Prof. Rogério Santos Pozza

Universidade Tecnológica Federal do Paraná
Campus Cornélio Procópio

Java Aplicada em Redes de Computadores

12 de abril de 2018

Roteiro da disciplina

Conceitos e Redes TCP

- Servidor básico de redes
- Atividades Concorrentes
- Aplicação exemplo
- Exercícios

Redes UDP

- Redes com UDP
- Aplicativo exemplo
- Exercícios

Java RMI

- Conceito de Invocação de Métodos Remotos
- Aplicativo exemplo
- Exercícios

Redes de Computadores

- As redes de computadores permitem compartilhar recursos computacionais independentemente da posição geográfica, tais como:
  - Compartilhamento de arquivos
  - Compartilhamento de impressora
  - Compartilhamento de conexão à Internet
  - **Aplicações distribuídas**

- Para se comunicar, os computadores utilizam protocolos de comunicação (conjunto de formatos e regras usados para transmitir informação)

- Computadores distintos devem obedecer a estas **regras** e **formatos** para se comunicar

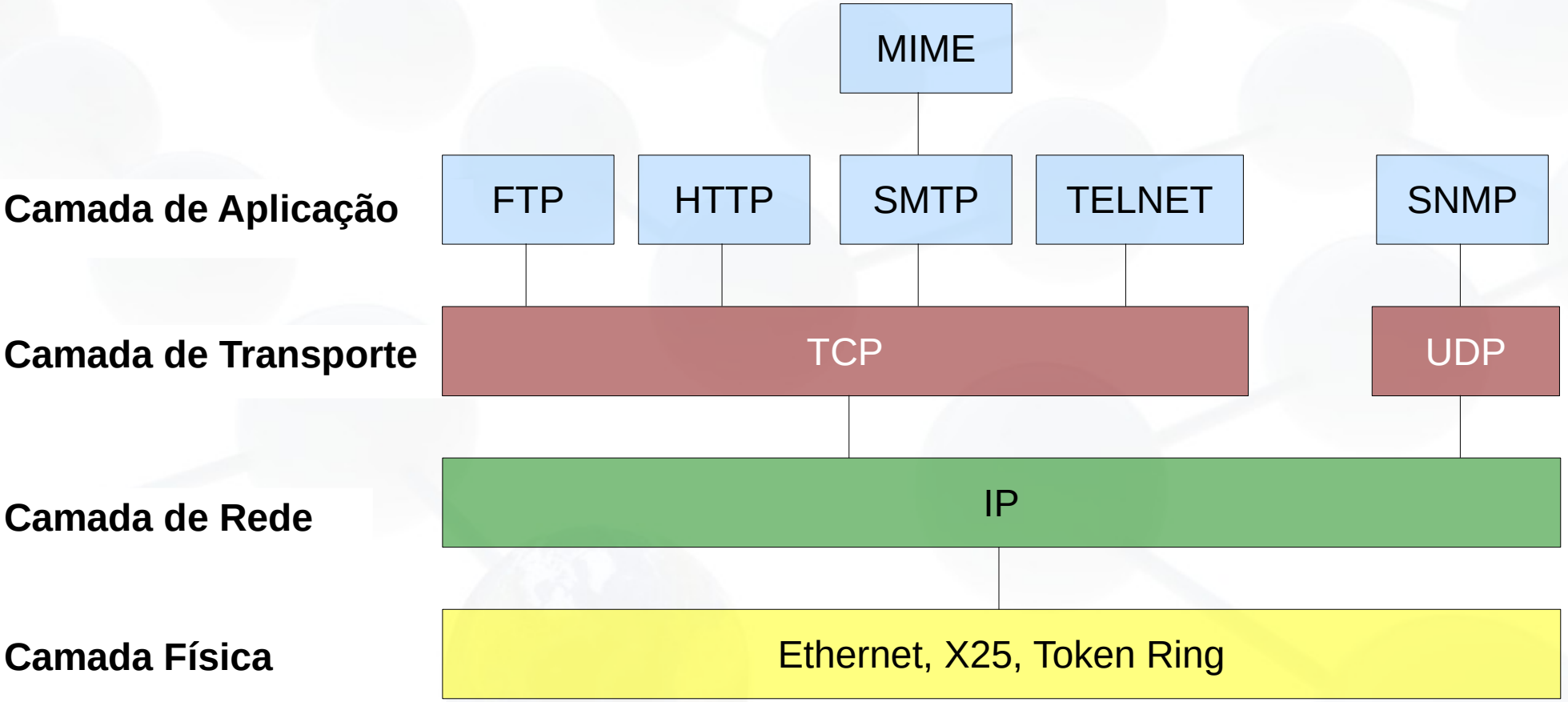
Redes de Computadores

Regras e formatos

Hello, nice to meet you!

Nice to meet you too!

Camadas de rede

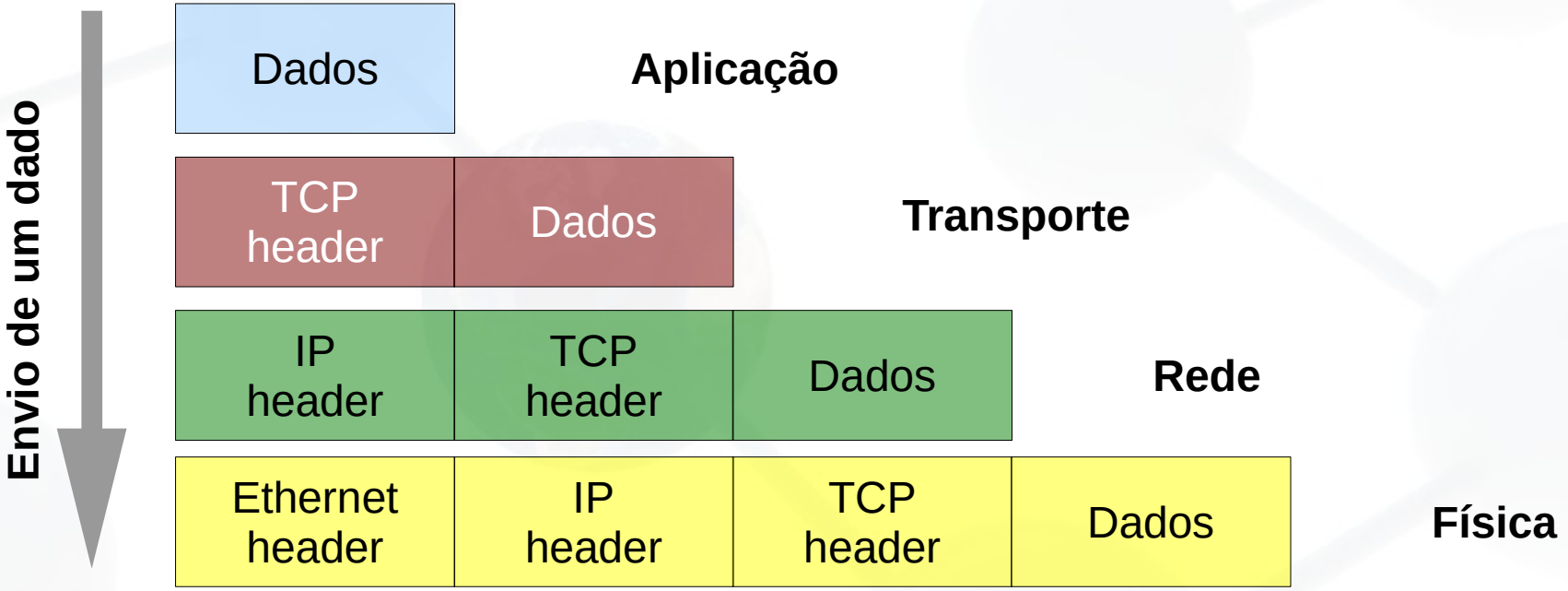


TCP – Transmission Control Protocol
HTTP – Hypertext Transfer Protocol
SNMP – Single Network Management Protocol
MIME – Multi-purpose Internet Mail Extension

