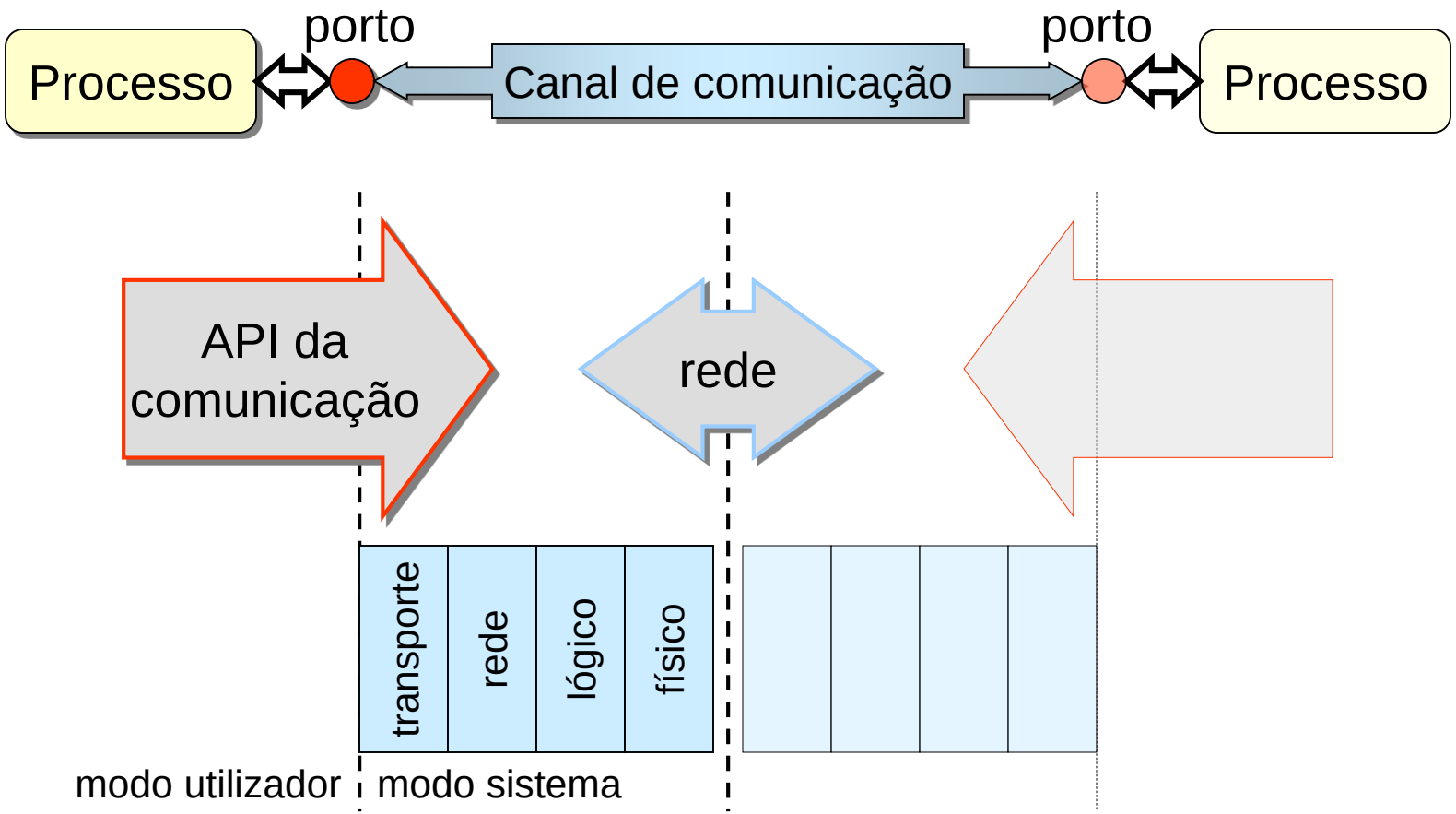
Sistemas Distribuídos

A rede que interliga o sistema distribuído

Objetivo

- Rever o modelo de arquitetura das redes
- Rever a forma de programação distribuída baseada em mensagens (aulas práticas)
- Compreender o modelo cliente-servidor e as suas evoluções
- Perceber as limitações do modelo de programação baseado em mensagens e a evolução para o RPC

Programação da comunicação: modelo



Redes de Dados

- Fornecer uma base mínima de compreensão das redes de dados
  - Arquitectura
  - Organização
  - Protocolos

Revisão

- Identificar os aspectos relevantes das redes de dados na concepção de sistemas distribuídos

Características habituais das Arquiteturas Físicas

- Redes Locais
  - Transmissão em difusão
  - Largura de Banda muito grande
  - Topologias de bus ou anel
  - Encaminhamento trivial
  - Menor escalabilidade
  - Maior tolerância a faltas

- Redes de Larga Escala
  - Transmissão ponto a ponto
  - Banda passante com limitações mas tecnologias tradicionais
  - Topologia malhada com redundância
  - Necessidade de encaminhamento
  - Grande escalabilidade
  - Menor tolerância a faltas

Arquitectura Lógica

- Porquê uma arquitectura Lógica nas redes?

- A arquitectura lógica define as propriedades da rede, adequadas ao seu campo de aplicações
  - Tipo de endereçamento
  - Desempenho
  - Garantia de entrega de mensagens
  - Ordenação das mensagens
  - Tolerância a faltas
  - Endereçamento em difusão
  - …
- A mesma arquitectura lógica pode ser realizada (com maior ou menor facilidade) sobre várias arquitecturas físicas

Modelo de Referência

- Um Modelo de Referência, ou Família de Protocolos, define as características lógicas e físicas das redes:
  - Normalmente divididos em níveis
  - Os níveis são independentes mas estão relacionados
  - Permitem várias realizações compatíveis

- Cada nível corresponde a um nível de abstração necessário no modelo
- Cada nível possui funções próprias e bem definidas
- As funções de cada nível foram escolhidas segundo a definição dos protocolos normalizados internacionalmente
- As fronteiras entre níveis devem ser definidas de modo a minimizar o fluxo de informação nas interfaces
- O número de níveis deve ser suficientemente grande para que funções distintas não precisem ser colocadas na mesma nível
- E, ao mesmo tempo, suficientemente pequeno que não torne a arquitectura difícil de controlar.

