MÓDULO 6

Programação de Sistemas de Comunicação

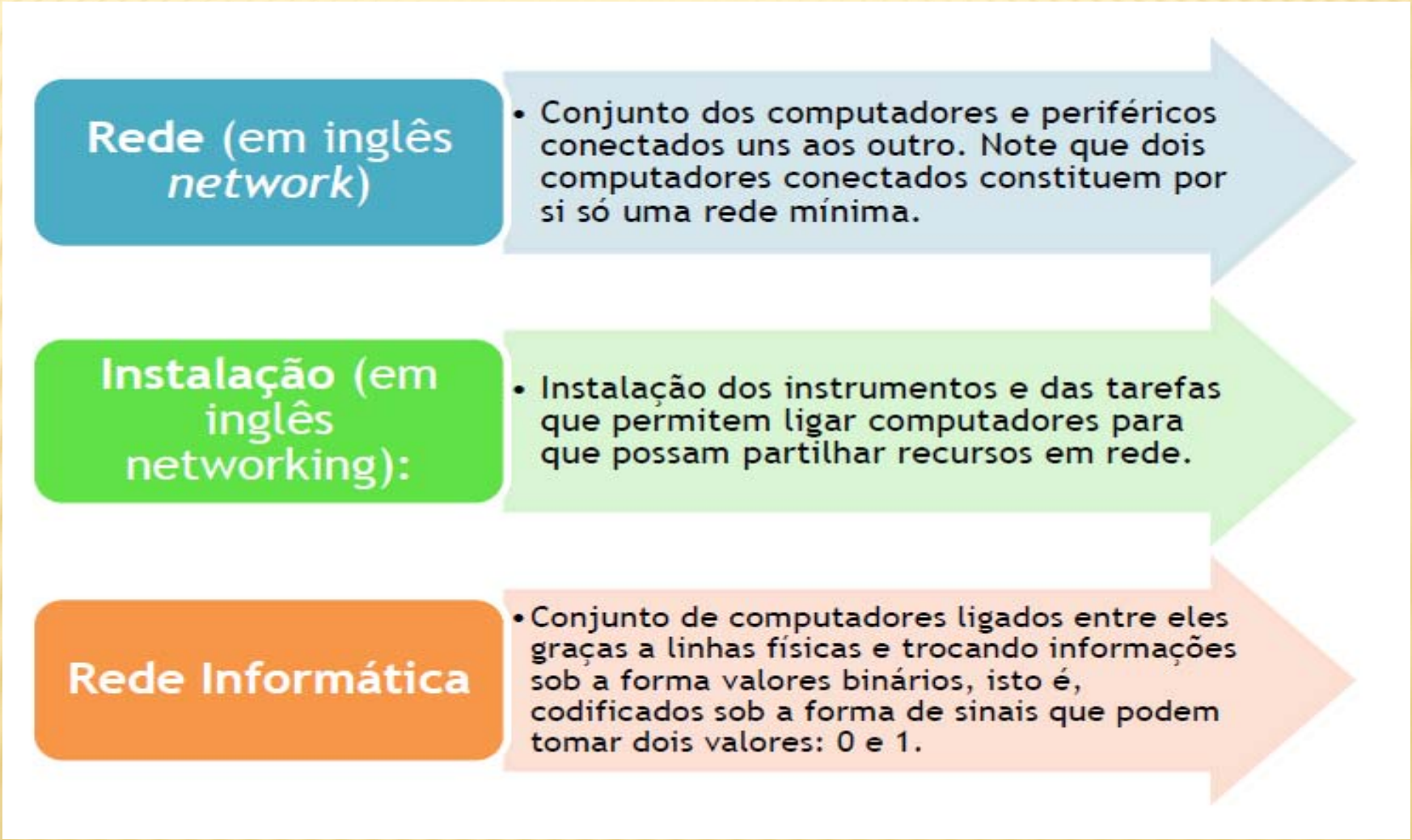
Curso Profissional de Técnico de Gestão e Programação de Sistemas Informáticos

12º Ano

ANO LETIVO: 2011/2012

Professores: Águeda Ramos e Paulo Quaresma

CONCEITO DE REDE



OBJETIVOS DE UMA REDE

- A partilha de recursos: ficheiros, aplicações ou materiais, ligação à Internet, etc.
- A comunicação entre pessoas: correio electrónico, conversa em direto, etc.
- A comunicação entre processos: entre computadores industriais, por exemplo.
- A garantia da unicidade e da universalidade do acesso à informação: bases de dados em rede.
- O jogo vídeo com vários jogadores.

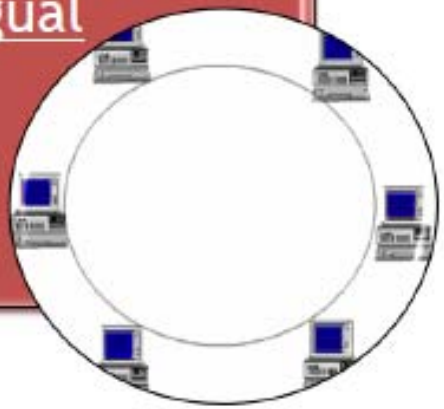
TIPOS DE REDES

- Não existe um só tipo de rede, porque historicamente existem tipos de computadores diferentes, comunicando de acordo com linguagens diversas e variadas.
- Isto deve-se igualmente à heterogeneidade dos apoios físicos de transmissão que os ligam:
  - quer seja a nível da transferência de dados;
    - circulação de dados sob a forma de impulsos eléctricos, luz ou ondas electromagnéticas;
  - quer a nível do tipo de apoio
    - cabo coaxial, pares entrançados, fibra óptica, etc.

DIFERENTES TIPOS DE REDE

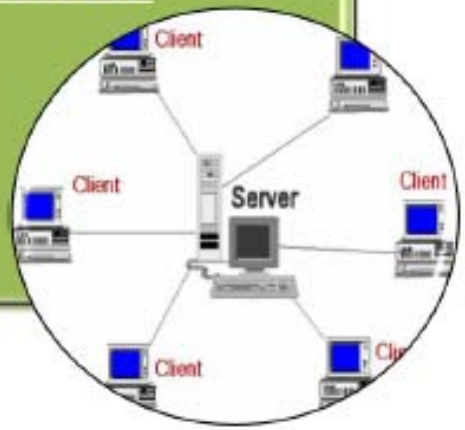
- As redes posto a posto

Peer to peer/de igual para igual



- Redes organizadas em torno de servidores

Cliente/Servidor



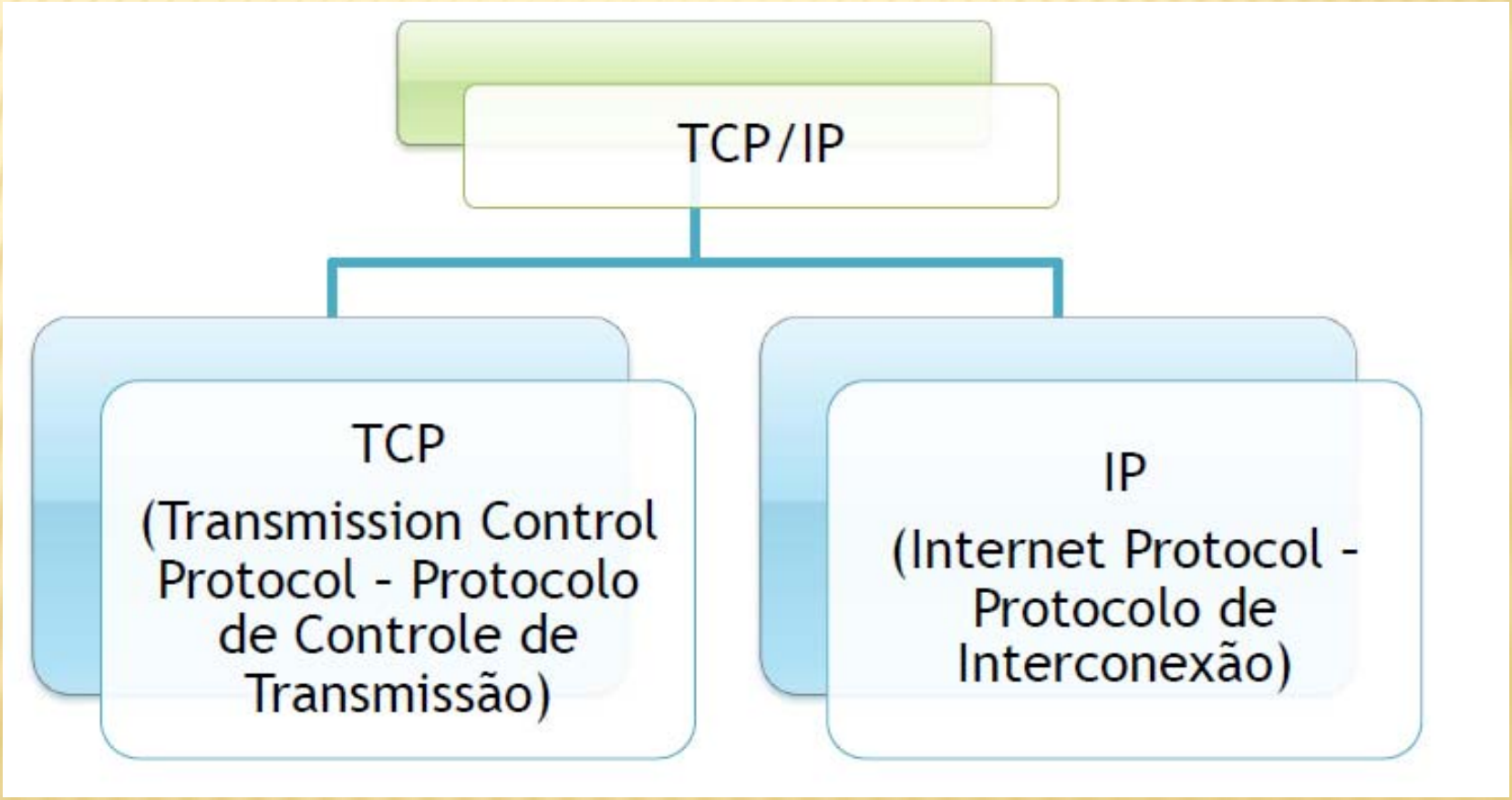
SEMELHANÇAS ENTRE TIPOS DE REDES

- Servidores
  - computadores que fornecem recursos partilhados aos utilizadores por um servidor de rede
- Clientes
  - computadores que acedem aos recursos partilhados fornecidos por um servidor de rede
- Apoio de conexão
  - condiciona a forma como os computadores estão ligados entre eles.
- Dados partilhados
  - ficheiros acessíveis nos servidores da rede
- Impressoras e outros periféricos partilhados
  - ficheiros, impressoras ou outros elementos utilizados pelos utentes da rede
- Recursos diversos
  - outros recursos fornecidos pelo servidor

