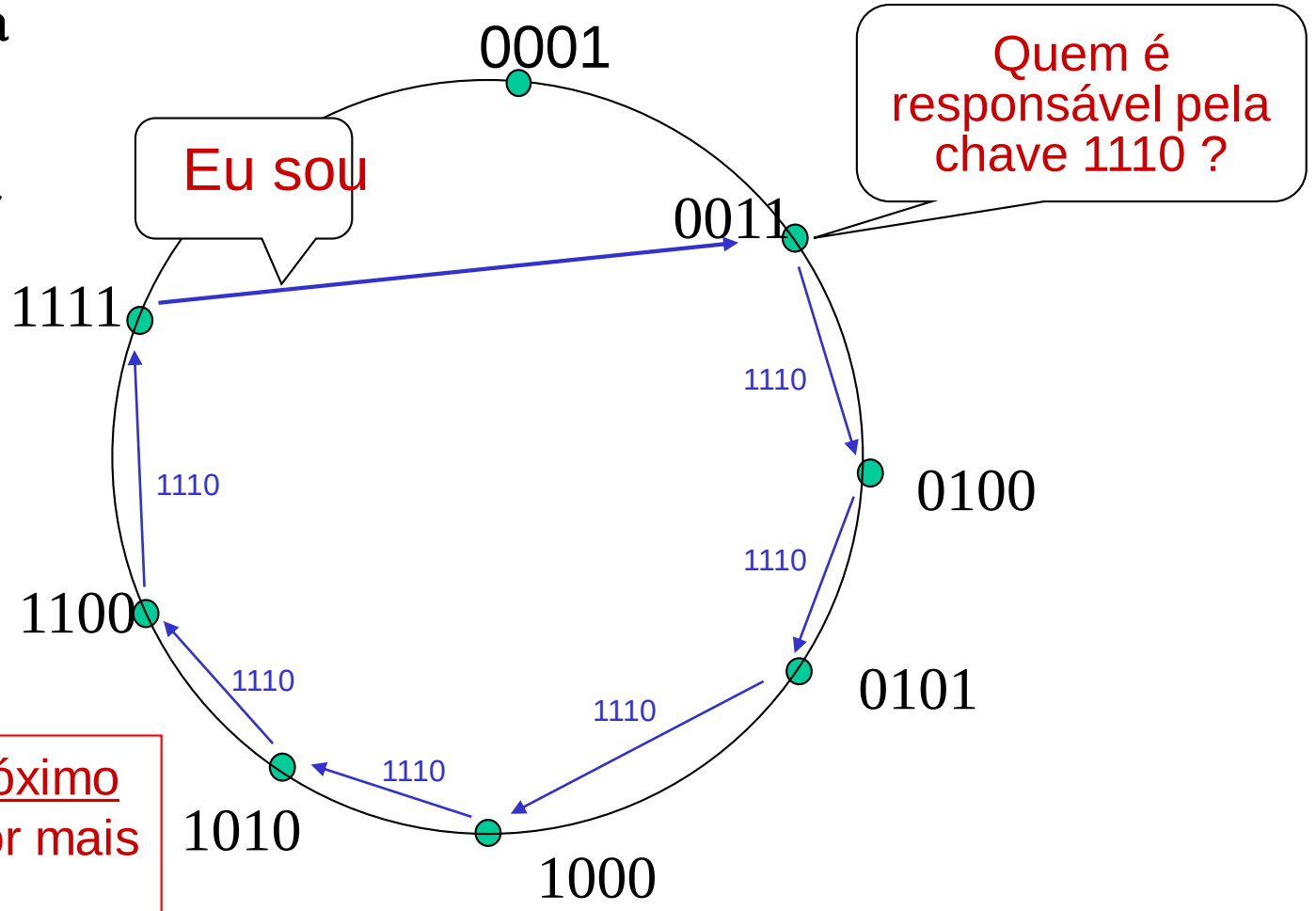
DHT circular (I)



- cada par rastreia apenas o seu sucessor e antecessor imediatos.
- "rede sobreposta (overlay)"

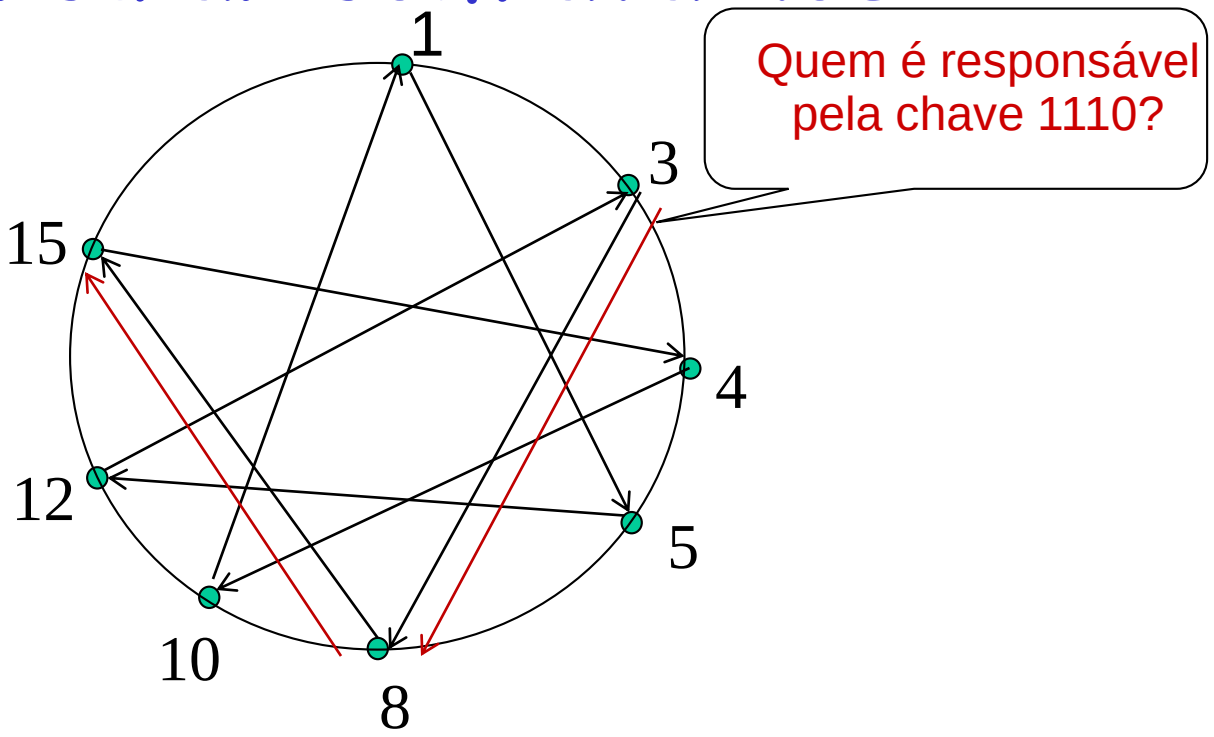
DHT circular (II)