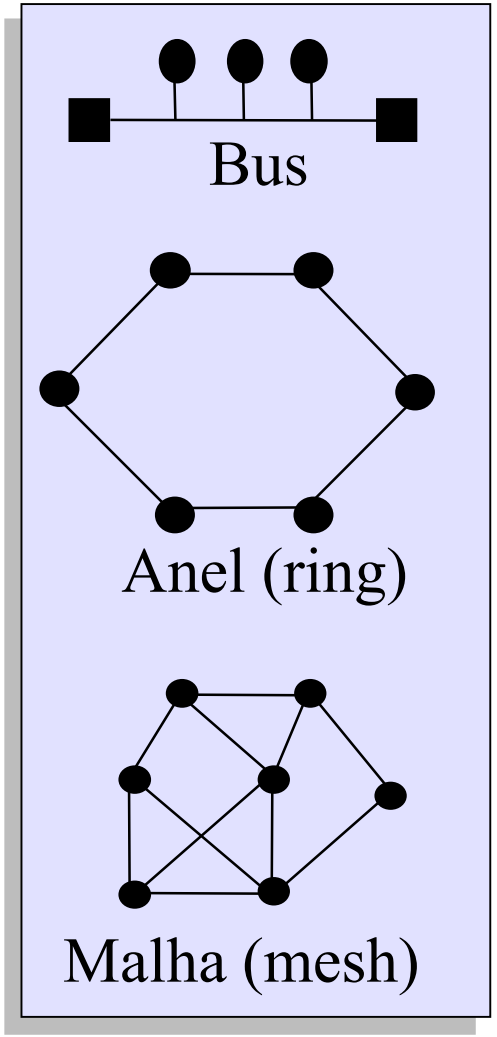
Modelo OSI

1. Físico (Ethernet, FDDI, B8ZS, V.35, V.24, RJ45)
2. Lógico / Ligação Dados (PPP, FDDI, ATM, IEEE 802.5/802.2 802.3/802.2, HDLC, Frame Relay)
3. Rede (AppleTalk DDP, IP, IPX)
4. Transporte (SPX, TCP, UDP)
5. Sessão (NFS, NetBios, RPC, SQL)
6. Apresentação (ASCII, EBCDIC, TIFF, GIF, PICT, JPEG, MPEG)
7. Aplicação (NFS, SNMP, Telnet, HTTP, FTP)

Open Systems Interconnection (OSI)
ISO, ITU-T
1977

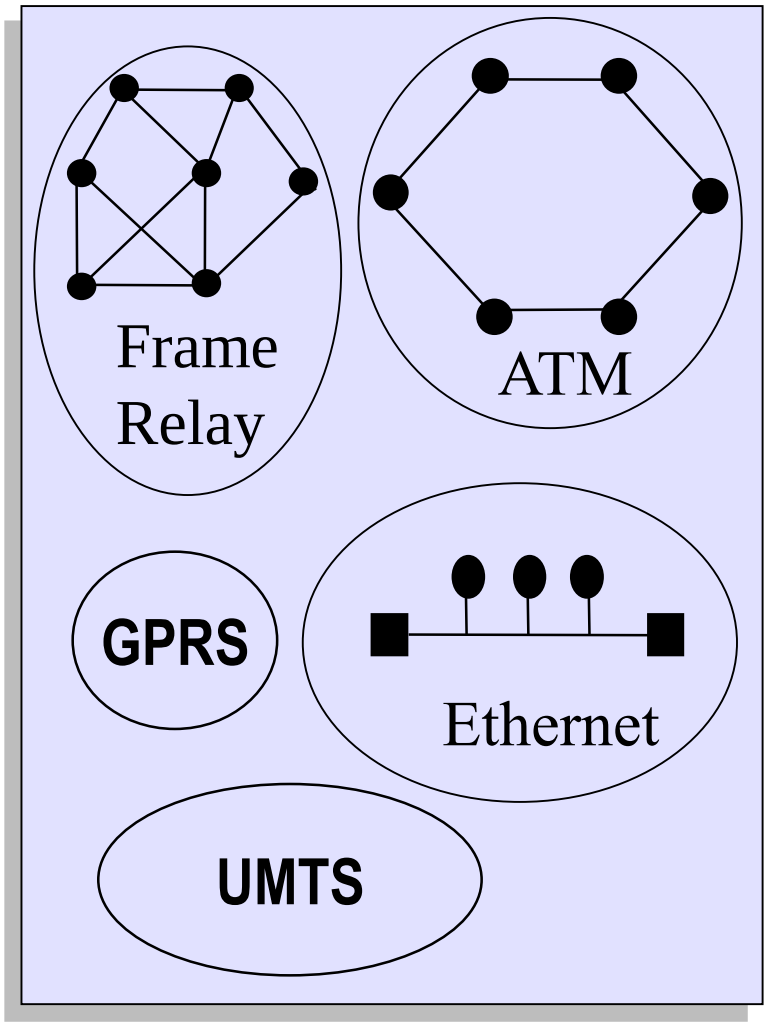
OSI - Nível Físico

- Funções: conseguir transmitir **1 bit** de informação sobre meio físico de interligação
  - Velocidade de propagação, atenuação, imunidade ao ruído, etc.
- Nível Físico define:
  - Níveis eléctricos do sinal, características temporais
  - Protocolos de codificação, baseados no funcionamento da rede (taxa de erros, recuperação de relógio, …)
  - Placas de interface (network cards)
    - Interface eléctrica
    - Aspectos mecânicos dos conectores