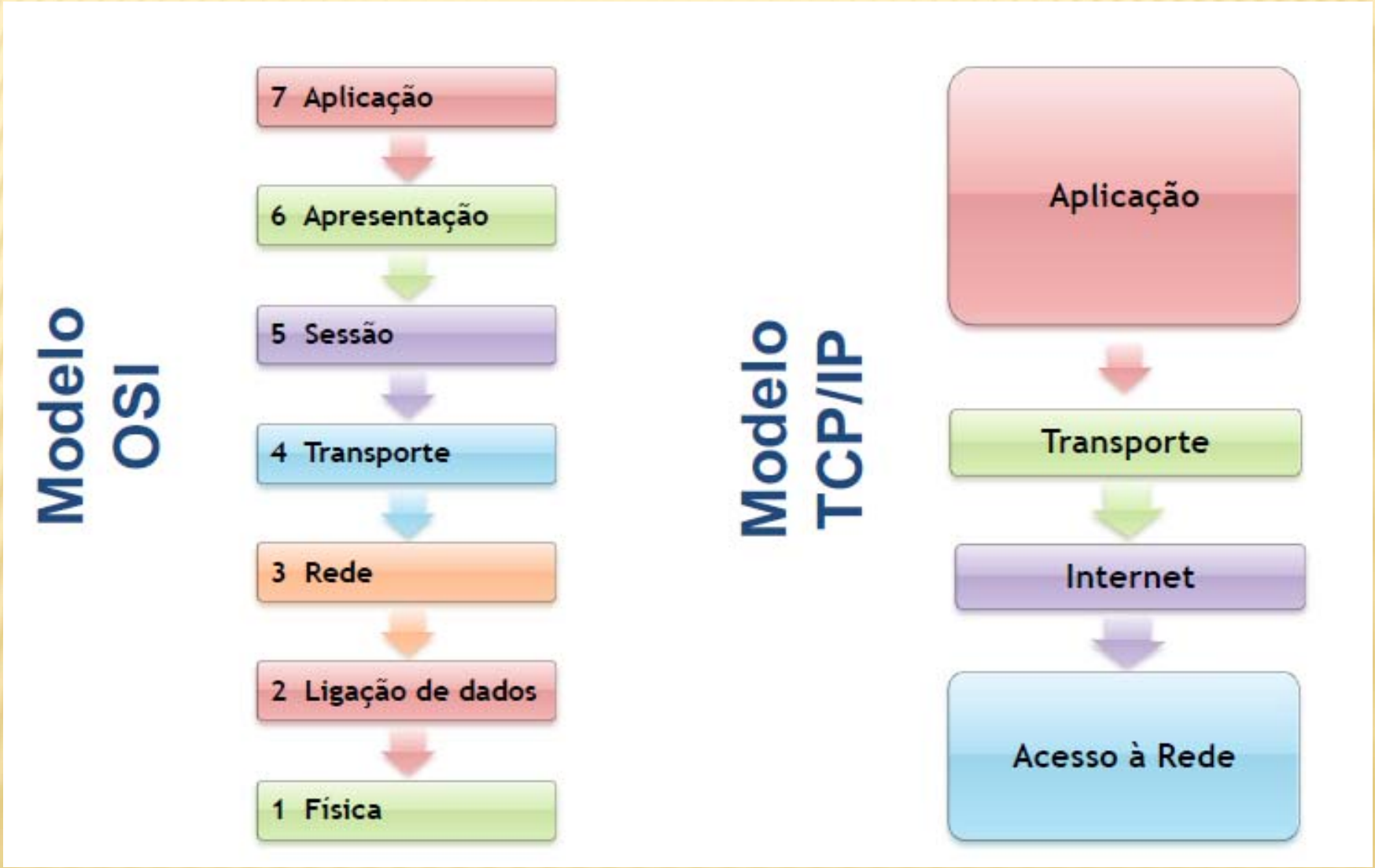
INSTALAÇÃO DE REDES

MODELO TCP/IP

- O TCP/IP é um conjunto de protocolos de comunicação entre computadores em rede.



COMPARAÇÃO ENTRE O MODELO OSI E TCP/IP

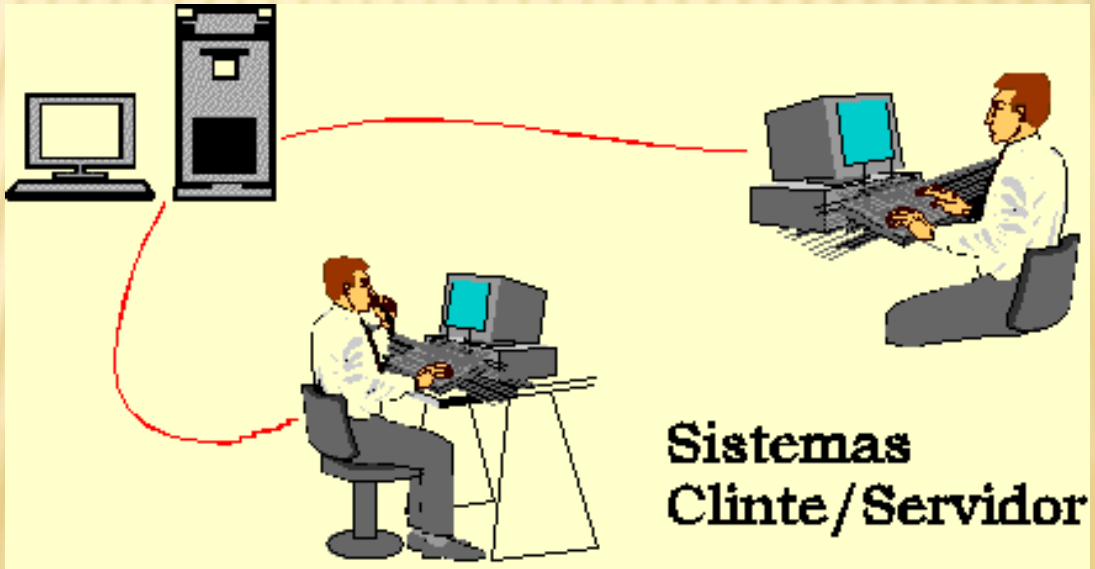


EXERCÍCIOS

1. Ipconfig
2. ipconfig/all
3. ping [endereçoIP]
4. tracert [endereço]
5. pathping [endereço]
6. nbtstat
   - nbtstat–a[nome do computador]
   - nbtstat–n
   - nbtstat–A [Endereço IP]
7. arp–a
8. hostname
9. Nslookup
   - netstat -a
   - netstat -e

ARQUITETURA DE UM SISTEMA CLIENTE/SERVIDOR

- **Máquinas clientes**(máquinas que fazem parte da rede) contactam um **servidor**, uma máquina geralmente muito potente em termos de capacidades de entrada/saída, que lhes fornece **serviços.**



CARATERÍSTICAS DE UM SISTEMA CLIENTE/SERVIDOR

- Serviço
  - Cliente/servidor é uma relação entre processos que estão a correr em máquinas diferentes. O processo servidor é o fornecedor dos serviços. O cliente é o consumidor dos serviços. Fundamentalmente esta arquitetura implementa uma separação lógica de funções baseada no conceito de serviço.
- Recursos Partilhados
  - Um servidor pode servir vários clientes ao mesmo tempo e gerir os acessos a recursos partilhados.
- Protocolos Assimétricos
  - Existe uma relação de muitos-para-um entre clientes e servidor. Os clientes iniciam o diálogo através da requisição de um serviço. Os servidores esperam passivamente os pedidos dos clientes.
- Localização Transparente
  - O servidor é um processo que pode residir na mesma máquina que o cliente ou numa máquina diferente que esteja ligada através de uma rede. Um programa pode ter o papel de cliente, servidor ou ambos.

CARATERÍSTICAS DE UM SISTEMA CLIENTE/SERVIDOR (2)

- Independência
  - O conceito inerente às arquiteturas cliente/servidor baseia-se em software que deve ser independente de hardware ou sistemas operativos.
- Baseado na transmissão de mensagens
  - Clientes e servidores devem estar ligados de forma “fraca”, ou seja, não deve ser obrigatório que o servidor esteja a correr para que o cliente possa correr. Sistemas deste tipo são normalmente baseados em mensagens. A mensagem é o mecanismo de transporte para os pedidos e respostas dos serviços.
- Encapsulamento de serviços
  - Um servidor deve ser um programa “especializado”. As mensagens transmitem o pedido de serviço ao servidor. O servidor é que deve ser responsável pela forma como implementa o serviço. A forma de implementar os serviços pode ser melhorada/alterada sem implicações ao nível dos clientes.

CARATERÍSTICAS DE UM SISTEMA CLIENTE/SERVIDOR (3)

- Escalabilidade
  - Os sistemas cliente/servidor podem evoluir facilmente quer por adição de novos clientes quer por evolução para novas máquinas servidoras mais potentes.
- Integridade
  - O código e dados do servidor devem ser mantidos centralmente. Desta forma reduzem-se os custos de manutenção e aumenta-se a integridade doa dados.

SERVIÇOS DA ARQUITETURA

- Os **serviços** são explorados por programas, chamados **programas clientes**, que se executam nas máquinas clientes.
- Cliente(cliente FTP, cliente de serviço de mensagens, etc.)
  - programa que funciona numa máquina cliente, capaz de tratar de informações que recupera junto de um servidor:
    - no caso do cliente FTP trata-se de ficheiros,
    - enquanto para o cliente de serviço de mensagens trata-se de correio electrónico.

VANTAGENS DA ARQUITETURA

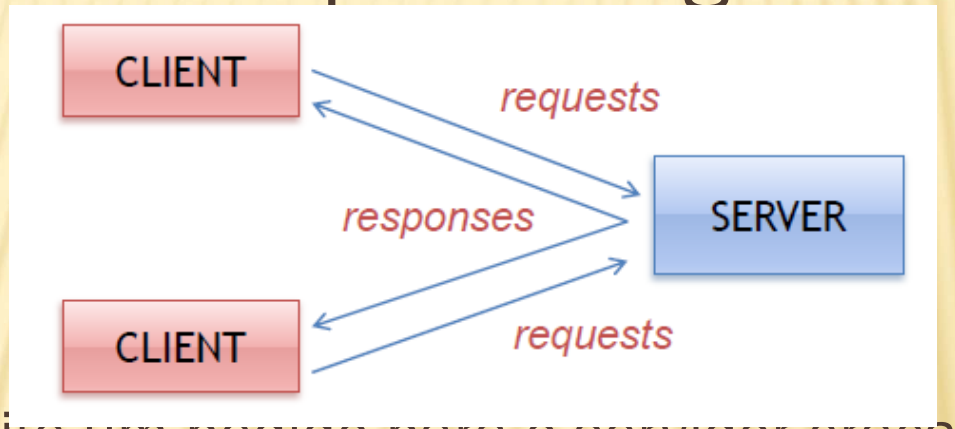
- O modelo cliente/servidor é particularmente recomendado para redes que necessitam de um grande nível de fiabilidade e as suas principais vantagens são:
  - **Recursos centralizados:** já que o servidor está no centro da rede, pode gerir recursos comuns a todos os utilizadores, como por exemplo uma base de dados centralizada, a fim de evitar os problemas de redundância e de contradição;
  - **Uma melhor segurança:** porque o número de pontos de entrada que permitem o acesso aos dados é menos importante;
  - **Uma administração a nível do servidor:** como os clientes têm pouca importância neste modelo, têm menos necessidade de ser administrados;
  - **Uma rede evolutiva:** graças a esta arquitetura, é possível suprimir ou acrescentar clientes sem estar a perturbar o funcionamento da rede e sem modificação essencial.

INCONVENIENTES DA ARQUITETURA

- A arquitetura cliente/servidor tem no entanto algumas lacunas, entre as quais:
  - **Um custo elevado:** devido ao tecnicismo do servidor;
  - **Um elo fraco:** o servidor é o único elo fraco da rede cliente/servidor, já que toda a rede está estruturada em redor dele! Felizmente, o servidor tem uma grande tolerância às avarias (nomeadamente graças ao sistema RAID).

FUNCIONAMENTO DE UM SISTEMA CLIENTE/SERVIDOR

- Um sistema cliente/servidor funciona de acordo com o esquema seguinte:



  - O cliente emite um pedido para o servidor graças ao seu endereço IP e a porta, que designa um serviço específico do servidor.
  - O servidor recebe o pedido e responde com a ajuda do endereço da máquina cliente e da sua porta.

COMPONENTES DE UMA ARQUITEURA CLIENTE/SERVIDOR

Cliente

- Baseia-se fundamentalmente no interface gráfico da aplicação. Cada vez mais este é um Object Oriented User Interface(OOUI).
- Acede aos serviços através do middleware.