Em média O(N) mensagens para resolver a consulta, quando houver N pares



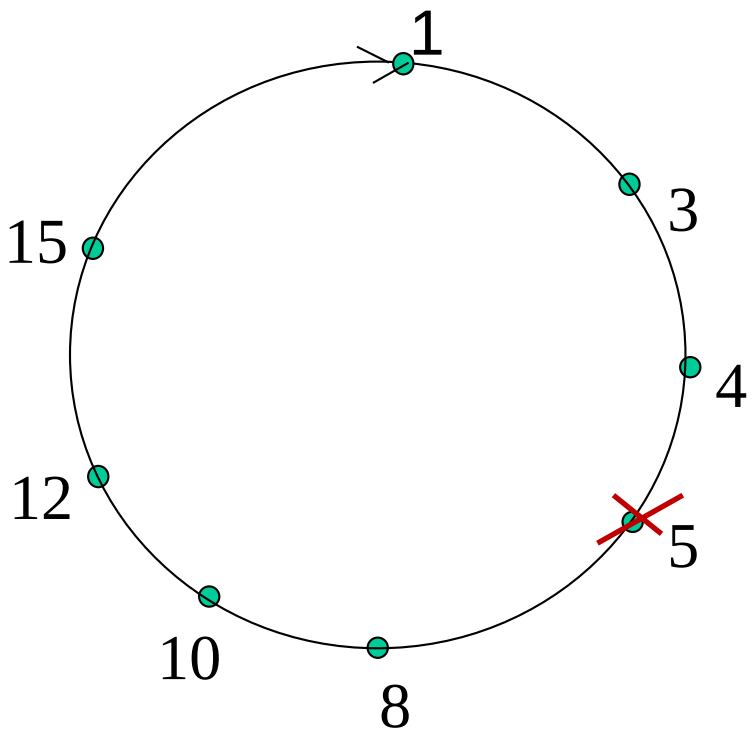
Defina mais próximo como o sucessor mais próximo

DHT circular com atalhos



- cada par rastreia os endereços IP do antecessor, sucessor e atalhos.
- reduz de 6 para 2 mensagens.
- Permite projetar atalhos de modo que para *O(log N)* vizinhos, *O(log N)* mensagens na consulta

Peer churn



tratando peer *churn*:

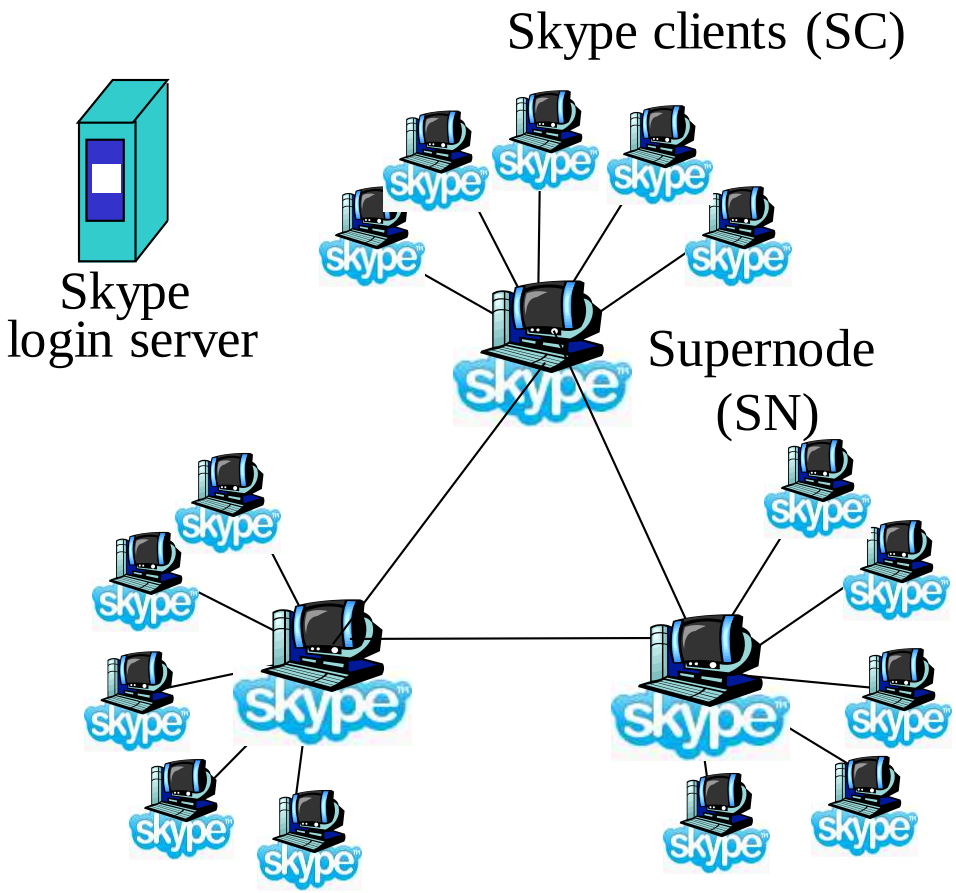
- pares podem chegar e sair (*churn*)
- cada para conhece o endereço dos seus dois sucessores
- cada par periodicamente envia um ping aos seus dois sucessores para verificar se estão vivos
- se o sucessor imediato sair, escolha o próximo sucessor como o sucessor imediato.

exemplo: par 5 sai abruptamente

- par 4 detecta a saída do par 5; torna 8 o seu sucessor imediato; pergunta a 8 quem é o seu sucessor imediato; torna o sucessor imediato de 8 como o seu segundo sucessor.
- o que fazer caso o par 13 resolva entrar?

