

Programação em Rede Baseada em Java

Luiz Affonso Guedes

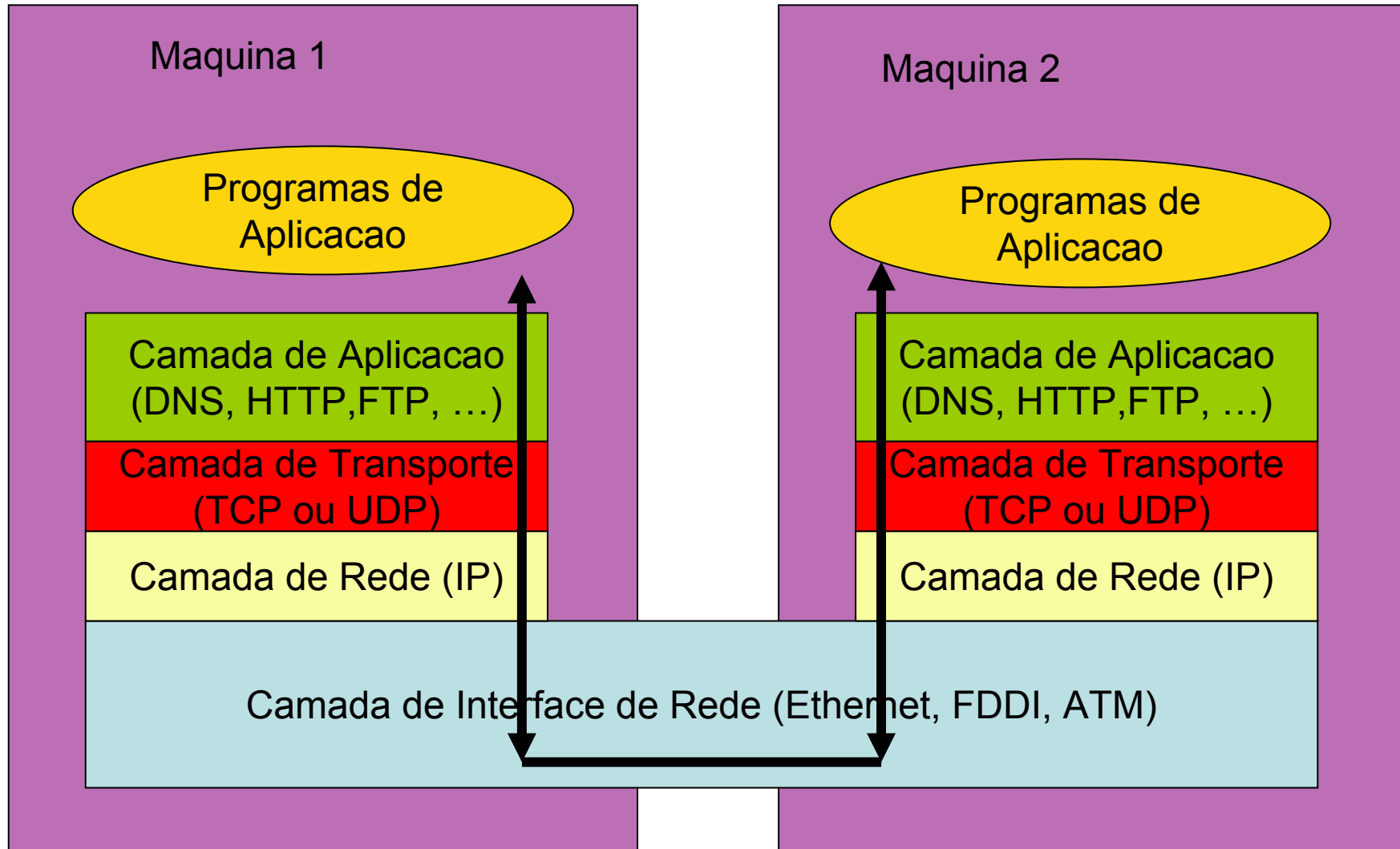
Tópicos em Redes de
Computadores – Programação
Distribuída

www.dca.ufrn.br/~affonso/cursos

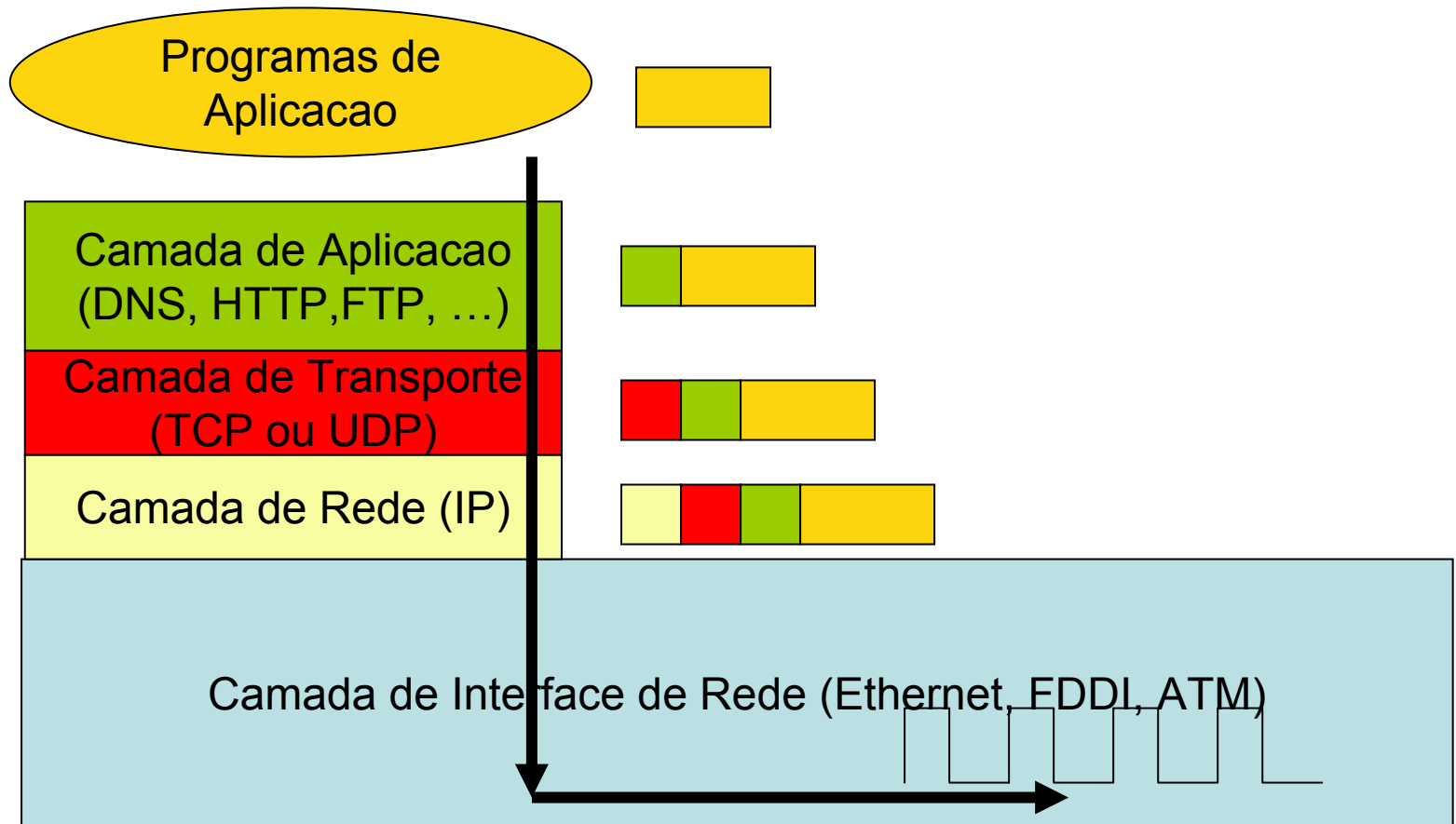
Definições Básicas

- Uma rede é um conjunto de computadores e outros dispositivos interligados por um meio físico.
 - Estes podem enviar e receber dados entre si, mais ou menos em tempo real.
- Redes de Pacotes
 - O canal de comunicação é compartilhado entre os diversos computadores.
 - Os computadores de reversam na utilização do canal, enviando mensagens: pacotes.

