

30-769

Sistemas Distribuídos

MSc. Fernando Schubert

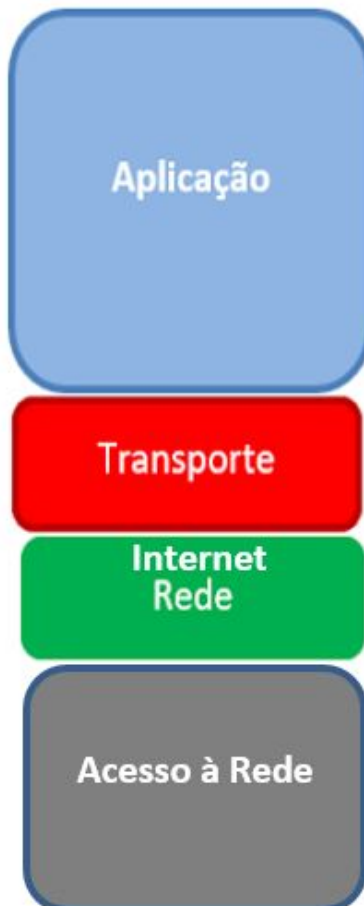
COMUNICAÇÃO EM SISTEMAS DISTRIBUÍDOS

- A comunicação interprocessos está no cerne de todos os sistemas distribuídos.
- Baseada na troca de mensagens utilizando-se da pilha (*stack*) de rede.
 - Maior complexidade do que em plataformas não-distribuídas
 - Memória não compartilhada
 - Comunicação não-confiável

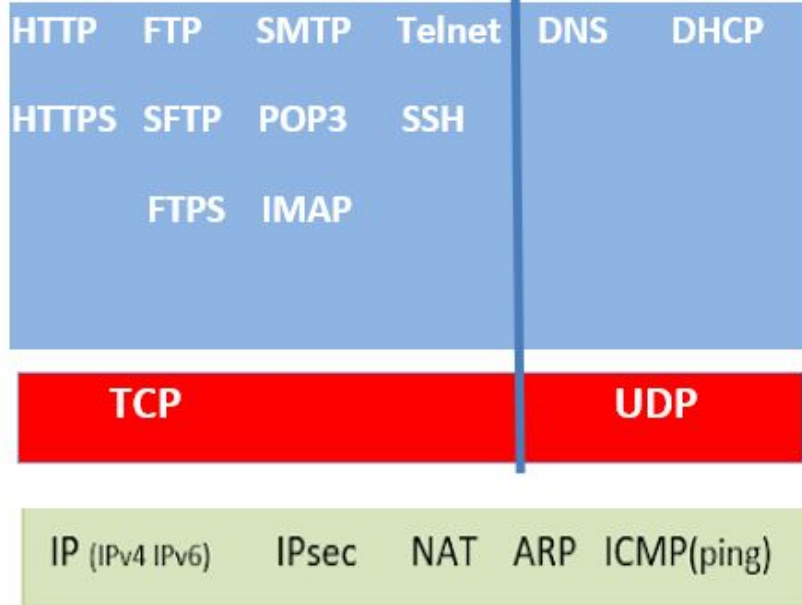
MODELO OSI



MODELO TCP/IP



Web	Transferência Ficheiros	Email	Acesso Remoto	Sistemas Nomes	Configuração Hosts
-----	----------------------------	-------	------------------	-------------------	-----------------------



Ethernet	Token Ring	Frame Relay	ATM	802.11 Wireless
----------	---------------	----------------	-----	--------------------

MODELO OSI



Computador 1



Cada camada, passa para a imediatamente inferior.

Cada camada, passa para a imediatamente superior.