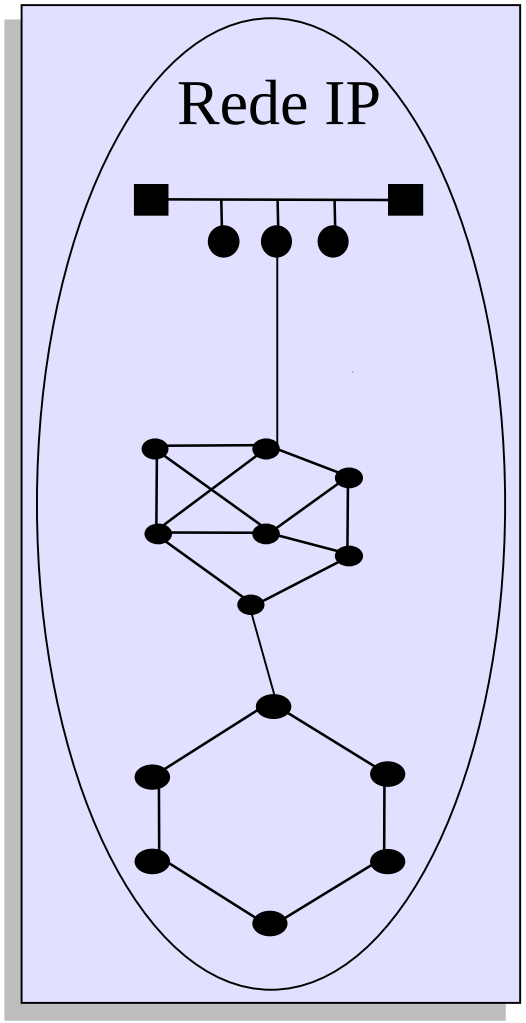
OSI - Nível Lógico ou Ligação de Dados

- Funções: transmissão de pacotes, ou tramas, entre máquinas ligadas à mesma rede física
- Nível Lógico define:
  - Delimitadores de trama
  - Endereço físico do destinatário
  - Multiplexagem do meio de transmissão (emissor)
  - Deteção do endereço do destinatário (receptor)
  - Definição da unidade básica de informação (bit, octeto)
  - Recuperação de erros de transmissão
  - Controlo de fluxo



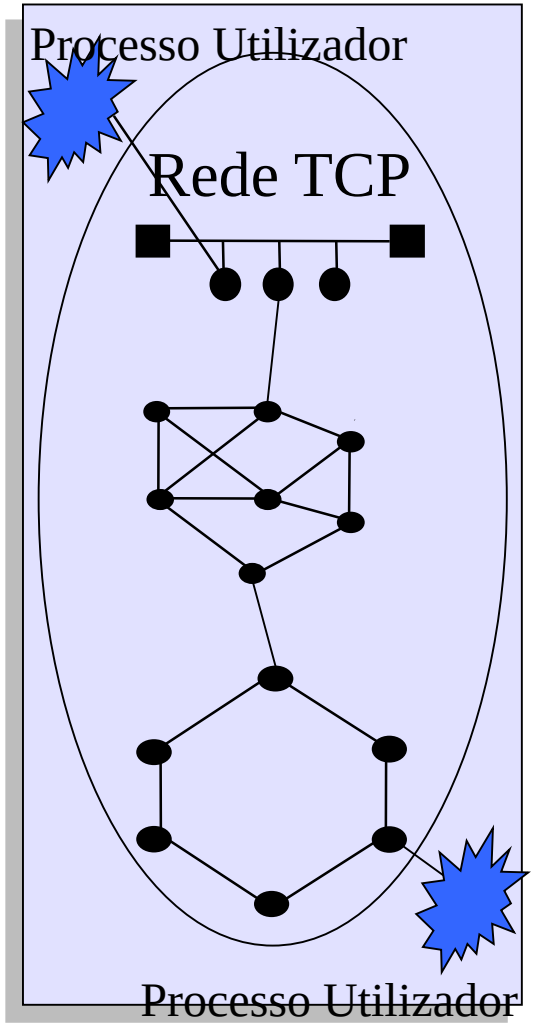
OSI - Nível Rede

- Funções: interligar máquinas independentemente da rede física a que estão ligadas
- Uma rede lógica passa a ser composta pela interligação de várias redes físicas
- Nível Rede define:
  - Formato dos pacotes de dados
  - Mecanismos de encaminhamento entre redes
    - Fundamental para redes malhadas
    - Normalmente baseados em tabelas de encaminhamento
  - Protocolo de rede OSI: X.25
    - Com ligação, sequencialidade, controlo de fluxo
  - Protocolo de rede Internet: IP
    - Sem ligação nem garantias de qualidade



Nível Transporte

- Funções: oferecer um serviço de transmissão de informação que permita a comunicação entre utilizadores finais
- Características
  - Com ou sem ligação
  - Comunicação fiável
    - Garantia de entrega
    - Garantia de ordem
  - Segmentação
  - Controlo de fluxo
  - Notificação de excepções na comunicação



Níveis superiores (aplicação)