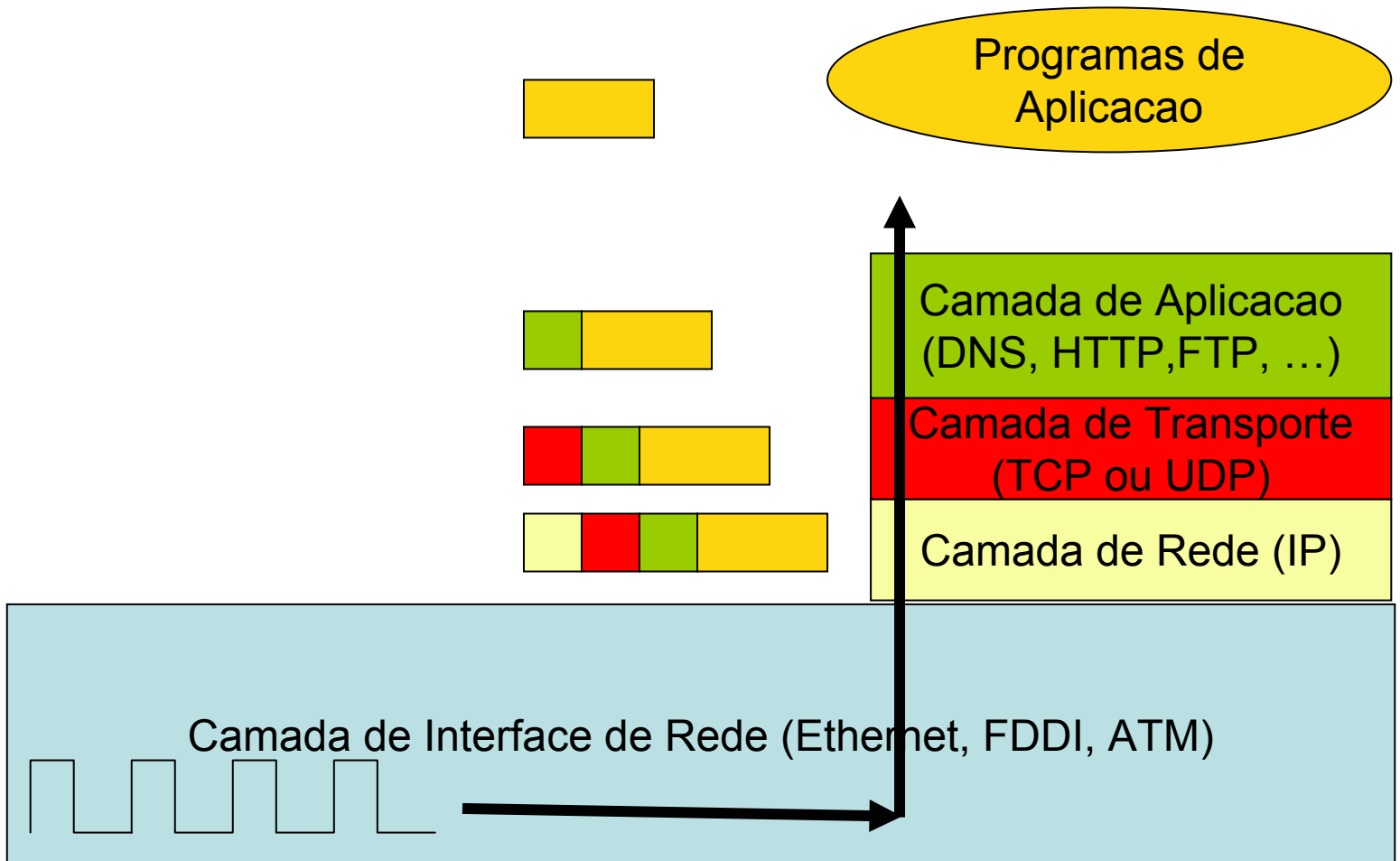
Camadas de uma Redes



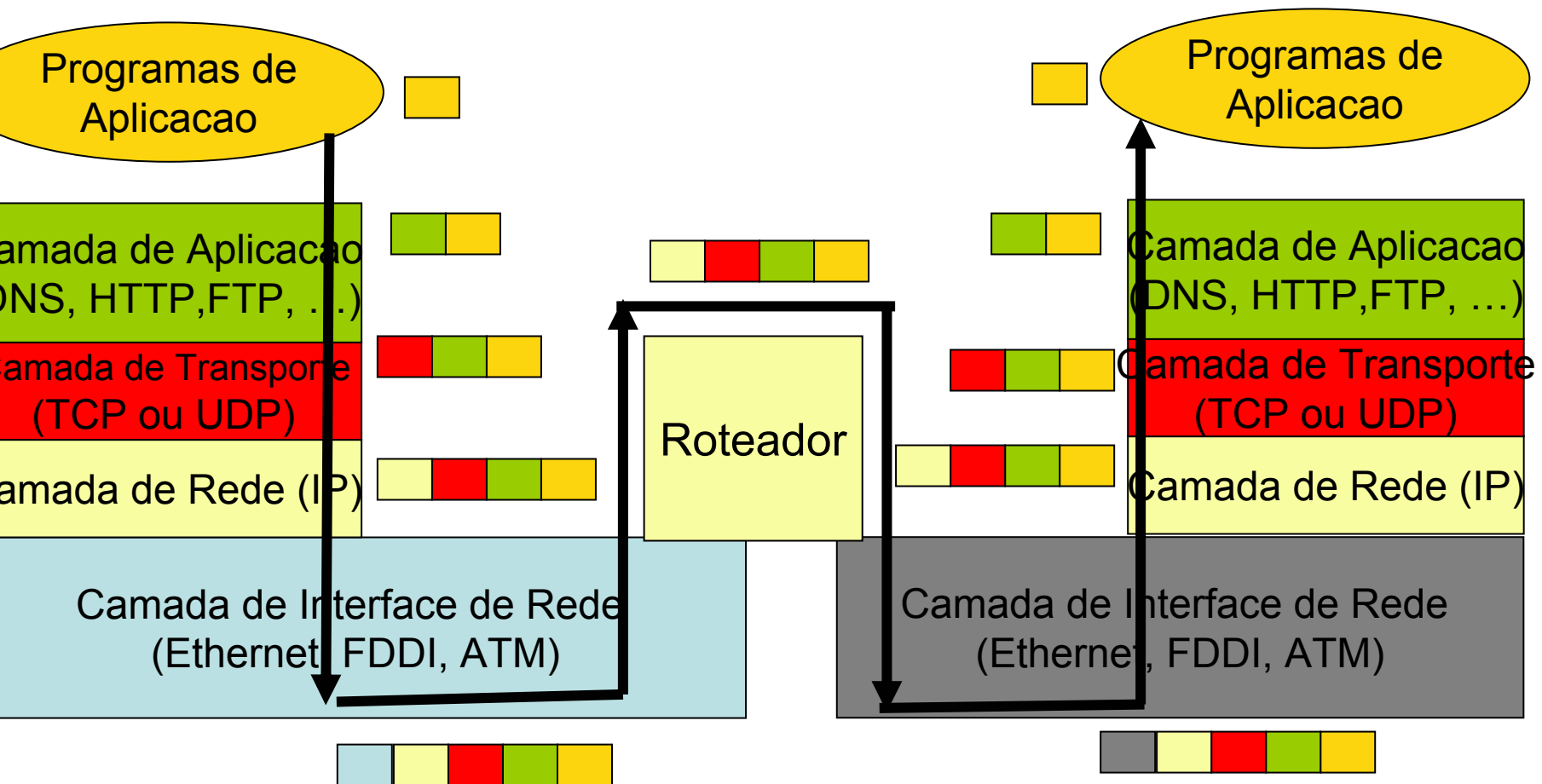
Encapsulação de Dados



Encapsulação de Dados



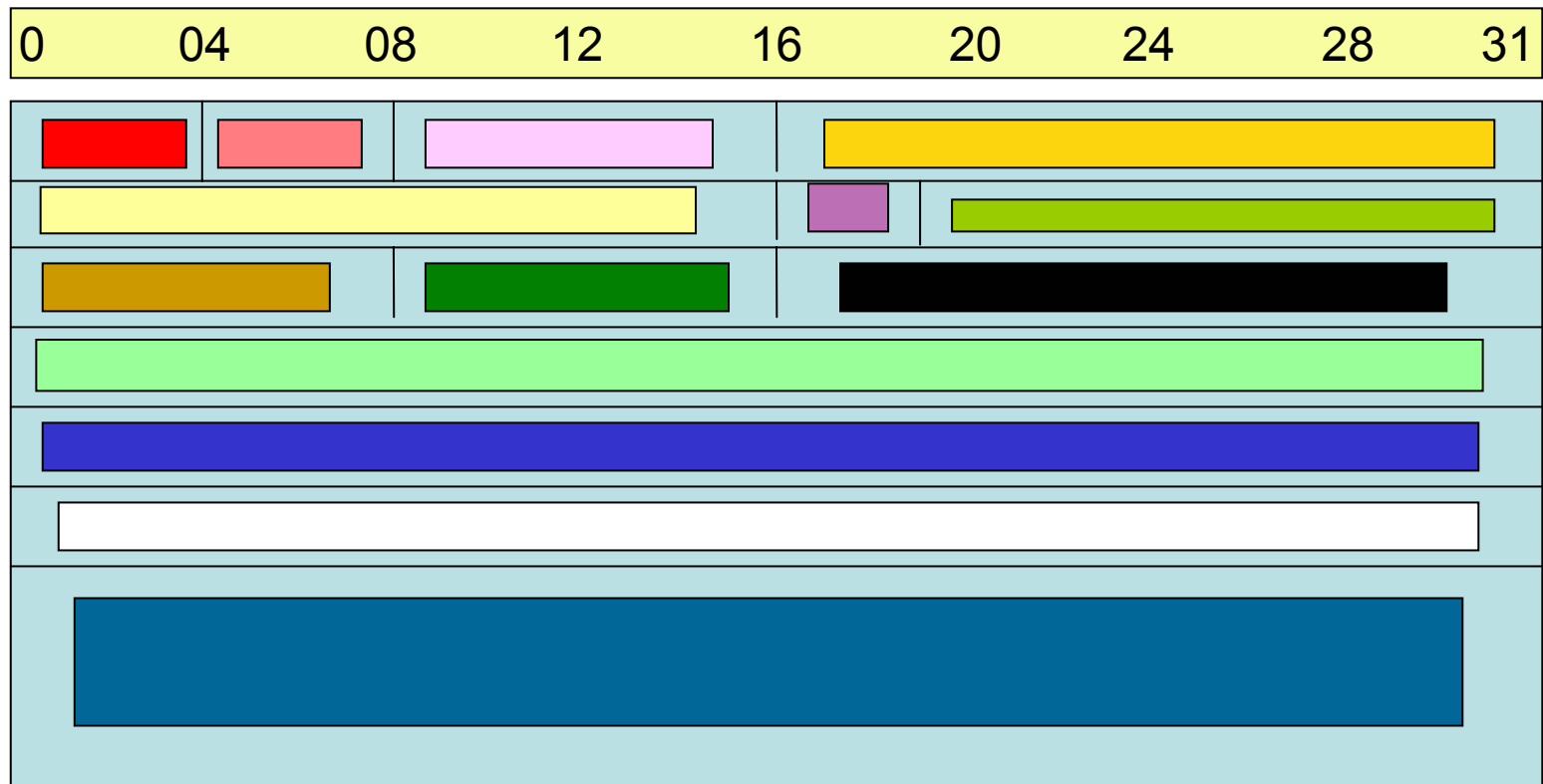
Roteamento de Pacotes


















Camada de Rede

- Protocolos IP (Internet Protocol)
 - Responsável pelo roteamento dos pacotes entre as sub-redes.
 - Numeração IP:
 - 04 bytes.
 - 05 Classes de números: A, B, C, D, E.
 - Cada interface de rede tem de ter pelo menos um endereço IP.
 - 127.0.0.1 → endereço de loop-back (localhost)

Formato do Pacote IP



	versão		Identificação		Protocolo		Opções
	Tamanho cabeçalho		Flags		Checksum/cab.		Dados
	Tipo de serviço		Offset/frag		End. fonte		
	Tamanho do pacote		TTI		End. destino		

