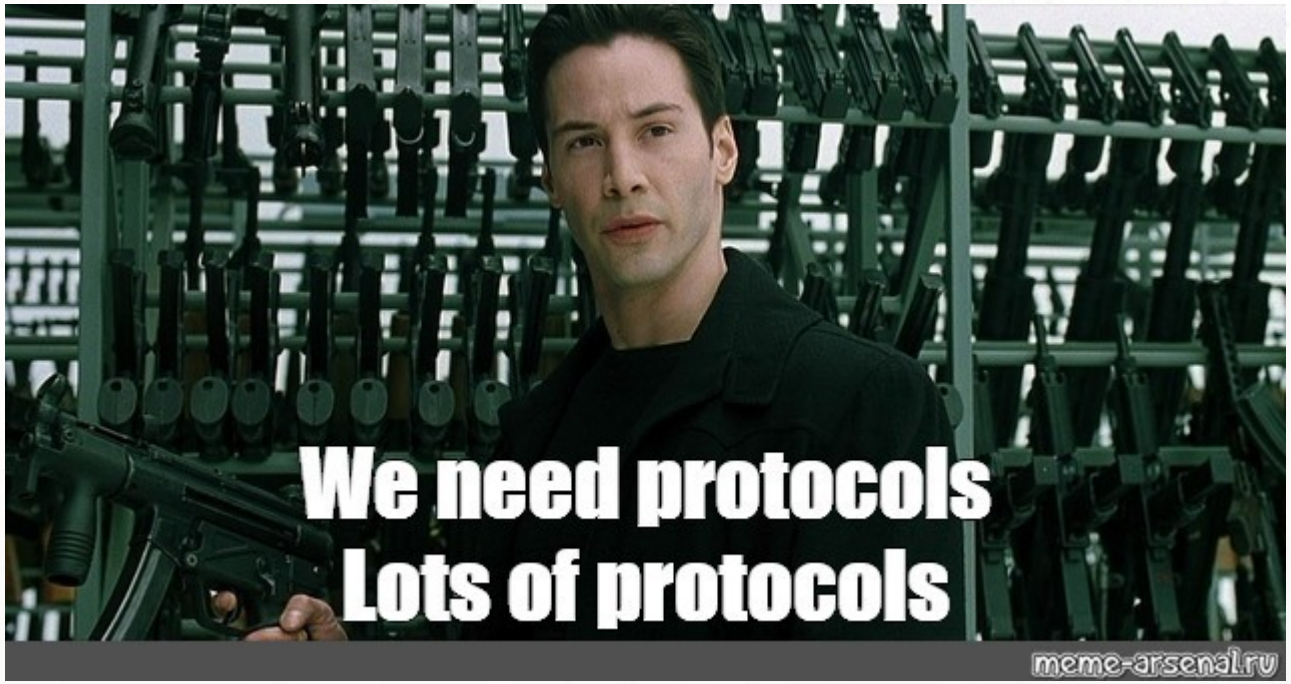
Sockets

Sistemas Distribuídos e Mobile
Gustavo Torres Custodio
gustavo.custodio@anhembi.br

Agenda

- Protocolos
  - TCP
  - UDP
- Sockets
  - TCP
  - UDP

Protocolos



ăProtocolos

- O que é um protocolo de comunicação?
  - Conjunto de regras e procedimentos para que duas entidades distintas possam estabelecer um canal de comunicação
    - Etapas de estabelecimento e finalização de conexão
    - Cabeçalhos indicativos de numeração e ordem dos pacotes
    - Endereço do emissor e receptor

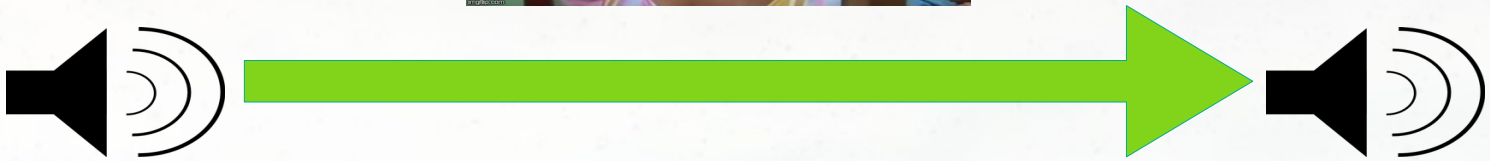
Protocolos

Preciso do contrato de compra.