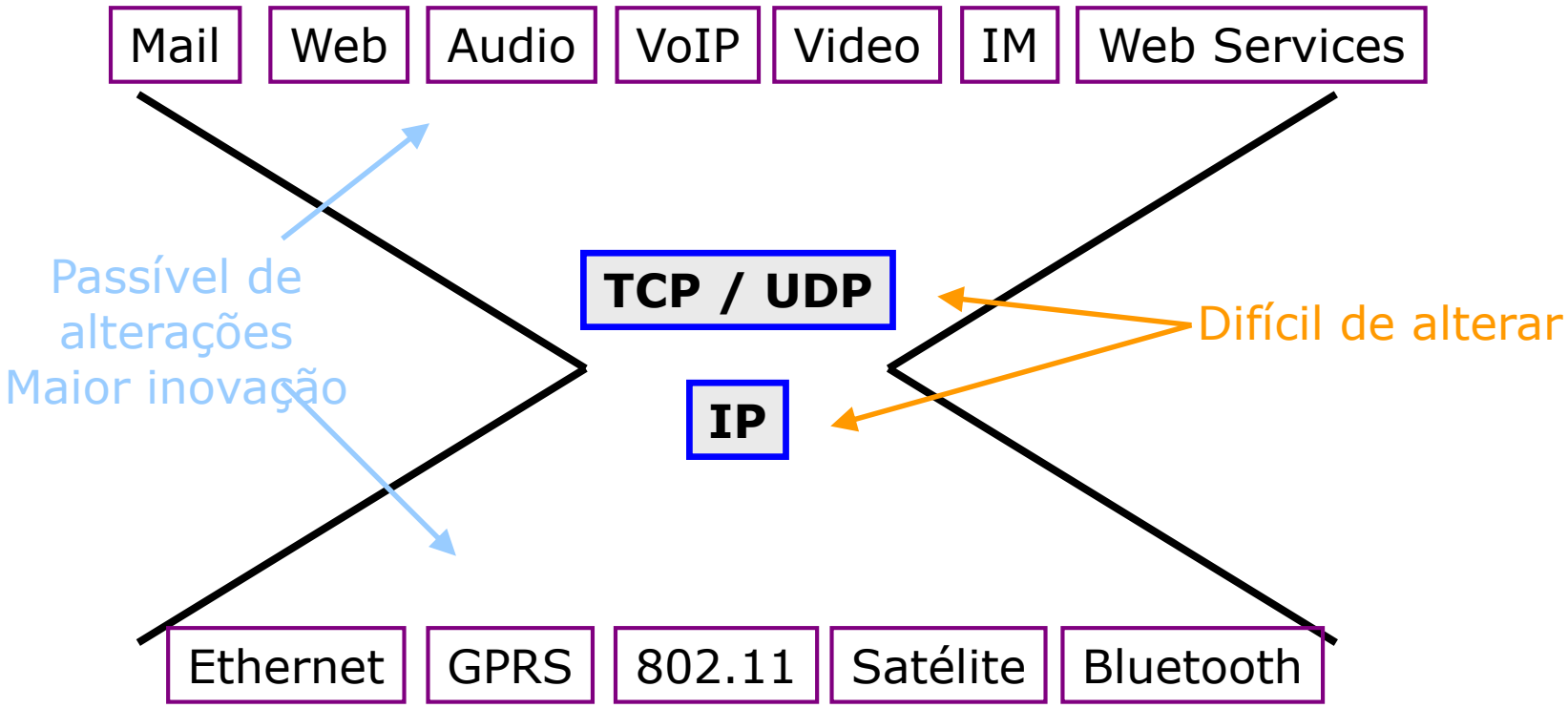
Exemplos

- telnet, rlogin, Winrdp- aplicações de terminal remoto
- ftp, samba – Transferência de ficheiros
- SMTP – Correio electrónico

Problemas?

- Cada aplicação possui um protocolo próprio
- Dificulta a utilização do protocolo por terceiros
- Desempenho
  - Executado em modo utilizador

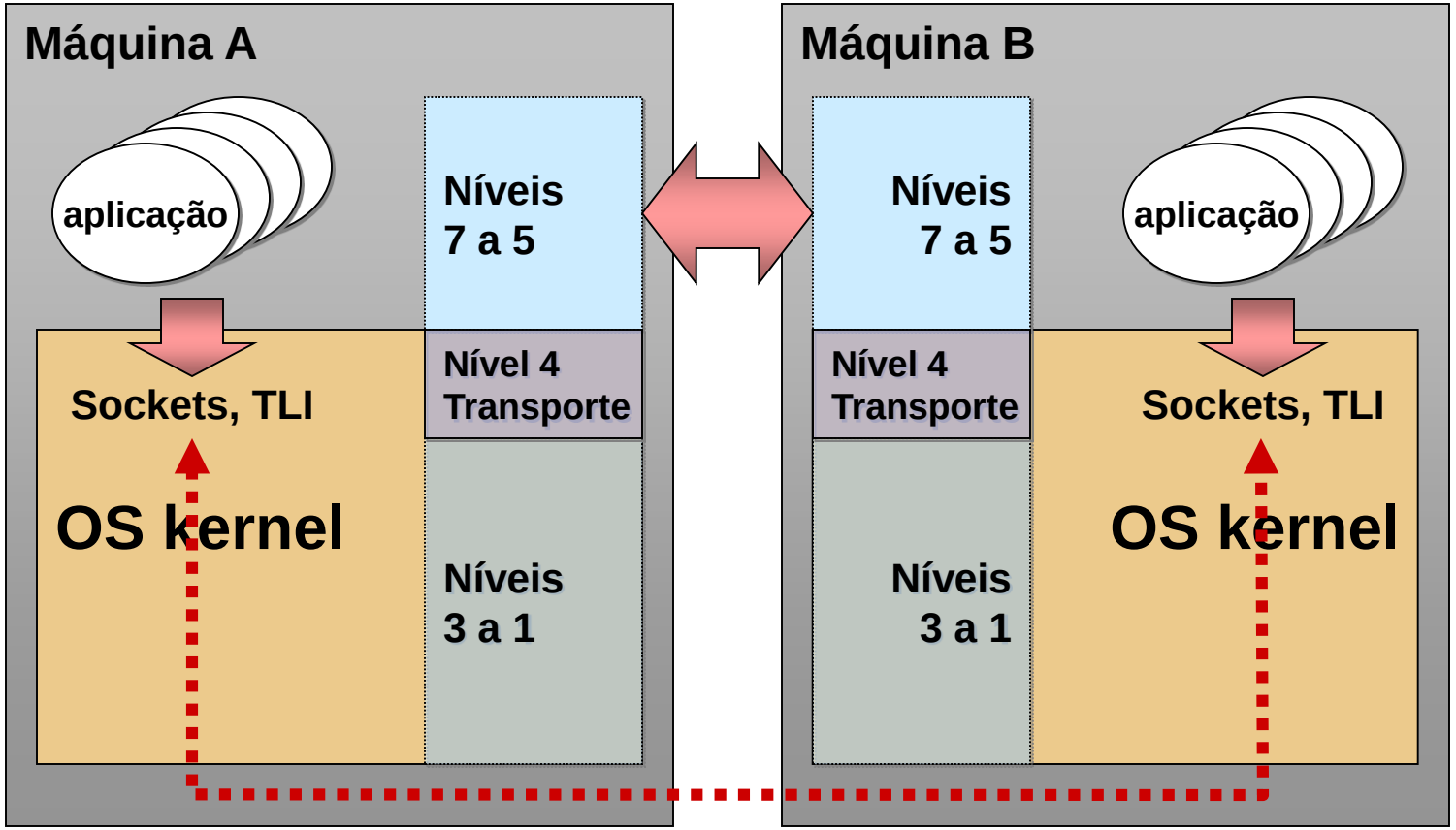
A Internet como um Relógio de Areia



Interfaces de Comunicação

- **Interação baseada na troca de mensagens**
  - Facilidade de transporte para múltiplos sistemas
- **Exploração das APIs normais de comunicação**
  - Tipicamente da API de transporte (sockets)