Formato do Pacote IP

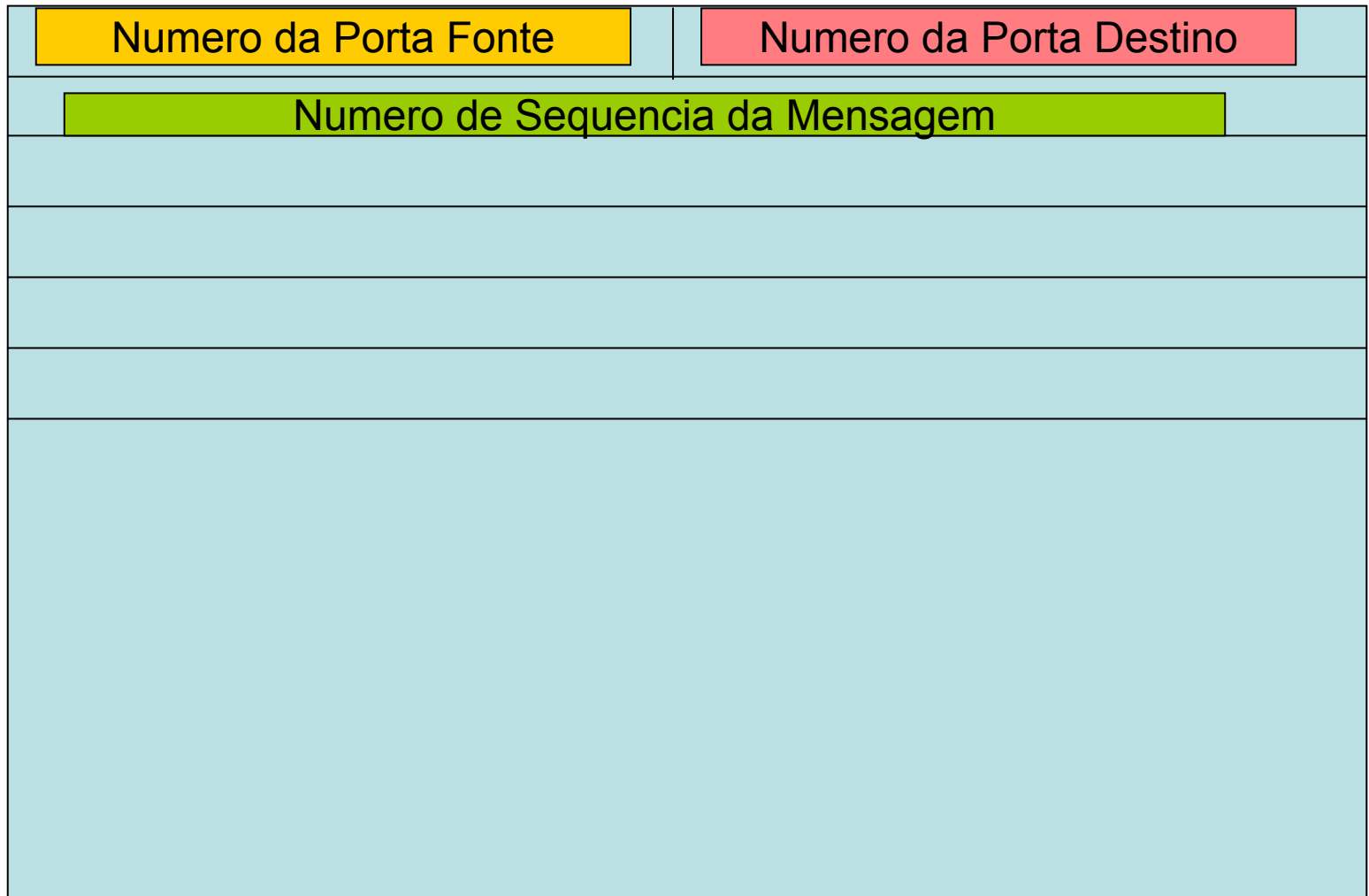
- Campo Versão:
 - Atualmente versão quatro: 4
 - IPv6 (Versao 6): 6
- Campo PROTOCOLO:
 - Protocolo TCP: 6
 - Protocolo UDP: 17
- Campos TTL – Time To Live e Offset/Flags:
 - Únicos campos que são alterados durante o percurso de roteamento dos pacotes

Camada de Transporte

- Responsável pela comunicação fim-a-fim.
- TCP – Transmission Control Protocol
 - Protocolo com orientação de conexão e garantia de entrega de mensagem.
- UDP – User Datagram Protocol
 - Protocolo sem orientação de conexão e sem garantia de entrega de mensagem.

Formato da Mensagem TCP

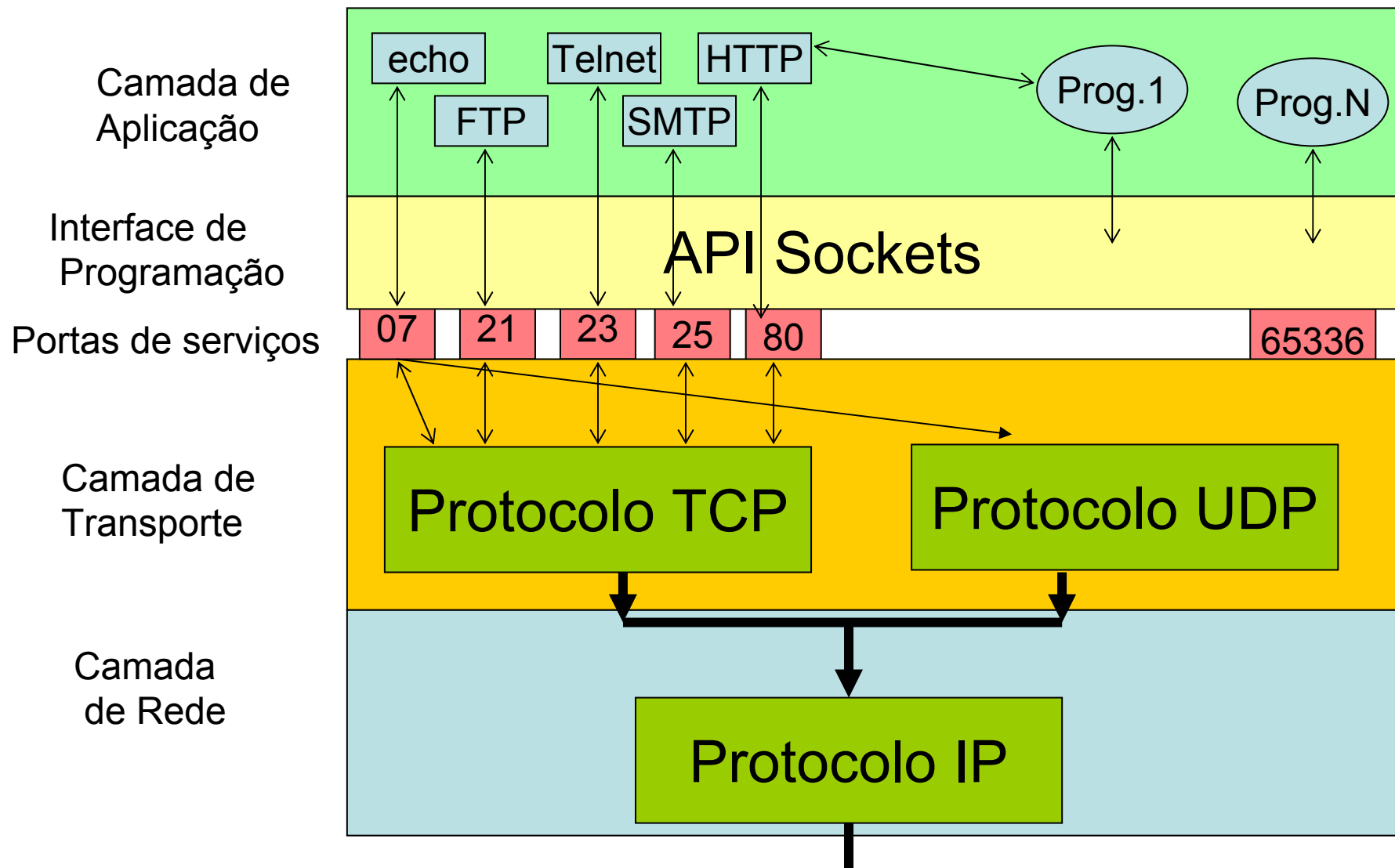
0	04	08	12	16	20	24	28	31
---	----	----	----	----	----	----	----	----