Executiva no Brasil

Executivo no Canadá

O processo é o mesmo na internet



Protocolos na Internet

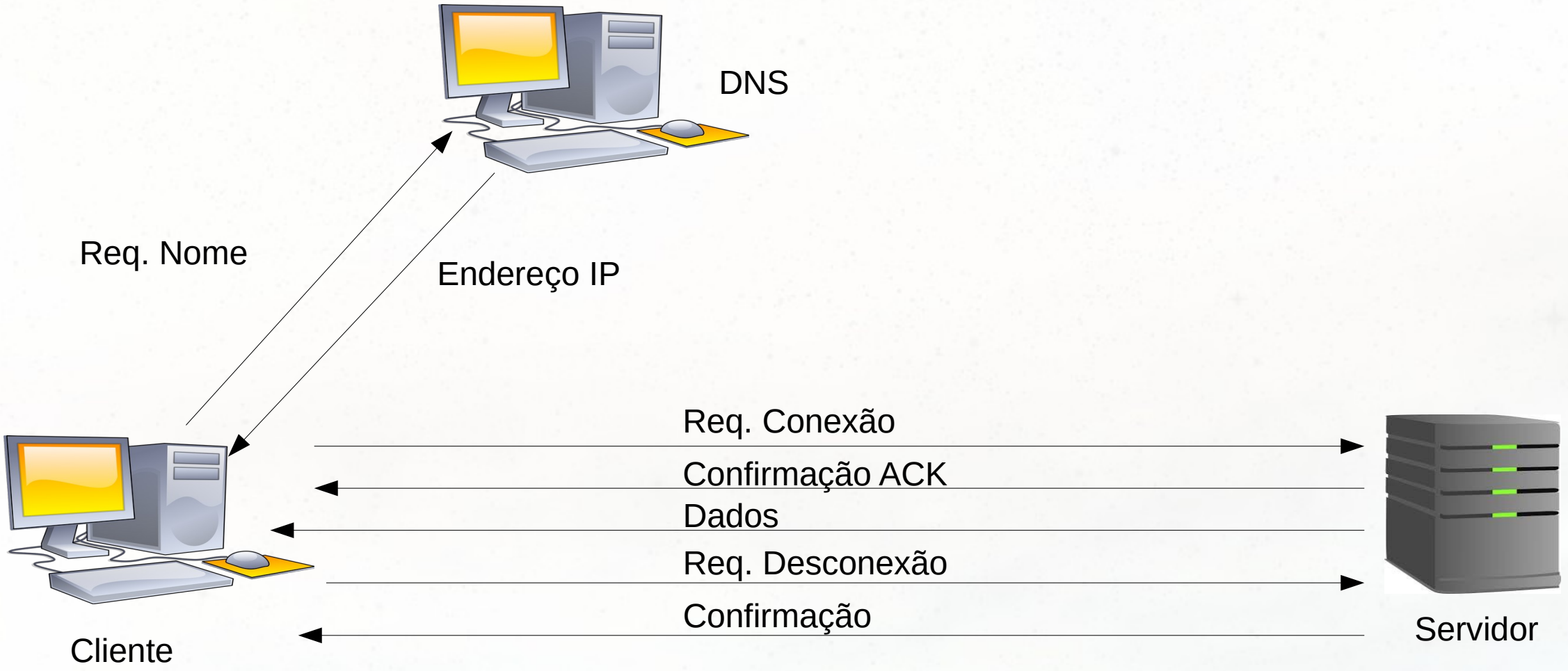
ăProtocolo TCP/IP

- Protocolo atualmente utilizado na Internet.
- Baseado no modelo OSI da ISO (7 camadas).
- TCP
  - protocolo de transporte que utiliza os serviços do protocolo IP para garantir estabelecimento de conexões e integridade de dados.
- IP
  - protocolo de rede responsável pelo endereçamento das máquinas (endereço IP) e rotas entre dispositivos.

Protocolo TCP

- *Transmission Control Protocol* (TCP):
  - Protocolo orientado à conexão:
    - Exige o estabelecimento de um canal lógico para iniciar a transmissão de dados, em 3 fases:
      - Fase de conexão;
      - Fase de dados;
      - Fase de desconexão.
  - Exemplos de aplicação:
    - TELNET, Web Browser, ...

Conexão TCP

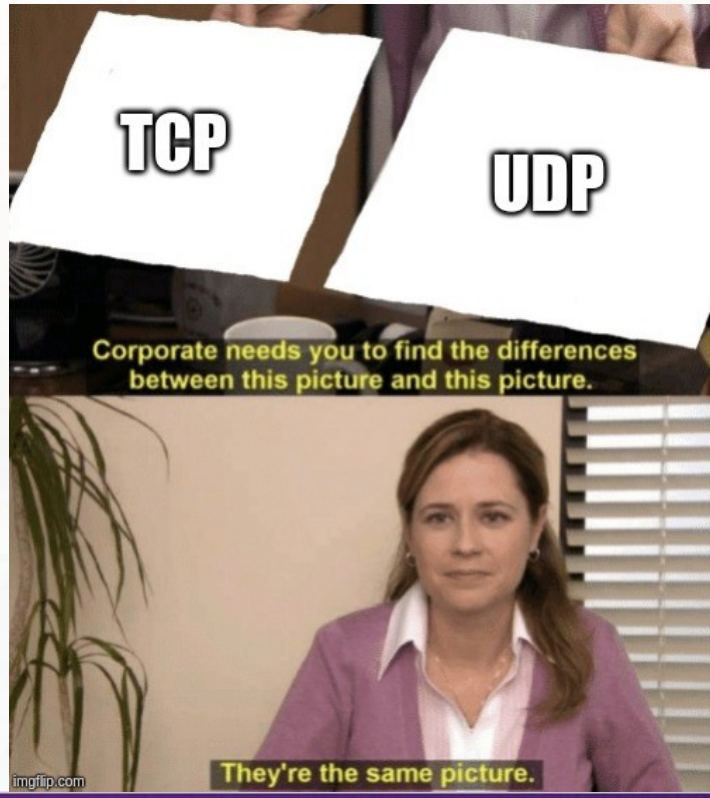

DNS
Req. Nome
Endereço IP
Req. Conexão
Confirmação ACK
Dados
Req. Desconexão
Confirmação
Cliente
Servidor

ăProtocolo UDP

- *User Datagram Protocol* (UDP):
  - Protocolo não orientado à conexão;
  - Não há garantia de entrega dos dados (não há mensagens de confirmação);
  - Perdas durante as transmissões não são tratadas por este protocolo;
  - Usado em redes com alta confiabilidade, onde as taxas de perda são baixas;

Diferença entre TCP e UDP

- TCP
  - Garante que o destinatário recebeu os pacotes;
  - Mais lento, pois faz checagem dos pacotes enviados e recebidos.
- UDP
  - Não garante que o destinatário recebeu os pacotes;
  - Mais rápido.




Sockets

Sockets

- Qual é o método para comunicarmos uma troca de mensagem entre um cliente e um servidor?
  - Via Rede
- Mas para isso precisamos de um mecanismo de serviço de transporte...

Sockets

- Um dos mecanismos é o **Socket**;
- Socket é a maneira mais popular de utilizar as funcionalidades de comunicação TCP/IP;
- Todos os mecanismos Sockets são gerenciados pela camada de transporte;
- Existem diversas APIs Sockets (Application Program Interface) e as mais populares são do ambiente Unix, bem como a WinSock do Windows.

Sockets

- Um dos mecanismos é o **Socket**;
- Socket é a maneira mais popular de utilizar as funcionalidades de comunicação TCP/IP;
- Todos os mecanismos Sockets são gerenciados pela camada de transporte;
- Existem diversas APIs Sockets (Application Program Interface) e as mais populares são do ambiente Unix, bem como a WinSock do Windows.

ăDefinição - Sockets

- Um **Socket** é um ponto final (*endpoint*) de um canal bidirecional de comunicação entre dois programas rodando em uma rede;
- Cada Socket tem os seguintes endereços de *endpoint*:
  - **Endereço local** (número da porta) que refere-se ao endereço da porta de comunicação para camada de transporte;
  - **Endereço global** (nome *host*) que refere-se ao endereço do computador (*host*) na rede.