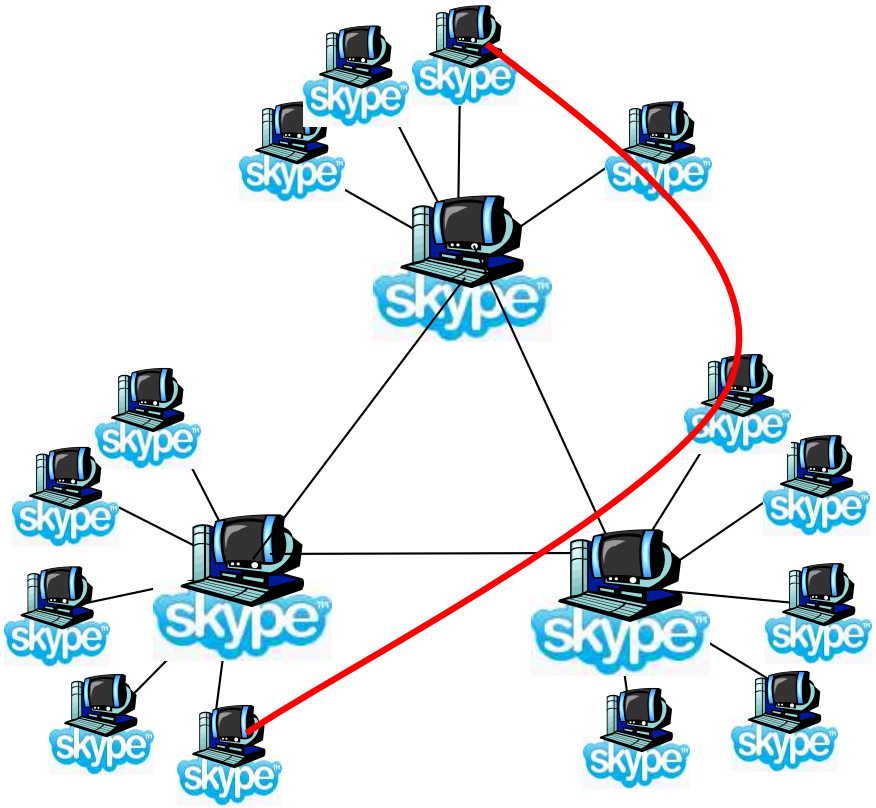
Estudo de caso P2P: Skype

- inerentemente P2P: comunicação entre pares de usuários.
- protocolo proprietário da camada de aplicação (inferido através de engenharia reversa)
- overlay hierárquico com SNs
- Índice mapeia nomes dos usuários a endereços IP; distribuído através dos SNs



Pares como intermediários (*relays*)

- Problema quando tanto Alice como Bob estão atrás de "NATs".
  - O NAT impede que um par externo inicie uma chamada com um par interno
- Solução:
  - Intermediário é escolhido, usando os SNs de Alice e de Bob.
  - Cada par inicia sessão com o intermediário
  - Pares podem se comunicar através de NATs através do intermediário



Programação com sockets

Meta: aprender a construir aplicação cliente/servidor que se comunica usando sockets

API Sockets

- apareceu em BSD4.1 UNIX, 1981
- 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

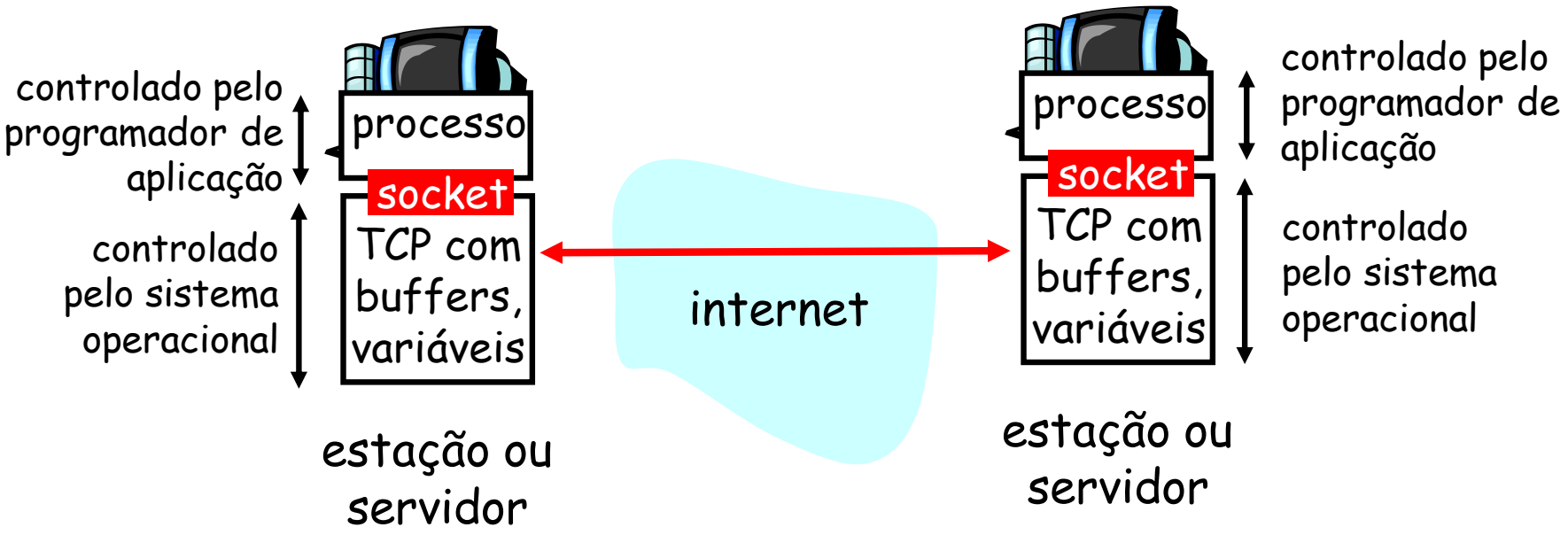
socket

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:

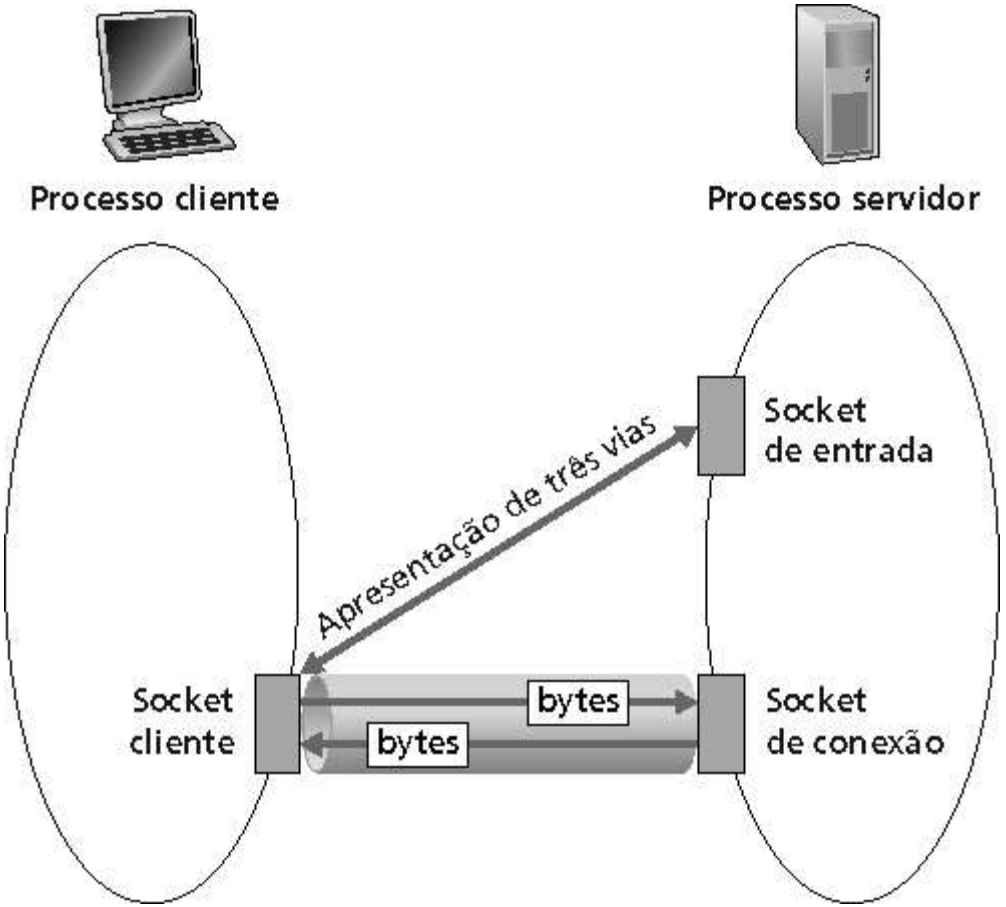
- criar socket TCP local ao cliente
- especificar endereço IP, número de porta do processo servidor
- Quando cliente cria socket: TCP cliente cria conexão com TCP do servidor