Camada de Transporte

- Conceito de serviço
 - Serviços estão associados com portas.
 - Portas são números inteiros representados por 02 bytes:
 - 65.536 portas possíveis
 - As portas de 1 a 1023 são destinadas a serviços específicos.

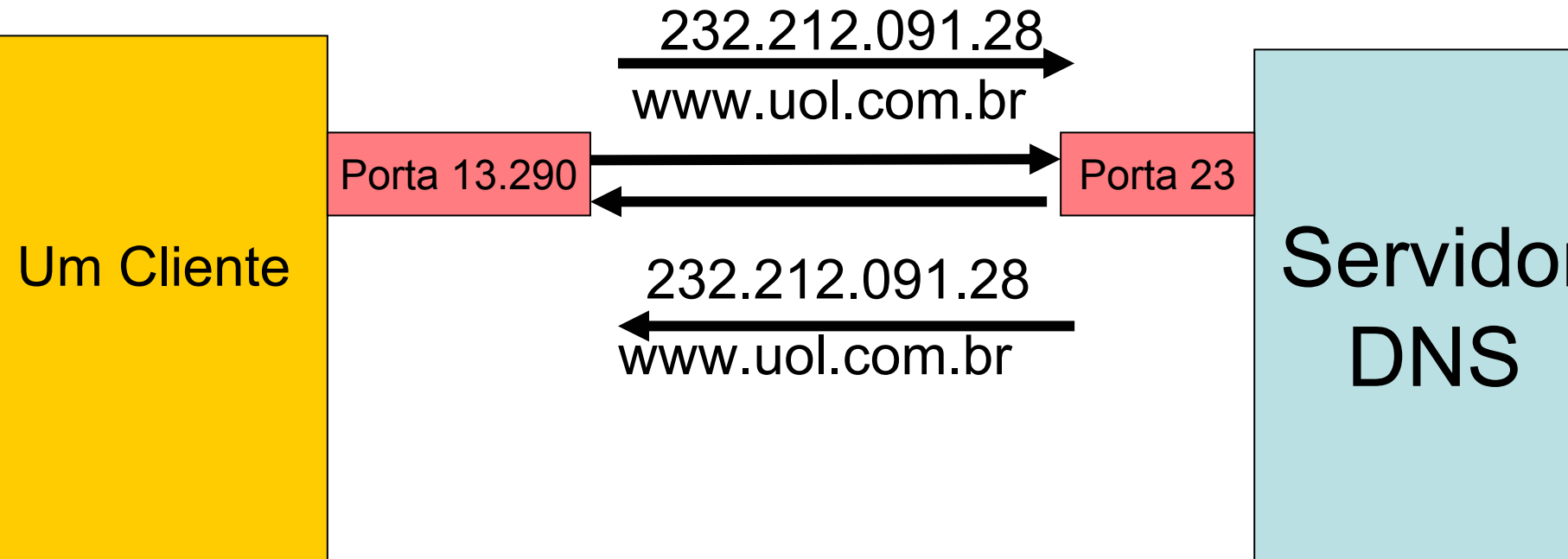
Comunicação num Host



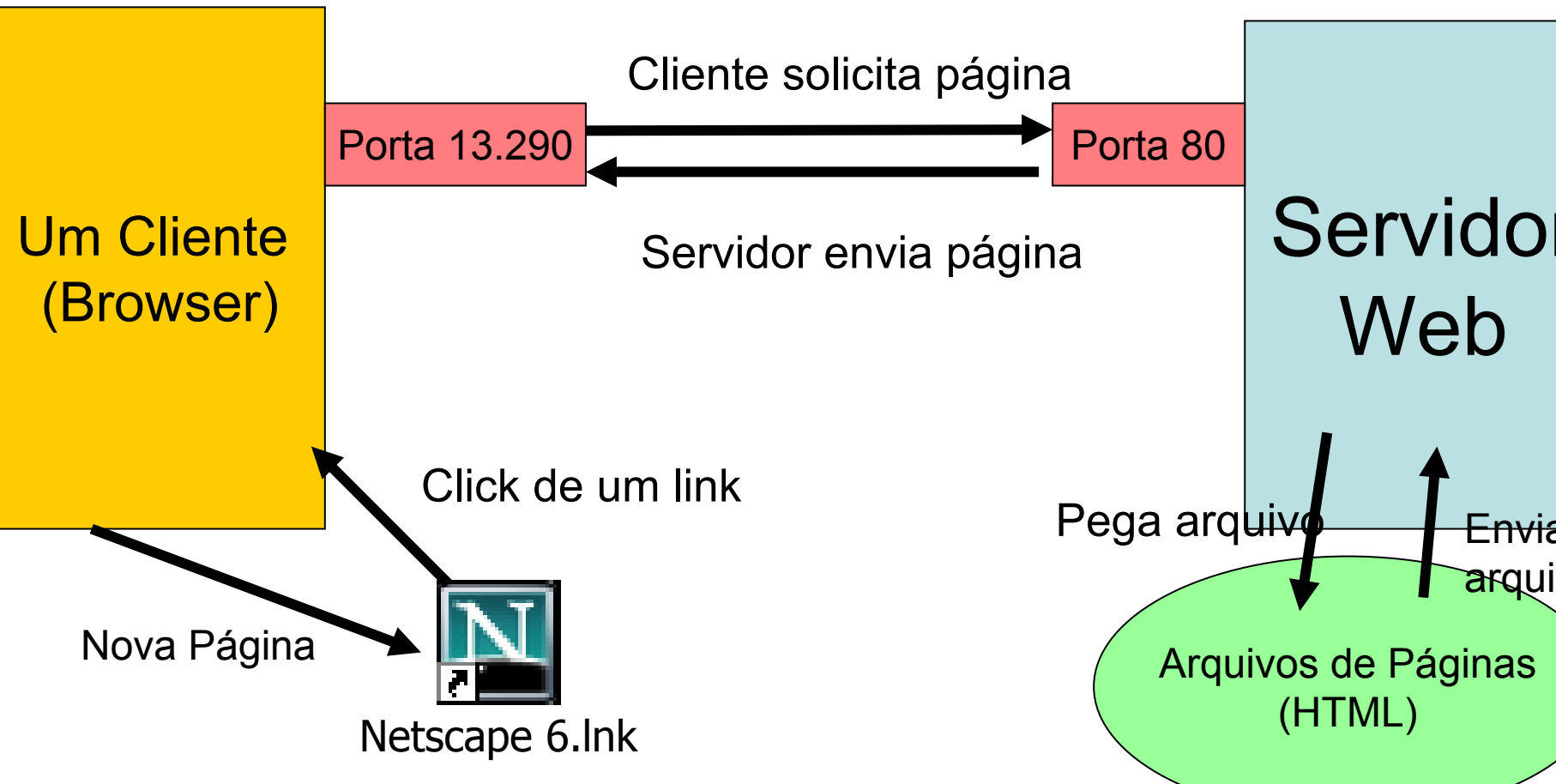
Protocolo da Camada de Aplicação

Protocolo	Porta	Protocolo de Transporte	Proposito
FTP	20 e 21	TCP	Transferência de arquivos
HTTP	80	TCP	Transferência de hipertextos
SMTP	25	TCP	Envio de e-mail para servidor de e-mail
POP3	110	TCP	Transferência de e-mails acumulados no servidor para um cliente
DNS	23 (?)	TCP ou UDP	Converter nome de máquina em número e vice-versa