Computador 2



Meio Físico



MODELO OSI

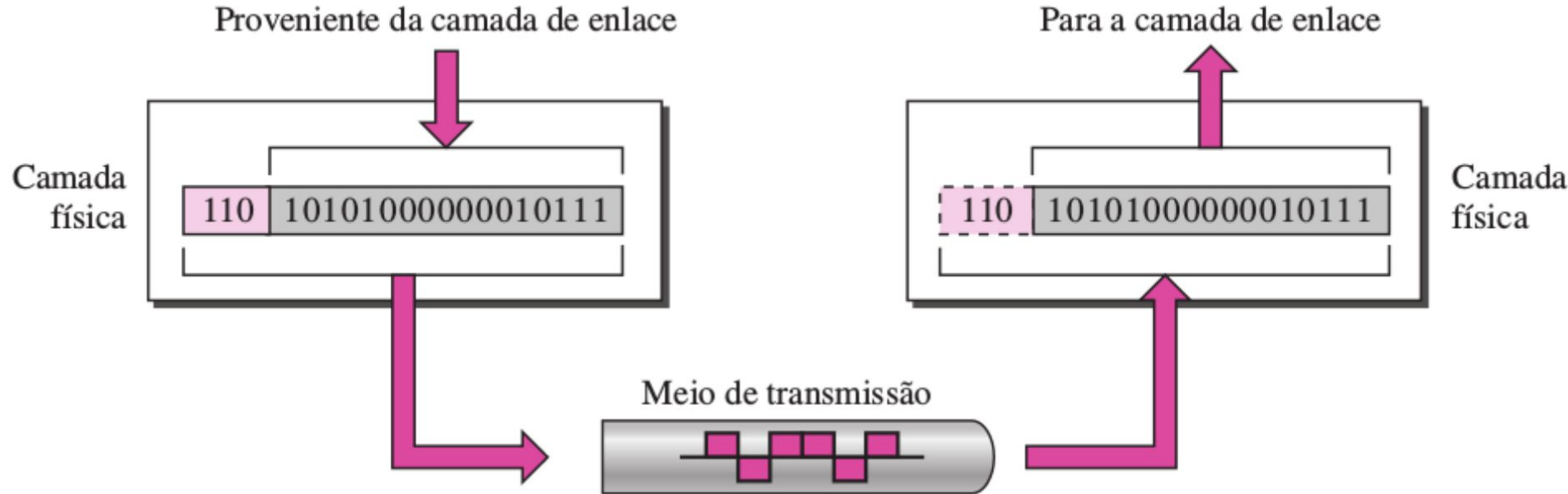
- **Prós:** bastante geral e continua válido até hoje
- **Contras:** protocolos associados ao modelo OSI são raramente usados
- **Críticas:**
 - Complexidade
 - Cada camada deve desempenhar a sua função antes de encaminhar os dados para a camada seguinte
 - Rigidez de modelagem
 - Camadas diferentes não devem compartilhar informações
 - Mesmos serviços implementados por diferentes camadas
 - Ex.: correção de erros

MODELO OSI - CAMADA FÍSICA

Funções:

- E responsável pela transmissão de bits.
- Características físicas (mecânicas e elétricas) das interfaces (conectores) e dos meios de transmissão.
- Define quais os tipos de meio de transmissão devem ser utilizados (cabo par trançado, fibra óptica, cabo coaxial, etc.). Quantos pinos o conector de rede terá e qual será a finalidade de cada pino;
- Representação dos dados: define a codificação dos dados em sinais (elétricos ou ópticos);
- Taxa de transferência dos dados: corresponde ao número de bits por segundo, isto é, define o tempo de duração de um bit no meio;
- Sincronização dos bits: os relógios do transmissor e do receptor devem estar sincronizados.
- Modo de transmissão: define o sentido da transmissão (simplex, half-duplex ou full-duplex).
- Topologia.