- 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



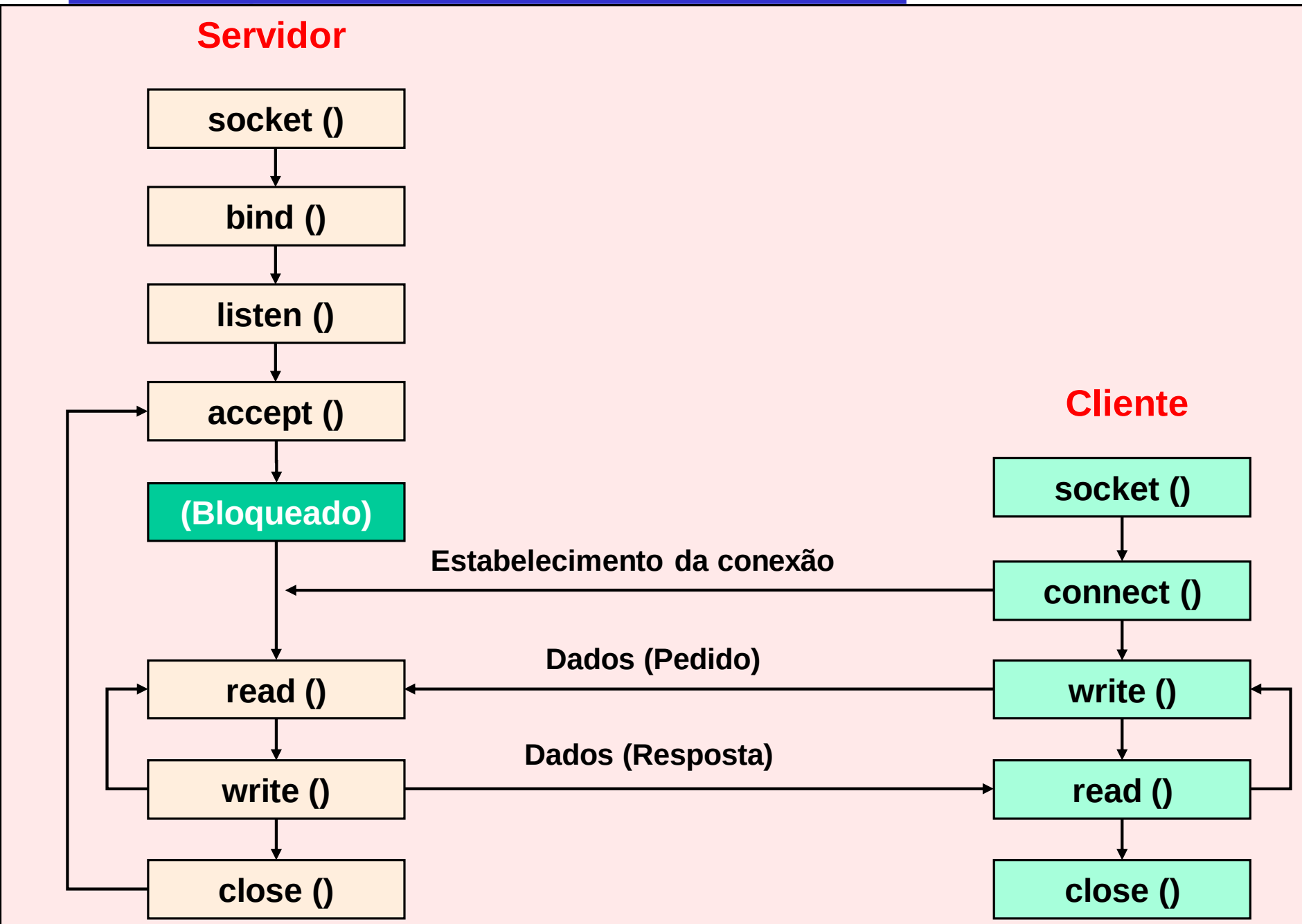
Sockets em C/C++

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

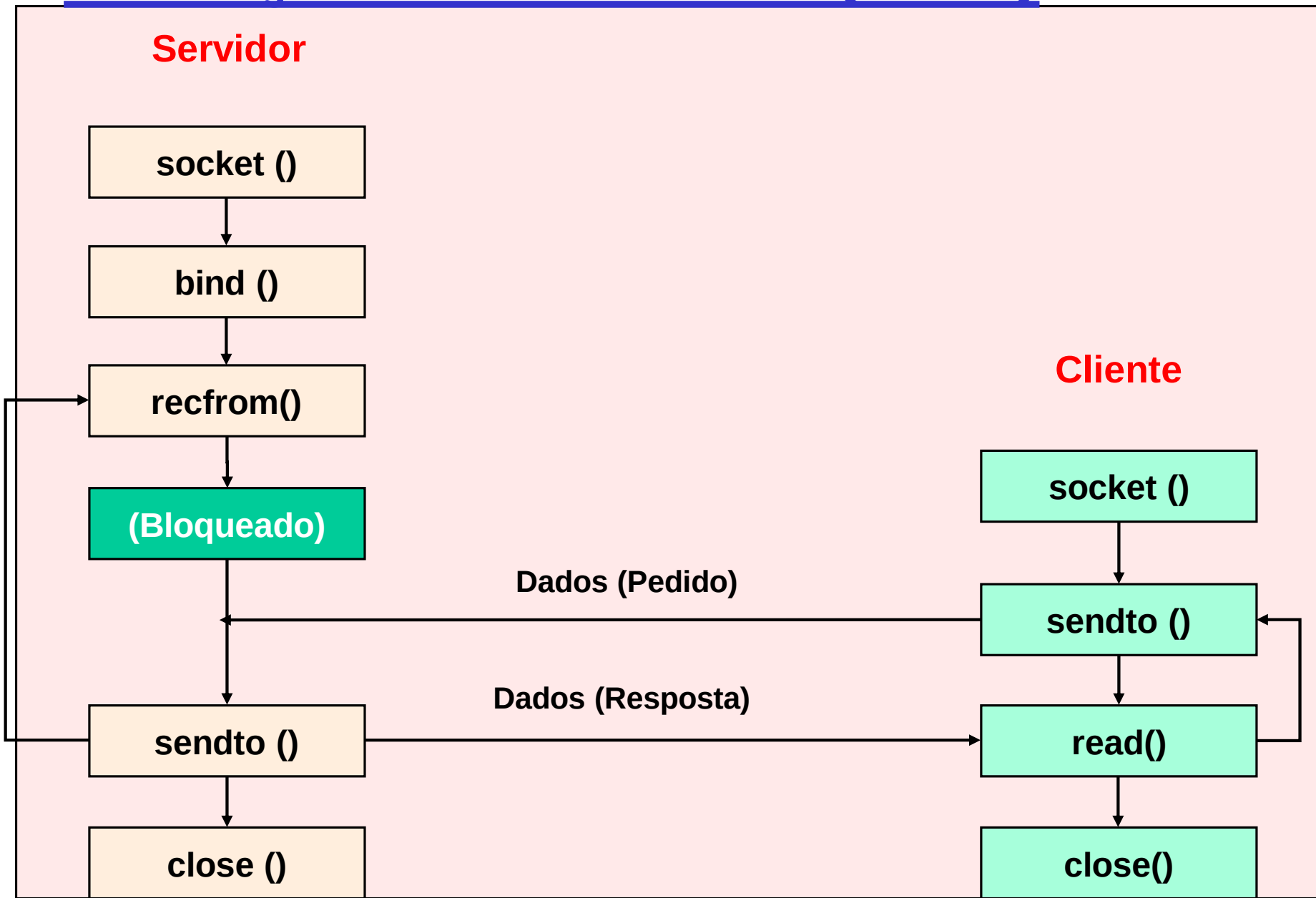
Principais funções da API sockets

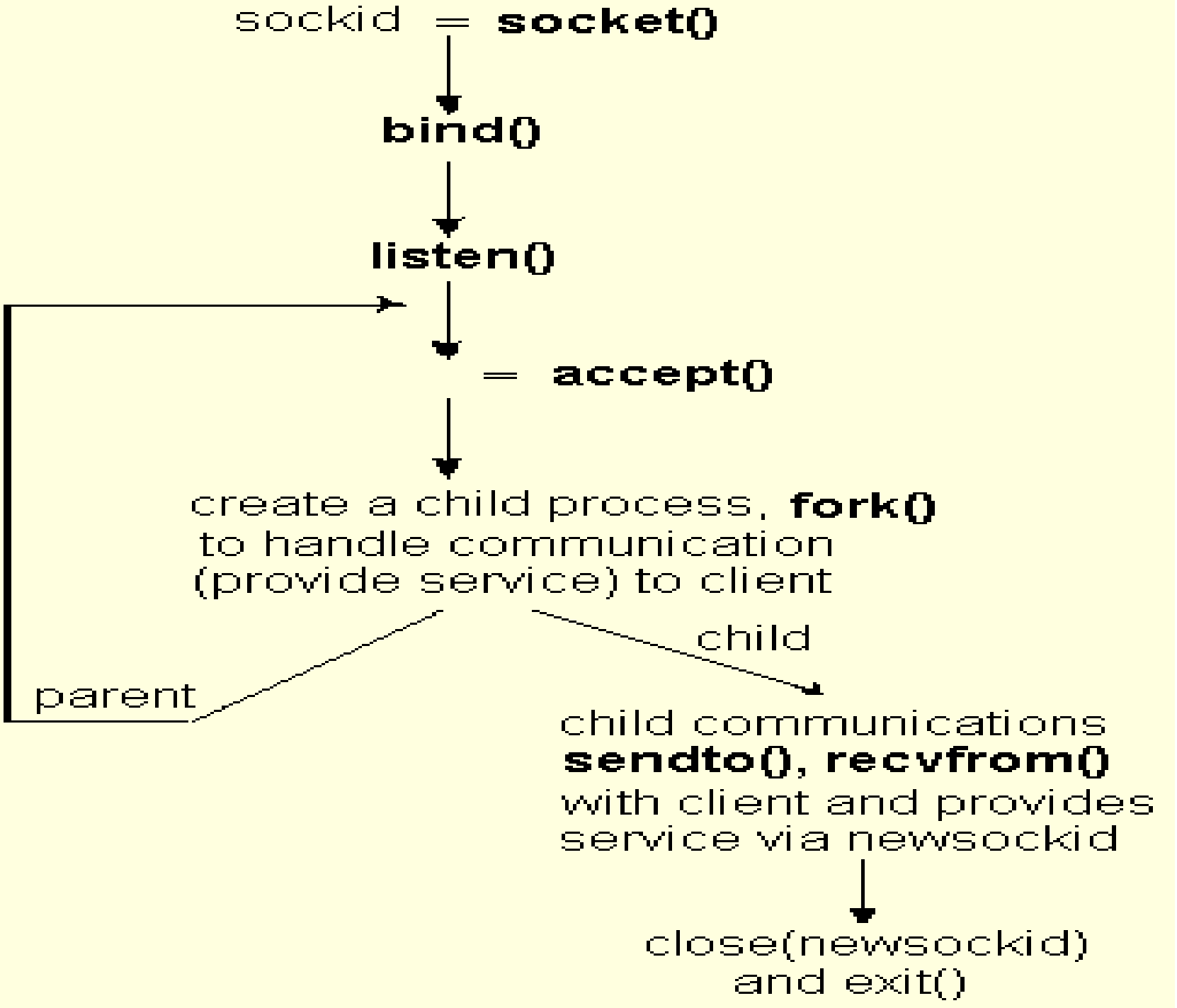
socket	Cria um novo descritor para comunicação
connect	Iniciar conexão com servidor
write	Escreve dados em uma conexão
read	Lê dados de uma conexão
close	Fecha a conexão
bind	Atribui um endereço IP e uma porta a um socket
listen	Coloca o socket em modo passivo, para “escutar” portas
accept	Bloqueia o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

Serviço com Conexão (TCP)



Serviço sem Conexão (UDP)




sockid = socket()
bind()
listen()
= accept()
create a child process, fork()
to handle communication
(provide service) to client
child
parent
child communications
sendto(), recvfrom()
with client and provides
service via newsockid
close(newsockid)
and exit()

CLIENTE

```
#include ...
#include <sys/socket.h>
int main(int argc, char **argv)
{
   int s;
   struct sockaddr_in dest;
   char msg_write[100], msg_read[100];
   s = socket(AF_INET, SOCK_STREAM, 0));
   bzero(&dest, sizeof(dest));
   dest.sin_family = AF_INET;
   dest.sin_port = htons(9999);
   inet_aton("127.0.0.1", &dest.sin_addr.s_addr);
   connect(s, (struct sockaddr*)&dest, sizeof(dest));
   do {
       scanf("%s",msg_write);
       write (s, msg_write, strlen(msg_write)+1);
       read (s, msg_read, MAXBUF);
   } while (strcmp(msg_read,"bye"));
    close(s);
}
```

