

# Redes de Computadores

Sockets, FTP, Correio Eletrônico, P2P

Prof. Jó Ueyama

Março/2017

# Capítulo 2.7 e 2.8 - Camada de Aplicação

## Programação de sockets

# Programação de Sockets

## Objetivo:

- aprender a construir aplicações cliente-servidor que se comunicam usando sockets.

## Socket:

- interface local, criada por aplicações, controlada pelo OS (uma “porta”) na qual os processos de aplicação podem tanto enviar quanto receber mensagens de e para outro processo de aplicação (local ou remoto).

# API Socket

- Introduzida no BSD4.1 UNIX, 1981.

∀ Sockets são explicitamente criados, usados e liberados pelas aplicações.

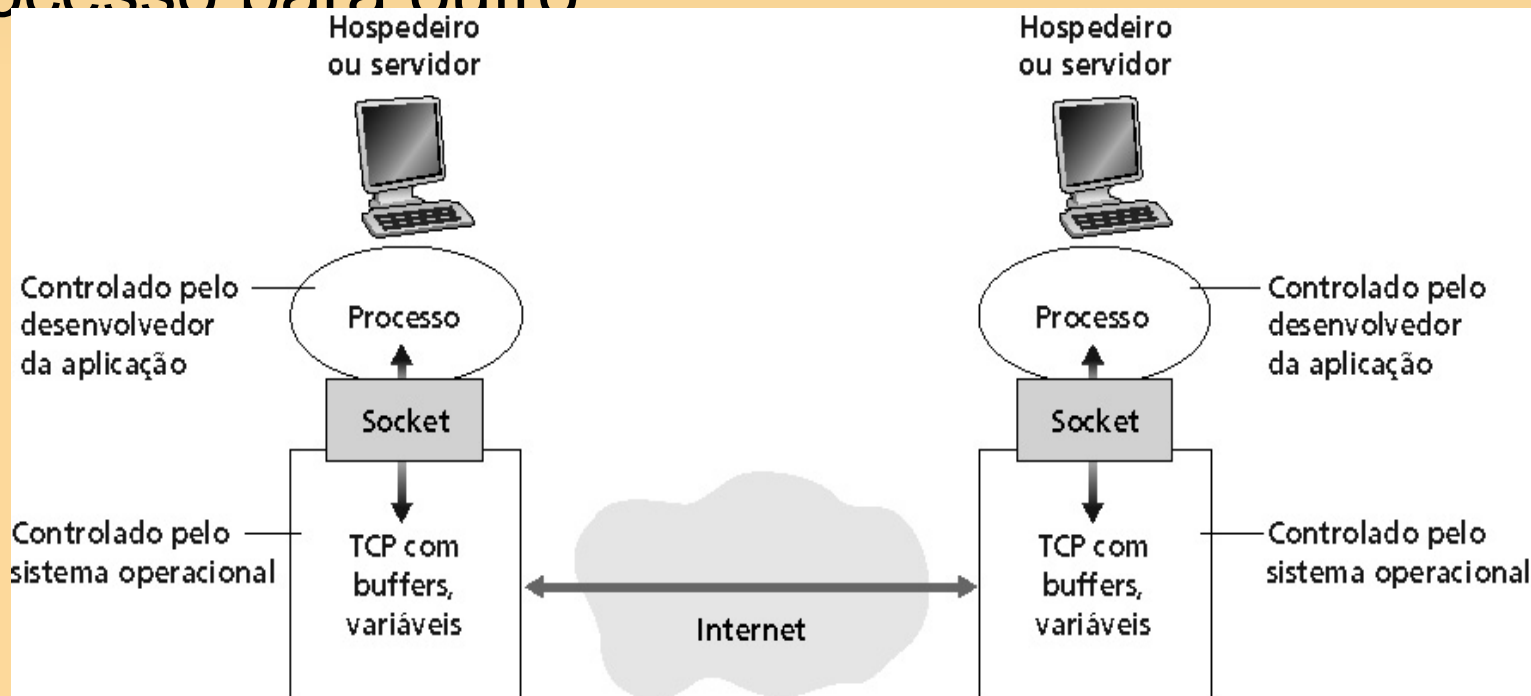
∀ Implementam paradigma cliente-servidor.

∀ Dois tipos de serviço de transporte via socket API:

- datagrama não confiável (UDP);
- confiável, orientado a cadeias de bytes (TCP).

# Programação de sockets com TCP

- Socket: uma porta entre o processo de aplicação e o protocolo de transporte fim-a-fim.
- Serviço TCP: transferência confiável de bytes de um processo para outro



# Sockets TCP - Cliente

- Processo servidor já deve estar em execução.
- Servidor deve ter criado socket (porta) que aceita o contato do cliente.
- Cliente contata o servidor:
  - criando um socket TCP local;
  - especificando endereço IP e número da porta do processo servidor.
- Quando o cliente cria o socket:
  - cliente TCP estabelece conexão com o TCP do servidor.

# Sockets TCP - Servidor

- Quando contatado pelo cliente, o servidor cria um novo socket para o processo servidor comunicar-se com o cliente.
- Permite ao servidor conversar com múltiplos clientes
- Números da porta de origem são usados para distinguir o cliente (mais no Capítulo 3).

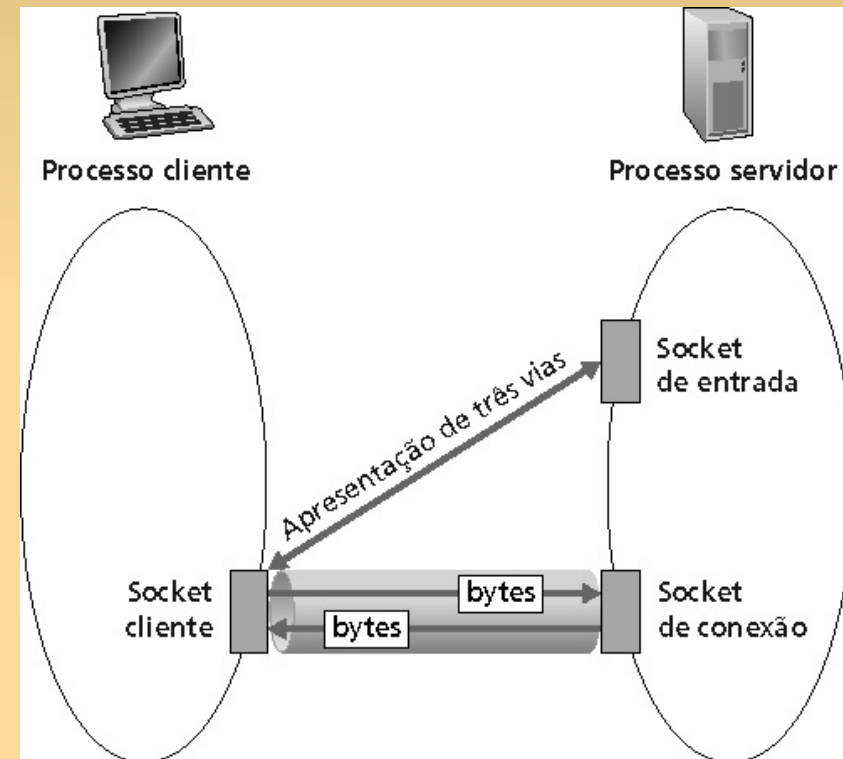
# Terminologia: stream

- Um stream é uma seqüência de caracteres que fluem para dentro ou para fora de um processo.
- Um stream de entrada é agregado a alguma fonte de entrada para o processo, ex.: teclado ou socket.
- Um stream de saída é agregado a uma fonte de saída, ex.: monitor ou socket.