Servidor

- Executa a parte de serviços da aplicação.
- Utiliza normalmente servidores específicos de software (SGBDs–relacionais ou OO, Monitores TP, servidores de groupware, servidores de objectos e servidores de Web).

Middleware

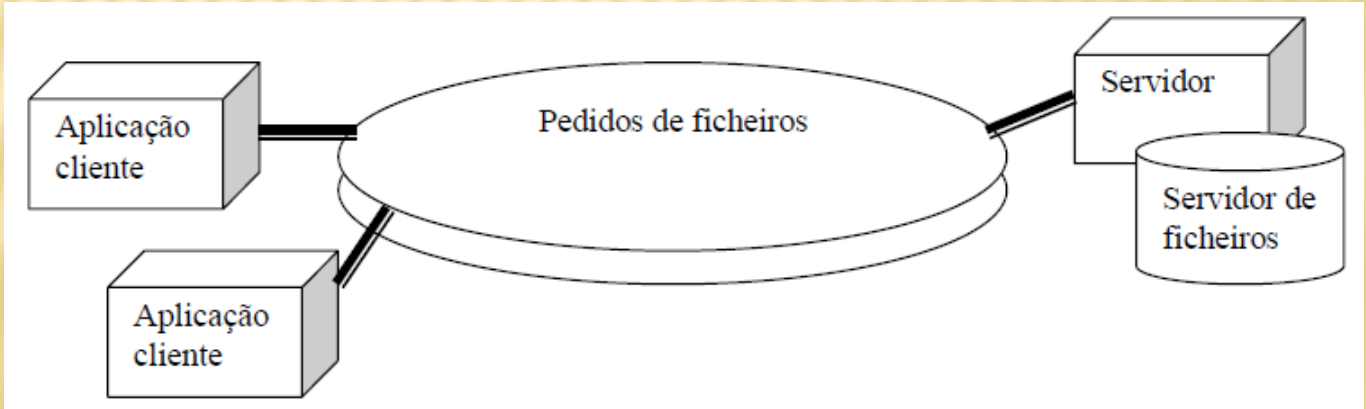
- Executa no cliente e no servidor.
- Normalmente está dividido em stacks de transporte (TCP/IP, NetBios, IPX/SPX, SNA, etc), Sistemas Operativos de Rede ou serviços de rede (NOS = serviços de directório, segurança, RPC, mensagens, etc) e serviços específicos de middleware (ODBC, Mail, ORB, HTTP, etc).

VERSÕES DE UM SISTEMA CLIENTE/SERVIDOR

- Servidores de **Ficheiros;**
- Servidores de **Base de Dados;**
- Servidores de **Transações;**
- Servidores de ***Groupware;***
- Servidores de **Objetos;**
- Servidores de **Web.**

VERSÕES DE UM SISTEMA CLIENTE/SERVIDOR (2)

- Servidor de Ficheiros
  - O cliente executa pedidos de ficheiros ao servidor de ficheiros através da rede.
  - É uma forma muito primitiva de serviço de dados e provoca uma troca muito elevada de mensagens de rede.
  - São, no entanto, sistemas necessários para a partilha de repositórios de ficheiros em rede (documentos, imagens, desenhos, etc).



VERSÕES DE UM SISTEMA CLIENTE/SERVIDOR (3)

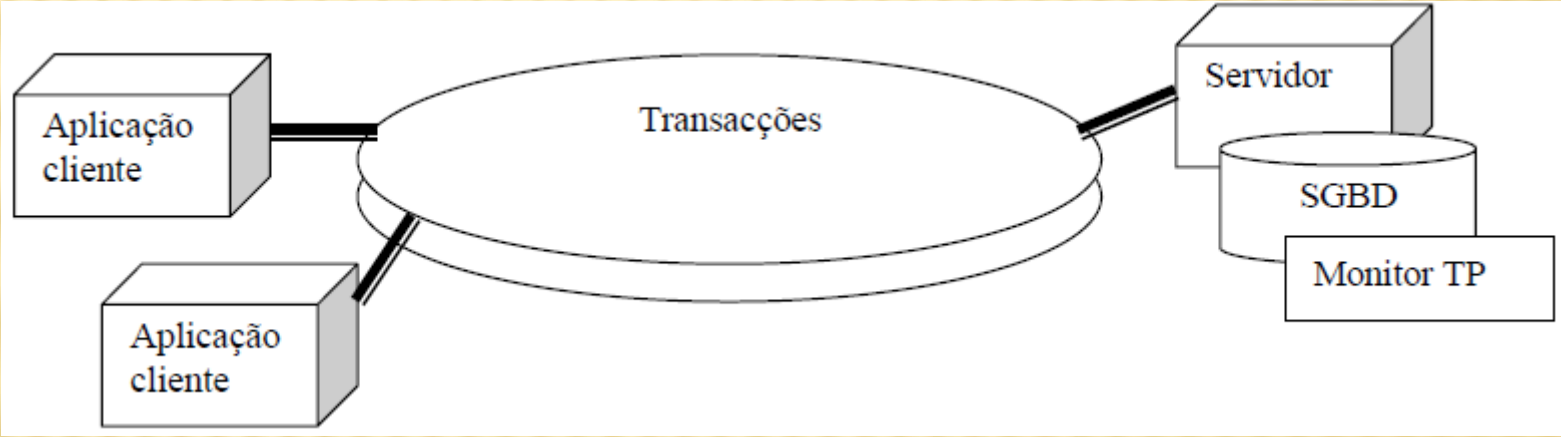
- Servidor de Bases de dados
  - O que é transmitido na rede são instruções SQL.
  - O resultado das instruções de SQL são enviadas para o cliente.
  - O código que processa as instruções SQL e os dados residem na mesma máquina (servidor).
  - É o servidor que determina quais os registos resultantes da instrução e são apenas estes que são enviados pela rede.

Nos servidores de ficheiros todos os registos são enviados pela rede e é o cliente que determina aqueles que interessam.

VERSÕES DE UM SISTEMA CLIENTE/SERVIDOR (4)

Servidor de Transações

- Os clientes invocam procedimentos remotos que residem no servidor (com uma base de dados). Estes procedimentos são constituídos por grupos de instruções SQL.
- As instruções do procedimento são executadas na totalidade ou então falha tudo.
- Estes sistemas designam-se de OLTP (*OnlineTransaction Processing*).

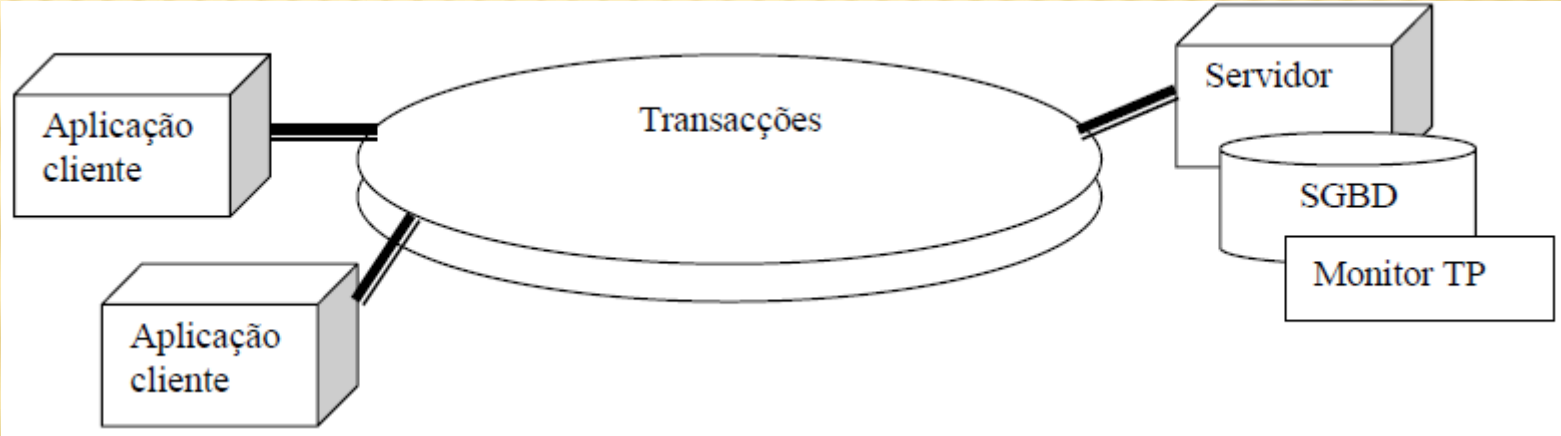


Ao contrário do simples servidor de bases de dados neste caso o programador tem que escrever código no cliente e no servidor.

VERSÕES DE UM SISTEMA CLIENTE/SERVIDOR (5)

Servidor de *GroupWare*

- Facilita a gestão de informação semiestruturada (ou não estruturada) tal como texto, imagem, e-mail, etc.
- Implementam capacidades de automação de *workflow*.
- Estes sistemas suportam-se sobre sistemas de transmissão de mensagens.



Existem diversos sistemas que embora se possam interligar são implementados de formas diferentes.

VERSÕES DE UM SISTEMA CLIENTE/SERVIDOR (6)

Servidor de *Objetos*

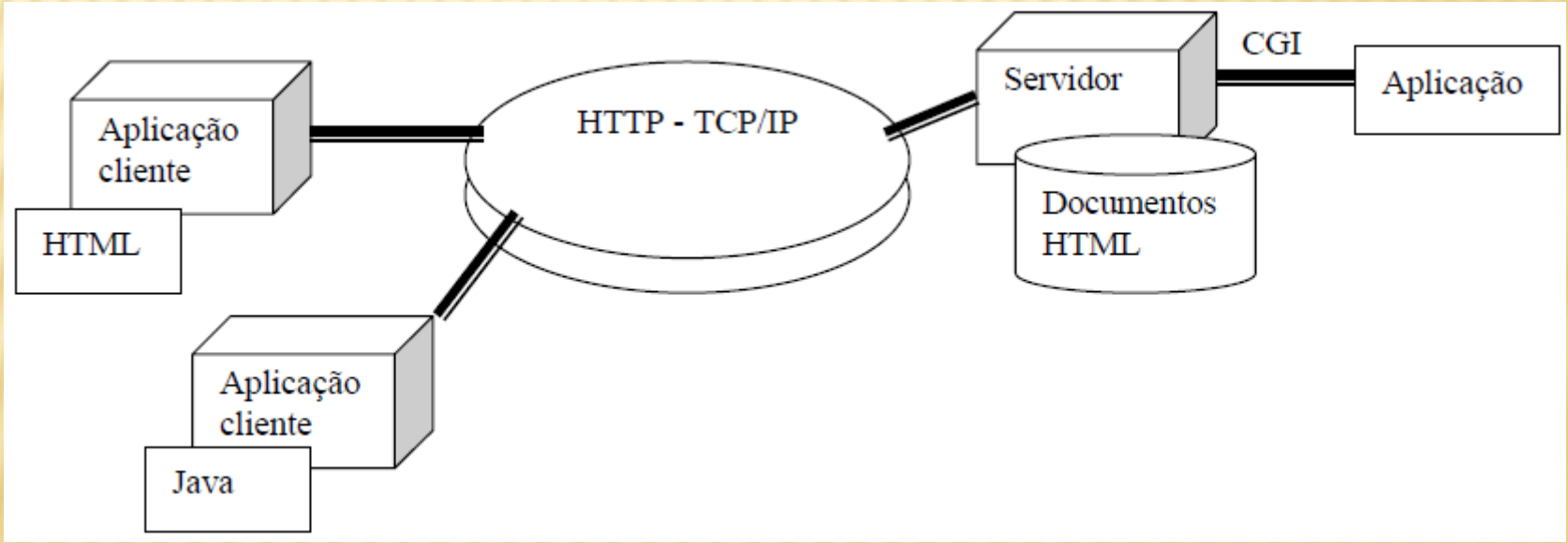
- É implementado através de um conjunto de objetos que podem comunicar entre si;
- Objetos cliente comunicam com objetos servidores através do *Object Request Broker* (ORB);
- Quando o cliente invoca um método num objecto remoto o ORB localiza a instância do objecto servidor, invoca o método e retorna o resultado ao objecto cliente.