Exemplo de Comunicação com um Servidor DNS



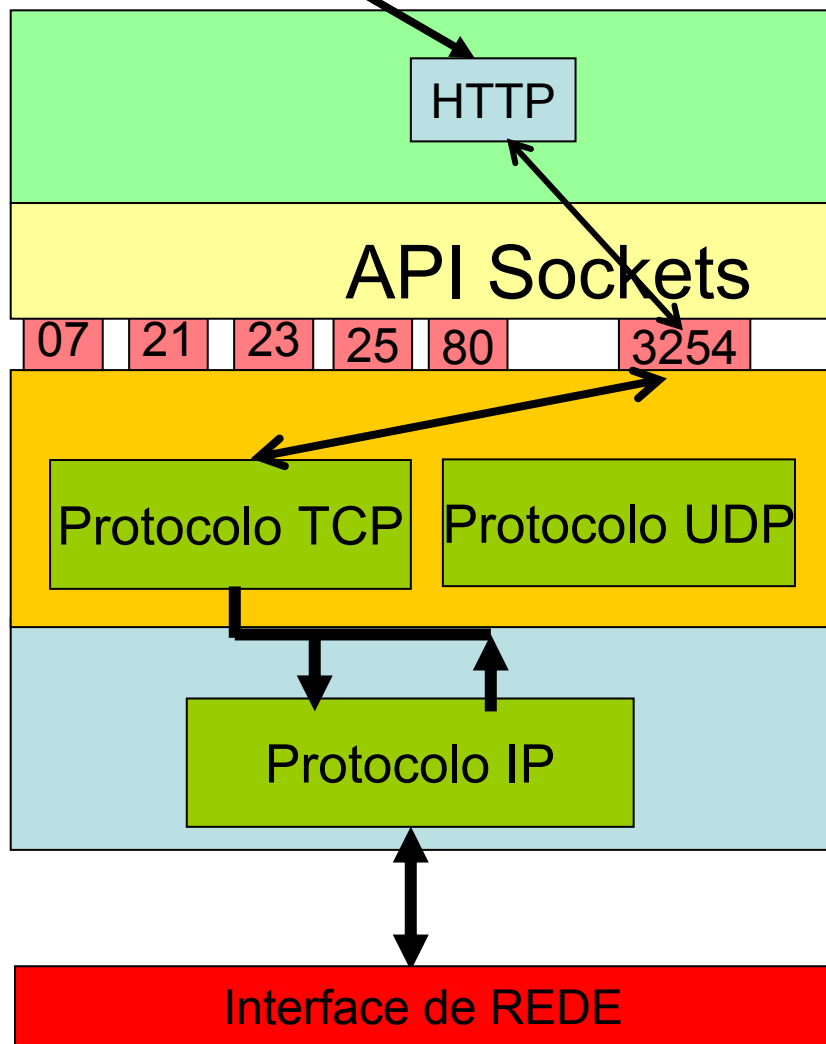
Exemplo de Comunicação com um Servidor WEB