ăComo funciona? - Analogia

1) Criação do Socket

O que é?

Processo 1:
Preciso de um Socket

Analogia

Conector
Telefone

- Processo solicita um socket. Como se pedisse ao SO uma tomada de telefone

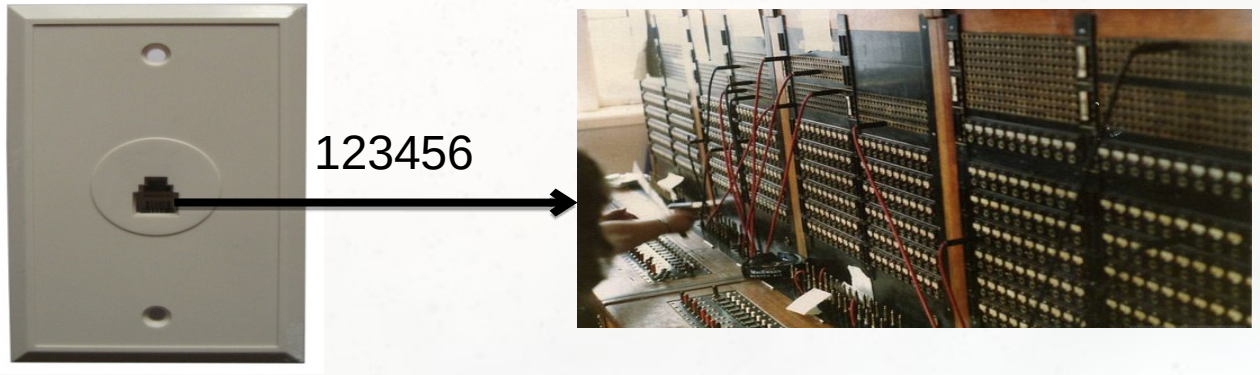
ăComo funciona? - Analogia

2) Binding

O que é?

Processo 1

Analogia



- Deve-se associar um endereço ao socket (o mesmo que associar um número de telefone à tomada)

ăComo funciona? - Analogia

3) Listening

O que é?

Processo 1:
“Listening to port”

Analogia

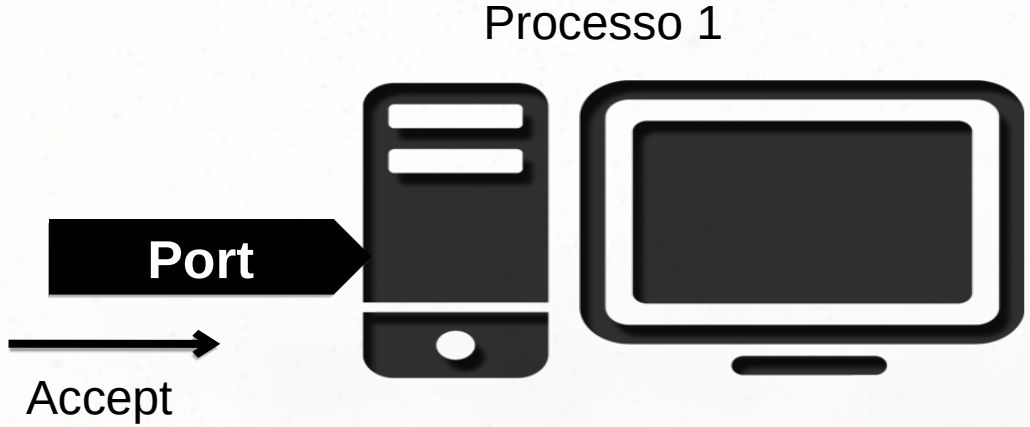
“Atendente”

- Processo fica aguardando por pedidos de conexão (esperando para ver se o telefone toca)

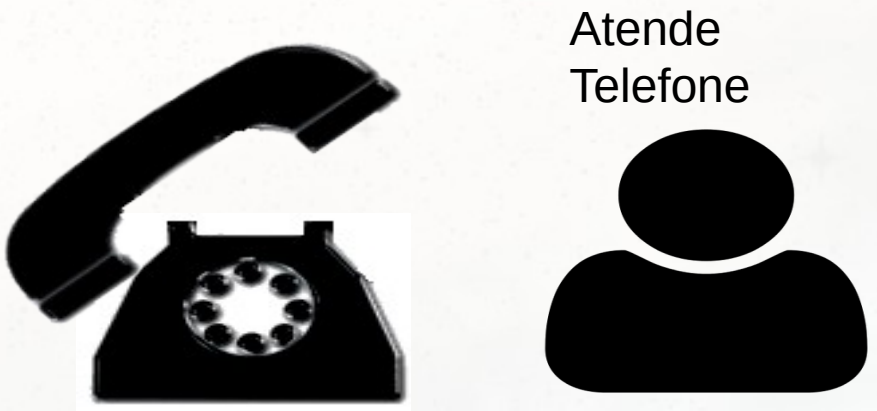
ăComo funciona? - Analogia

4) Accept

O que é?



Analogia

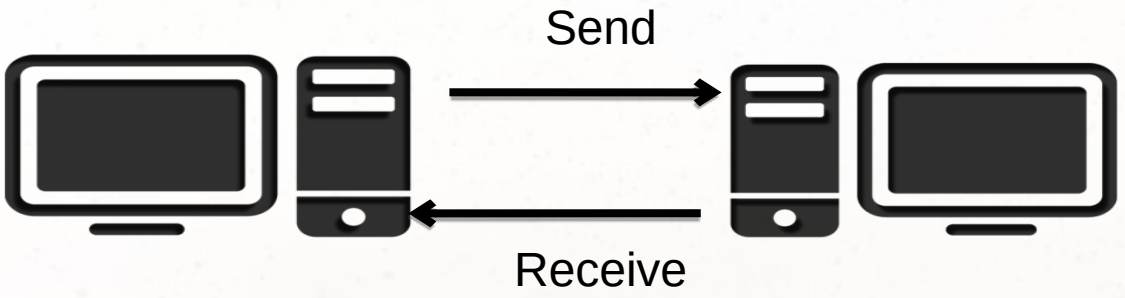


- Processo aceita pedidos de conexão (como se o telefone fosse atendido)