As tecnologias concorrentes nesta área são o CORBA e o DCOM.

VERSÕES DE UM SISTEMA CLIENTE/SERVIDOR (7)

Servidor Web

- O novo modelo introduzido pela internet consiste em clientes “leves”, “portáteis” e “universais” que comunicam com servidores “super pesados” (servem milhares ou milhões de clientes).
- A comunicação é feita por um protocolo do tipo RPC designado por HTTP.



MIDDLEWARE

- É a camada que se encontra entre o cliente e o servidor.
- Começa com a API que o cliente utiliza para invocar um serviço, inclui a transmissão do pedido pela rede assim como da resposta.
- O *Middleware* não inclui o software que executa o serviço (esta é a função do servidor).
- O *Middleware* não inclui o interface (esta é uma função do cliente).

CATEGORIAS DE MIDDLEWARE

- *Middleware* genérico:
  - Normalmente inclui tudo o que tem a ver com transporte (*stacks* de comunicação, serviços de diretório, serviços de autenticação, RPCs, etc.)
- Serviços de *Middleware*:
  - *Middleware* específico de base de dados (odbc, etc);
  - *Middleware* específico de *groupware* (MAPI, VIM, etc);
  - *Middleware* específico de serviços de objetos (CORBA, DCOM, etc);
  - *Middleware* específico de internet (HTTP, SSL, etc);
  - *Middleware*de gestão específico (ORB, etc.)

BALANCEAMENTO CLIENTE/SERVIDOR

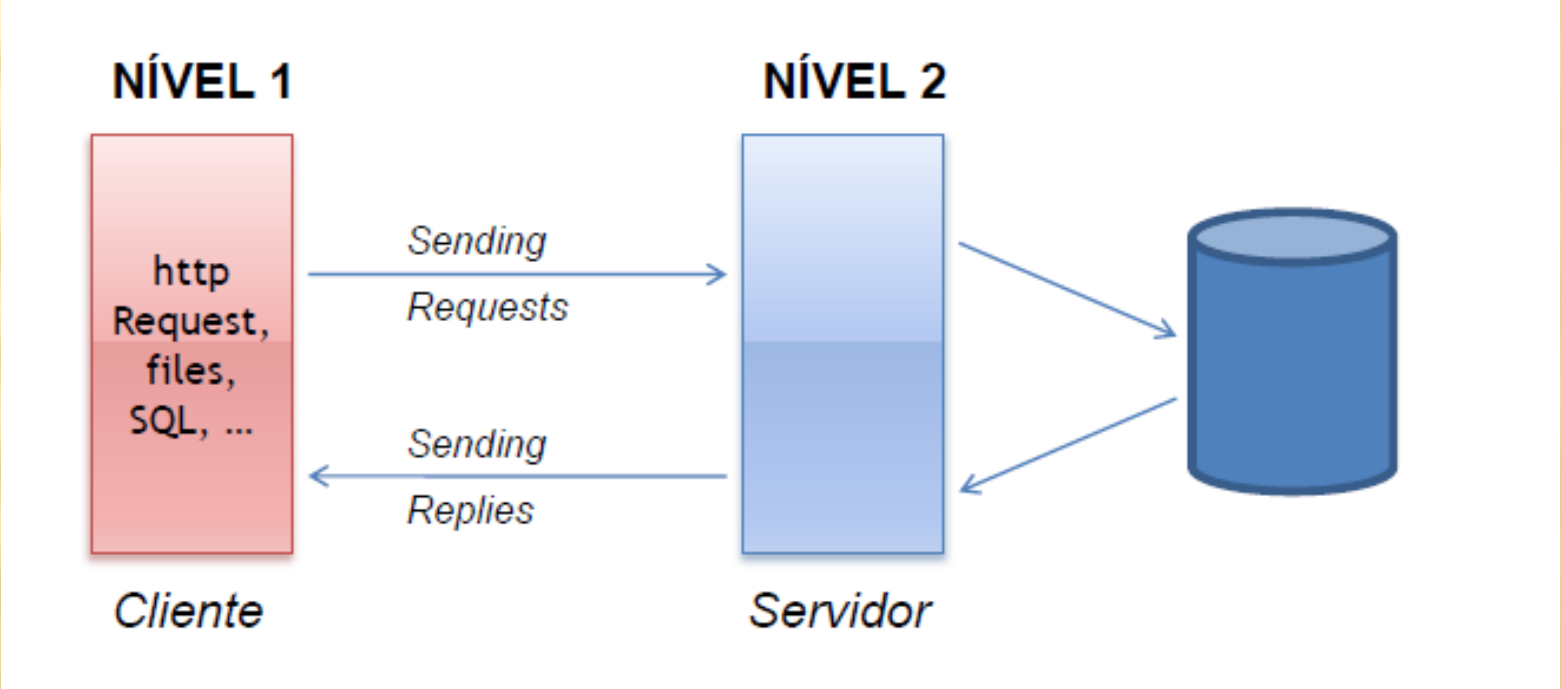
- Como distribuir uma aplicação entre cliente e servidor?
- *Groupware* e servidores de Web são servidores "pesados"
- Servidores de ficheiros e de base de dados são exemplos de clientes "pesados"
- Objetos distribuídos estão no meio (normalmente)

TIPOS DE ARQUITETURA CLIENTE/SERVIDOR

- Arquitetura Cliente/Servidor de 2 níveis versus 3 níveis
  - As aplicações podem normalmente ser divididas logicamente em interface, lógica de negócio e base de dados.
  - No chamado cliente/servidor a **2 níveis** a lógica de negócio está dividida entre cliente e servidor.
  - No servidor a **3 níveis** (ou ***n* níveis**) existe um nível independente só para a lógica de negócio. Esta solução permite um desempenho e uma evolução superior.
  - O exemplo por excelência desta arquitetura são as soluções de objetos distribuídos.
  - As soluções também se enquadram normalmente nesta categoria.

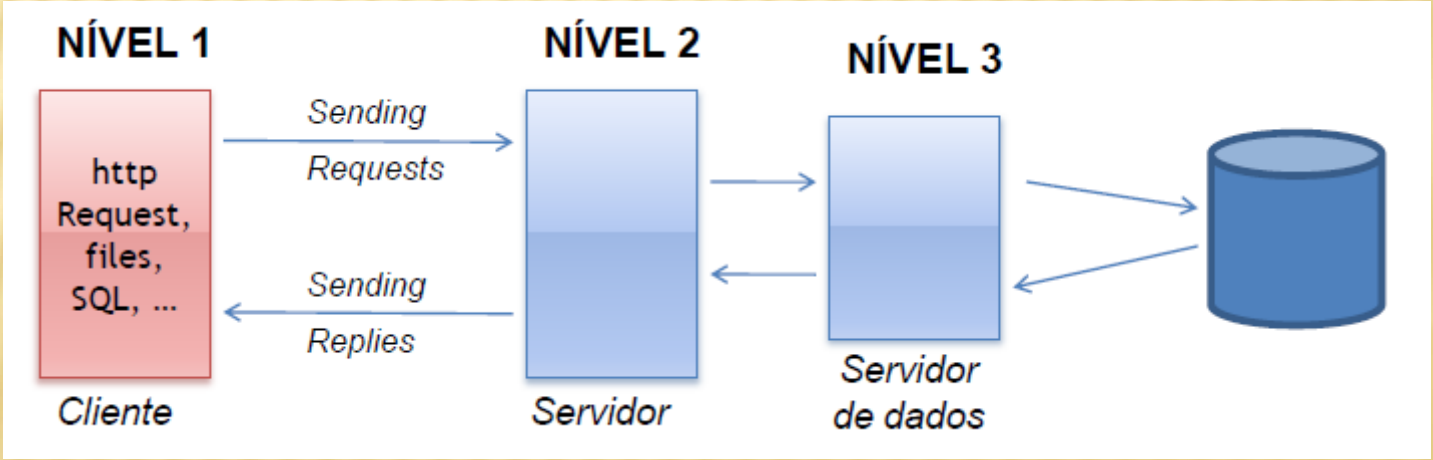
2 NÍVEIS -*2-TIER*

- Caracteriza os sistemas clientes/servidores pelos quais o cliente pede um recurso e o servidor lho fornece diretamente, utilizando os seus próprios recursos.



3 NÍVEIS -*3-TIER*

- Arquitetura compartilhada entre:
  - Um cliente, quer dizer o computador requerente de recursos, equipado de um interface utilizador (geralmente um navegador web) encarregado da apresentação;
  - O servidor de aplicação (chamado igualmente ***middleware***), encarregado de fornecer o recurso mas que recorre a um outro servidor
  - O servidor de dados, fornecendo ao servidor de aplicação os dados dos quais este tem necessidade.