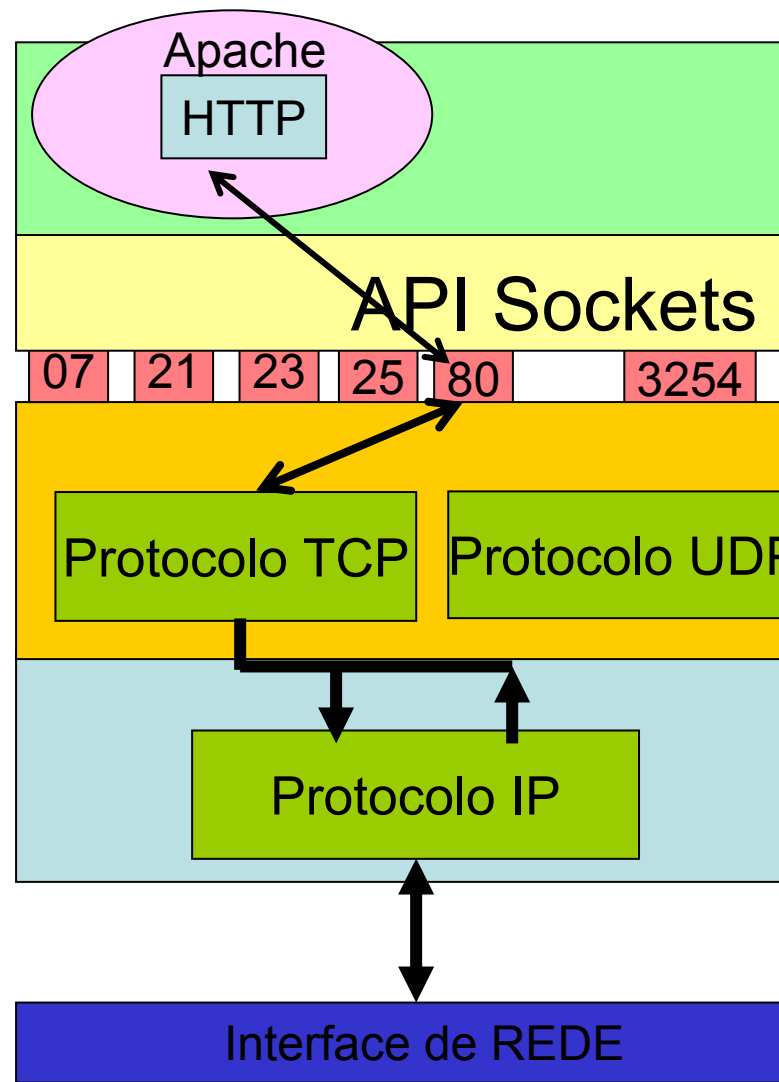


Netscape 6.0

LADO CLIENTE

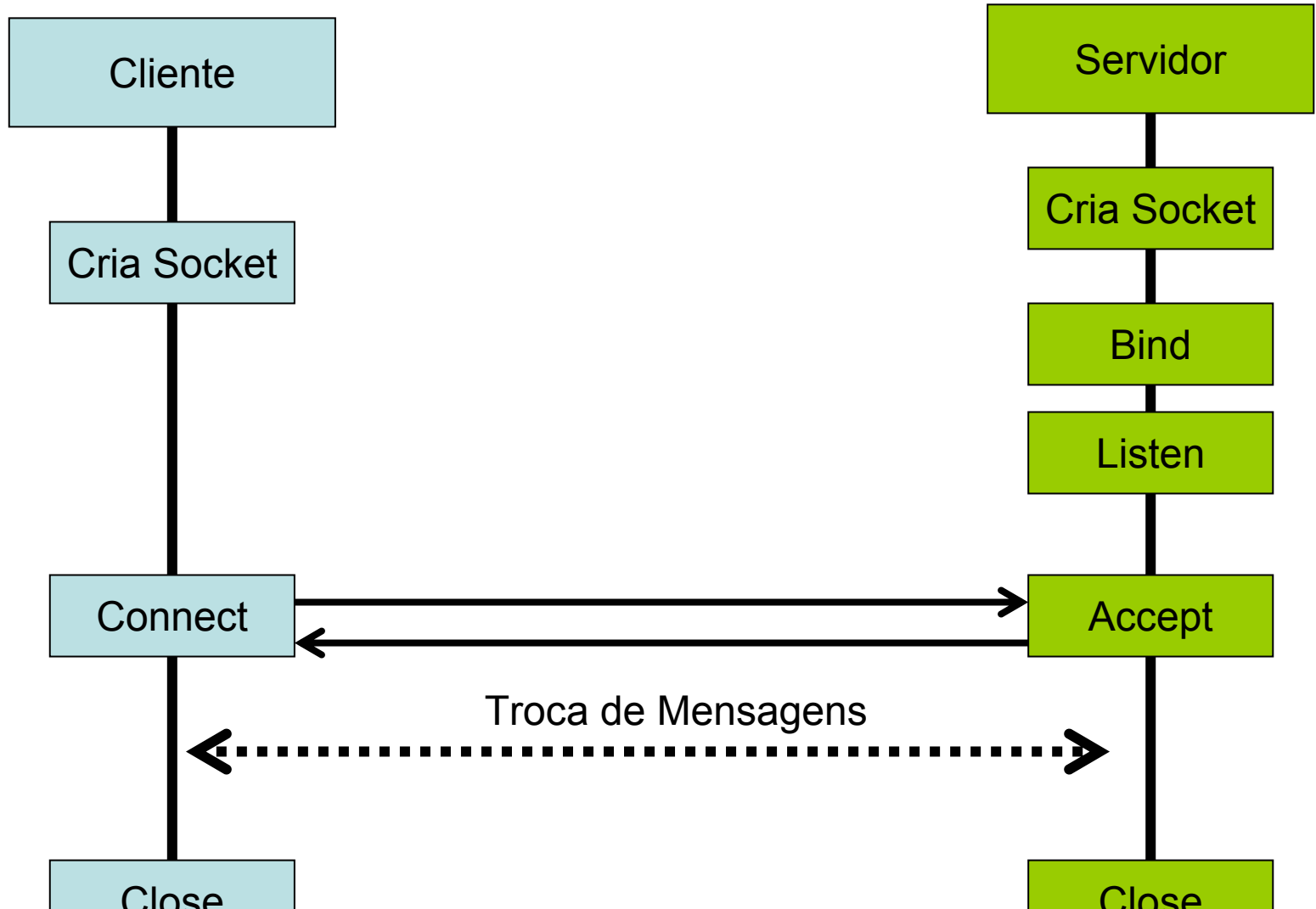


LADO SERVIDOR

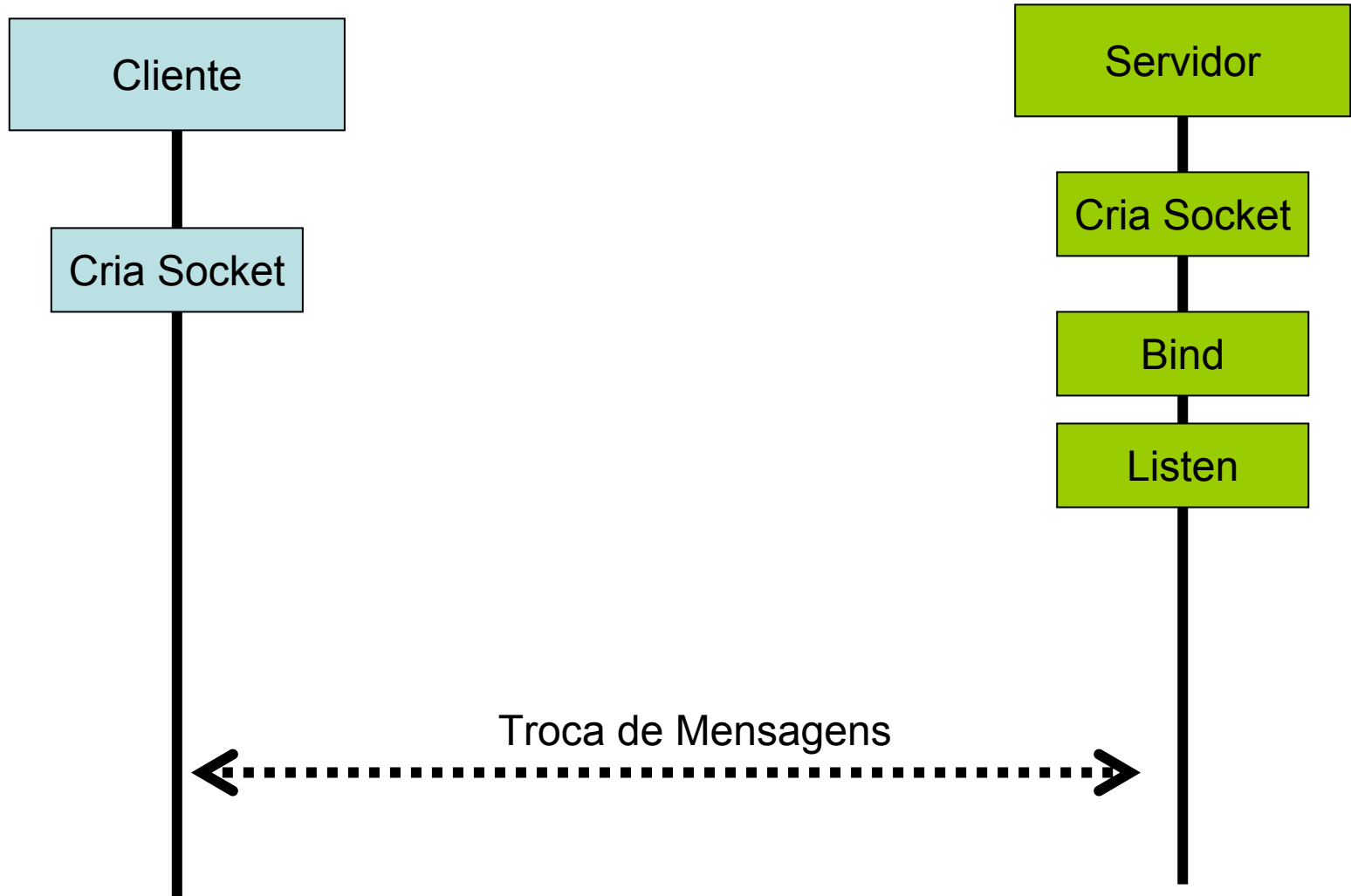


internet

Socket TCP



Socket UDP

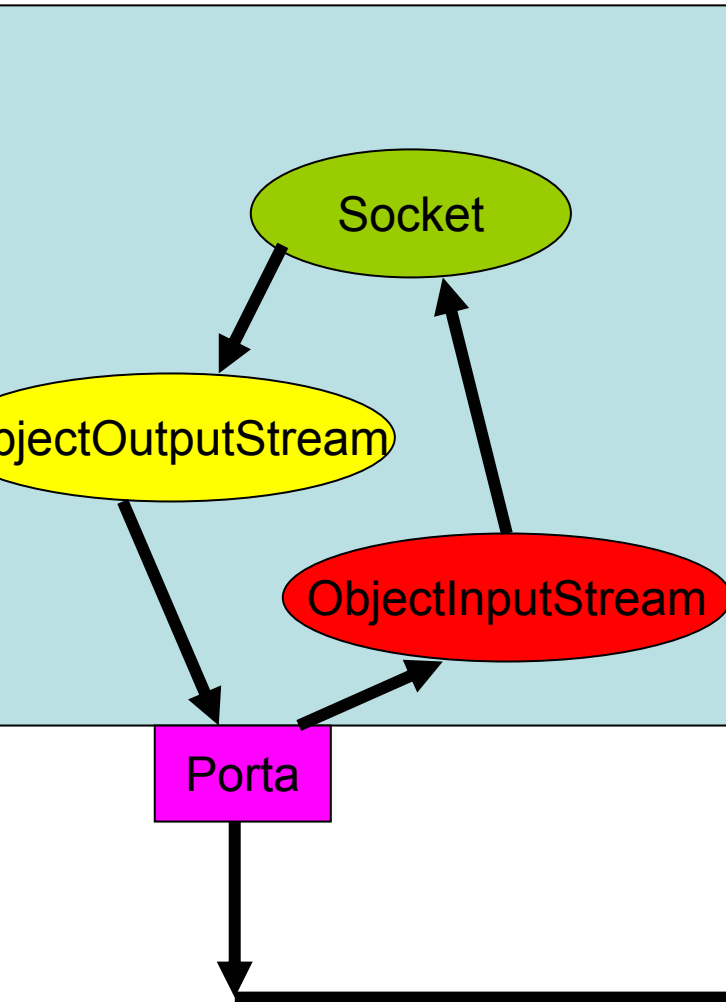


Sockets em Java

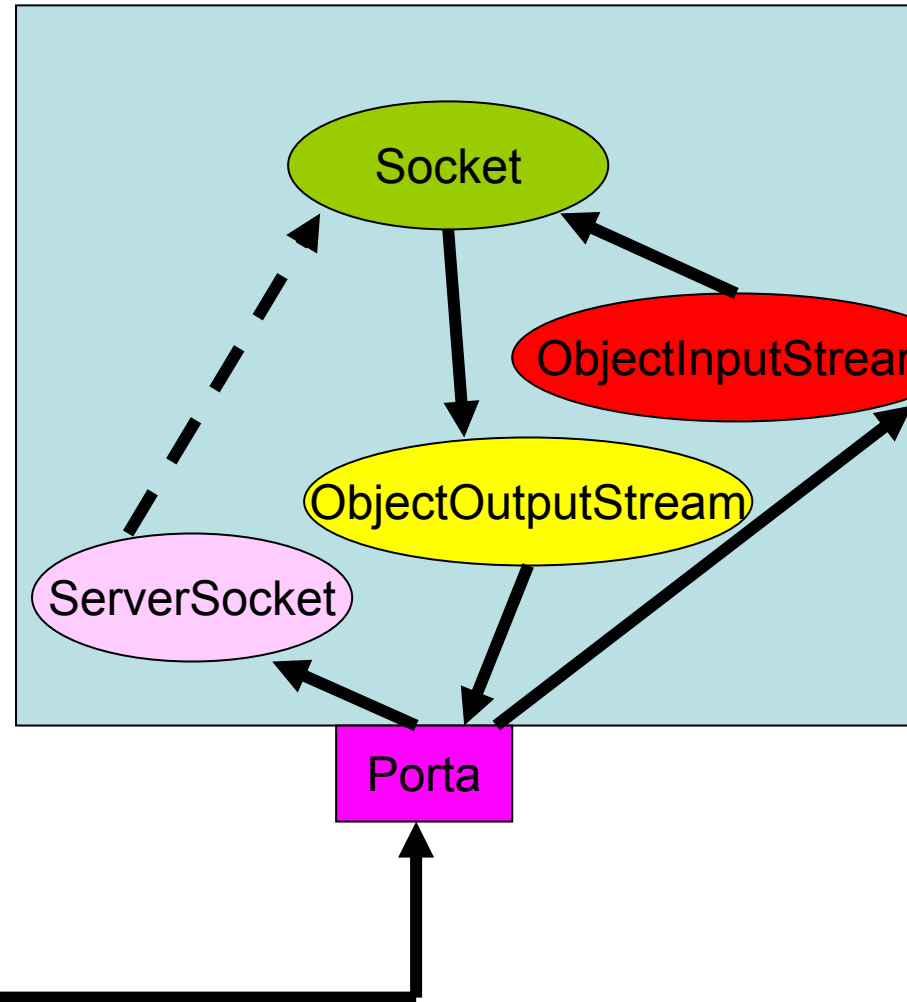
- Classes
 - Socket
 - ServerSocket
- Enviam objetos da Classe OutputStream
- Recebem objetos da Classe InputStream