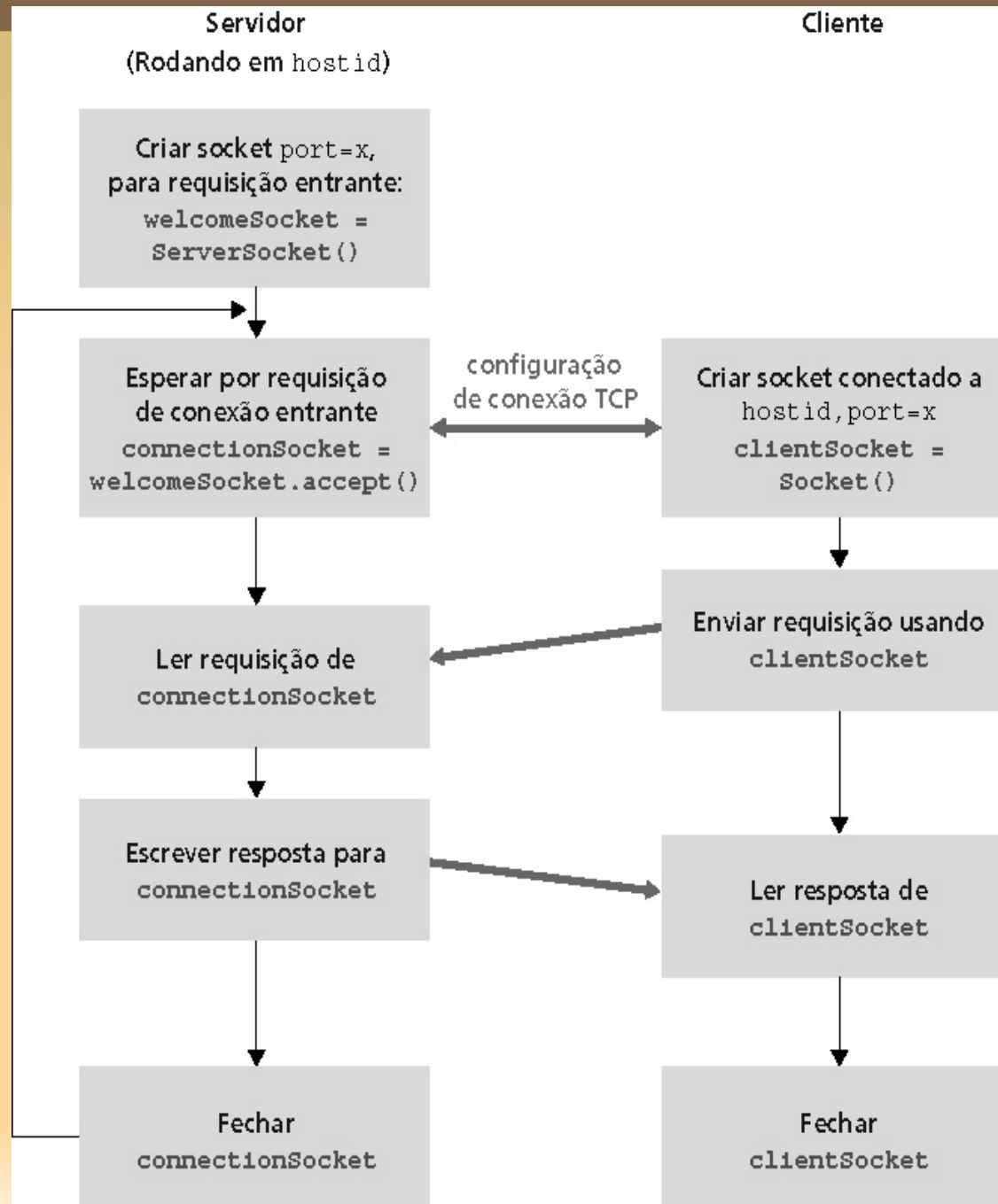


# Exemplo de aplicação cliente-servidor TCP

- 1) Cliente lê linha da entrada-padrão do sistema (**inFromUser** stream), envia para o servidor via socket (**outToServer** stream).
- 2) Servidor lê linha do socket.
- 3) Servidor converte linha para letras maiúsculas e envia de volta ao cliente.
- 4) Cliente lê a linha modificada através do (**inFromServer** stream).



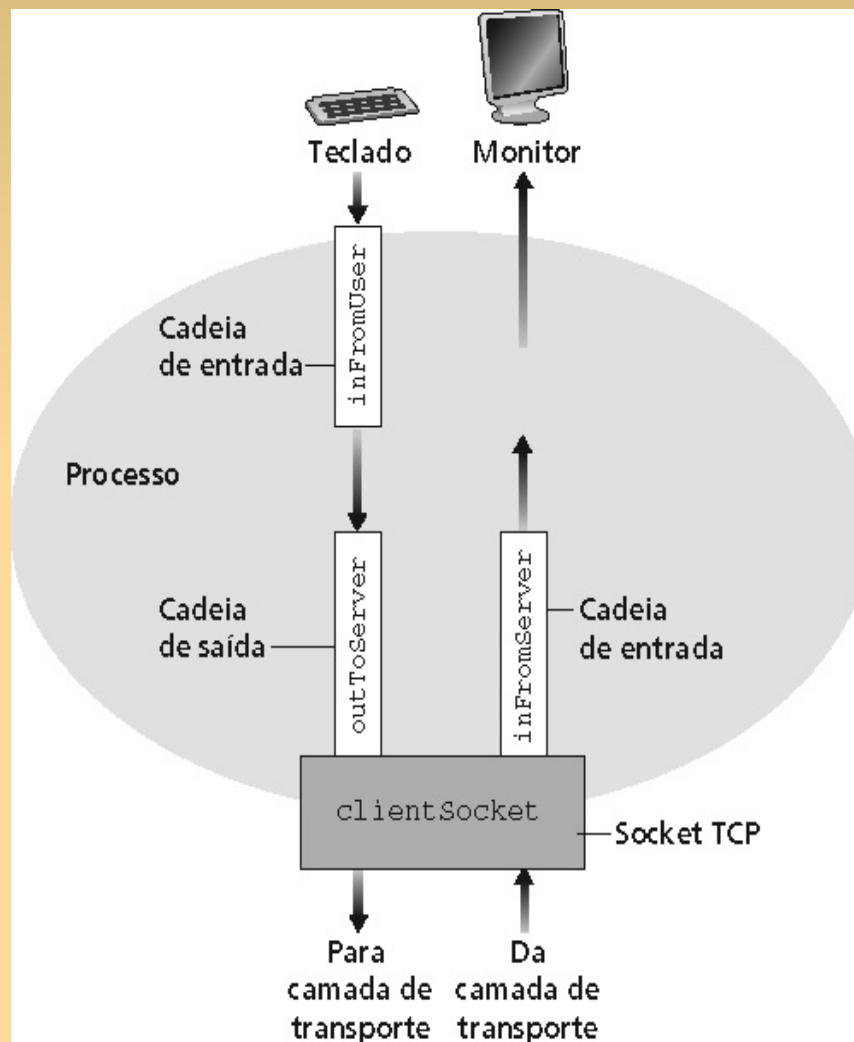
# Interação cliente-servidor TCP



# Programação de sockets com UDP

- Não há conexão entre o cliente e o servidor.
- Transmissor envia explicitamente endereço IP e porta de destino em cada mensagem.
- Servidor deve extrair o endereço IP e porta do transmissor de cada datagrama recebido.
- Dados transmitidos podem ser recebidos fora de ordem ou perdidos.