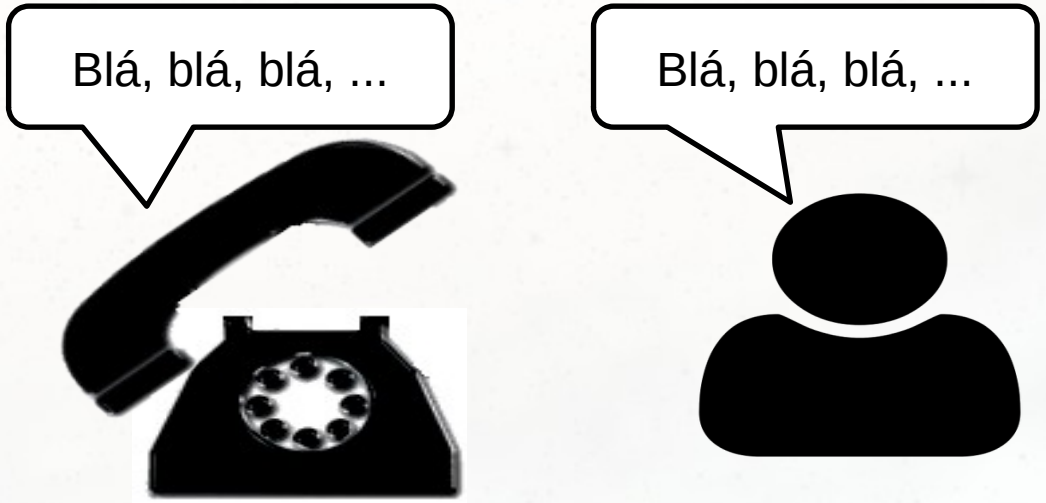
ăComo funciona? - Analogia

5) Send / Receive

O que é?



Analogia



- A conversação é realizada

ăComo funciona? - Analogia

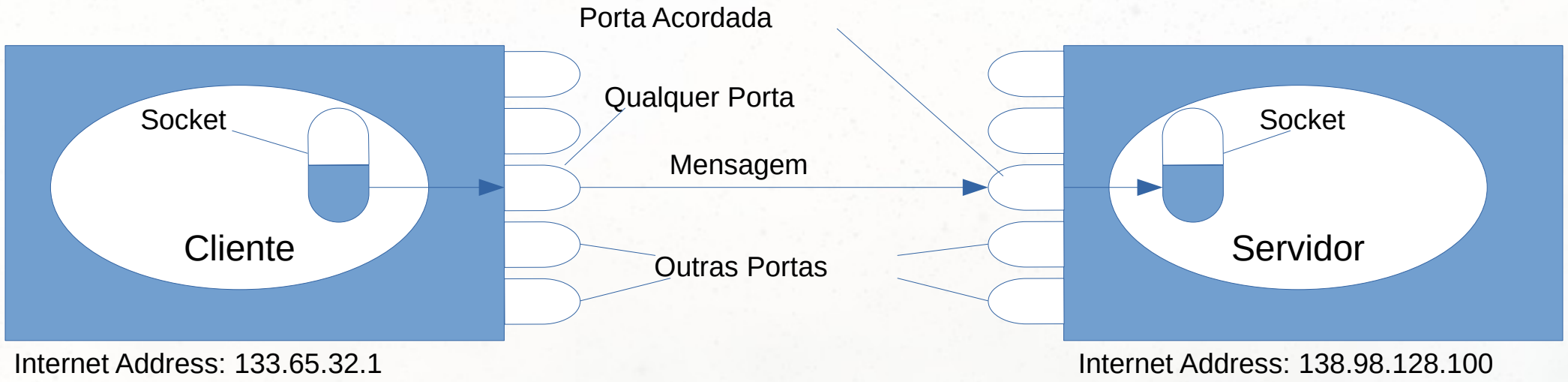
6) Desconexão

O que é?

Analogia

- Fim da comunicação. O canal é fechado.