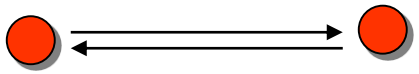
Interfaces de Comunicação



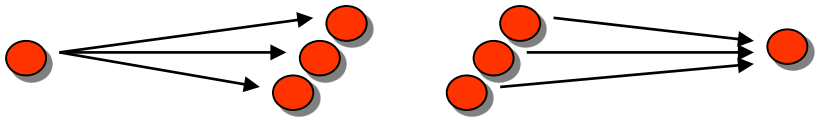
Caracterização do canal de Comunicação

Tipos de canais

- Com ligação
  - Normalmente serve 2 interlocutores



  - Normalmente fiável, bidireccional e garante sequencialidade
- Sem ligação
  - Normalmente serve mais de 2 interlocutores



  - Normalmente não fiável: perdas, duplicação, reordenação
- Canal com capacidade de armazenamento em fila de Mensagens
  - Normalmente com entrega fiável das mensagens

Portos – Extremidades do Canal de Comunicação

- São extremidades de canais de comunicação
  - Em cada máquina são representados por objectos do modelo computacional local
- Possuem 2 tipos de identificadores:
  - O do objecto do modelo computacional
    - Para ser usado na API pelos processos locais
      - Ex.: File descriptors, handles
  - O do protocolo de transporte
    - Para identificar a extremidade entre processos (ou máquinas) diferentes
      - Ex.: Endereços TCP/IP, URL

Interface sockets

- Interface de programação para comunicação entre processos introduzida no Unix 4.2 BSD
- Objectivos:
  - Independente dos protocolos
  - Transparente em relação à localização dos processos
  - Compatível com o modelo de E/S do Unix
  - Eficiente

Interface sockets

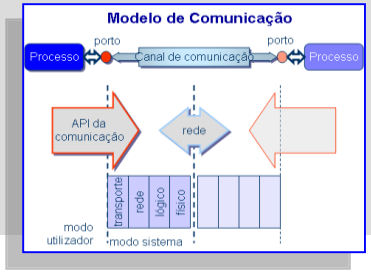
- Domínio do socket: define a família de protocolos associada a um socket
  - INET: família de protocolos Internet
  - Unix: comunicação entre processos da mesma máquina
  - Outros…
- Tipo do socket: define as características do canal de comunicação
  - Stream: canal com ligação, bidireccional, fiável, interface tipo sequência de octetos
  - Datagram: canal sem ligação, bidireccional, não fiável, interface tipo mensagem
  - Raw: permite o acesso directo aos níveis inferiores dos protocolos (ex: IP na família Internet)

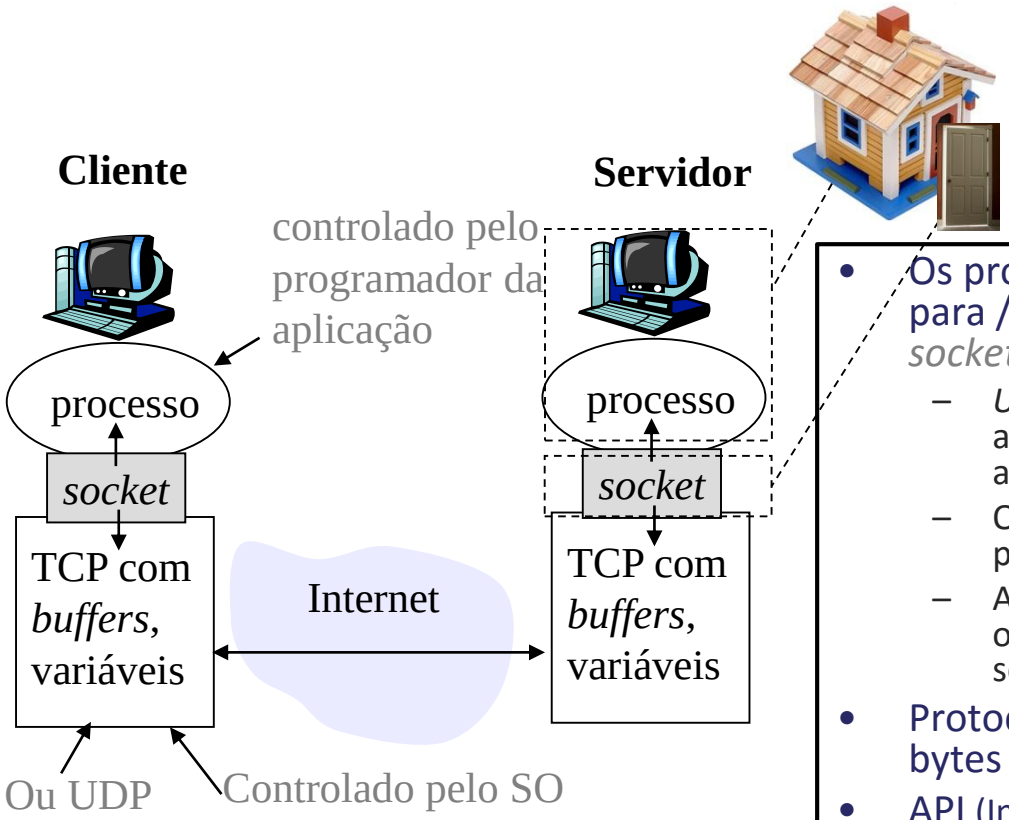
Interface sockets

- Relação entre domínio, tipo de socket e protocolo

	UNIX	INET	NS
Stream	Sim	TCP	SPP
Datagram	Sim	UDP	IDP
Raw	-	IP	Sim
Seq Packet	-	-	SPP