# Exemplo de aplicação cliente-servidor UDP



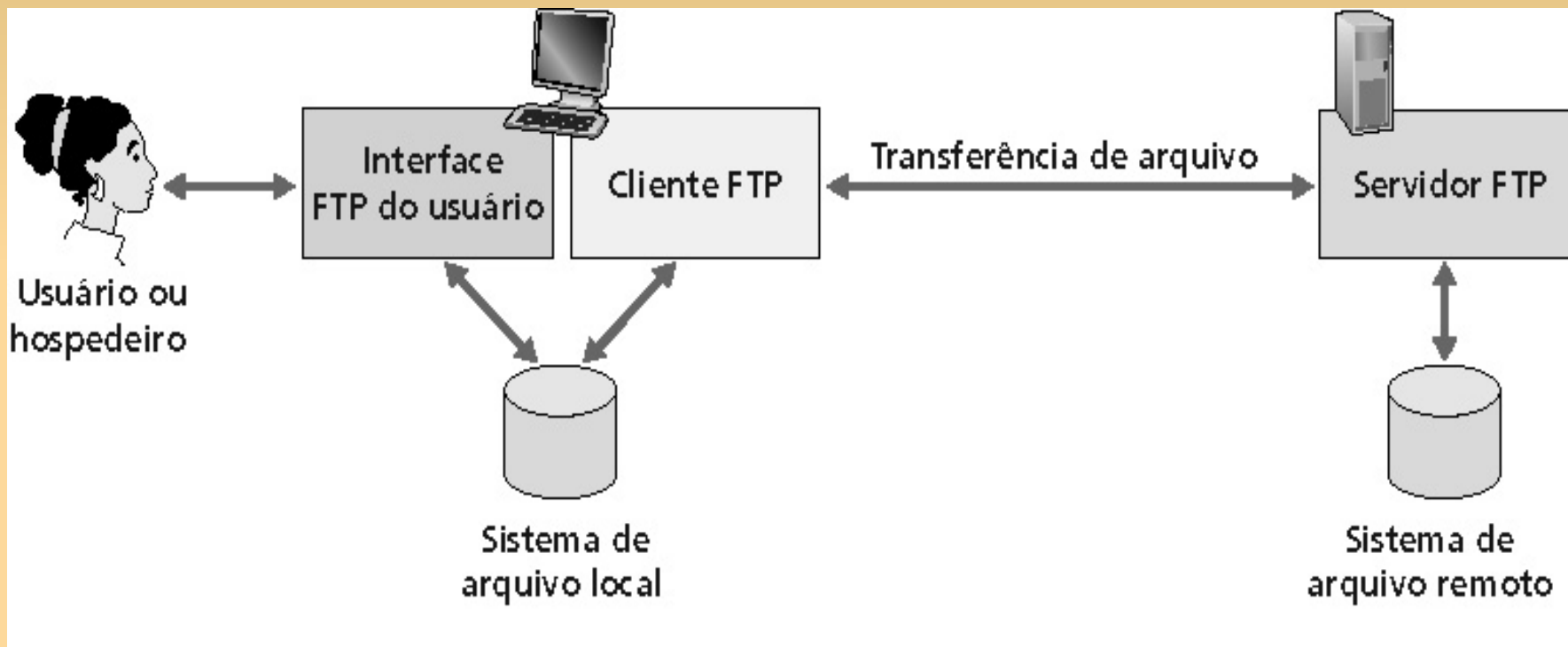
# EP: Implementação de um servidor Web

# Capítulo 2.3 - FTP

# FTP (*File Transfer Protocol*)

- Transferência de arquivos de e para o computador remoto.
- Modelo cliente servidor:
  - Cliente: lado que inicia a transferência (seja de ou para o lado remoto)
  - Servidor: hospedeiro remoto
- RFC 959.
- Servidor FTP: porta 21.

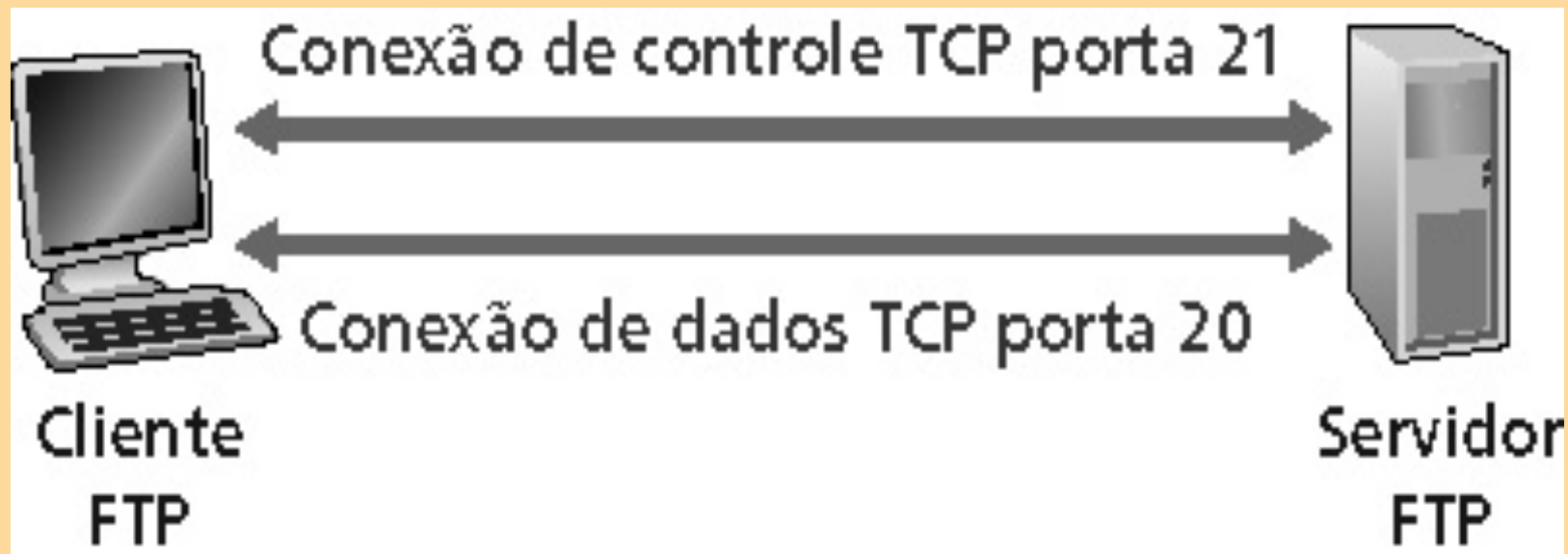
# FTP





# FTP: conexão de controle e de dados

- Duas conexões:
  - controle;
  - dados.
- Conexão de controle: “fora da banda”.



# FTP: conexão de controle e de dados

∀ Cliente FTP contata o servidor FTP na porta 21.

- TCP como protocolo de transporte;
  - Por que?
- estabelece conexão de controle.

∀ Conexão de Controle:

- cliente obtém autorização;
- cliente procura o diretório remoto;
- cliente envia comando para transferência de arquivo.

# FTP: conexão de controle e de dados

## ∀ Conexão de Dados:

- servidor abre essa conexão TCP quando recebe um comando para transferência de arquivo;
- após a transferência de um arquivo, o servidor fecha a conexão.
- servidor abre uma segunda conexão de dados TCP para transferir outro arquivo;

∀ Servidor FTP mantém “estado”: diretório atual, autenticação anterior.

# FTP: Comandos

∀ texto ASCII sobre canal de controle.

∀ USER *username*

∀ PASS *password*

∀ LIST retorna listagem do arquivo no diretório atual.

∀ RETR filename recupera (obtém) o arquivo.

∀ STOR filename armazena o arquivo no hospedeiro remoto.

# FTP: Códigos de retorno

∀ Código de status e frase (como no HTTP)

∀ 331 Username OK, password required

∀ 125 data connection already open; transfer starting

∀ 425 Can't open data connection

∀ 452 Error writing file

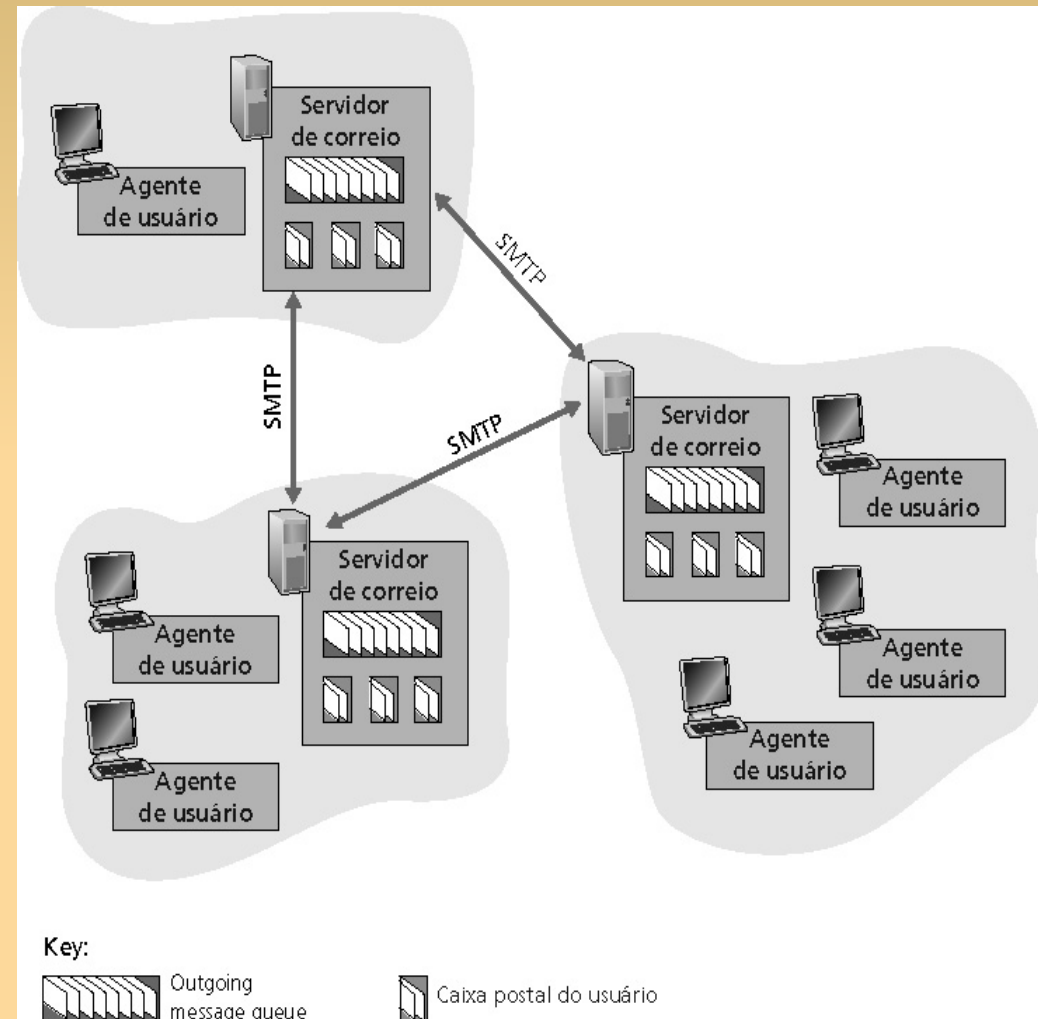
# Mas hoje....