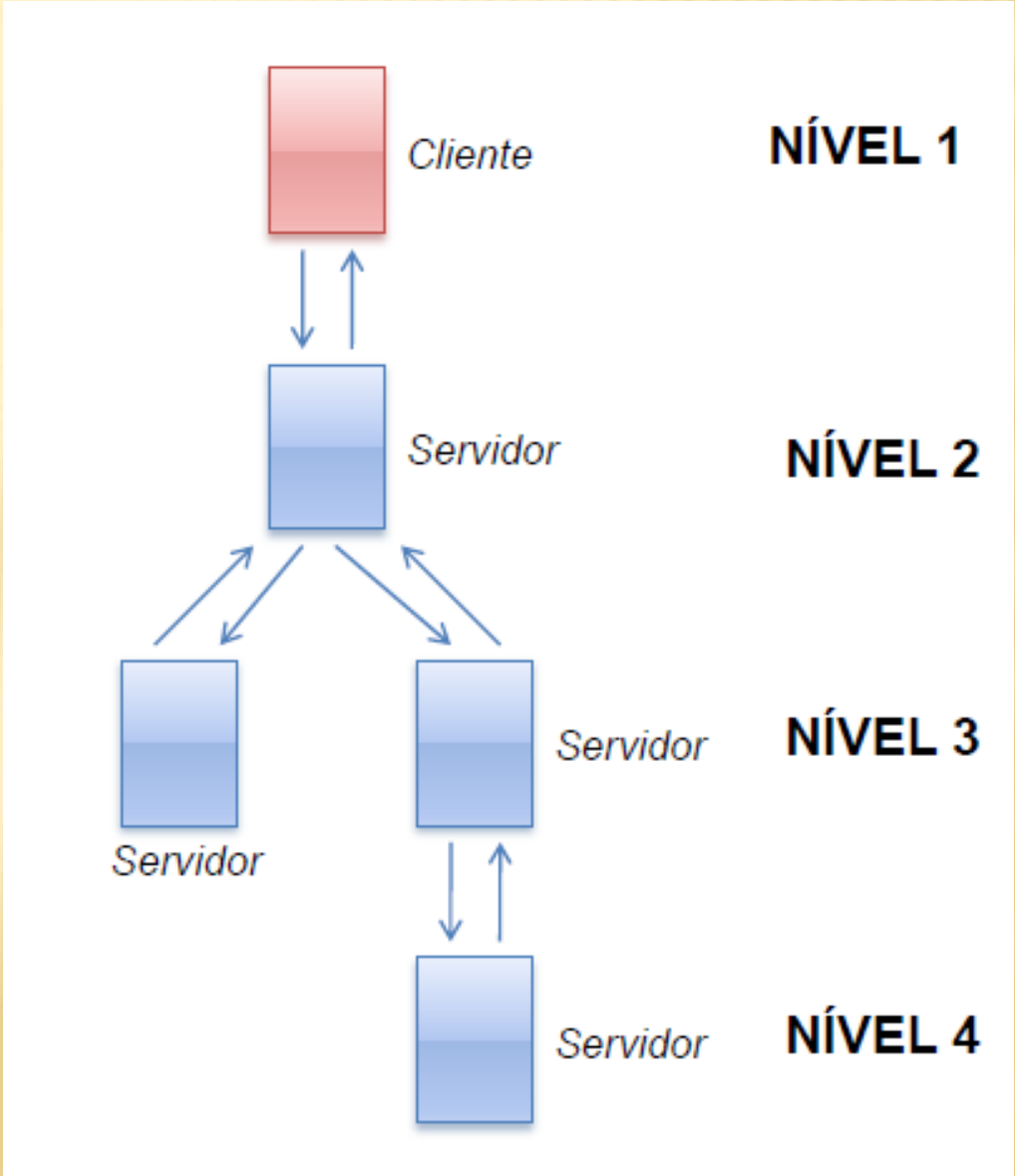


COMPARAÇÃO

- A **arquitetura de dois níveis** o servidor é polivalente, quer dizer que é capaz de fornecer diretamente o conjunto dos recursos pedidos pelo cliente.
- Na **arquitetura de três níveis**, em contrapartida, as aplicações ao nível do servidor são deslocalizadas, o que quer dizer que cada servidor é especializado numa tarefa (servidor web/servidor de base de dados, por exemplo).
- A arquitetura de três níveis permite:
  - Maior flexibilidade;
  - Uma segurança acrescida porque a segurança pode ser definida independentemente para cada serviço, e a cada nível;
  - Melhores desempenhos, dada a divisão das tarefas entre os diferentes servidores.

ARQUITETURA MULTINÍVEIS

- Um servidor pode, utilizar os serviços de um ou vários outros servidores a fim de fornecer o seu próprio serviço.
- A arquitetura de 3 níveis é potencialmente uma arquitetura de *n* níveis...



PROGRAMAÇÃO CLIENTE/SERVIDOR

- O paradigma de programação distribuída através da separação das aplicações entre servidores e clientes foi a arquitetura de distribuição predominante nos anos 1990. Um dos seus atrativos é:

Aumento da fiabilidade (a falha de uma máquina não inviabiliza necessariamente a operação do sistema como um todo).
Redução do custo (máquinas mais simples podem executar os serviços isoladamente, ao invés de ter uma grande máquina fazendo todos os serviços).

PROGRAMAÇÃO CLIENTE/SERVIDOR (2)

- Outros exemplos de aplicações que utilizam o modelo cliente/servidor:
  - Email (SMTP)
  - Transferência de arquivos (FTP)
  - Terminal remoto (telnet e SSH)
  - Sistema de nomes (DNS)
  - Base de dados (SQL)
  - etc

PROGRAMAÇÃO CLIENTE/SERVIDOR (3)

- Características de programas do tipo cliente:
  - Inicia a solicitação;
  - Aguarda pela resposta;
  - Normalmente interage com um número pequeno de servidores ao mesmo tempo.
- Características de programas do tipo servidor:
  - É passivo (aguarda solicitações dos clientes);
  - Quando recebe uma solicitação, processa e envia resposta;
  - Pode interagir com um grande número de clientes ao mesmo tempo.

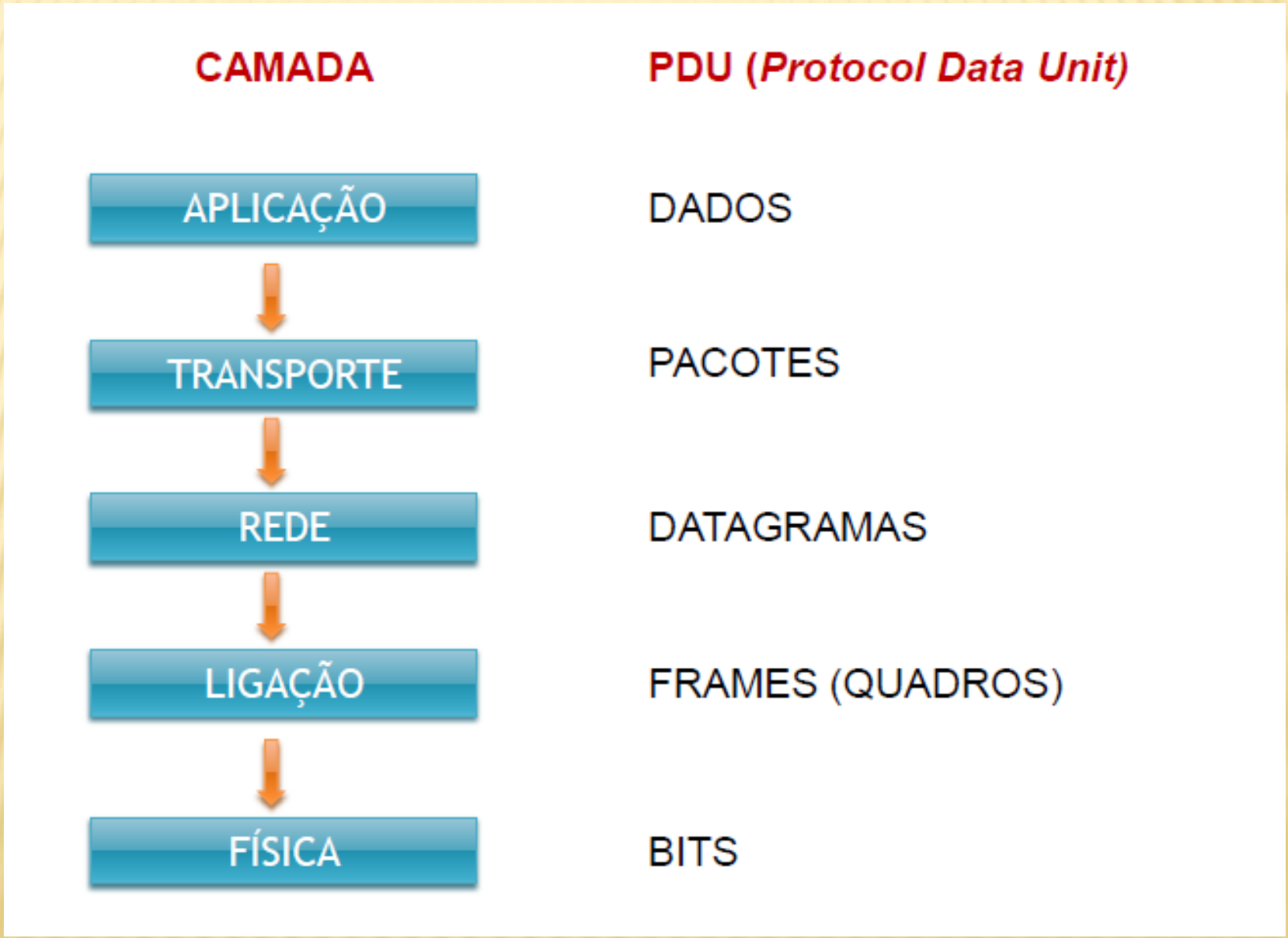
SERVIDOR MONOCLIENTE

- Um **monocliente**, é um computador cliente numa rede de modelo cliente/servidor de duas camadas.
  - Tem poucos ou nenhum aplicativo instalado;
  - Depende primeiramente de um servidor central para o processamento de atividades.
- A palavra mono refere-se a uma pequena imagem de *boot* apenas com o necessário para fazer a conexão com a rede e iniciar um navegador *web* dedicado ou uma conexão de “Área de Trabalho Remota”.

SERVIDOR MULTICLIENTE

- **Multicliente** é um computador cliente no cliente/servidor que normalmente fornece a funcionalidade independentemente da central servidor.
- Multicliente necessita, ainda, de pelo menos uma ligação periódica a uma rede ou servidor central, mas é muitas vezes caracterizado pela capacidade de executar várias funções.

MODELO DE CAMADAS TCP/IP



MODELO TCP/IP

- Segundo *Tanenbaum* o Modelo TCP/IP possui somente **quatro** camadas e não cinco como mostra o quadro abaixo.

Camada	Exemplo
5 – Aplicação *(camadas OSI 5 até 7)*	HTTP, FTP, DNS *(protocolos de routing como BGP e RIP, que, por uma variedade de razões, são executados sobre TCP e UDP respectivamente, podem também ser considerados parte da camada de rede)*
4 – Transporte *(camadas OSI 4 e 5)*	TCP, UDP, RTP, SCTP *(protocolos como OSPF, que é executado sobre IP, pode também ser considerado parte da camada de rede)*
3 – Rede *(camada OSI 3)*	Para TCP/IP o protocolo é IP *(protocolos requeridos como ICMP e IGMP é executado sobre IP, mas podem ainda ser considerados parte da camada de rede; ARP não roda sobre IP)*
2 – Enlace *(camada OSI 2)*	ARP
1 - Interface com a Rede *(camada OSI 1)*	Ethernet, Wi-Fi, MPLS,Modem, etc.

RESOLUÇÃO E FORMAÇÃO DE ENDEREÇOS IP

- Os endereços da Internet são conhecidos pelos nomes associados aos endereços IP.
  - (por exemplo, www.google.pt). Para que isto seja possível, é necessário traduzir (***resolver***) os nomes em endereços IP.
- O *Domain Name System*(DNS) é um mecanismo que converte os nomes em endereços IP e endereços IP em nomes.
- Os nomes DNS são hierárquicos e permitem que faixas de espaços de nomes sejam delegados a outros DNS.

RESOLUÇÃO E FORMAÇÃO DE ENDEREÇOS IP (2)