Sockets

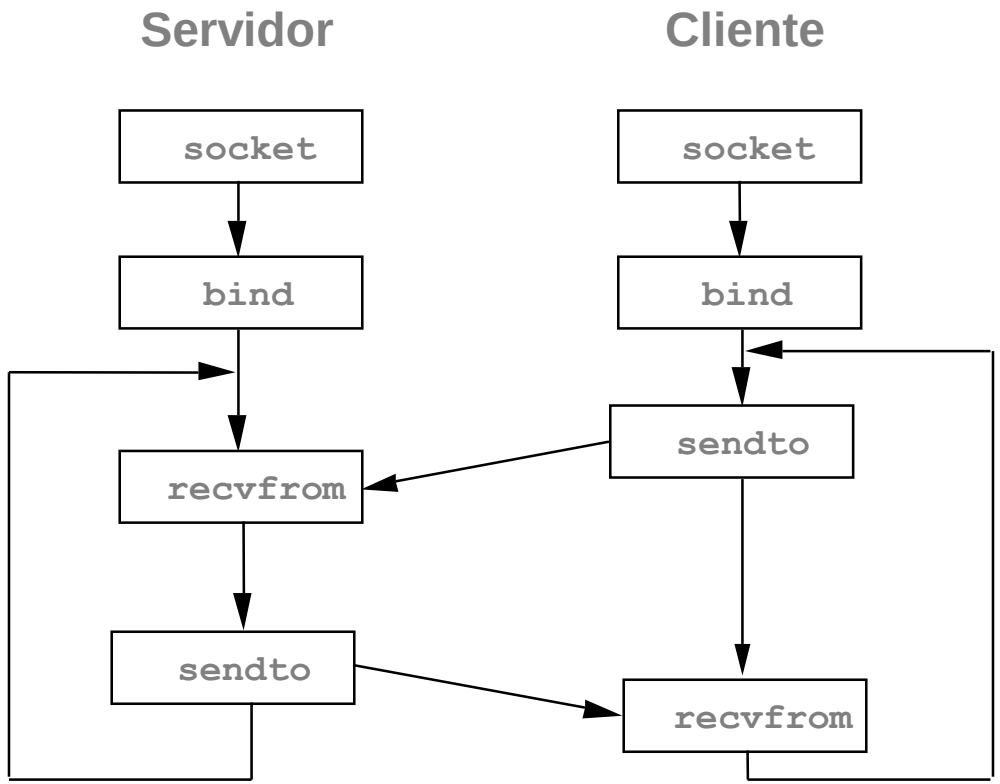




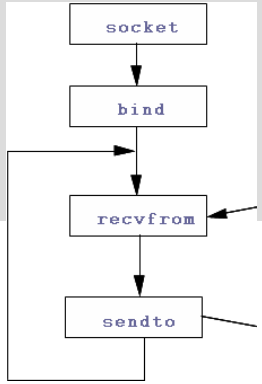
socket

- Os processos enviam /recebem mensagens para /de outros processos através dos seus *sockets*
  - *Um socket* corre no sistema terminal e é análogo a uma porta entre os processos da aplicação e o protocolo de transporte
  - O processo que envia empurra a mensagem para fora da porta
  - Assume que a infraestrutura de transporte do outro lado da porta leva a mensagem até ao socket do processo que a recebe
- Protocolo de transporte: transferência de bytes de um processo para outro
- API (Interface de Programação da Aplicação)
  - Permite escolher o protocolo de transporte
  - E definir alguns parâmetros

Sockets sem Ligação



Sockets UDP em Java (Servidor)



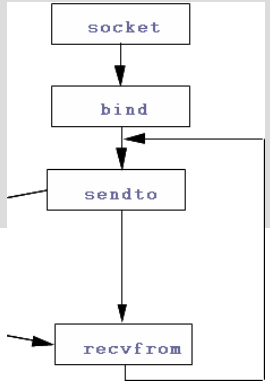

```
import java.net*;
import java.io*;
public class UDPServer{
   public static void main(String args[]){
      DatagramSocket aSocket = null;
       try{
            aSocket = new DatagramSocket(6789);
            byte[] buffer = new byte [1000];
            while(true){
               DatagramPacket request = new DatagramPacket(buffer, buffer.length);
               aSocket.receive(request);
               DatagramPacket reply = new DatagramPacket(request.getData(),
                  request.getLength(), request.getAddress(), request.getPort());
            aSocket.send(reply);
          }
        } catch (SocketException e){System.out.println("Socket:"+
                    e.getMessage());
      } catch (IOException e){System.out.println("IO:" + e.getMessage());
      } finally {if(aSocket ! = null) aSocket.close();}
   }
}
```

Constrói um socket datagram (associado ao porto 6789)

Recebe mensagem

Extrai da mensagem o IP e porto do processo origem para responder

Sockets UDP em Java (Cliente)