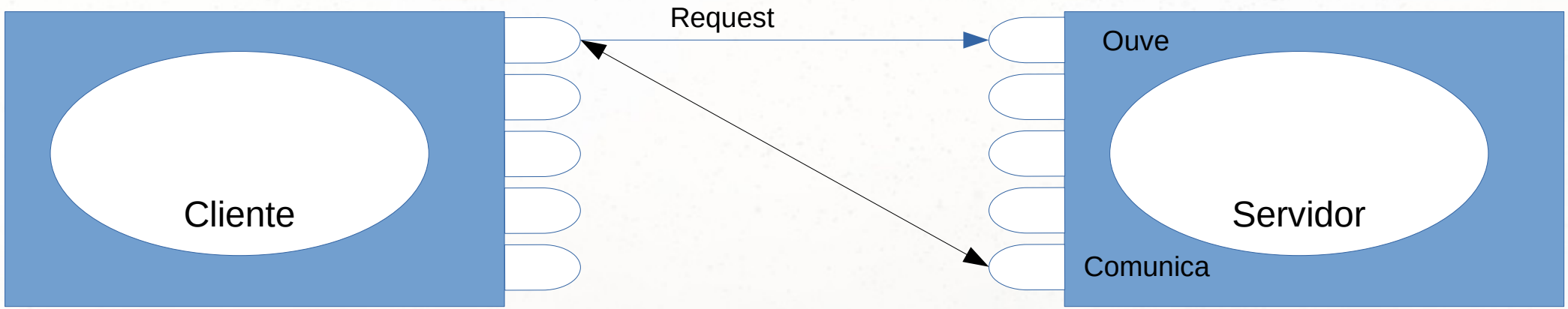
Sockets e Portas



ăConexão - Sockets

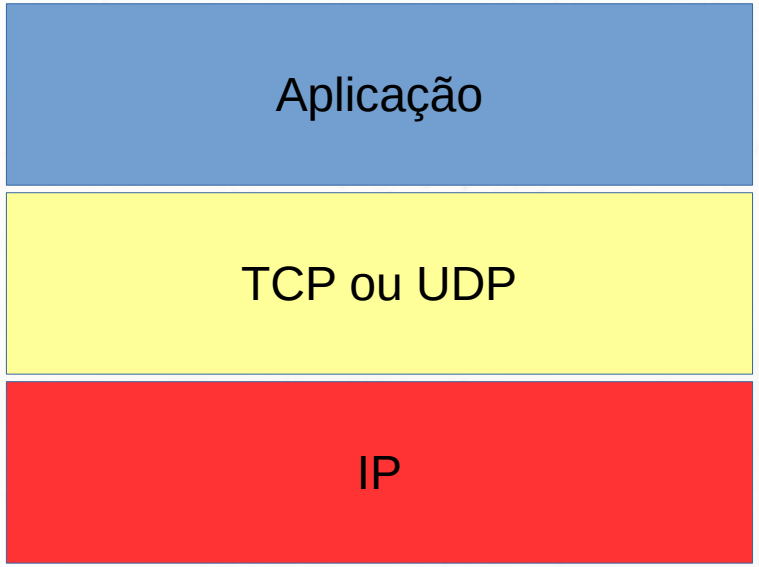
- Como acontece a conexão com os sockets?
  - O servidor apenas fica “ouvindo” o Socket aguardando um pedido de conexão do cliente;
  - O cliente sabe o nome do *host* e qual porta está associada à aplicação servidora;
  - Assim que o servidor aceitar a conexão, este cria um novo Socket (e consequentemente o associa a uma nova porta) e pode ficar esperando novas conexões no Socket original enquanto atende às requisições do cliente pelo novo Socket.

Conexão - Sockets



ăProtocolos TCP e UDP - Sockets

- Protocolos de Transporte TCP e UDP;
- Ambos utilizam a camada IP como camada de Rede.



API Sockets

- **Socket**: (cliente e servidor)
  - Cria um Socket e retorna um descritor;
  - O descritor é a referência para que as outras funções utilizem o Socket criado.
- **Bind**: (servidor)
  - Provê o número da porta que o servidor espera contato;
  - Função utilizada apenas pelo servidor, uma vez que associa um determinado endereço IP e porta TCP ou UDP para o processo servidor.

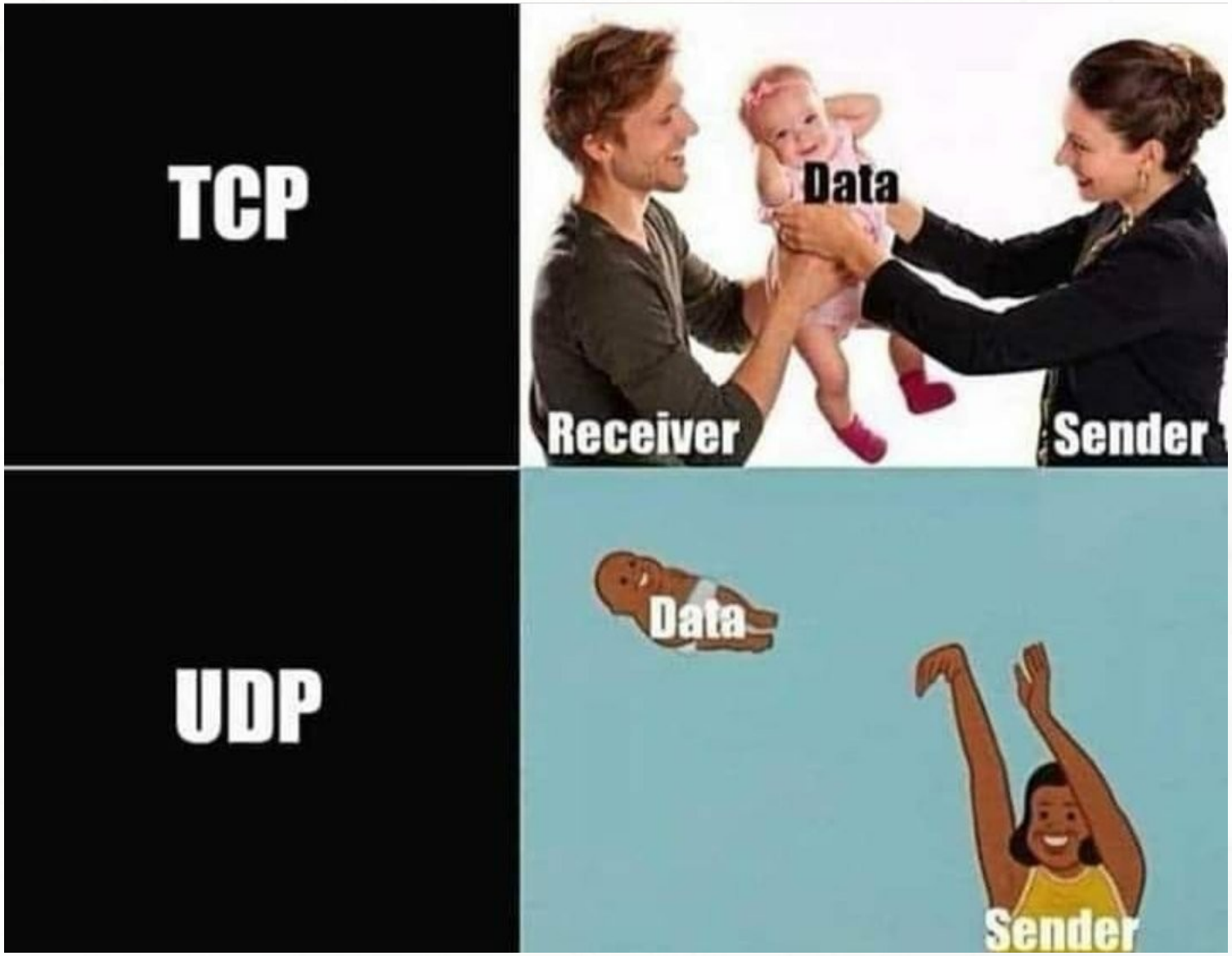
API Sockets

- **Listen**: (servidor)
  - Indica ao sistema operacional para colocar o Socket em modo de espera (passivo) para aguardar conexões de clientes.
- **Accept**: (servidor)
  - Cria um novo Socket a partir do estabelecimento de uma conexão para iniciar a comunicação (leitura e escrita).
- **Connect**: (cliente)
  - Função que o cliente utiliza para se conectar ao socket de um servidor.