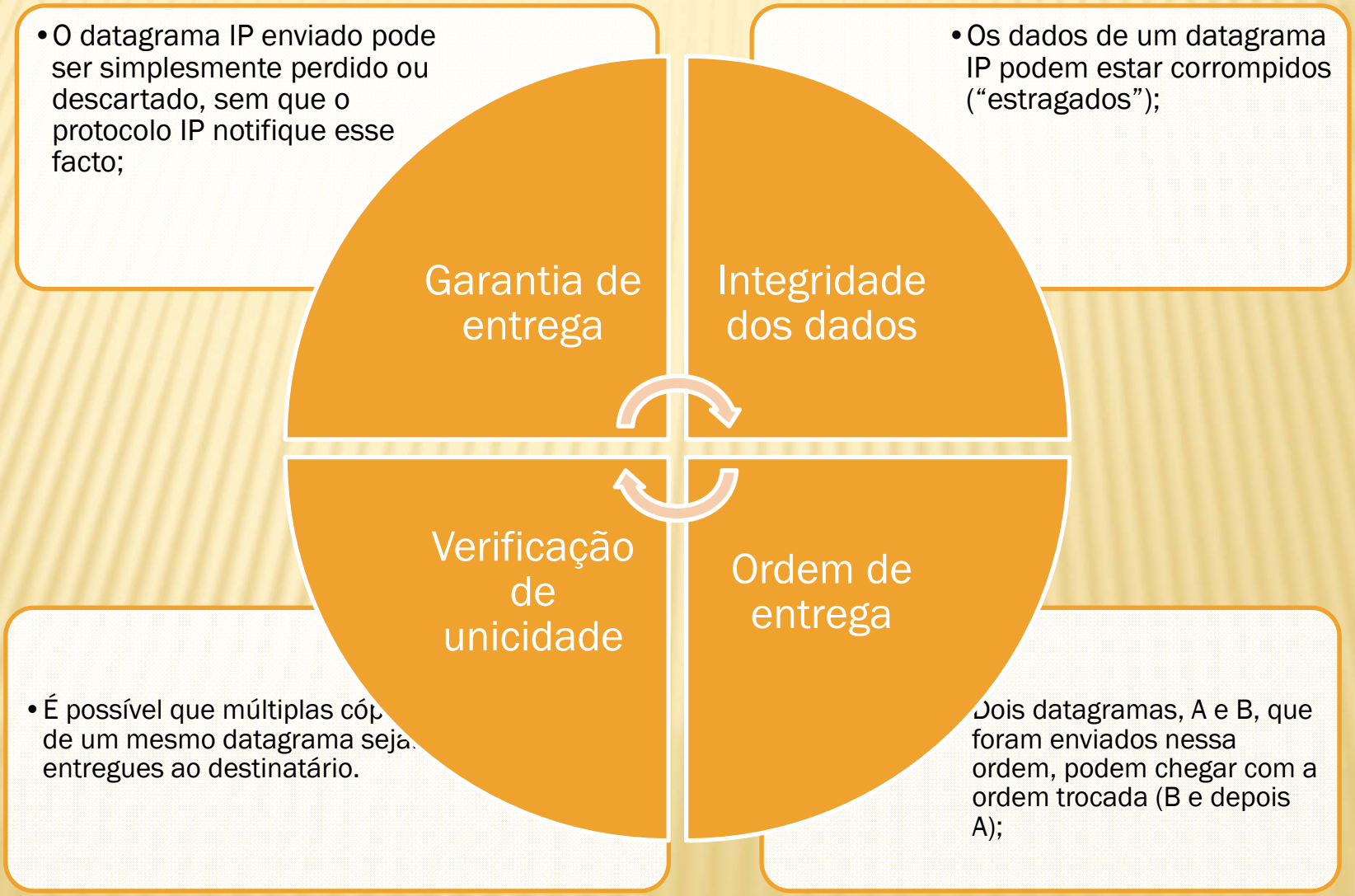
- Os computadores comunicam-se entre eles graças ao protocolo IP (*Internet Protocol*), que utiliza endereços numéricos, chamados endereços IP.
- São compostos por 4 números inteiros (4 bytes) entre 0 e 255 e escritos sob a forma de xxx.xxx.xxx.xxx. Por exemplo: 194.153.205.26.
- Estes endereços, numa rede, servem para que os computadores comuniquem entre si, **cada computador tem um endereço IP único**.
- O ICANN (*Internet Corporation for Assigned Names and Numbers*) é encarregue de atribuir endereços IP públicos, quer dizer os endereços IP dos computadores que estão ligados a uma rede pública de Internet.

RESOLUÇÃO E FORMAÇÃO DE ENDEREÇOS IP (3)

- Devemos notar que um endereço IP não identifica uma máquina individual, mas uma conexão à rede. Assim, um *gateway* conectado a *n redes tem 'n' endereços IP diferentes, um* para cada conexão.
- O endereço IP, na versão 4 (IPv4), é um número de 32 bits escrito com quatro octetos representados no formato decimal. (exemplo: 128.6.4.7).
- A versão 6 (IPv6) utiliza um número de 128 bits. Com isso dá para utilizar 25616endereços.
  - Exemplo: 2002:0000:0000:0015:0000:0000:0000:0001

PROTOCOLO IP

O protocolo IP, não provê os seguintes serviços:



PROTOCOLOS DA CAMADA DE TRANSPORTE

- Os serviços que o protocolo IP, da camada de rede não fornece, podem ser oferecidos pelos protocolos da camada de transporte (camada 4).
- Os dois principais protocolos da camada de transporte do modelo TCP/IP são:
  - TCP – *Transmission Control Protocol*
  - UDP – *User Datagram Protocol*

FORMATO DE UM PACOTE TCP E UDP

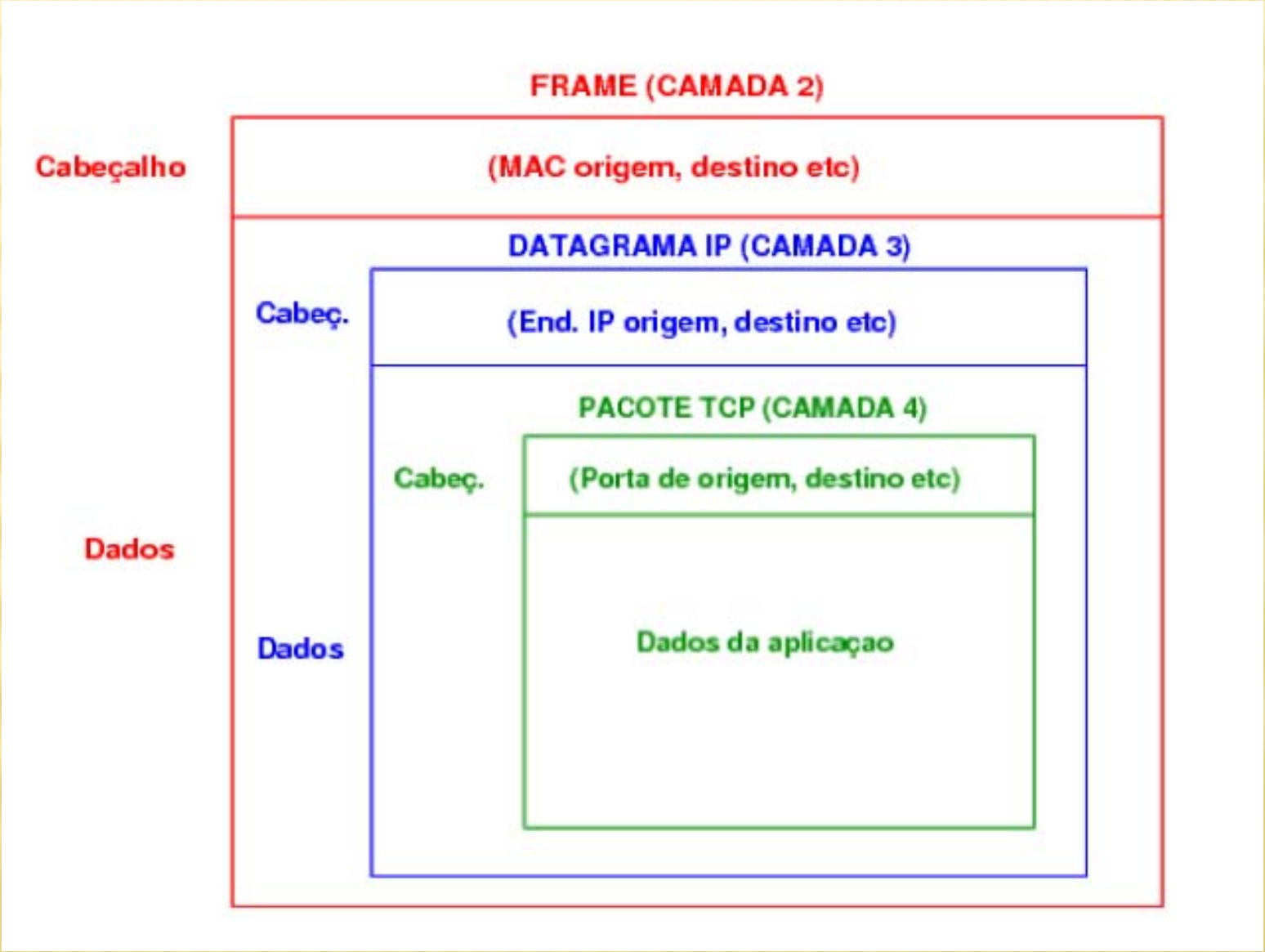
	8 bits	8 bits	8 bits	8 bits
0	Porta de Origem		Porta Destino	
32	Sequence Number			
64	Acknowledgment number			
96	Data Offset / Reserved	CWR ECE URG ACK PSH RST SYN FIN	Window	
128	Checksum		Urgent Pointer	
160	Options (Optional)			
160 ou >=192	Dados			

Pacote TCP

	8 bits	8 bits	8 bits	8 bits
0	Porta de Origem		Porta Destino	
32	Tamanho		Checksum	
64	Dados			

Pacote UDP

ENCAPSULAMENTO (EXEMPLO)



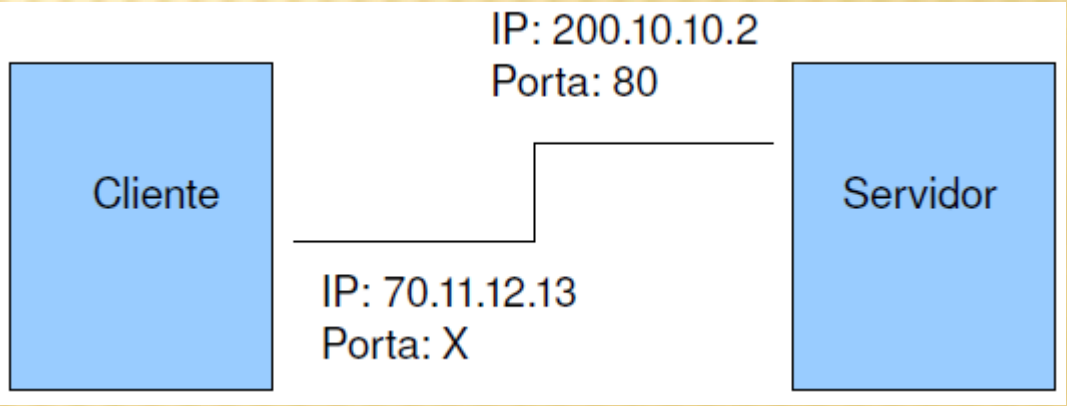
PORTAS

- Muitas vezes os servidores oferecem diversos tipos de serviços, na mesma máquina, através de diferentes protocolos:
  - Web (HTTP)
  - Email (SMTP)
  - Nomes (DNS)
  - etc
- Assim, de modo a permitir que tenhamos diversos "fluxos" simultâneos de informações vindos de diversos servidores para diversos clientes no computador do utilizador, existem as Portas.

"Portas" são números que identificam um "canal" de dados entre dois *hosts* trocando informações pela rede.

LIGAÇÕES TCP E UDP

- Para estabelecer a conexão TCP é preciso identificar as extremidades dessa conexão tanto no processo cliente como no processo servidor. Essas extremidades são soquetes, identificados por um endereço de rede e um número de porta.
- Uma conexão entre dois computadores utilizando TCP/IP (ou UDP/IP) é dada por quatro informações:
  - Endereço IP do servidor
  - Porta TCP ou UDP do servidor
  - Endereço IP do cliente
  - Porta TCP/UDP do cliente



PORTAS EFÉMERAS

- Antes de ocorrer a troca de informações, apenas três das quatro informações necessárias estão pré-estabelecidas:
  - O endereço IP do servidor;
  - A porta TCP ou UDP do servidor;
  - O endereço IP do cliente;
- A quarta informação (porta do cliente) é definida "aleatoriamente", serve e existe apenas durante uma troca de informações. Se houver nova troca, o número da porta será provavelmente diferente.
- Essa porta "aleatória" utilizada temporariamente pelo cliente é chamada de "porta efémera" (*ephemeral port*).
- Utilizando o comando "netstat -an" Pode-se ver as ligações TCP abertas, e respectivas portas utilizadas.