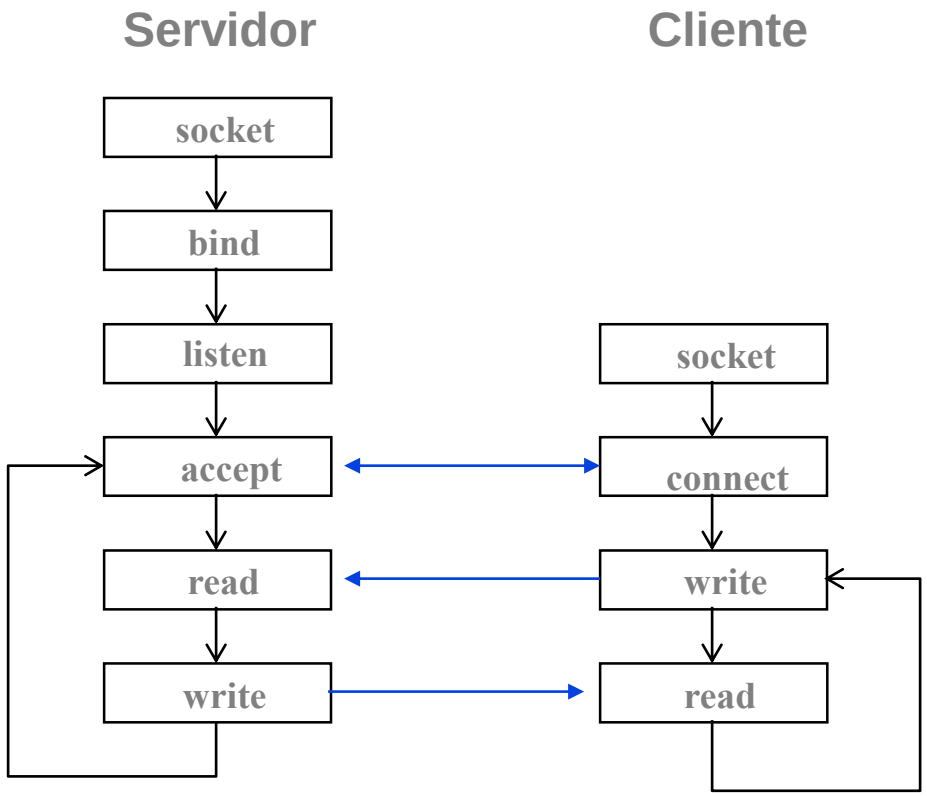

```
import java.net*;
import java.io*;
public class UDPClient{
   public static void main(String args[]){
      // args give message contents and server hostname
      DatagramSocket aSocket = null;
       try {
              aSocket = new DatagramSocket();
              byte [] m = args [0].getBytes();
              InetAddress aHost = InetAddress.getByName(args[1]);
              Int serverPort = 6789;
              DatagramPacket request =
                              new DatagramPacket(m, args[0].length(), aHost, serverPort);
              aSocket.send(request);
              byte[]buffer = new byte[1000];
              DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
              aSocket.receive(reply);
              System.out.println("Reply:" + new String(reply.getData()));
          } catch (SocketException e){System.out.println("Socket:" +
                          e.getMessage());
        } catch (IOException e){System.out.println("IO:" + e.getMessage());
      } finally { if(aSocket ! = null) aSocket.close();}
   }
}
```

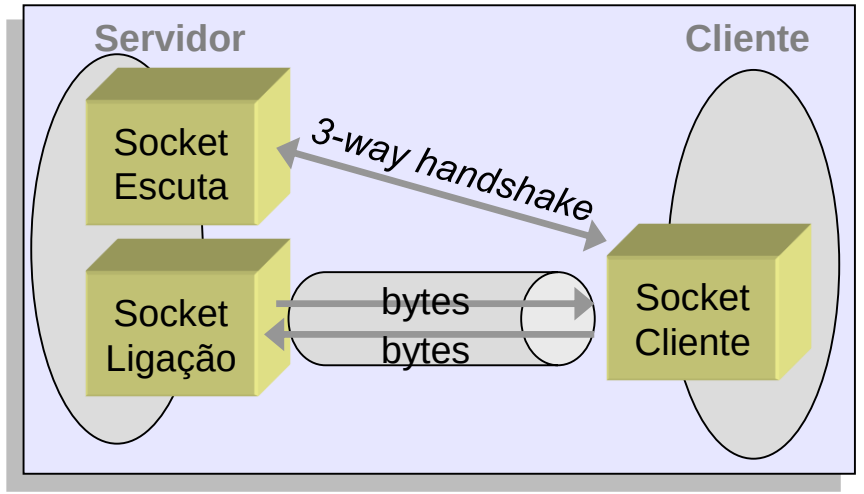
Constrói um socket datagram (associado a qualquer porto disponível)

Conversão do nome DNS para endereço IP

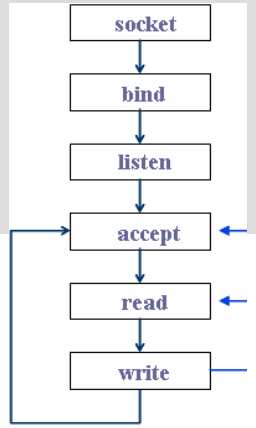
Cada mensagem enviada tem que levar junto identificador do processo destino: IP e porto

Sockets com Ligação





Sockets Stream em Java (Servidor)



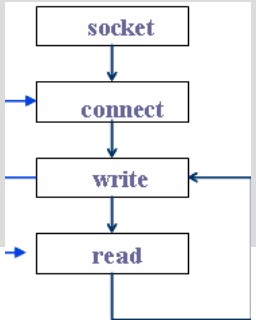

```
import java.net*;
import java.io*;
public class TCPServer{
   public static void main(String args[]){
      try{
            int server Port = 7896;
            ServerSocket listenSocket = new ServerSocket(serverPort);
            while(true){
                Socket connectionSocket = listenSocket.accept();
                myConnection c = new myConnection(connectionSocket);
                          }
       }catch (IOException e){System.out.println("Listen:"
                          +e.getMessage());}
     }
   }
```

Cria socket servidor que fica à escuta no porto "serverPort"

Bloqueia até cliente estabelecer ligação.

Cria novo socket servidor com quem é estabelecida ligação com o cliente e onde os dados são recebidos

Sockets Stream em Java (Cliente)




```
import java.net*;
import java.io*;
public class TCPClient{
   public static void main(String args[]){
   // args: message and destin. hostname
      Socket s = null;
      try{
            int server Port = 7896;
            s = new Socket (args[1], serverPort);
            DataInputStream = new DataInputStream(s.getInputStream());
            DataOutputStream out = newDataOutputStream (s.getOutputStream());
            out.writeUTF(args[0]);
            String data = in.readUTF();
             System.out.prtintln("Received: " + data);
         }catch (UnknownHostException e){
                        System.out.println("Sock:" + e.getMessage());
        }catch (EOFException e){System.out.println("EOF:"e.getMessage());
       }catch (IOException e){System.out.println("IO:"e.getMessage());
      }finally {if(s!=null) try{s.close();}catch (IOException e}
   }
```

- classe Socket – suporta o socket cliente. Argumentos: nome DNS do servidor e o porto.
- Construtor não só cria o socket como efectua a ligação TCP

Métodos getInputStream / getOutputStream – permitem aceder aos dois streams definidos pelo socket

WriteUTF / readUTF – para Universal transfer format / para as cadeias de caracteres

Sockets em Unix

Sockets Datagram em Unix

Cliente

- s=socket(AF_NET, SOCK_DGRAM,0)
- bind(s, ClientAddress)
- sendto(s, “message”, ServerAddress)