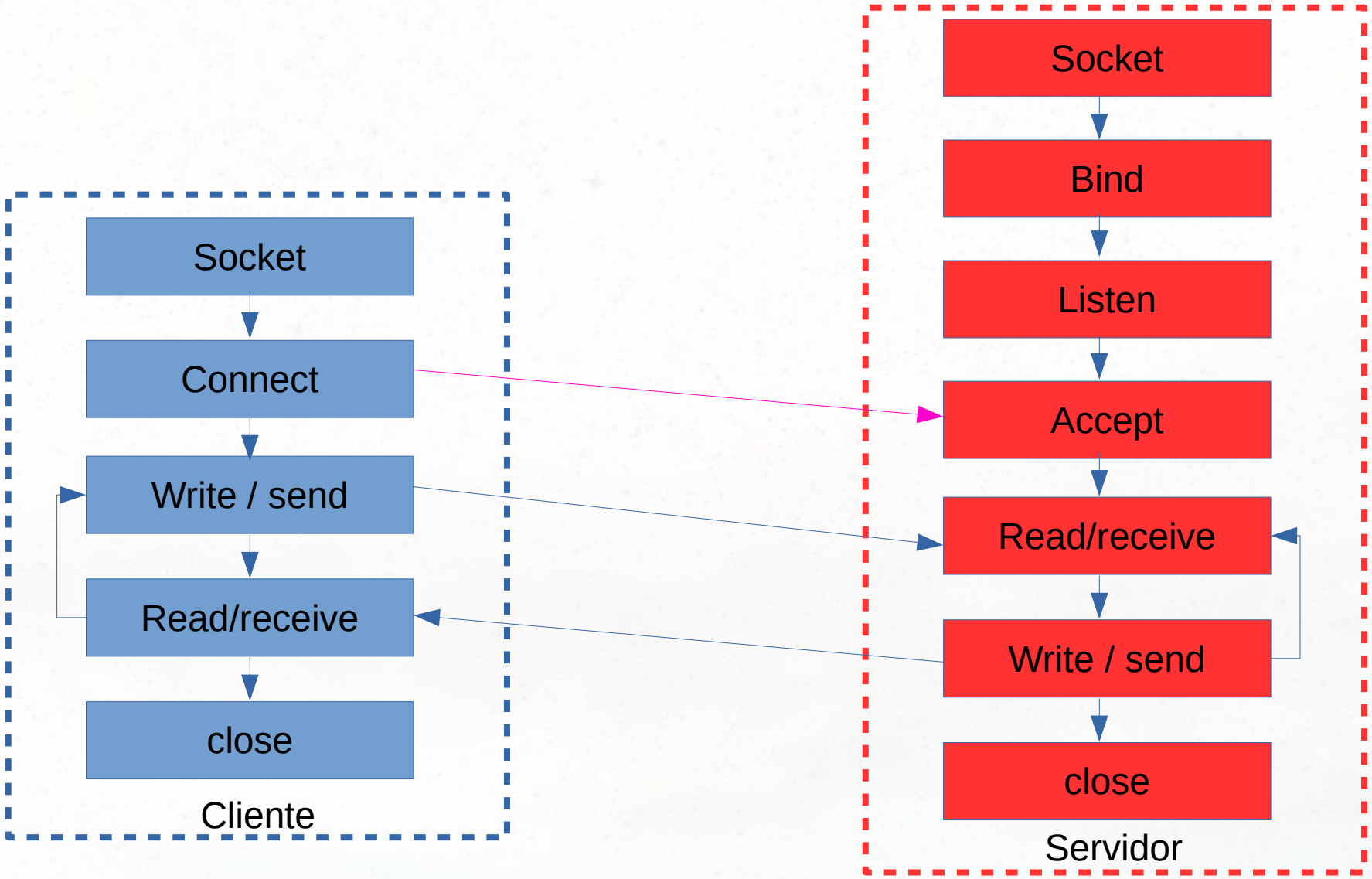
API Sockets

- **Read**:
  - Lê o conteúdo do buffer associado ao Socket.
- **Write**:
  - Escreve dados em um buffer associado ao Socket.
- **Close**: (cliente e servidor)
  - Informa ao sistema operacional para terminar o uso de um Socket.

Sockets TCP



Comunicação Cliente / Servidor – Sockets TCP



ăComunicação Cliente / Servidor – Sockets TCP

- Funções do Servidor:
  - Efetua a criação de um Socket;
  - Associa o Socket a um endereço local;
  - Aguarda por conexões da parte cliente;
  - Aceita conexões;
  - Segundo Socket Criado:
    - Lê requisições;
    - Opcionalmente envia resposta;
    - Fecha o Socket.
  - Fecha o Socket