MODELO OSI - CAMADA FÍSICA

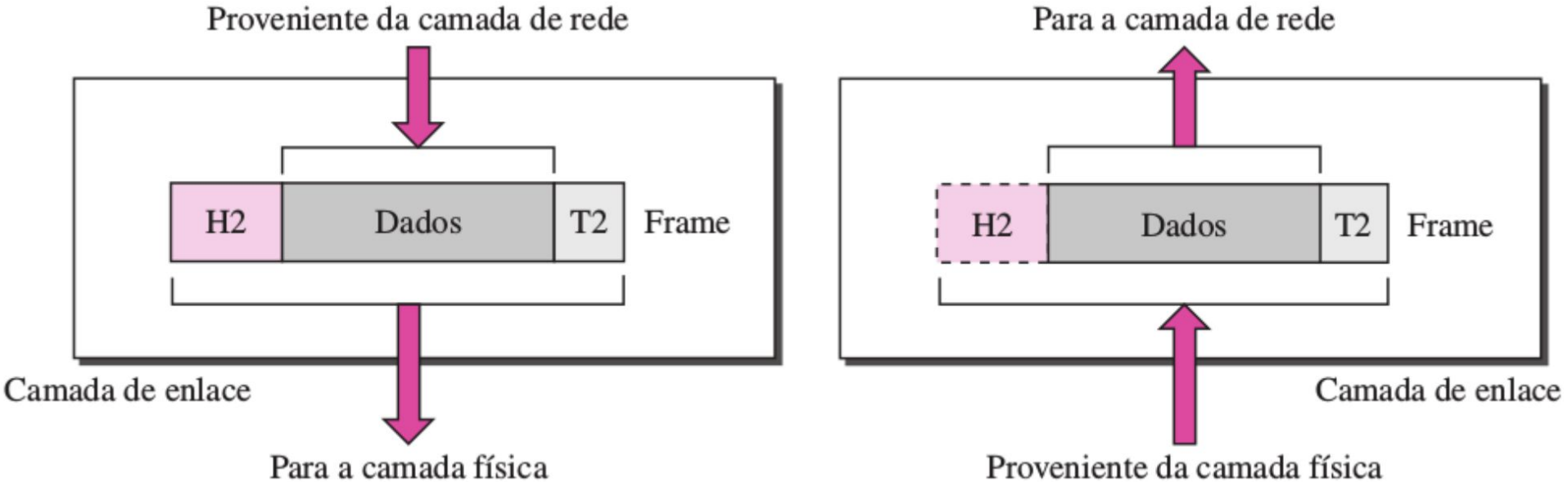


MODELO OSI - CAMADA DE ENLACE DE DADOS

Funções:

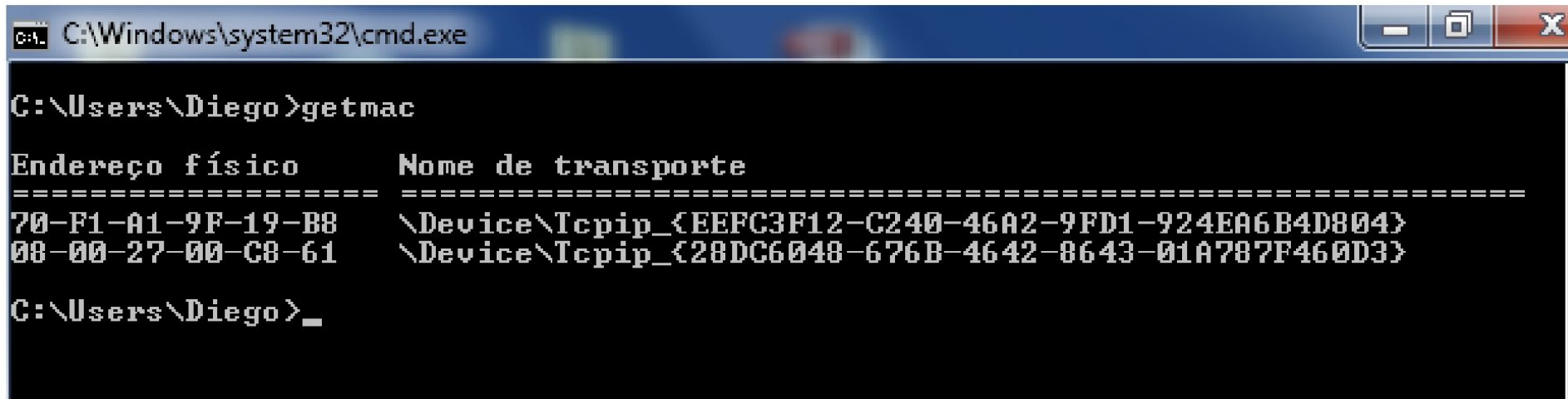
- Endereçamento físico (MAC): define o transmissor e o receptor local do quadro específico;
- Enquadramento: divide a cadeia de bits recebidos em unidades denominados quadros ou frames.
- Controle de fluxo: para evitar que o transmissor envie uma quantidade de dados maior do que o receptor pode processar;
- Controle de erro: tem a finalidade de propor confiabilidade aos dados recebidos, através de um mecanismo de detecção de erros e descarte de quadros;
- Controle de acesso: se existirem muitos computadores e todos desejarem enviar os dados ao mesmo tempo.