Servidor

- s=socket(AF_NET, SOCK_DGRAM,0)
- bind(s, ServerAddress)
- amount=recvfrom(s, buffer, from)

Sockets Stream em Unix

Cliente

- s=socket(AF_NET, SOCK_STREAM,0)
- connect(s, ServerAddress)
- write(s, “message”, length)

Servidor

- s=socket(AF_NET, SOCK_STREAM,0)
- bind(s, ServerAddress)
- Listen(s, 5)
- sNew=accept(s, ClientAddress)
- n=read(sNew, buffer, amount)

Aula de Laboratório – 2ª Semana

Labs SD >

Ferramentas

Objectivos da semana

- Instalar/testar o software necessário
- Usar o Maven para compilar projectos Java
- Usar o Eclipse para programar e depurar projectos Java
- Usar sockets para transferir dados entre cliente e servidor

Documentação de apoio à aula

- Java tools reference card
- Introdução ao Maven
- Configurar um projecto Maven no Eclipse
- Dicas de utilização do Eclipse
- Secção 1.6 do livro da cadeira e slides das teóricas sobre World Wide Web e Sockets
- (Breve) introdução à World Wide Web

Exemplos

- Exemplo de aplicação Java simples - utiliza o Maven para compilar e executar
- Exemplo de aplicação Java com testes JUnit - utiliza o Maven para compilar, testar e executar
- Servidor de Sockets TCP/IP - transferência de texto com sockets TCP/IP
- Cliente de Sockets TCP/IP

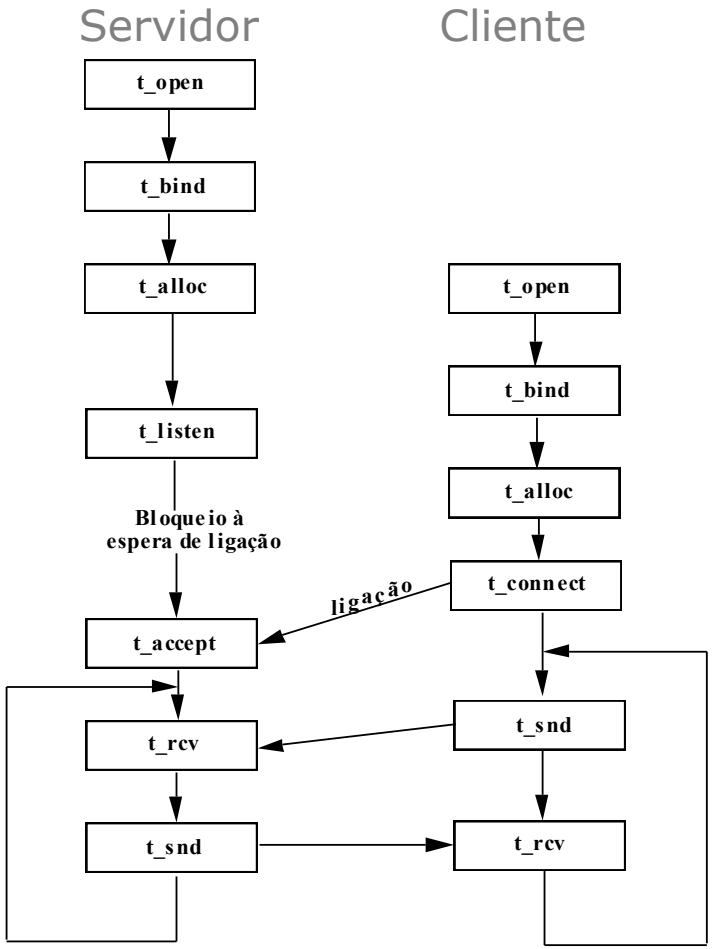
Exercício a resolver até ao fim da aula

1. Obter o exemplo de **aplicação Java simples**.

Transport Layer Interface

- Introduzida no Sistema V versão 3 (1987)
- A implementação da TLI é suportada nos stream Unix. Objectivo total integração com os mecanismos de E/S.
- As funções são muito semelhantes às dos sockets mas existe uma maior uniformização com a interface genérica dos streams.
- Actualmente pouco utilizados (sockets são o standard *de facto*)

Transport Layer Interface



OSI – Níveis superiores do Modelo

- Os restantes níveis do modelo OSI implicam a integração com os sistemas operativos e com as aplicações
- São em grande parte o objecto desta cadeira,
  - Embora alguns protocolos de nível aplicacional possam ser vistos como de transporte de informação

Aplicação	HTTP, FTP, SMTP, Corba, IIOP, SOAP, RMI
Apresentação	XML, XDR
Sessão	Binding protocol, DCE-RPC

Integração da Comunicação no Sistema Operativo

Integração da Comunicação no Sistema Operativo

- As aplicações invocam uma API que lhes permite aceder ao mecanismos de transporte
- A API deve ser conceptualmente independente de uma determinada pilha de protocolos de transporte
- Alternativas de implementação
  - Funções de ES genéricas
    - Ex: sockets – parcialmente
  - Funções de comunicação específicas
    - Ex: Algumas funções dos sockets
    - Ex: TLI
  - Mecanismo básico de comunicação entre processos do sistema operativo
    - Ex: IPC dos micro-núcleos

Unix – 4.4 BSD

<table>
<tr><td colspan="5">System calls</td><td colspan="3">Interrups and traps</td></tr>
<tr><td colspan="2">Terminal handing</td><td>Sockets</td><td>File naming</td><td>Map-ping</td><td>Page faults</td><td rowspan="2">Signal handling</td><td rowspan="2">Process Creation and Termination</td></tr>
<tr><td rowspan="2">Raw TTY</td><td>Cooked TTY</td><td>Network protocols</td><td>File systems</td><td colspan="2">Virtual memory</td></tr>
<tr><td>Line disciplines</td><td>Routing</td><td>Buffer cache</td><td colspan="2">Page cache</td><td colspan="2">Process scheduling</td></tr>
<tr><td colspan="2">Character devices</td><td>Netwok device drivers</td><td colspan="2">Disk device drivers</td><td colspan="2">Process dispatching</td></tr>
<tr><td colspan="8">Hardware</td></tr>
</table>

Winsock Implementation