ăComunicação Cliente / Servidor – Sockets TCP

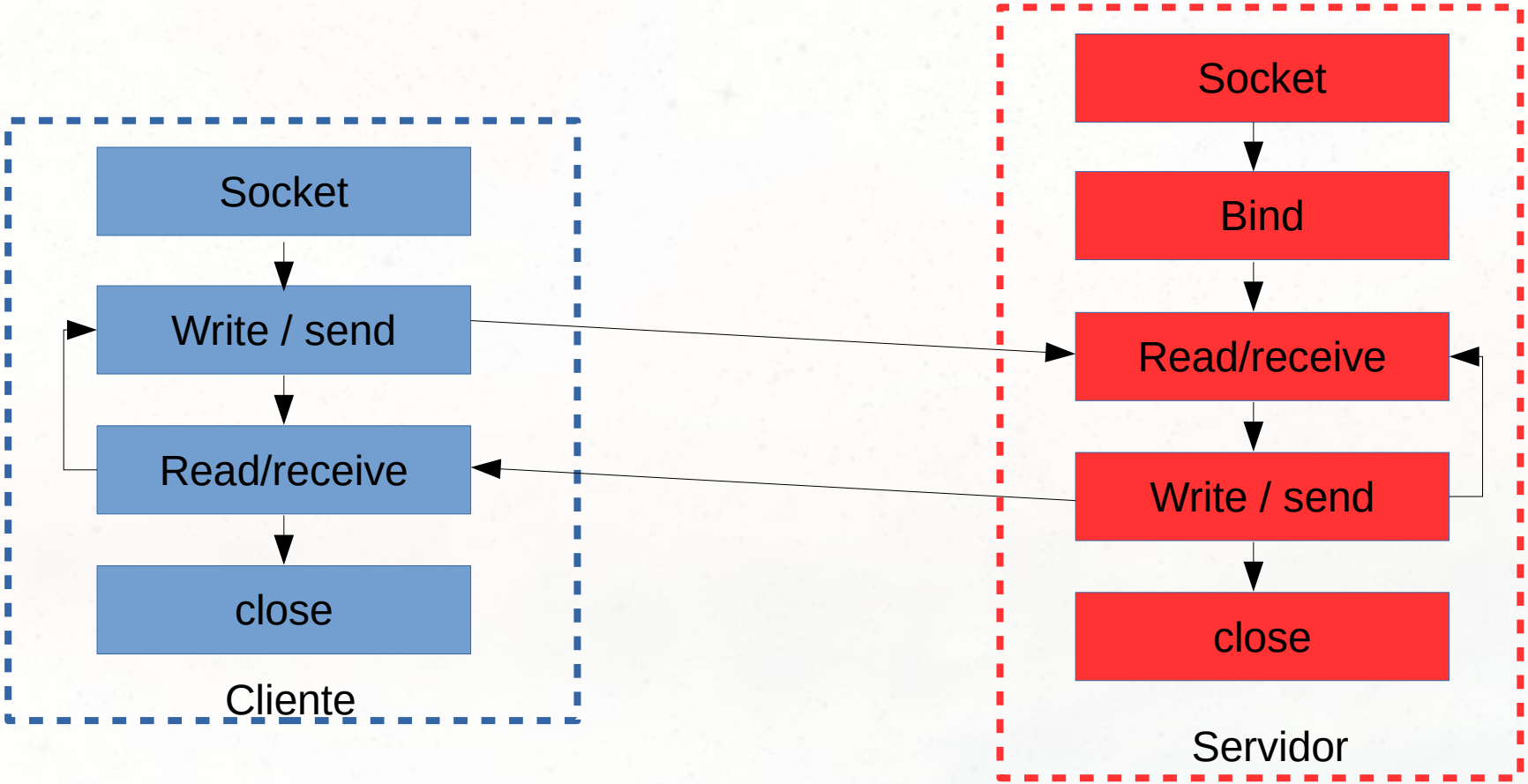
- Funções do Cliente:
  - Efetua a criação do Socket;
  - Estabelece a conexão;
  - Envia a requisição;
  - Opcionalmente aguarda resposta;
  - Fecha o Socket.

Sockets UDP

TCP

UDP

ăComunicação Cliente / Servidor – Sockets UDP



ăComunicação Cliente / Servidor – Sockets UDP

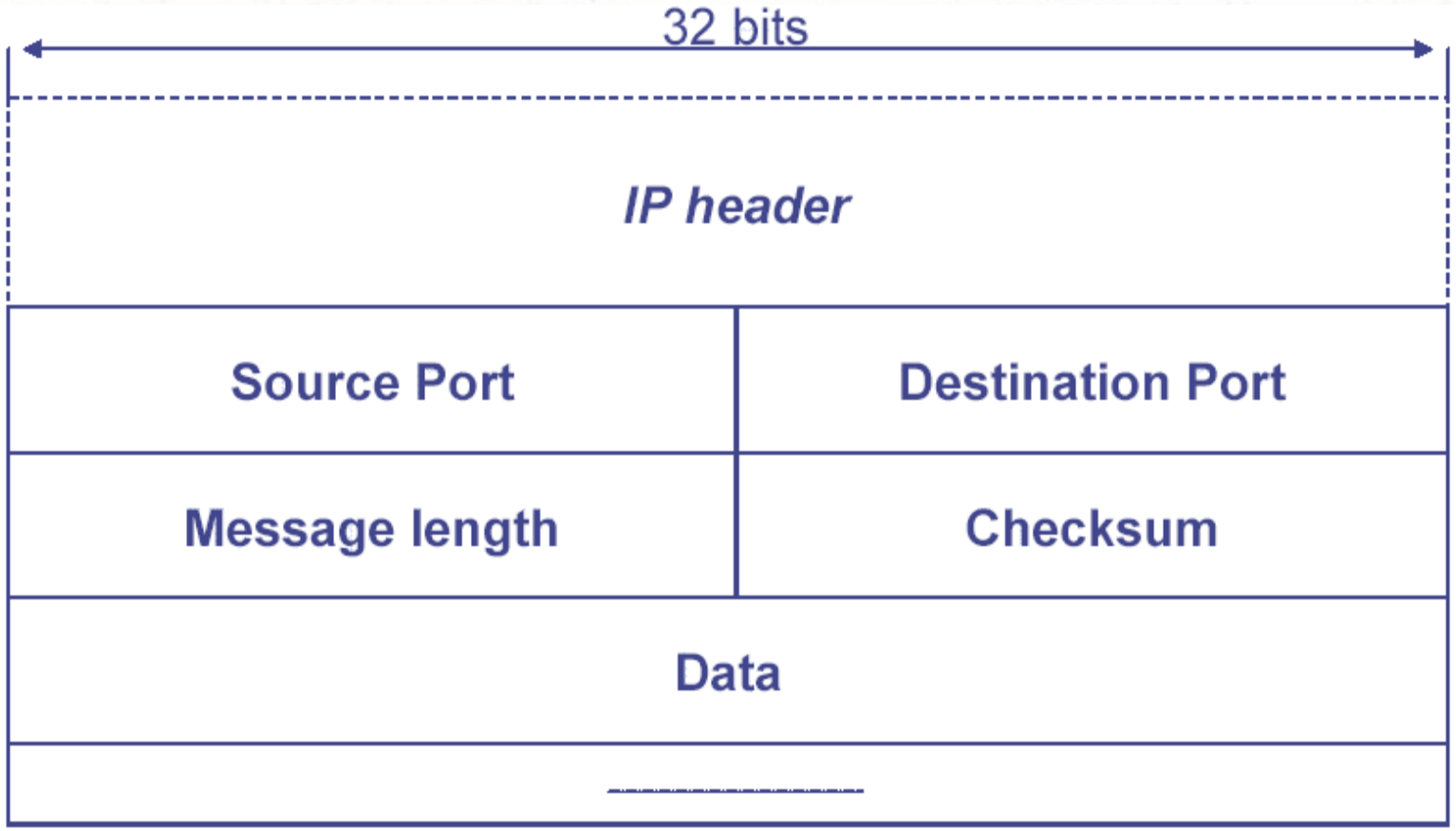
- Funções do Servidor:
  - Efetua a criação de um Socket;
  - Associa o Socket a um endereço local;
  - Ouve do cliente;
  - Lê requisições;
  - Opcionalmente envia resposta;
  - Fecha o Socket.
  - Perdas durante as transmissões não são tratadas por este protocolo;

ăComunicação Cliente / Servidor – Sockets UDP

- Funções do Cliente:
  - Efetua a criação do Socket;
  - Envia a requisição;
  - Opcionalmente aguarda resposta;
  - Fecha o Socket.