SMTP – Single Mail Transfer Protocol
FTP – File Transfer Protocol
IP – Internet Protocol
UDP – User Datagram Protocol

Redes de Computadores

- No processo de transmissão de dados em uma rede **TCP/IP** os dados são divididos em grupos chamados de **pacotes**
- Cada camada adiciona alguns dados a mais no **início** de cada pacote para permitir que o mesmo chegue ao **destino**
- Os dados adicionados são chamados de ***headers* (cabeçalhos)**



Redes de Computadores – TCP e UDP

TCP

- Serviços confiáveis
  - Sem perdas de dados na rede
  - Garantia de ordenação dos pacotes de dados enviados
- Possibilidade de utilização de fluxos de dados(*DataStreams*)

Desvantagens:
- É mais lento do que o modo orientado a datagrama.
- Comportamento do servidor diferente do comportamento do cliente

UDP

- Serviços não confiáveis
  - Mensagens podem ser perdidas
  - Ordem das mensagens não é garantida
- Cada mensagem é um datagrama:

Vantagem:
- É muito mais rápido do que o modo orientado a conexão (TCP);

Redes de Computadores – Sockets

- Em sistemas comerciais geralmente existe a necessidade de um computador (**cliente**) usufruir de recursos oferecidos por outro (**servidor**), como, por exemplo*:*
  - consulta em base de dados
  - Validações remotas
  - *Ex.: consulta a preços de produtos*

- Para tanto, a comunicação entre *cliente-servidor* deve ser **confiável.** Nenhum dado pode ser perdido e a sequência de recebimento pelo cliente deve estar de acordo com a sequência de envio do servidor.

Redes de Computadores – Sockets

- O Java oferece suporte a ***sockets*** que são mecanismos de comunicação interprocessos (IPC - *InterProcess Communication*), que podem estar numa mesma máquina ou em máquinas diferentes conectadas em rede

- Esta biblioteca é responsável pela programação em rede, contendo ***sockets unicast*** (TCP e UDP - comunicação ponto a ponto) e ***multicast*** (UDP - grupo de *hosts* que se comunicam por um mesmo endereço IP)

Redes de Computadores
- Modo Orientado à Conexão

1. Servidor tem uma porta e fica aguardando conexões nesta porta

2. Cliente deve saber qual a máquina servidora e porta que está aguardando requisições

3. Cliente solicita conexão em ***host/porta***

4. Se nenhum problema ocorrer, o servidor aceita a conexão gerando um ***socket*** (canal de comunicação) em uma porta qualquer do lado do servidor

5. Em Java a comunicação confiável (modo orientado a conexão) acontece por ***sockets*** utilizando duas classes: ***Socket*** (de dados) e ***ServerSocket*** (do servidor)

Redes de Computadores
- Modo Orientado à Conexão

Servidor

Porta 80

Aguardando conexões...

Solicita conexão (IP/porta)

10001010001001110101101 00

Canal de comunicação

Socket

Cliente

Redes de Computadores
- Modo Orientado à Conexão

Principais Características

- Protocolo da camada de transporte
- Complexo
- Orientado à conexão
- Confiável
- Detecção de Erros
- Controle de Congestionamento
- Menor desempenho que UDP

Redes de Computadores
- Modo Orientado à Conexão

- O protocolo TCP usa duas abstrações chamadas de s*ockets* e portas

- Em Java, um socket é um objeto que sabe como enviar e receber dados de (e para) um outro computador através de uma rede

- Java *Socket* encapsula as regras de transmissão de dados por meio do protocolo TCP, e isenta o usuário de conhecer o padrão de empacotamento dos dados

Redes de Computadores
- Modo Orientado à Conexão

- Servidores são computadores que oferecem serviços de rede através de **portas** de comunicação
  - *FTP, SSH, WEB, Jogos, VNC, SMTP, POP, etc.*