PORTAS BEM-CONHECIDAS

- Diversos serviços da Internet são bastante utilizados (WWW, Email, FTP, telnet, etc) e por isso os servidores desses serviços utilizam um número de porta fixo para o serviço.
- Esses números de portas “fixos” são conhecidos como “Portas Bem Conhecidas” (*Well Known Ports*).
- Algumas portas bem conhecidas:
  - HTTP (WWW): Porta 80/TCP
  - Email (SMTP): Porta 25/TCC
  - Telnet: Porta 23/TCP
  - POP3: Porta 110/TCP
  - Eco (retornar o mesmo que for enviado): Porta 7 –echo
  - Transferência de arquivos: Porta 21 –ftp
  - Tempo/data: Porta 37 -time

DIFERENÇAS TCP E UDP

TCP é orientado a conexão e o UDP não utiliza conexões	Antes de haver troca de dados entre dois hosts é necessário que o cliente “abra” uma ligação entre eles através de uma sequência chamada “handshake TCP” (descrito a seguir).
TCP é “pesado”	O TCP provê diversos serviços adicionais que o UDP não provê mas, justamente por isso, ele é mais complexo (e mais lento) que o UDP.
Os pacotes TCP são ordenados	Números de sequência no cabeçalho TCP permitem que o receptor dos pacotes ordene os mesmos caso cheguem fora de ordem. O UDP também não possui esse recurso.
TCP é confiável	Caso haja a perda de um pacote TCP no caminho e ele não chegar ao destino, o sistema TCP do remetente providencia o reenvio do pacote perdido. O UDP não tem essa funcionalidade

PORTAS TCP E UDP

- Os principais campos de um cabeçalho TCP ou UDP são:
- **Porta de destino (*destination port*):** identifica o serviço do servidor ao qual se deseja conectar.
  - (ex: porta 80 –WWW)
- **Porta de origem (*source port*):** identifica a porta do cliente a ser utilizada pelo servidor para responder a requisição feita.

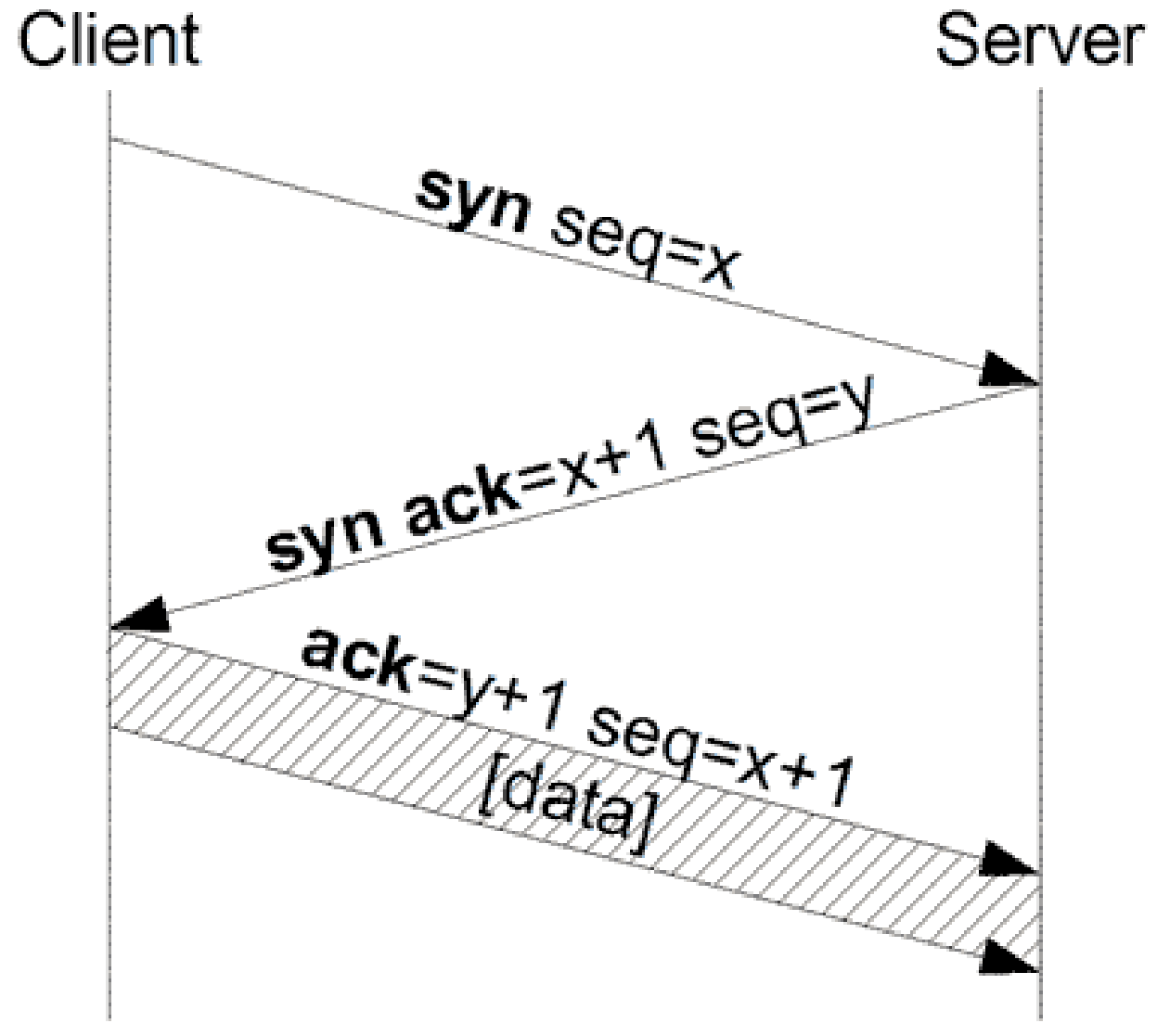
LOCALHOST

- A rede 128.0.0.0/8 é especial e não aparece em uso na Internet.
- Ela é reservada como “localnet” e todos os *hosts* dessa rede são tratados como “localhost”, especialmente o endereço 127.0.0.1.
- Qualquer ligação para este endereço é “retornada” para o próprio computador de origem.
  - Ex: o comando ping 127.0.0.1 é respondido pelo *localhost*

HANDSHAKE TCP

- Ao contrário do protocolo UDP, o protocolo TCP é “orientado a conexão”. Isso significa que antes de trocar dados o TCP necessita “abrir” uma ligação.
- Essa conexão é estabelecida através de uma sequência chamada “TCP handshake” (aperto de mão TCP) e é mostrada a seguir.
- Essa sequência envolve a troca de três pacotes e por isso é chamado de “three way handshake” (aperto de mão em três etapas).
- O pacote inicial do *handshake* é chamado pacote de sincronização e tem o bit SYN (*flag*) ligado.

HANDSHAKE TCP (2)



ESTADOS DAS PORTAS TCP

- O protocolo TCP é orientado a conexão, o que significa que antes de haver troca de dados é necessário que um computador estabeleça uma conexão TCP com outro através do “handshake TCP”;
- Quando um programa aceita receber conexão na porta TCP (servidor) dizemos que essa porta está em modo **LISTEN (ESCUTA)**;
- Quando há uma ligação estabelecida, a conexão passa para **ESTABLISHED (ESTABELECIDA)**;
- O comando *netstat* -a do Windows e do Unix mostra o estado das conexões.

MODELO OSI

- A ISSO (*International Standard Organization), é um organismo* de carácter geral que define padrões em praticamente todas as áreas da ciência e tecnologia.
- É responsável pela elaboração de um modelo de referência englobando todo o tipo de comunicações de dados, o **Modelo OSI** (*Open Systems Interconnection).*
  - P*ropõe uma arquitetura de 7 camadas para os* sistemas de comunicação de dados, dividindo assim modularmente as tarefas envolvidas no processo de comunicação de dados de modo a diminuir a sua complexidade e aumentar a sua compatibilidade;
  - Possibilita os equipamentos poderem ligar-se e comunicar entre si, independentemente das suas diferenças quer ao nível do *hardware, quer ao nível do software.*

MODELO OSI (2)

Modelo OSI		
7 - Aplicação	Localizam-se aqui as aplicações que requisitam serviços de rede através das outras camadas abaixo.	
6 - Apresentação	Entidades estabelecem os standards para a troca de dados como códigos de carácter e outros.	
5 - Sessão	Os pedidos de serviços são suportados nesta camada. Estabelece pontos de conexão entre duas entidades.	Sockets
4 - Transporte	Permite estabelecer níveis de transferência fiáveis entre as entidades que estão nas extremidades da comunicação.	Ex: TCP, UDP
3 - Rede	Funções de endereçamento, controlo de fluxo e encaminhamento.	Ex: IP
2 - Ligação de Dados	Funções de detecção e correcção de erros e controlo de fluxo.	Ex: Ethernet
1 - Física	Interface físico e adaptação ao meio.	

SOCKETS

- A comunicação entre processos de software tornou-se indispensável nos sistemas atuais. O elo de ligação entre os processos do servidor e do cliente é o *socket.*
- Ele é a “porta” na qual os processos enviam e recebem mensagens.
- De acordo com JAMES F KUROSE:

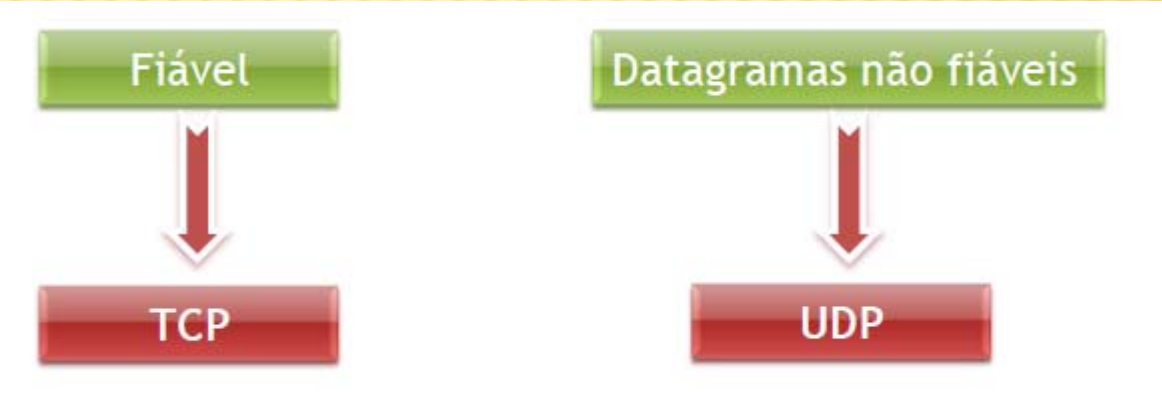
“socket é a interface entre a camada de aplicação e a de transporte dentro de uma máquina”.

SOCKETS (2)

- São utilizados nas redes entre programas cliente/servidor.
- Usados na camada de transporte por protocolos TCP e UDP.
- Permitem a comunicação entre dois processos (aplicações) a correr em máquinas distintas de uma rede.
  - Forma como o software comunica entre si.