ăComunicação Cliente / Servidor – Sockets UDP

- Header UDP:



Sockets em Java

ăTipos de Sockets em Java

- Pacote java.net
  - java.net.Socket
  - java.net.ServerSocket
- Socket - utilizado em cada lado do canal de comunicação bidirecional.
- ServerSocket - responsável por ficar aguardando pedidos de conexão dos clientes

ăTipos de Sockets em Java

- Socket e ServerSocket são sockets do tipo StreamSocket
  - Utilizam protocolo TCP
  - Orientado à conexão (o servidor precisa aceitar o pedido de conexão do cliente)
- Outro tipo é o DatagramSocket
  - Utiliza protocolo UDP
  - Não é orientado à conexão

ăLeitura / Escrita em Sockets

- Sockets enviam e recebem dados na forma de bytes
- Cada conjunto de bytes é chamado stream
- Utilizam os seguintes métodos
  - getInputStream
    - read
  - getOutputStream
    - write

ăExemplo: Escrita em Sockets

```
String txt;  // string da mensagem a ser enviada
OutputStream out;// objeto para escrita no socket

try {
  out = socket.getOutputStream(); // habilita a escrita
  out.write(txt.getBytes()); // escreve (envia) mensagem
}
catch (Exception e)
{...}
```

Exemplo: Leitura em Sockets

```
InputStream in;          // objeto para leitura
InputStream byte btxt[];// array de bytes
int bt                   // objeto (número de bytes lidos)

try
{
  in = socket.getInputStream(); // habilita a leitura
  bt = in.read(btxt);           // le a cadeia de bytes
}
catch(Exception e)
{ ... }
```

ăSockets em Java

- Classe Conexao.java
  - utilizada pelo servidor e pelo cliente para o envio e recebimento dos dados
- Classe Servidor.java
  - é a classe que explicita o funcionamento do servidor
- Classe Cliente.java
  - classe que faz a conexão com o servidor para a troca de dados

ăExemplo com o Cliente.java

- Neste exemplo, o cliente:
  - Faz a conexão com o servidor,
  - Fica em um loop de 10 iterações enviando a mensagem “Cliente envia: Testando” para o servidor,
  - O servidor responde a cada envio com a resposta “Servidor envia : Olá Cliente” ,
  - Ao final o cliente fecha a conexão com o servidor.

ăConexão

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

public class Conexao {

  public static void send(Socket socket,String txt)
  {
    OutputStream out;
    try {
      out = socket.getOutputStream();
      out.write(txt.getBytes());
    }
    catch (Exception e)
    {
      System.out.println(”Exceção no OutputStream");
    }
  }
```

ăConexão

```
  public static String receive(Socket socket)
  {
    InputStream in;
    int bt;
    byte btxt[];
    String txt="";
    btxt = new byte[79];
    try
    {
      in = socket.getInputStream();
      bt = in.read(btxt);
      if (bt > 0) txt = new String(btxt);
    }
    catch(Exception e)
    {
      System.out.println("Excecao no InputStream: "+e);
    }
    return txt;
  }
}
```

ăCliente

```
public class Cliente {
  static Conexao c;
  static Socket socket;
  int i;
  public Cliente()
  {
    try {
      socket = new Socket("200.18.98.106",9600); }    // fase de conexão
    catch (Exception e) {
      System.out.println("Nao consegui resolver o host...");}
  }
  public static void main(String args[]){
     String msg = "Cliente envia : Olá Servidor";
     String texto;
     new cliente();
     for(i=0;i<10;i++){
        c.send(socket,msg);                          //
        texto = c.receive(socket);                   // fase de dados
        System.out.println(texto);                   //
     }
     socket.close();                              // fase de desconexão
  }
}
```

ăServidor

```
public class Servidor
{
  static ServerSocket serversocket;
  static Socket client_socket;
  static Conexao c;
  static String msg;

  public Servidor() {
    try {
      serversocket = new ServerSocket(9600);
      System.out.println(”Criando o Server Socket"); }

    catch (Exception e) {
      System.out.println("Nao criei o server socket..."); }
  }
```

Servidor

```
 public static void main(String args[]) {
     String texto;
     new servidor();
     c = new Conexao();
     if (connect()) {
       for(int i=0; i<10; i++){
          texto = c.receive(client_socket);                    //
          System.out.println(texto);                          // fase de dados
          c.send(client_socket,”Servidor Envia: Olá Cliente"); //
       }
     try {client_socket.close(); serversocket.close();}      // desconexao
     catch (Exception e) {...}
  }
  static boolean connect() {
    boolean ret;
    try {
      client_socket = serversocket.accept();              // fase de conexão
      ret = true; }
    catch (Exception e){
      System.out.println(”Não fez conexão"+e.getMessage());
      ret = false;
    }
    return ret;
  }
}
```

Dúvidas??

Referências

- Livro:
  - Sistemas Distribuídos: Conceitos e Projetos
    - Coulouris
      - Cap. 3 - Redes de Computadores e Interligação de Redes
        - 3.4 - Protocolos Internet
      - Cap 4 – Comunicação Interprocessos
        - 4.2 – API para protocolos da Internet
  - Sistemas Distribuídos Princípios e Práticas
    - Tanenbaum
      - Cap. 2 – Comunicação
        - 2.4 Comunicação Transiente orientada a mensagem

Obrigado!

gustavo.custodio@anhembi.br

https://gustavotcustodio.github.io/sdmobile.html


ecossistema
ânima