- Cada serviço é disponibilizado em uma porta:

Serviço	**Porta**
FTP	21
SSH	22
HTTP	80
VNC	5900
Jogos	20000 ~ 65535
SMTP	25
POP	110

Redes de Computadores
- Modo Orientado à Conexão

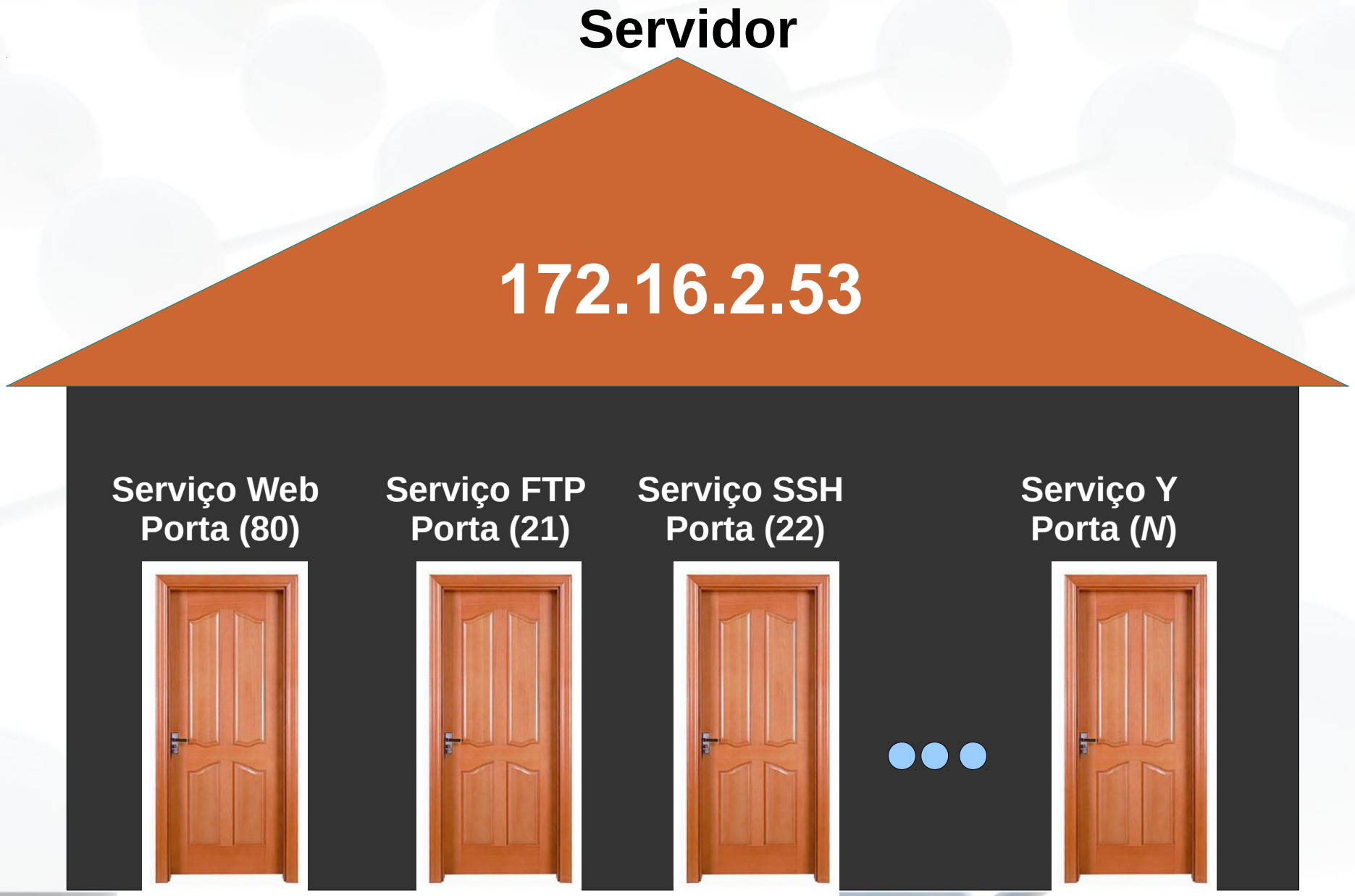
- O protocolo TCP dispõe 65536 portas de comunicação