Com conexão

SERVIDOR

Com conexão

```
#include ...
#include <sys/socket.h>
int main(int argc, char **argv)
{  int s, client_s;
   struct sockaddr_in self, client;
   int addrlen = sizeof(client);
   char msg_write[100], msg_read[100];

   s = socket(AF_INET, SOCK_STREAM, 0);
   bzero(&self, sizeof(self));
   self.sin_family = AF_INET;
   self.sin_port = htons(9999);
   self.sin_addr.s_addr = INADDR_ANY;

   bind(s, (struct sockaddr*)&self, sizeof(self));
   listen(s, 5);
   while (1) {
        client_s = accept(s, (struct sockaddr*)&client, &addrlen);
        do {
               read (client_s, msg_read, MAXBUF);
               write (client_s, msg_read, strlen(msg_read)+1);
        } while (strcmp(msg_read,"bye"));
        close(client_s);
   }
}
```

Sockets sem Conexão (C)

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

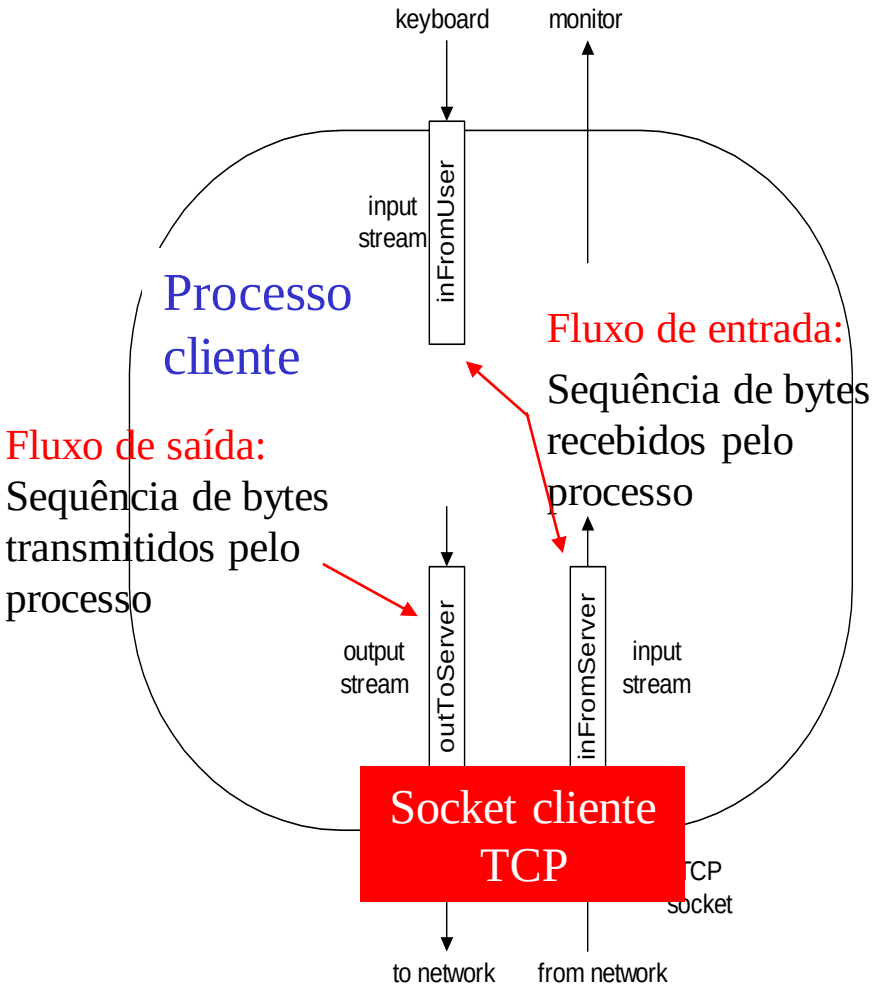
Jargão para Fluxo (*Stream*)

- Um fluxo (*stream*) é uma sequência 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*.