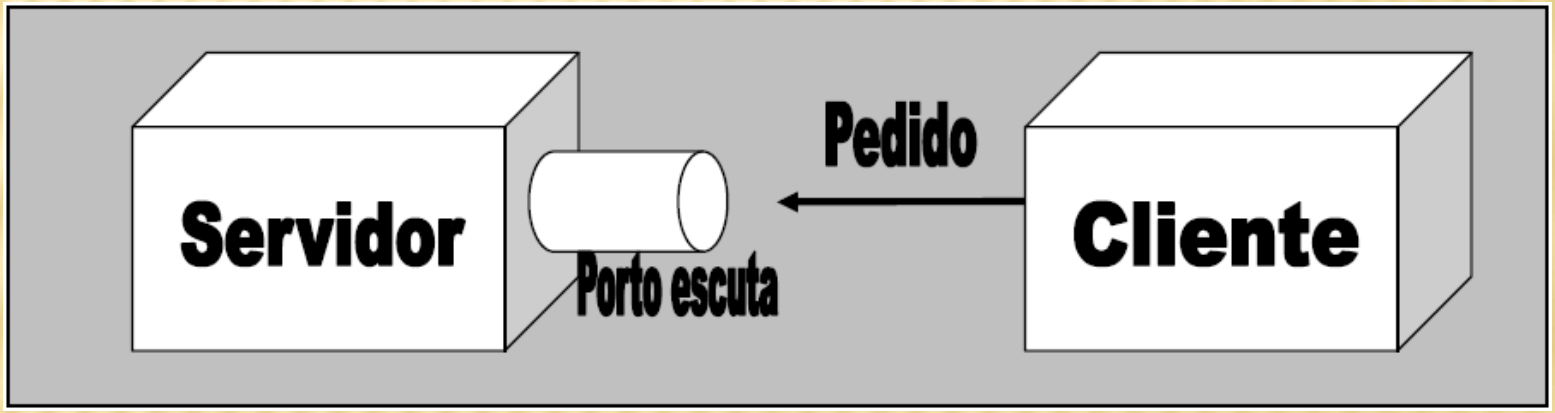
SOCKETS (3)

- Tudo acima da interface do *socket*, na camada de aplicação, é controlado pelo criador da aplicação. O controlo da camada de transporte é feito pelo Sistema Operativo.
- Temos dois tipos de serviços de transporte via *socket:* ***o fiável*** orientado à cadeia de bytes (byte *steam)* e os ***datagramas não fiáveis.***



SOCKETS TCP/IP

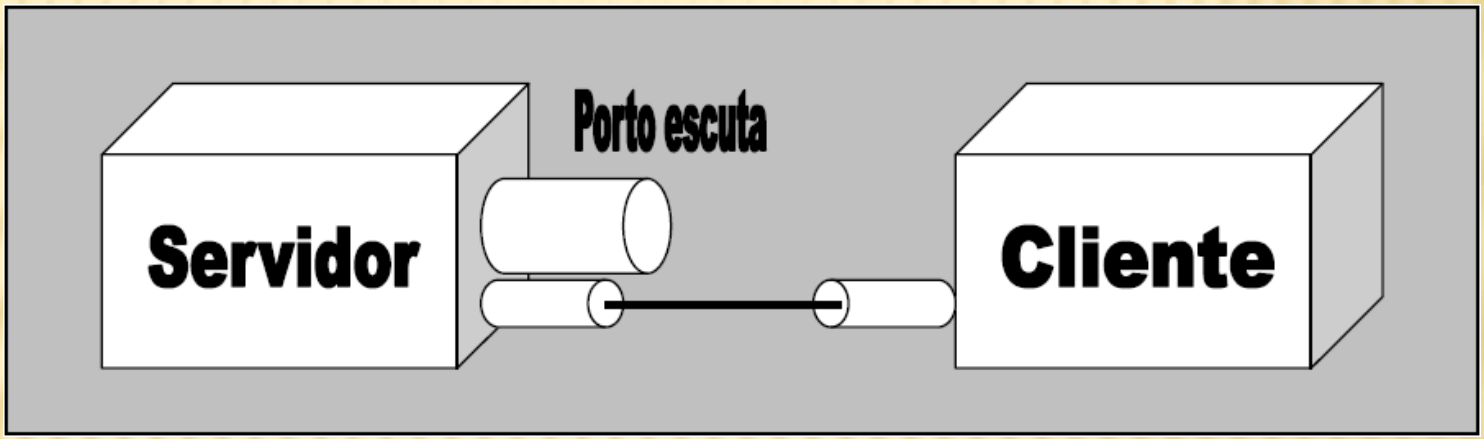
- O processo de comunicação no modo orientado à conexão ocorre da seguinte forma:
  - O servidor escolhe um determinado porto e ai aguarda alguma conexão.
  - O cliente, sabendo previamente qual o servidor e o porto em que aguarda, solicita uma conexão.



Pedido do cliente ao porto de escuta do servidor

SOCKETS TCP/IP (2)

- Se não ocorrer nenhum problema, o servidor aceita o pedido, gerando um *socket*, num porto qualquer criando assim um canal de comunicação.



Criação do Socket de comunicação entre o servidor e cliente

- Tipicamente, dependendo da utilização do programa, o servidor fica em escuta contínua onde aguarda novas conexões, gerando *sockets,* atendendo ás solicitações dos clientes.

CRIAÇÃO DE SOCKETS (EM PHP)

Socket_create(int domain, int type, int protocol)

1º Parâmetro - DOMAIN

AF_INET refere-se a protocolos baseados em IPV4 como TCP e UDP

AF_INET6 refere-se a protocolos baseados em IPV6, normalmente TCP e UDP

2º Parâmetro - TYPE

SOCK_STREAM para usar um tipo de comunicação baseada em TCP

(orientado à conexão)

OCK_DGRAM para usar um tipo de comunicação baseada em UDP

(não orientado à conexão)

3º Parâmetro - PROTOCOL

SOL_TCP para usar o TCP

SOL_UDP para usar o UDP

ASSOCIAÇÃO DE SOCKETS COM A LIGAÇÃO

A associação de *sockets* com a ligação é feita através da seguinte instrução:

Socket_bind(resource $socket, string $address, int $port)

1º Parâmetro - SOCKET

Nome do *socket* criado em "Criação de *Sockets*" (*socket* servidor)

2º Parâmetro - ADDRESS

Endereço IP da nossa máquina (servidor)

3º Parâmetro - PORT

Porta a associar ao *socket* (Porta livre acima de 1024)

ESPERAR POR CLIENTES (TCP)

Um servidor necessita de esperar que algum cliente procure ligar-se a ele (TCP). Através da função que se segue será possível para um servidor estar constantemente a “ouvir” o meio, à espera que alguém se tente ligar a ele na porta escolhida que disponibilizou para o efeito.

Socket_listen(resource $socket, int $backlog)

1º Parâmetro - SOCKET

Nome do *socket* criado em “Criação de *Sockets*” (*socket* servidor)

2º Parâmetro - BACKLOG

Número de clientes em fila de espera

ACEITAR CLIENTES (TCP)

A aceitação de clientes é realizada apenas quando se trata de um *socket* em TCP. Para além de ser necessário ouvir o meio é fundamental poder aceitar os clientes quando estes tentam comunicar com o servidor. Isto é conseguido através da função seguinte:

Socket_accept(resource $socket)

1º Parâmetro - SOCKET

Nome do *socket* criado em "Criação de *Sockets*" (*socket* servidor)

ESTABELECIMENTO DE UMA LIGAÇÃO (TCP)

Esta função é utilizada do lado do cliente em TCP para que se estabeleça uma ligação entre este e o servidor. A função é muito semelhante ao ***socket_bind()***que vimos anteriormente:

Socket_connect(resource $socket, string $address, [int $port=0])

1º Parâmetro - SOCKET

Nome do *socket* criado em "Criação de *Sockets*" (*socket* servidor)

2º Parâmetro - ADDRESS

Endereço IP do servidor que criou o *Socket*

3º Parâmetro - PORT

Porta a associar ao *socket* no servidor

ENVIO/RECEPÇÃO VIA SOCKET TCP

Para realizar a recepção de informação através de um socket em TCP, utiliza-se a seguinte função:

Socket_read(resource $socket, int $length, [int $type])

1º Parâmetro - SOCKET

Nome do *socket* criado em "Criação de *Sockets*" (*socket* servidor)

2º Parâmetro - LENGTH

Tamanho em bytes da mensagem a enviar

3º Parâmetro - TYPE

Campo opcional mas que por defeito fará a sua leitura em modo binário. No âmbito desta disciplina deverá ser mudado para o modo normal de leitura. Assim, neste campo deverá ser escrito **PHP_NORMAL_READ**. Neste caso a leitura irá parar sempre que encontrar \r ou \ n

ENVIO/RECEPÇÃO VIA SOCKET TCP (2)

Por outro lado se pretendermos escrever no socket em TCP, utiliza-se a seguinte função:

Socket_read(resource $socket, string $buffer, [int $length = 0])

1º Parâmetro - SOCKET

Nome do *socket* criado em "Criação de *Sockets*" (*socket* servidor)

2º Parâmetro - BUFFER

Texto a enviar

3º Parâmetro - LENGTH

Campo opcional onde se poderá fazer referência ao número de bytes a escrever no *Socket*

ENVIO/RECEPÇÃO VIA SOCKET UDP

Para realizar a recepção de informação através de um socket em UDP, utiliza-se a seguinte função:

Socket_recvfrom(resource $socket, string & $buf, int $len, int $flags, string & $name, [int & $port])
1º Parâmetro - SOCKET Nome do *socket* criado em "Criação de *Sockets*" (*socket* servidor)
2º Parâmetro - BUF Informação que estiver no *Socket* será enviada para esta variável
3º Parâmetro - LEN Tamanho da informação a ler do *Socket*
4º Parâmetro - FLAGS O campo não é relevante para a disciplina. Por defeito Colocar 0
5º Parâmetro - NAME IP da estação (passada por variável) da qual está a receber a informação
6º Parâmetro - PORT Porta (passada por variável) utilizada no *Socket*

ENVIO/RECEPÇÃO VIA SOCKET UDP (2)

Por outro lado se pretendermos escrever no socket em UDP, utiliza-se a seguinte função:

Socket_sendto(resource $socket, string $buf, int $len, int $flags, string $addr, [int & $port])
1º Parâmetro - SOCKET Nome do *socket* criado em "Criação de *Sockets*" (*socket* servidor)
2º Parâmetro - BUF Informação a enviar para o *socket*
3º Parâmetro - LEN Tamanho da informação a enviar para o *Socket*
4º Parâmetro - FLAGS O campo não é relevante para a disciplina. Por defeito Colocar 0
5º Parâmetro - ADDR IP remoto da estação para a qual se pretende enviar informação
6º Parâmetro - PORT Porta remota da estação para a qual se pretende enviar informação

DESTRUIÇÃO DE UM SOCKET

Tanto do lado do cliente como do lado do servidor é necessário, no final da comunicação, desligar-se o *socket*, ou melhor, destruí-lo. Para o fazer corretamente é necessário utilizarem-se duas funções, pela seguinte ordem:

Socket_shutdown(resource $socket, [int $how = 2])

1º Parâmetro - SOCKET

Nome do *socket* criado em "Criação de *Sockets*" (*socket* servidor)

2º Parâmetro - HOW

Valor inteiro entre 0 e 2

0 - encerra a possibilidade de leitura no Socket

1 - encerra a possibilidade de escrita no Socket

2- encerra a possibilidade de leitura e escrita no Socket

DESTRUIÇÃO DE UM SOCKET (2)

Posteriormente, deve-se fechar definitivamente o socket através da seguinte instrução:

Socket_close(resource $socket)
1º Parâmetro - SOCKET Nome do *socket* criado em "Criação de *Sockets*" (*socket* servidor)

TRATAMENTO DE ERROS

O PHP disponibiliza funções próprias para o tratamento de erros relacionados com *sockets*. A função que iremos utilizar é a seguinte:

Socket_strerror(int $errno)
1º Parâmetro - ERRNO Variável ou função que será analisada para erro

SOCKETS TCP

- Basicamente são necessários 4 passos para criar um *socket* TCP num servidor:
  1. Criação do *socket* TCP;
  2. União entre o *socket* criado com um IP e uma porta;
  3. Esperar por clientes;
  4. Aceitar clientes.
- A partir deste momento, é possível utilizarem-se as funções de leitura e escrita em *sockets* (TCP) para iniciar a sua utilização. No final, os *sockets* devem ser destruídos.

SOCKETS TCP (2)

- Do lado do cliente o processo é idêntico mas mais simples. Vejamos:
  1. Criação do socket TCP;
  2. Estabelecimento de ligação com o servidor.
- A partir deste momento é possível trocar informação com o servidor usando as funções para esse efeito. No final o socket deverá ser destruído.

ESQUEMA GLOBAL