- Qualquer comunicação em rede baseada no protocolo TCP precisa obrigatoriamente definir um endereço e uma porta de comunicação
  - *Por exemplo, o serviço HTTP por padrão atende pela porta 80*

- Entretanto, existem outros servidores HTTP que atendem requisições em portas diferentes da 80.
  - *Exemplo: Tomcat Application Server atende pela porta 8080*

- A porta de um serviço de rede pode ser modificado através de um arquivo de configuração

Redes de Computadores
- Portas de Comunicação



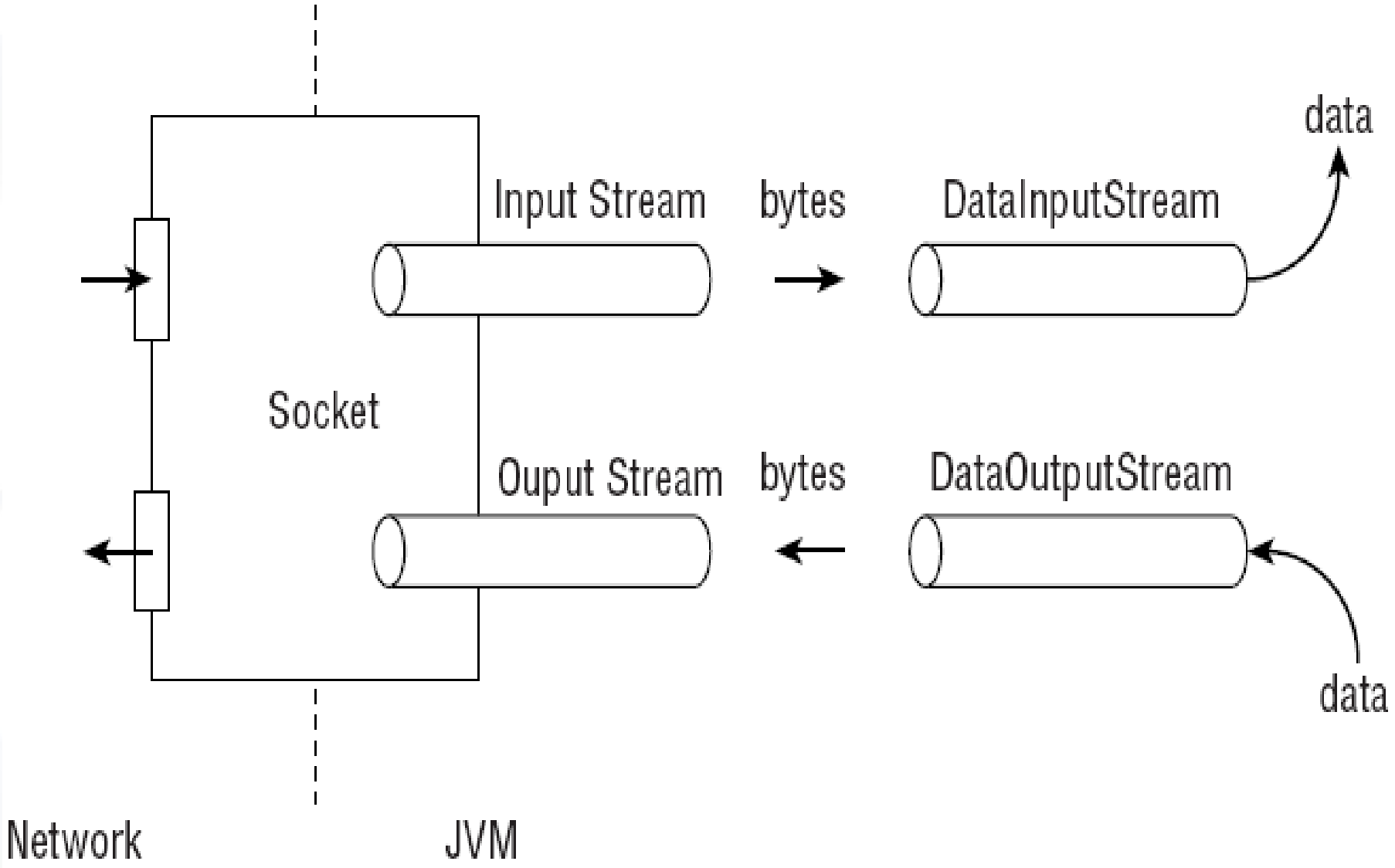
Redes de Computadores
- Modo Orientado à Conexão