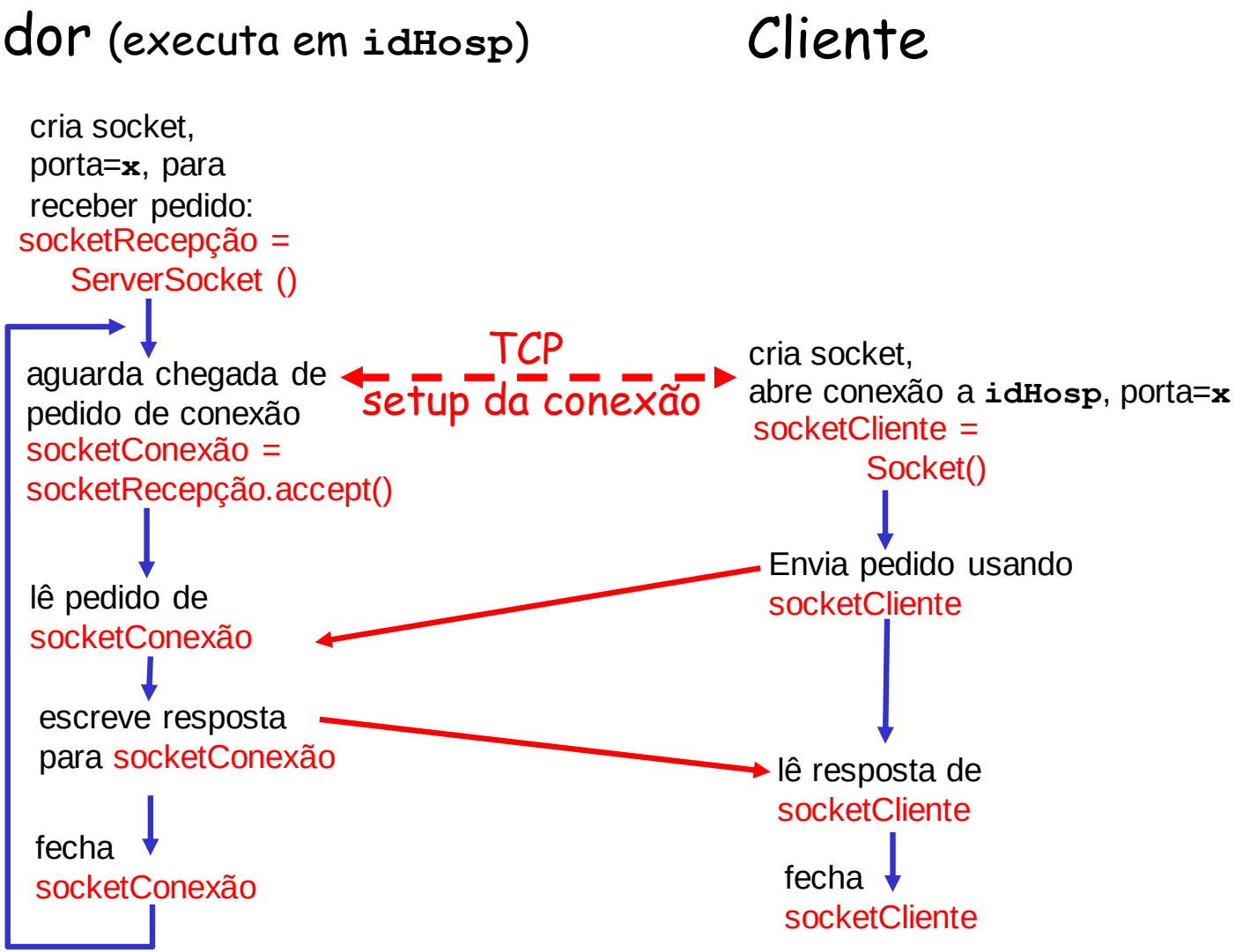
Programação com sockets usando TCP

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 com socket: TCP



Exemplo: cliente Java (TCP)

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

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

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

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

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

Cria fluxo de entrada

Cria socket de cliente, conexão ao servidor

Create output stream attached to 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;
     StringfFraseMaiusculas;

     ServerSocket socketRecepcao = new ServerSocket(6789);

     while(true) {

       Socket socketConexao = socketRecepcao.accept();

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

Cria socket para recepção na porta 6789

Aguarda, no socket para recepção, o contato do cliente

Cria fluxo de entrada, ligado ao socket

Exemplo: servidor Java (TCP), cont

Cria fluxo de saída, ligado ao socket

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

Lê linha do socket

```
fraseCliente= doCliente.readLine();

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

Escreve linha ao socket

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

Final do elo 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)

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)

```
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()
```

Programação com sockets usando UDP

UDP: não tem "conexão" entre cliente e servidor

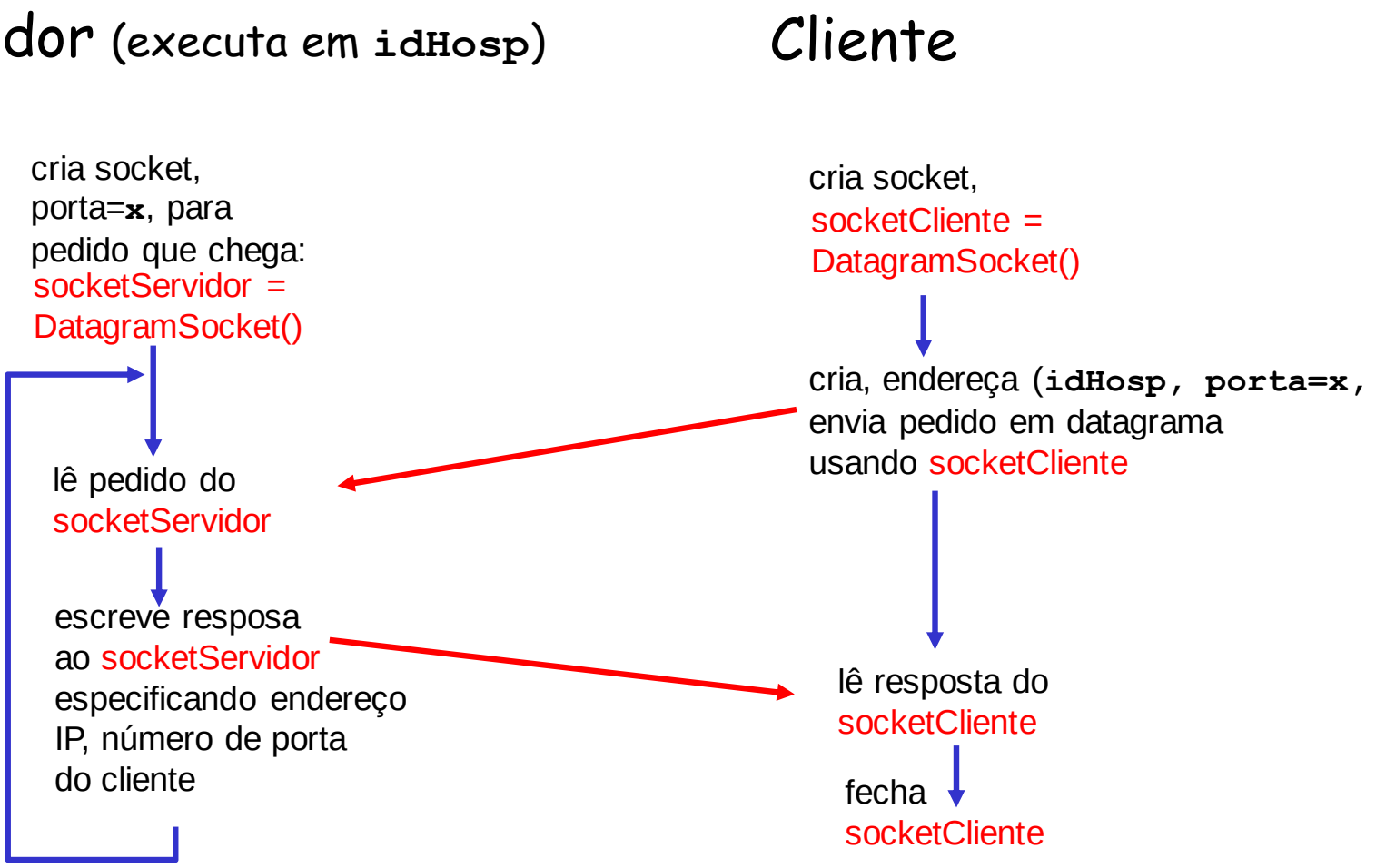
- não tem "handshaking"
- remetente coloca explicitamente endereço IP e porta do destino
- servidor deve extrair endereço IP, porta do remetente do datagrama recebido

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

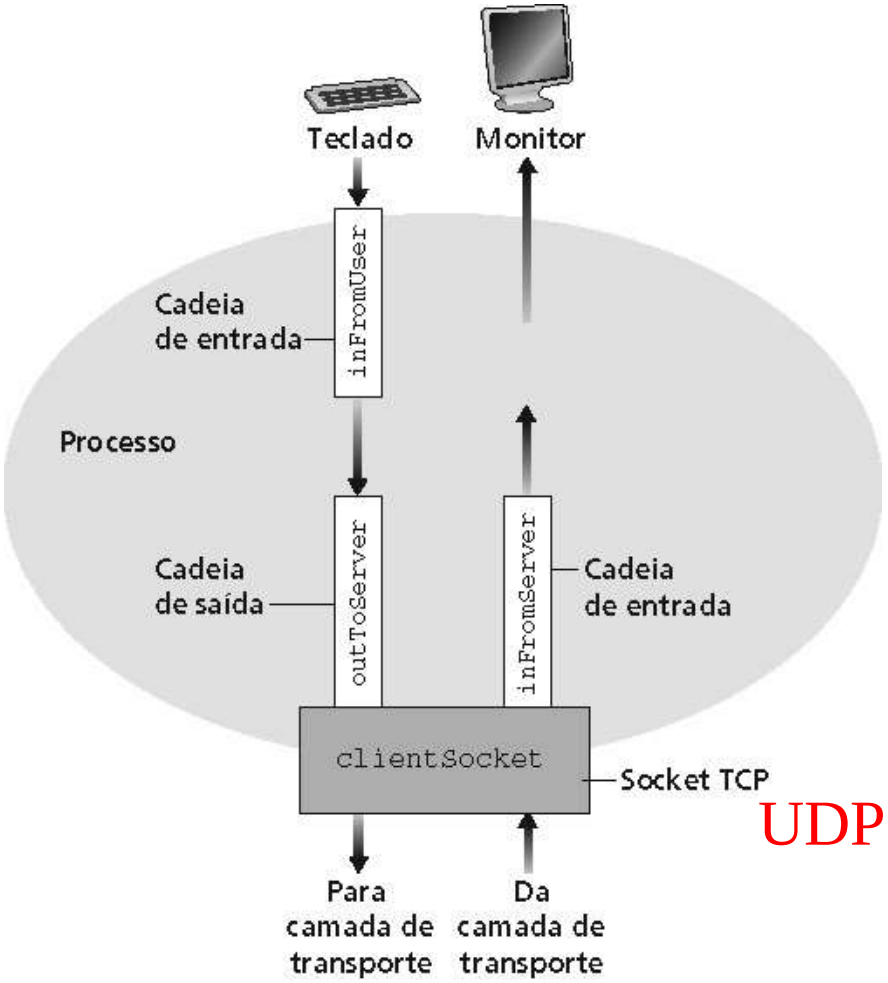
ponto de vista da aplicação

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

Interações cliente/servidor com socket: UDP



Exemplo: Cliente Java (UDP)



Exemplo: cliente Java (UDP)

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

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

      DatagramSocket socketCliente = new DatagramSocket();

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

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

      String frase = doUsuario.readLine();

      dadosEnvio = frase.getBytes();
```