Sockets em Java

Cliente



Servidor



Exemplo 1: Cria Sockets

- Um programa que faz o scan das portas baixas (1 até 1023).
- Apenas o lado Cliente.
- `Socket s = new Socket(String host, int porta)`
throws `IOException`,
`UnknownHostException`;
- `host = "localhost"; // comunicacao local`

Exemplo2: Utilizando o serviço de nome

- DNS: Domain Name Server
 - Classe InetAddress
 - Pacote `java.net.InetAddress`
 - Possui o endereço Internet armazenado em dois formatos:
 - Nome em `String` e Numero em `int`
 - Métodos
 - `public static InetAddress.getByName(String hostName)`
`Throws UnknownHostException`
 - `public static InetAddress.getLocalHost()`
`Throws UnknownHostException`

Exemplo2: Utilizando o serviço de nome

- Classe InetAddress

- Métodos

- public String getHostName();

- public byte[] getAddress();

- public String.getHostAddress ();

Exemplo3:

- Um programa que faz o **scan** das portas baixas (1 até 1023)
- Socket s =
 new Socket(InetAddress host, int porta)
 throws IOException, UnknowHostException;

Exemplo4

```
public Socket(String host, int port, InetAddress interface,  
               int portaLocal) throws IOException
```

```
public Socket(InetAddress host, int port, InetAddress interface,  
               int portaLocal) throws IOException
```

Exemplo4 - Métodos de informação

Método que obtém o **InetAddress** remoto de onde o Socket está conectado

```
– public InetAddress getInetAddress( )  
-----  
try {  
    Socket umSocket = new Socket("java.sun.com", 80);  
    InetAddress endereco = umSocket.getInetAddress( );  
    System.out.println("Conectado à máquina " + endereco);  
}  
catch(UnknownHostException e) {  
    System.err.println(e);  
}  
catch(IOException e) {  
    System.err.println(e);  
}  
-----
```

Exemplo4 - Métodos de informação

Método que obtém a **porta remota** de onde o Socket esté conectado

– public int getPort()

```
-----  
try {  
    Socket umSocket = new Socket("java.sun.com", 80);  
    int porta = umSocket.getPort( );  
    System.out.println("Conectado à porta " + porta);  
}  
catch(UnknownHostException e) {  
    System.err.println(e);  
}  
catch(IOException e) {  
    System.err.println(e);  
}  
-----
```

Exemplo4 - Métodos de informação

Método que obtém a **porta LOCAL** por onde o Socket está conectado

– public int getLocalPort()

```
-----  
try {  
    Socket umSocket = newSocket("java.sun.com", 80);  
    int porta        = umSocket.getLocalPort( );  
    System.out.println("Conectado a partir da porta " + porta);  
}  
catch(UnknownHostException e) {  
    System.err.println(e);  
}  
catch(IOException e) {  
    System.err.println(e);  
}  
-----
```

Exemplo4 - Métodos de informação

Método que obtém o **InetAddress LOCAL** por onde o Socket está conectado

– public InetAddress getLocalAddress()

```
-----  
try {  
    Socket umSocket          = newSocket("java.sun.com", 80);  
    InetAddress enderecoLocal = umSocket.getInetAddress( );  
    System.out.println("Conectado a partir da maquina " + enderecoLocal);  
}  
catch(UnknownHostException e) {  
    System.err.println(e);  
}  
catch(IOException e) {  
    System.err.println(e);  
}  
-----
```

Exemplo5

- Exemplo de um programa cliente que acessa um servidor de tempo remoto
 - Porta 13
 - `java GetTime <nome_do_host>`
 - Utiliza o método `getInputStream` da classe `Socket`
 - `public InputStream getInputStream()`
`throws IOException`
 - Retorna um fluxo de entrada, de modo a se poder ler dados a partir do `Socket`.

Fechamento do Socket

- Método `close()`
 - Public synchronized void `close ()`
throws `IOException`

```
Socket conexao = null;
try {
    Socket conexao = new Socket(www.ufrn.br, 1333);
    // codigo que vai interagir com o socket
}
catch (UnknownHostException e) { }
catch (IOException e) { }
finally {
    if(conexao != null) conexao.close( );
}
```


Exemplo6

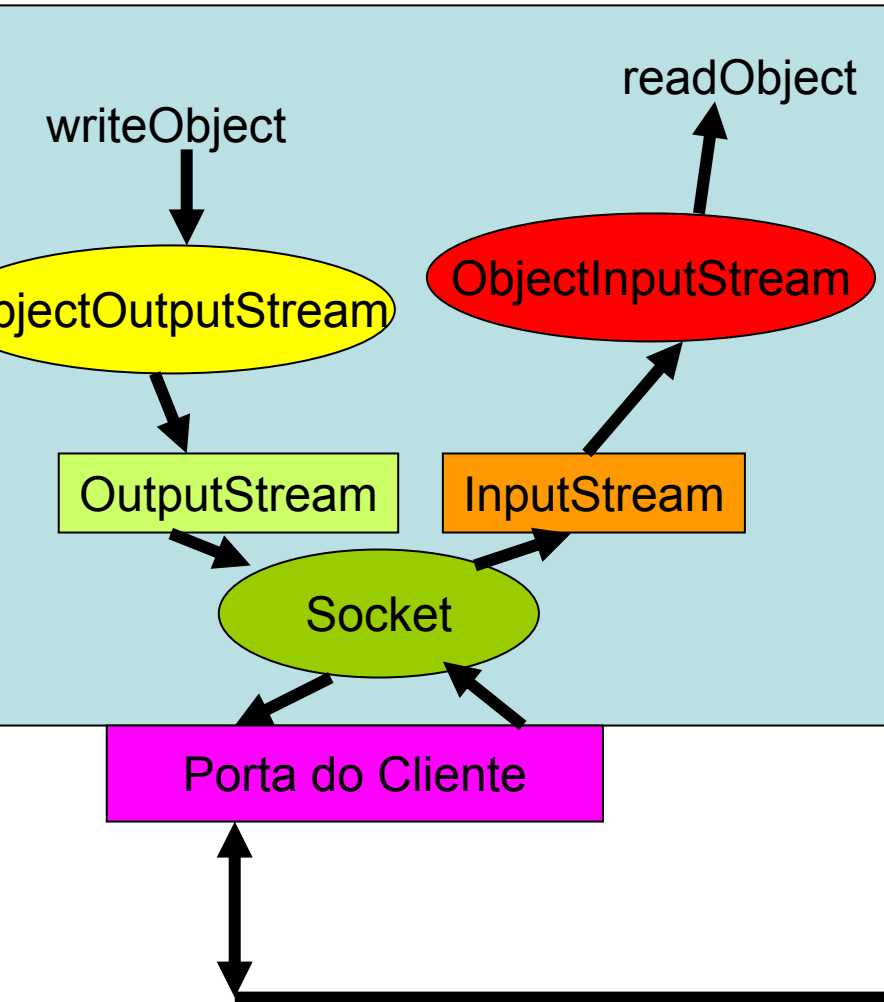
- Programa exemplificando a utilização do método de fechamento de **socket**, com seu tratamento de **exceção**.

Servidor Socket

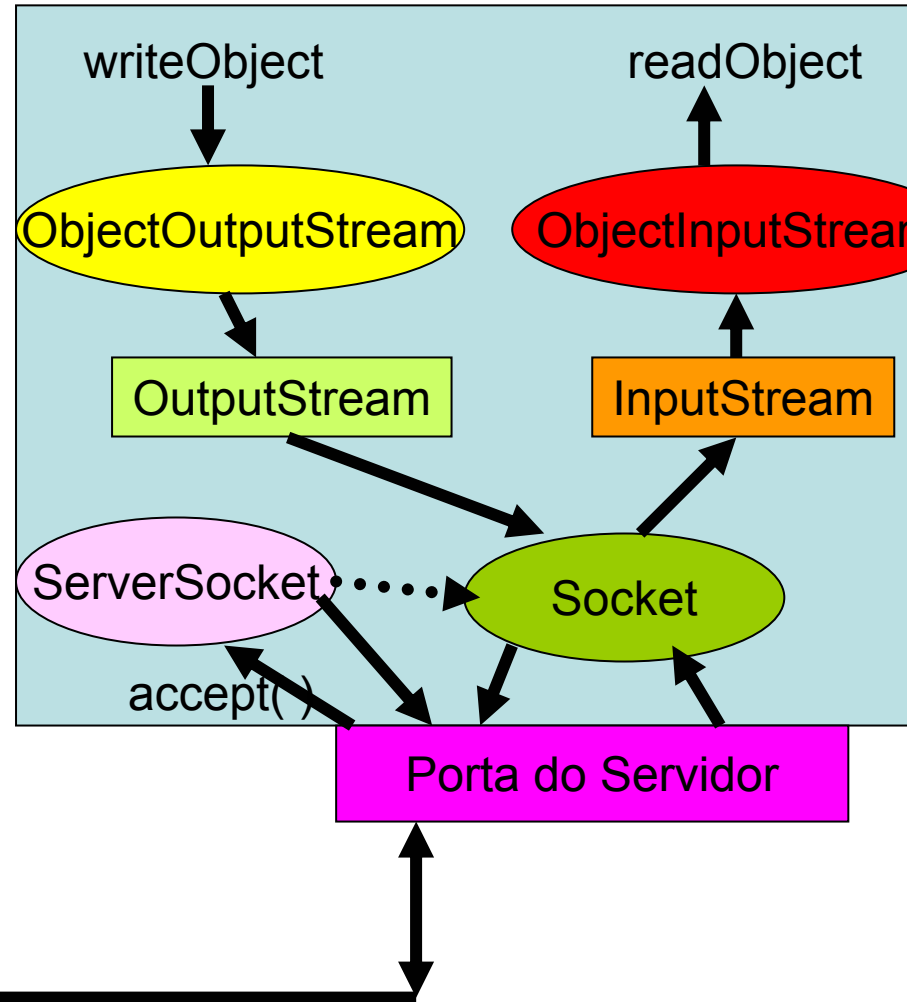
- Utiliza um objeto da **classe** **ServerSocket**
 - Método de escuta de porta (**listen**).
 - Método que aceita uma conexão.
 - Método que retorna um objeto tipo **Socket** quando uma conexão é efetuada.