• Um endereço IP é formado por 32 bits divididos em 4 octetos:

• Endereços com início 127.x.x.x são utilizados para *loopback*, representando a própria máquina

• Esses endereços podem ser organizados em sub-redes

Streams


data
Input Stream
bytes
DataInputStream
Socket
Ouput Stream
bytes
DataOutputStream
data
Network
JVM

Java e Redes

• A linguagem Java oferece a classe ***java.net.Socket*** para a comunicação entre dois redes

• Uma vez que um Socket usa o protocolo TCP, logo esse herda todas as características do protocolo, inclusive a confiabilidade de entrega de pacotes

• Um socket é uma conexão bilateral entre dois computadores

• Construtor:

```
public Socket(String ipserver, int porta) throws
IOException;
```

Java e Redes

• Após a instância do objeto da classe Socket, um canal de comunicação é aberto para que dados possam ser transmitidos de um *host* a outro

• Se o *socket* não conseguir se comunicar, uma **IOException** é lançada

• Para ler dados de um socket é preciso instanciar um objeto DataInputStream passando a referência de entrada de dados do socket por meio do método getInputStream(), conforme exemplo:

```
Socket socket = new Socket(host, porta);
DataInputStream entrada = new DataInputStream(socket.getInputStream());

entrada.readUTF();  // leitura de String
entrada.readInt();     // leitura de int
```

Java e Redes

- O envio de dados é realizado pela instância de um objeto DataOutputStream, instanciado pelo retorno do método getOutputStream() de um socket:

```
Socket socket = new Socket(host, porta);
DataOutputStream saida = new DataOutputStream(socket.getOutputStream());

saida.writeUTF(nome);  // envia uma String
saida.writeInt(idade);     // envia um int
```

Java Server Sockets

• Até o presente momento, vimos como se conectar a um serviço. A partir daqui, será apresentado como criar um serviço baseado em Socket.

• A classe ***ServerSocket*** cria um canal de comunicação e permite agudar conexões de um host cliente utilizando um **Socket**

• O construtor mais utilizado para instanciar um objeto ServerSocket é:

```
public ServerSocket(int porta) throws IOException;
```

Java Server Sockets

- Se o ***ServerSocket*** não conseguir se instalar na porta, será lançada uma IOException