Cria fluxo de enrada

Cria socket de cliente

Traduz nome de hospedeiro ao endereço IP usando DNS

Exemplo: cliente Java (UDP) cont.

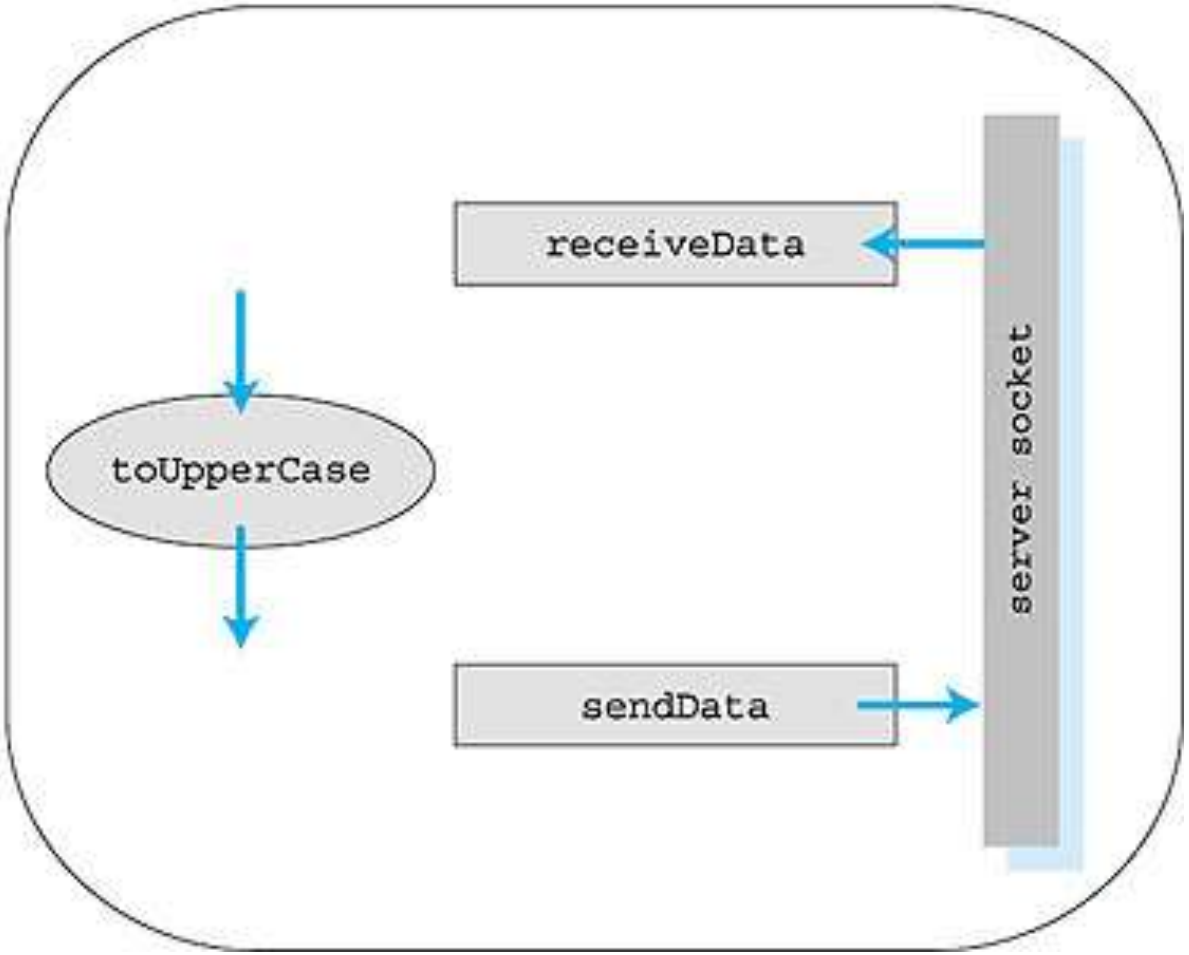
Cria datagrama com dados para enviar, comprimento, endereço IP, porta

Envia datagrama ao servidor

Lê datagrama do servidor

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

                socketCliente.send(pacoteEnviado);

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

                socketCliente.receive(pacoteRecebido);

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

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

Servidor UDP



Exemplo: servidor Java (UDP)

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

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

      DatagramSocket socketServidor = new DatagramSocket(9876);

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

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

          socketServidor.receive(pacoteRecebido);
```

Cria socket para datagramas na porta 9876

Aloca memória para receber datagrama

Recebe datagrama

Exemplo: servidor Java (UDP), cont

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

InetAddress IPAddress = pacoteRecebido.getAddress();

int port = 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 ao socket

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

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()
```

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