- FTP não é mais tão usado...
  - senhas passam em aberto na rede!
- SCP:
  - somente transferência de arquivos.
- SFTP (SSH File Transfer Protocol):
  - permite transferência e manipulação de arquivos;
  - normalmente usa SSH-2;
  - autenticação e segurança provida pelo protocolo abaixo (SSH).

# Capítulo 2.4 - Correio Eletrônico

# Correio Eletrônico

- Três componentes principais:
  - Agentes de usuário
  - Servidores de correio
  - SMTP (Simple Mail Transfer Protocol)





# Correio eletrônico: agentes de usuário

∀ Composição, edição, leitura de mensagens de correio.

∀ Ex.: Outlook, Thunderbird, Eudora, pine.

∀ Mensagens de entrada e de saída são armazenadas no servidor.

– alguns programas copiam as mensagens recebidas para o disco local (configuração).

# Correio eletrônico: servidores de correio

∀ Caixa postal contém mensagens que chegaram (ainda não lidas) para o usuário.

∀ Fila de mensagens contém as mensagens de correio a serem enviadas.

∀ Protocolo SMTP permite aos servidores de correio trocarem mensagens entre si:

- cliente: servidor de correio que envia;
- “servidor”: servidor de correio que recebe.

# SMTP [RFC 821]

∀RFC é de 1982.

∀Usa TCP.

∀Servidor ouve na porta 25.

∀Conexões persistentes.

∀Transferência direta:

– servidor que envia para o servidor que recebe.

∀Agentes de usuário usam o SMTP para enviar mensagens para o seu servidor.

# SMTP [RFC 821]

∀ Três fases de transferência:

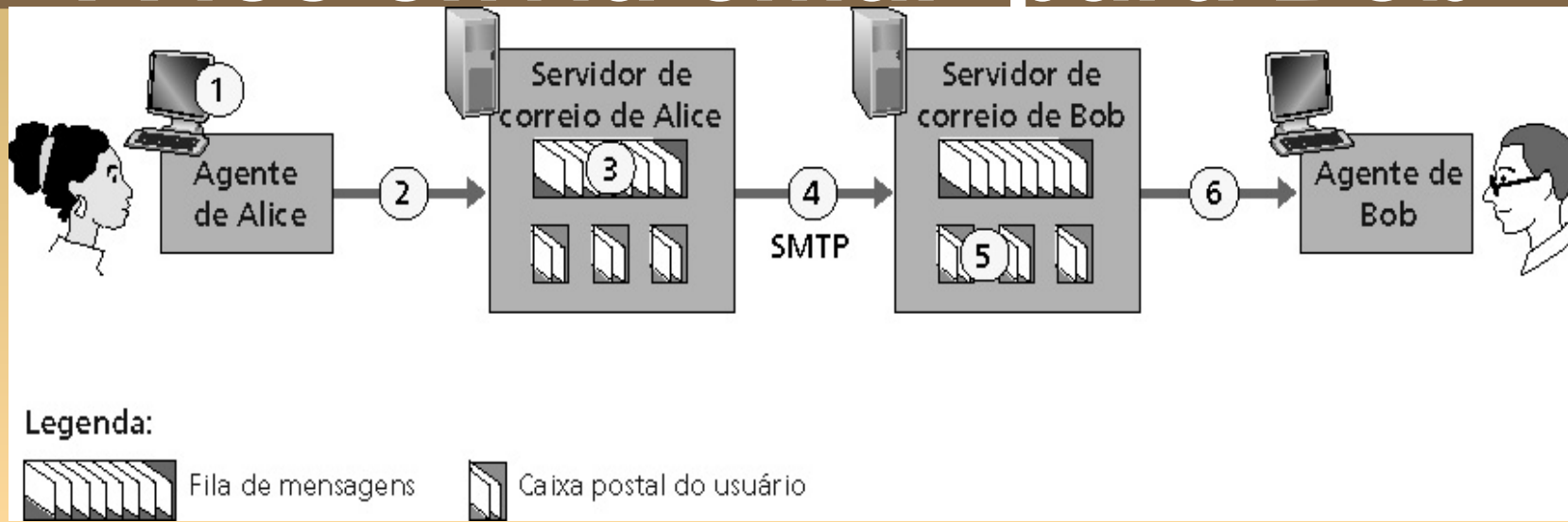
- handshaking (apresentação);
- transferência de mensagens;
- fechamento.

∀ Interação comando/resposta:

- Comandos: texto ASCII;
- Resposta: código de *status* e frase.

∀ Mensagens (cabeçalho e corpo) devem ser formatadas em código ASCII de 7 bits.

# Alice envia email para Bob



- 1) Alice usa o agente de usuário 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.

# Exemplo de iteração SMTP

```
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
```

# Tente você...

telnet nome-do-servidor 25

∀ Veja resposta 220 do servidor.

∀ Envie comandos HELO, MAIL FROM, RCPT TO, DATA, QUIT.

∀ A sequência acima permite enviar um comando sem usar o agente de usuário do remetente.

# SMTP: comparação com HTTP

∀HTTP: protocolo de recuperação de informações (*pull protocol*).

∀SMTP: protocolo de envio de informações (*push protocol*).

∀Ambos usam comandos e respostas em ASCII, interação comando/resposta e códigos de *status*.

∀HTTP: cada objeto encapsulado na sua própria mensagem de resposta.

∀SMTP: múltiplos objetos são enviados numa única mensagem.

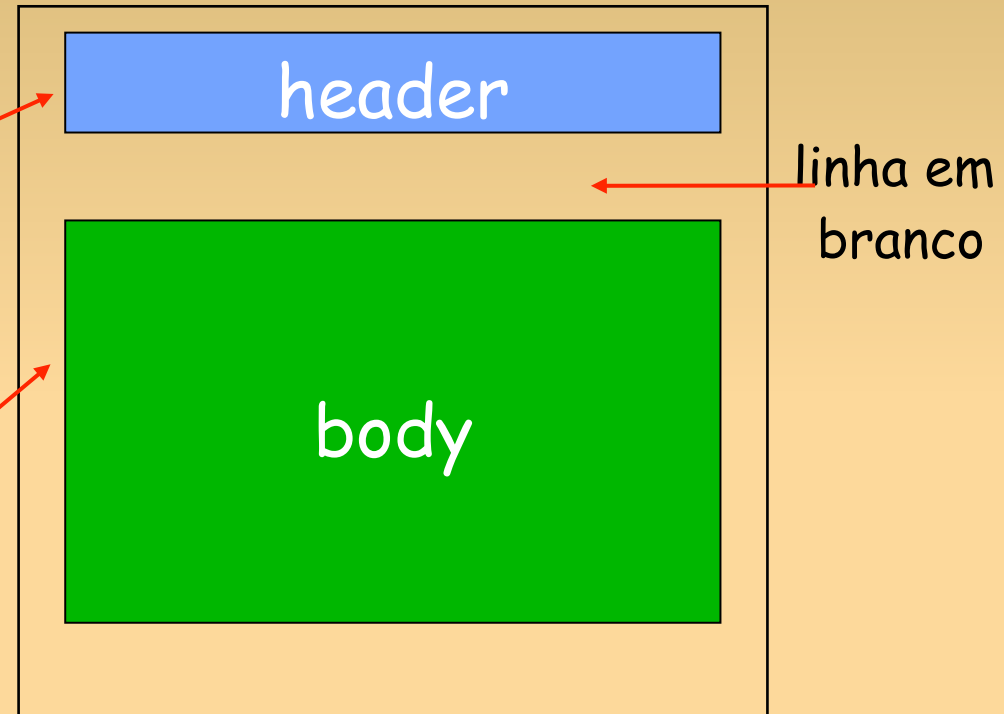


# Formato da Mensagem

RFC 822: padrão para mensagens do tipo texto:

- linhas de cabeçalho:
  - To:
  - From:
  - Subject:

**diferente** dos comandos SMTP!
- corpo:
  - a “mensagem”, ASCII somente com caracteres.



# Formato da Mensagem: extensões multimídia

∀MIME: multimedia mail extension, RFC 2045, 2056.

∀Linhas adicionais no cabeçalho declaram o tipo de conteúdo MIME.

∀Dados multimídia: codificados em ASCII de 7bits!

Versão da MIME

Método usado  
para codificar dados

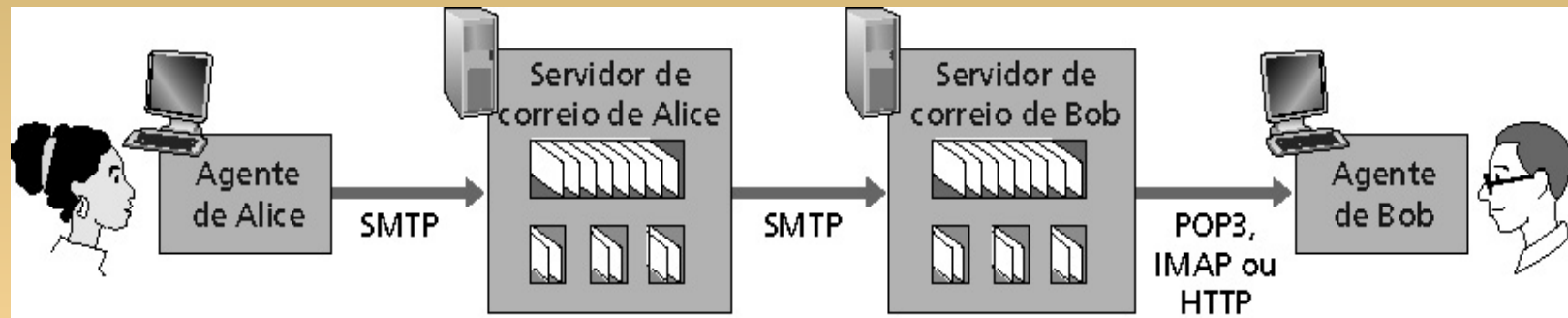
Dados multimídia  
tipo, subtipo,  
declaração de parâmetro

Dados codificados

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

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

# Protocolos de Acesso ao Correio



∀SMTP: entrega e armazena no servidor do destino.

∀Protocolo de acesso: recupera mensagens do servidor:

- POP: Post Office Protocol [RFC 1939]

- Autorização (agente<-->servidor) e download.

- IMAP: Internet Message Access Protocol [RFC 2060]

- mais recursos (mais complexo);

- manipulação de mensagens armazenadas no servidor.

- HTTP: Hotmail, Yahoo! Gmail etc.

# Protocolo POP3

## Fase de autorização

- comandos do cliente:
  - **user**: declara nome do usuário

```
S: +OK POP3 server ready
C: user alice
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

- **pass**: password
- respostas do servidor

- **+OK**
- **-ERR**

## Fase de transação, cliente:

- **list**: lista mensagens e tamanhos
- **retr**: recupera mensagem pelo número
- **dele**: apaga
- **quit**

```
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

- ∇ O exemplo anterior usa o modo “ler e apagar” (*download-and-delete*).
- ∇ O usuário não pode reler o e-mail se trocar o cliente.
- ∇ “ler e guardar” (*download-and-keep*):
  - mantém a mensagem no servidor;
  - cópias das mensagens em clientes diferentes.
- ∇ POP3 não mantém estado através das sessões.

# IMAP

- ∇ Mantém todas as mensagens em um lugar: o servidor.
- ∇ Permite que o usuário:
  - crie pastas;
  - mova as mensagens do Inbox para pastas.
- ∇ IMAP mantém o estado do usuário através das sessões:
  - Nomes das pastas e mapeamentos entre os IDs da mensagem e o nome da pasta.

# Webmail

- Agente de usuário: *browser*.
- Protocolo entre agente de usuário e servidor: HTTP.
- Algumas implementações utilizam um servidor IMAP:
  - scripts no servidor HTTP usam o protocolo IMAP para se comunicar com servidor IMAP;
  - vantagem: funcionalidades do IMAP.

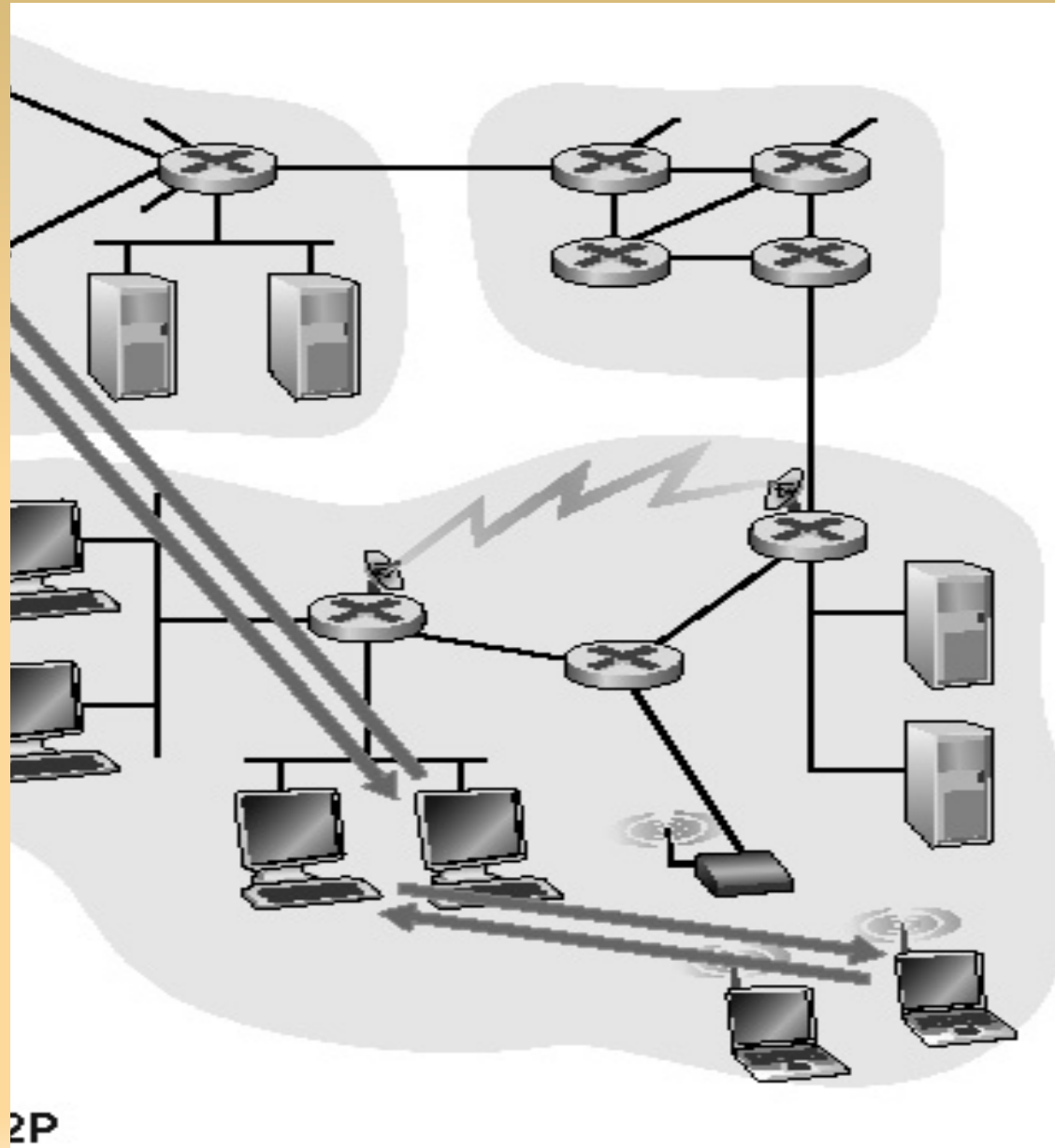
# Cap. 2.6 - Compartilhamento de Arquivos P2P



# Foco

- Protocolos de compartilhamento de arquivos em redes *peer-to-peer*.
- Outros aspectos não discutidos aqui, mas importantes:
  - segurança;
  - privacidade;
  - anonimato;
  - violação de direitos autorais e propriedade intelectual.