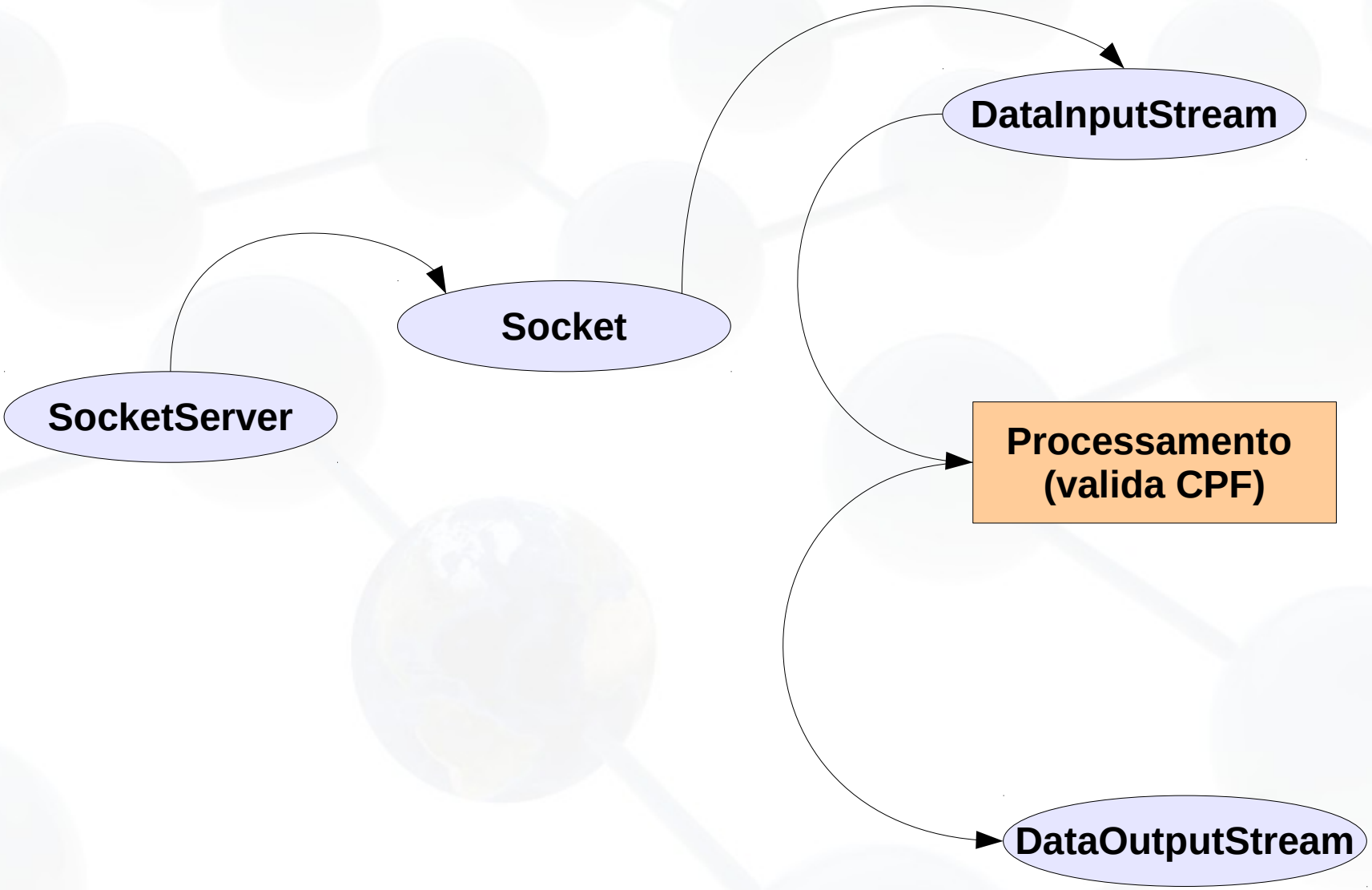
- Diferente do ***Socket***, após o construtor executar, o ***ServerSocket*** deverá chamar o método **`accept()`** para aguardar uma conexão

- *O* método **`accept()`** retorna um ***socket***. Portanto para se criar um serviço com Socket, temos:

Java Server Sockets

```
SocketServer server = new SocketServer(porta);

try {

    Socket socket = server.accept();
    DataInputStream entrada = new DataInputStream(socket.getInputStream());
    DataOutputStream saida = new DataOutpuStream(socket.getOutputStream());

    String dado = entrada.readUTF();

    //processa o dado

    saida.writeUTF(resultado);

} catch(IOException e) {
  //Tratamento

} finally {
    socket.close();
}
```

Modelo de comunicação em Sockets em Java!



Modelo de comunicação em Sockets em Java!