Sequência típica de uma aplicação Cliente-Servidor em Java

Cliente



Servidor



Ciclo de Vida de um Servidor

1. Um novo `ServerSocket` é criado.
2. Ele fica escutando uma porta: `accept()`.
3. Após aceitar uma conexão, `accept()` retorna um objeto do tipo `Socket`.
4. O objeto tipo `Socket` utiliza os métodos `getInputStream()` e `getOutputStream()` para receber e enviar dados, respectivamente.
5. Cliente e Servidor fecham a conexão.
6. O Servidor (`ServerSocket`) retorna ao passo 1.

Rede de Petri do Ciclo de Vida de um Servidor

Construtores do ServerSocket

- `public ServerSocket (int porta)`
throws `IOException, BindException`
- `public ServerSocket (int porta, int fila)`
throws `IOException, BindException`
- `public ServerSocket (int porta, int fila,
InetAddress clientAddress)`
throws `IOException, BindException`

Aceite de Conexão

- public Socket accept throws IOException

```
ServerSocket server = new ServerSocket(2666);  
while (true) {  
    Socket conexao = server.accept( ); // aguarda conexao  
    OutputStreamWriter saida  
        = new OutputStreamWriter(conexao.getOutputStream());  
    saida.write("Mensagem para o cliente\n");  
    conexao.close();  
}
```

Exemplo7

- **Servidor1.java e Cliente1.java**
 - Servidor envia um String para Cliente, dando boas-vindas.

Socket UDP

Comunicação Multicast

- Tipos de comunicação:
 - Unicast
 - Multicast
 - Broadcast
- Por que utilizar comunicação **multicast**?

Roteamento de Grupo

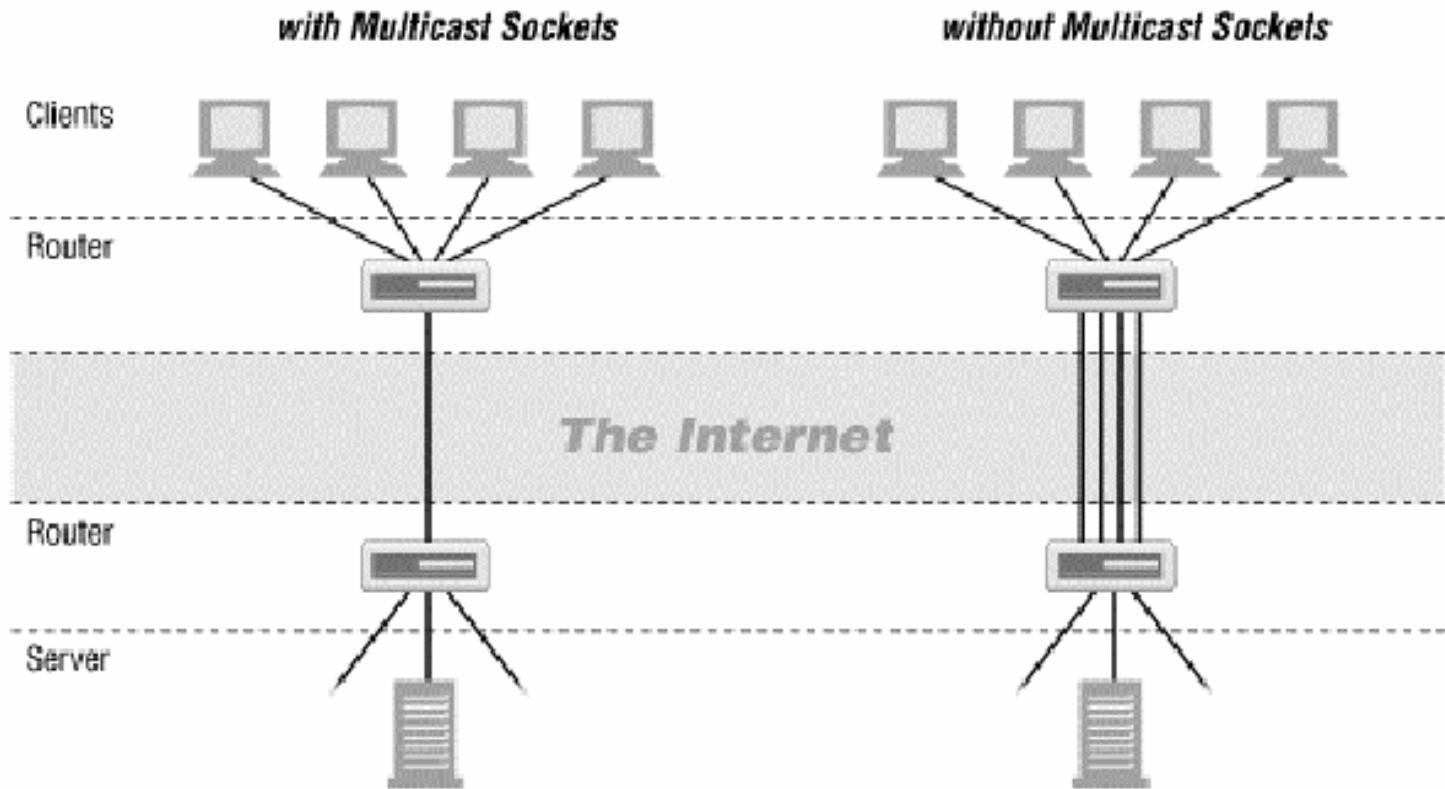
- Protocolo IGMP
- TTL – Time To Live
 - Máximo número de roteadores que o datagrama poderá ser roteado
 - TTL = 0 → Localhost
 - TTL = 1 → Rede Local
- Necessidade de ser transparente para o programador da aplicação
 - A aplicação deve enviar pacotes para o endereço multicast.
 - TCP ou UDP ???

- Conceito de GRUPO:
 - conjunto de hosts que compartilham um mesmo endereço multicast.
 - todo dado enviado para um endereço multicast é recebido por todos os membros do grupo.
 - membros se associam-se e desligam-se do grupo de forma autônoma.
- Endereçamento Multicast
 - Endereço do grupo
 - Classe D: 224.0.0.0 – 239.255.255.255

Endereços Multicast

- 224.0.0.0 – Endereço reservado.
- 224.0.0.1 - Grupo de todos os sistemas que suportam multicast na rede local.
- 224.0.1.1 – Network Time Protocol.
- 224.0..1.32 – Versão multicast do traceroute.
- 224.2.0.0 a 224.2.255.255 – MBONE (Multicast Backbone on the Internet)

Sockets Multicast



Classe Socket Multicast

- java.net.MulticastSocket
 - Subclasse de: java.net.DatagramSocket
 - `public` class `MulticastSocket` `extends` `DatagramSocket` throws `SocketException`
 - MulticastSocket `socket` = `new` MulticastSocket();
 - MulticastSocket `socket` = `new` MulticastSocket(`int` `porta`);

Estrutura de uma comunicação multicast

1. Criação do Socket
2. Associação a um grupo multicast (se for receber dados)
3. Envio/Recebimento de dados para/de membros do grupo
4. Desligamento do grupomulticast
5. Fechamento do Socket

- Associação a um grupo multicast
 - public void joinGroup(InetAddress endereco)
throws IOException
- Desligamento de um grupo multicast
 - public void leaveGroup(InetAddress endereco)
throws IOException
- Envio de dados
 - public void send(DatagramPacket pacote, byte ttl)
throws IOException
- Recebimento de dados
 - public void receive(DatagramPacket pacote)
throws IOException

Exemplo 9

- Programas Cliente e Servidor Multicast
 - O programa Cliente envia uma seqüência de mensagens ao programa Servidor.