# Arquitetura P2P (pura)



# Compartilhamento de Arquivos P2P

∇ Alice executa aplicação P2P:

- utiliza ADSL e obtém novos endereços IP para cada conexão.
- procura música “Hey Jude” (arquivo MP3);
- a aplicação exhibe outros pares que possuem uma cópia de “Hey Jude”;
- Alice escolhe um dos pares, Bob;
- o arquivo é copiado de Bob para Alice.

∇ Enquanto Alice faz o download, é possível fazer upload de arquivos em Alice.

# Compartilhamento de Arquivos P2P

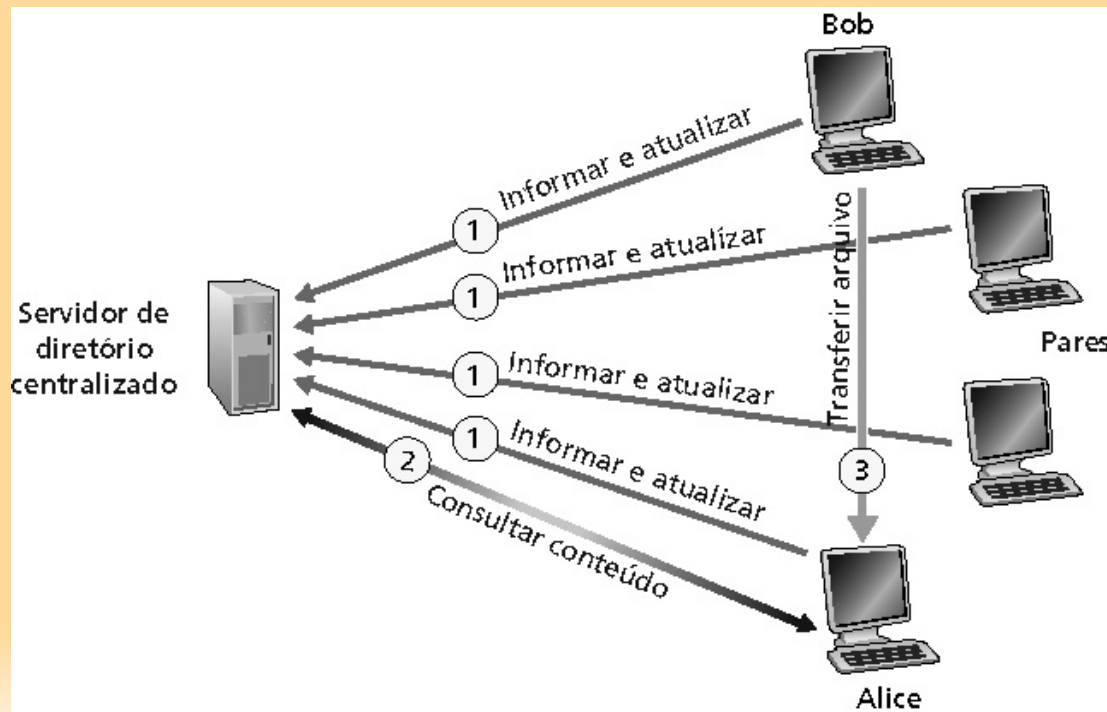
- Transferência de arquivos entre pares usa HTTP!
- Par que possui o arquivo é um **servidor web transitório!**
- Problema chave:
  - localização de arquivos!

# P2P: Diretório centralizado

- Localização de conteúdo através de diretório central.
- Ex.: “Napster”
  - <http://pt.wikipedia.org/wiki/Napster>
- Solução híbrida:
  - cliente/servidor: localização de conteúdo;
  - P2P: transferência de arquivos.

# P2P: Diretório centralizado

- Quando um par se conecta, ele informa ao servidor central: endereço IP e conteúdo.
- Usuário procura arquivo no servidor.
- Usuário requisita o arquivo do par que o contém.
- Servidor precisa verificar se usuários estão conectados.



# P2P: Diretório centralizado - Problemas

- Ponto único de falhas.
- Gargalo de desempenho.
- Violação de direitos autorais:
  - sanções judiciais podem levar ao desligamento dos servidores de diretório.

Transferência de arquivo é descentralizada, mas a localização de conteúdo é altamente centralizada.

# Distribuição de Arquivos P2P

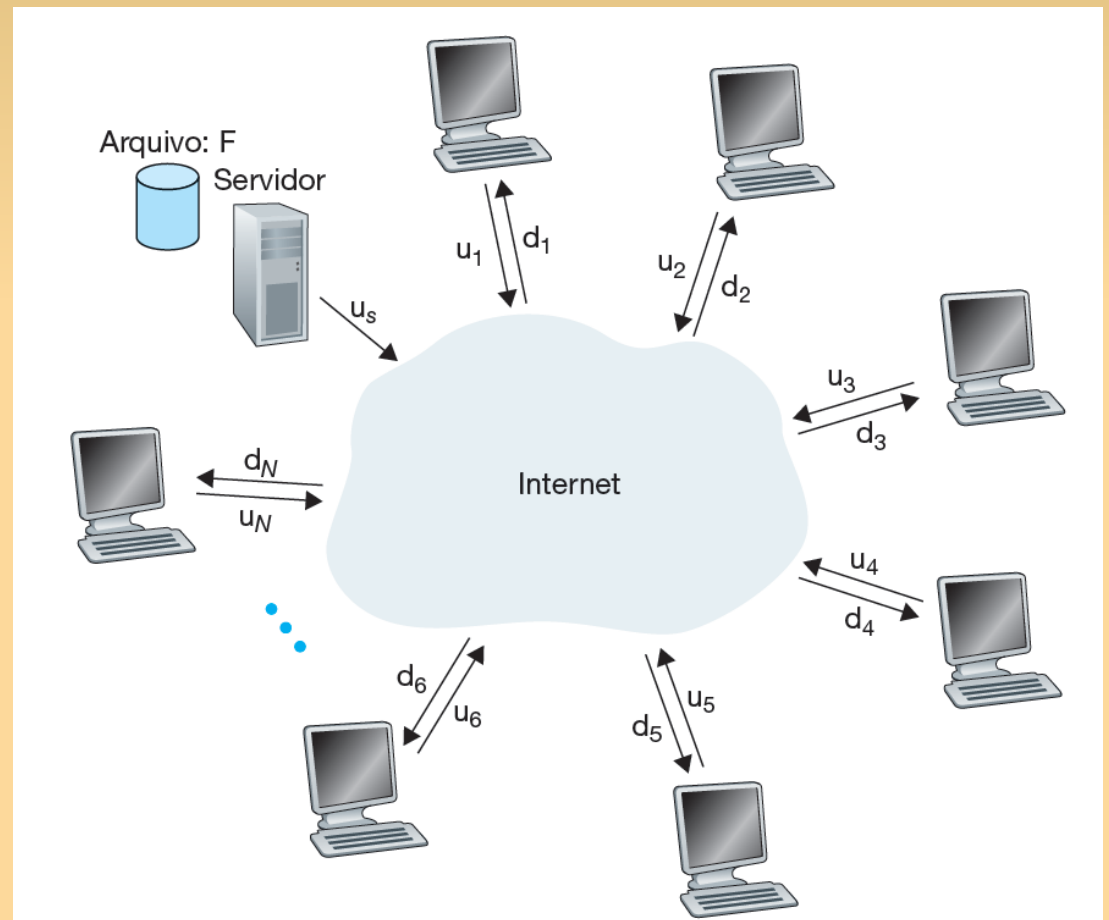
- Na distribuição de arquivos P2P, cada par pode redistribuir qualquer parte do arquivo recebido para outros pares
- auxiliando, assim, o servidor no processo de distribuição.
- O tempo de distribuição é o tempo necessário para que todos os  $N$  pares obtenham uma cópia do arquivo.
- O BitTorrent é um protocolo P2P popular para distribuição de arquivos.



# Distribuição de Arquivos P2P

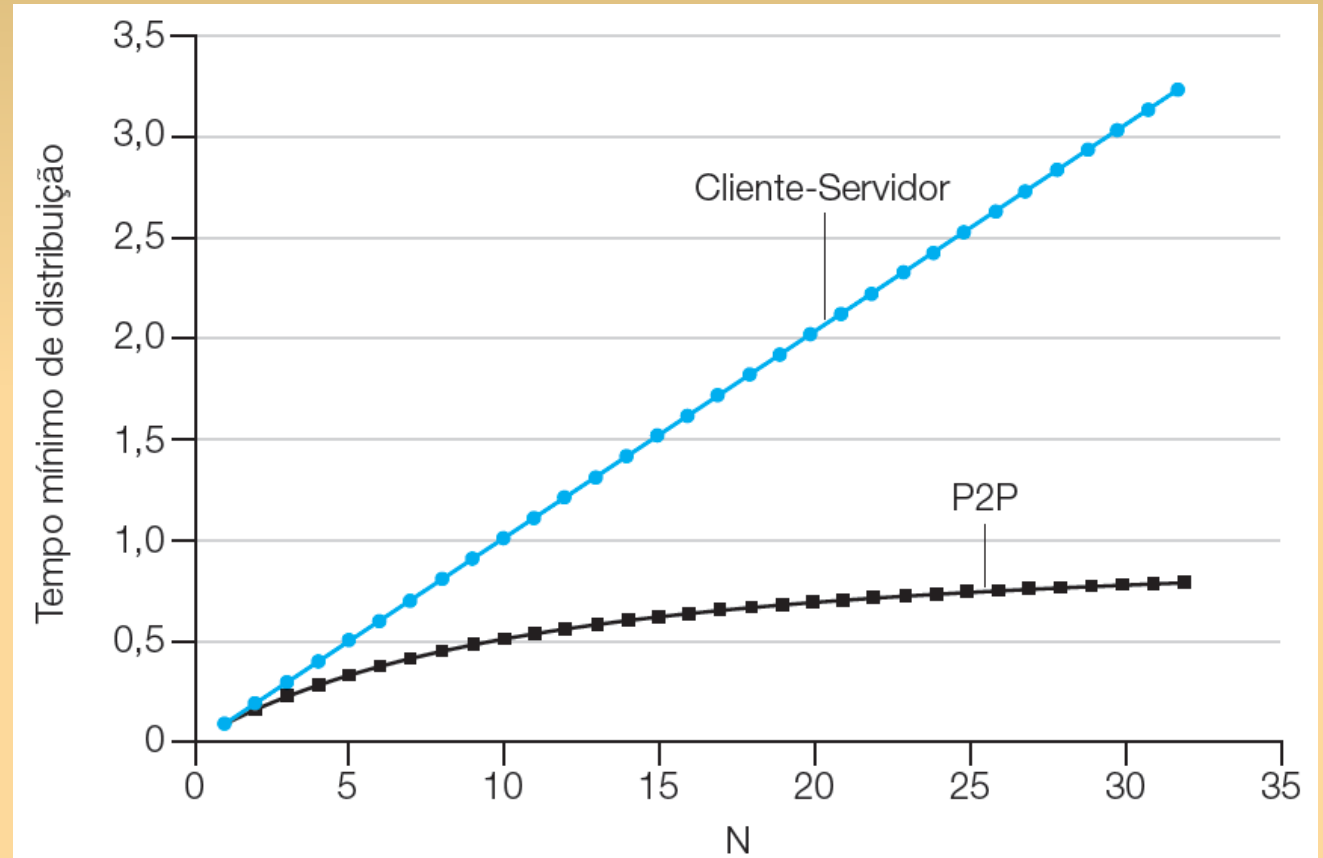
## Cenário de distribuição

Um problema ilustrativo de distribuição de arquivo



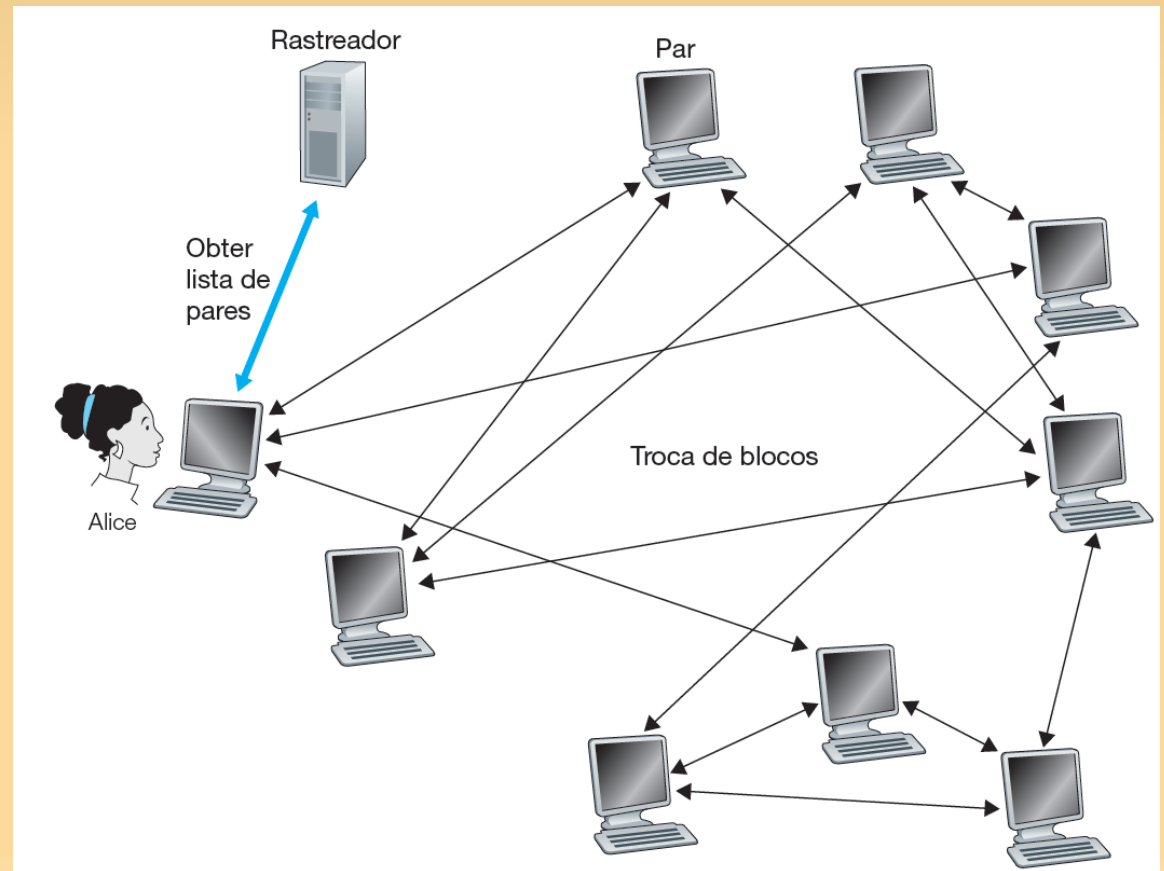
# Comparação P2P vs. Cliente Servidor

Tempo de distribuição para arquiteturas P2P e cliente-servidor



# Distribuição com BitTorrent

Distribuição de arquivos com o BitTorrent



# Distributed Hash Table (DHT)

- Vamos considerar como montar uma versão distribuída, P2P, de um banco de dados, que guardará os pares (chave, valor) por milhões.
- No sistema P2P, cada par só manterá um pequeno subconjunto da totalidade (chave, valor).
- Permitiremos que qualquer par consulte o banco de dados distribuído com uma chave em particular.

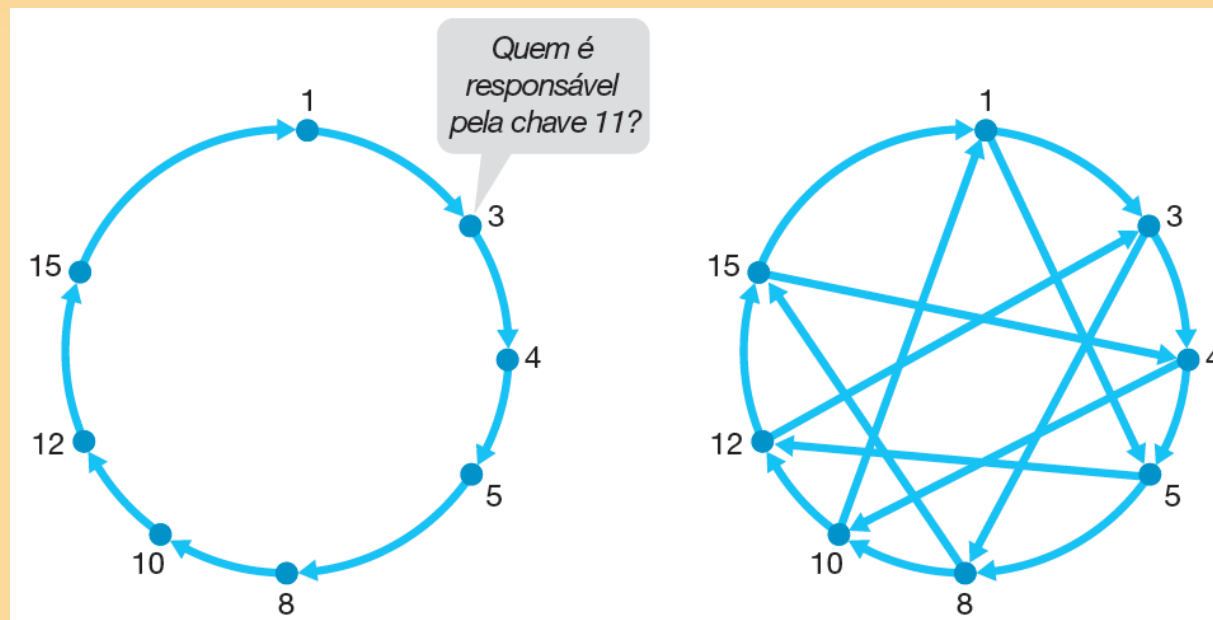
# Distributed Hash Tables (DHTs)

- O banco de dados distribuído, então, localizará os pares que possuem os pares (chave, valor) correspondentes e retornará os pares chave-valor ao consultante.
- Qualquer par também poderá inserir novos pares chave-valor no banco de dados.
- Esse banco de dados distribuído é considerado como uma **tabela *hash* distribuída (DHT — Distributed Hash Table)**.

# DHT Circular

## Distributed Hash Tables (DHTs)

- O DHT circular oferece uma solução bastante elegante para reduzir a quantidade de informação sobreposta que cada par deve gerenciar.



# Alteração e Atualização de DHTs

## Distributed Hash Tables (DHTs)

- Em sistemas P2P, um par pode vir ou ir sem aviso.
- Suponha que o par 5 da figura anterior saia de modo abrupto.
- Os dois pares precedentes ao que saiu (4 e 3) saberão que o par saiu, pois não responde mais às mensagens de ping.
- Os pares 4 e 3 precisam, portanto, atualizar as informações do estado de seu sucessor.

# Alteração e Atualização de DHTs

## Distributed Hash Tables (DHTs)

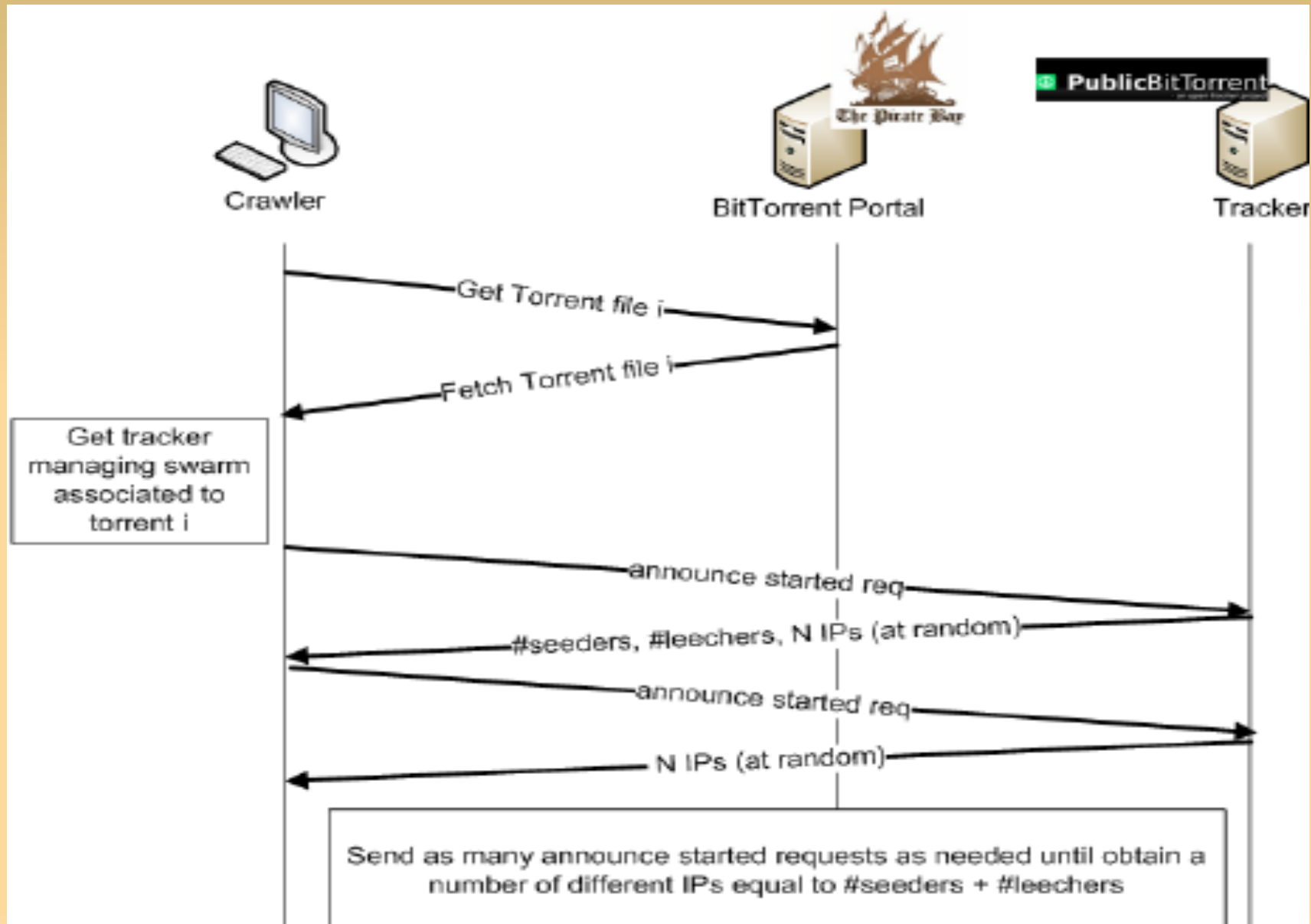
- Consideraremos agora como o par 4 atualiza seu estado:
  1. O par 4 substitui seu primeiro sucessor (par 5) por seu segundo sucessor (par 8).
  2. O par 4, então, pergunta a seu novo primeiro sucessor (par 8) o identificador e o endereço IP de seu sucessor imediato (par 10). O par 4, então, torna o par 10 seu segundo sucessor.



# BitTorrent

- Torrent: coleção de todos os pares que participarão na distribuição de um arquivo
- Cada par distribui “pedaços” de arquivos.
- *Tracker*:
  - coordena distribuição de arquivos;
  - Um por Torrent; cada par avisa periodicamente que ainda encontra-se no Torrent
  - Pode ter 10 a 1K pares em um Torrent
- **Exercício:** como BitTorrent se compara as outras soluções P2P?

# BitTorrent



# Perguntas???