MODELO OSI - CAMADA DE ENLACE DE DADOS



MODELO OSI - CAMADA DE ENLACE DE DADOS

- MAC Address(48 bits)
 - 3 octetos identificam o fabricante;
 - 3 octetos identificam a interface;



```
C:\Windows\system32\cmd.exe

C:\Users\Diego>getmac

Endereço físico      Nome de transporte
=====
70-F1-A1-9F-19-B8    \Device\Tcpip_{EEFC3F12-C240-46A2-9FD1-924EA6B4D804}
08-00-27-00-C8-61    \Device\Tcpip_{28DC6048-676B-4642-8643-01A787F460D3}

C:\Users\Diego>_
```

Para visualizar o endereço MAC da interface digite no prompt `ipconfig /all` ou `getmac` em máquinas Windows, ou `ifconfig` em ambiente Linux.

MODELO OSI - CAMADA DE REDE

Funções:

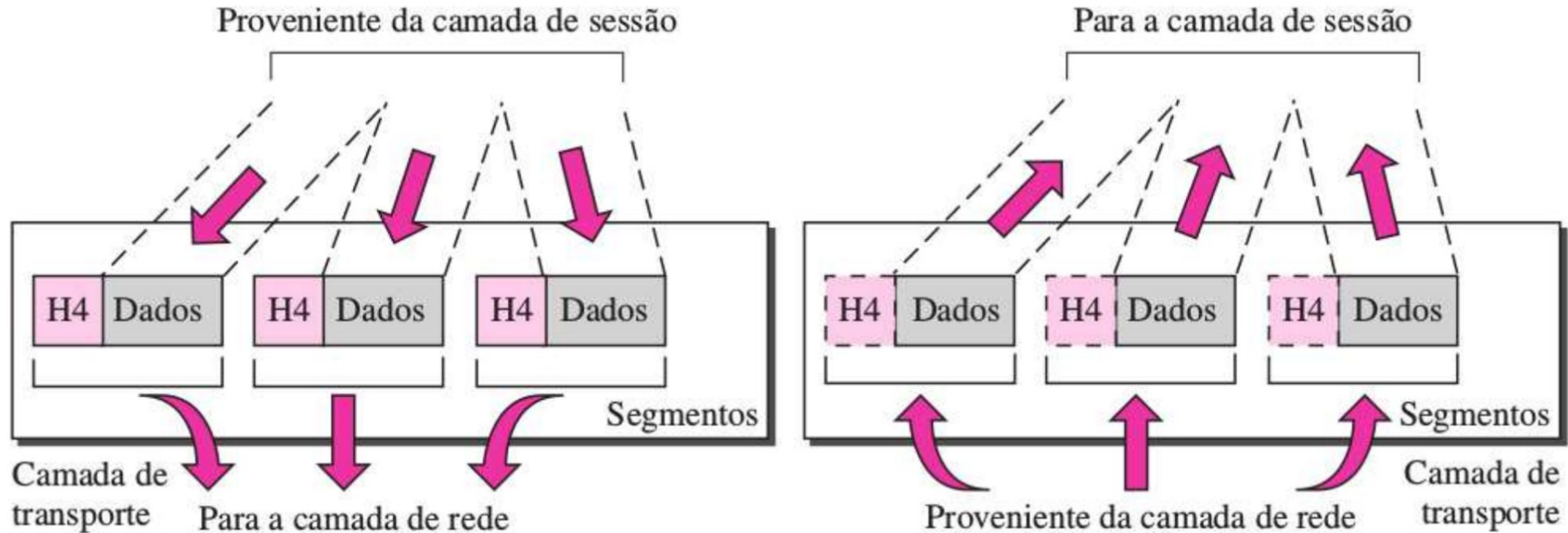
- Roteamento: determinar a maneira como os pacotes são roteados da origem até o destino;
- Qualidade do serviço fornecido (retardo, tempo de trânsito, instabilidade etc.) através da escolha das melhores rotas.
- Endereçamento lógico dos pacotes;
 - Tradução de endereços lógicos em endereços físicos;

MODELO OSI - CAMADA DE TRANSPORTE

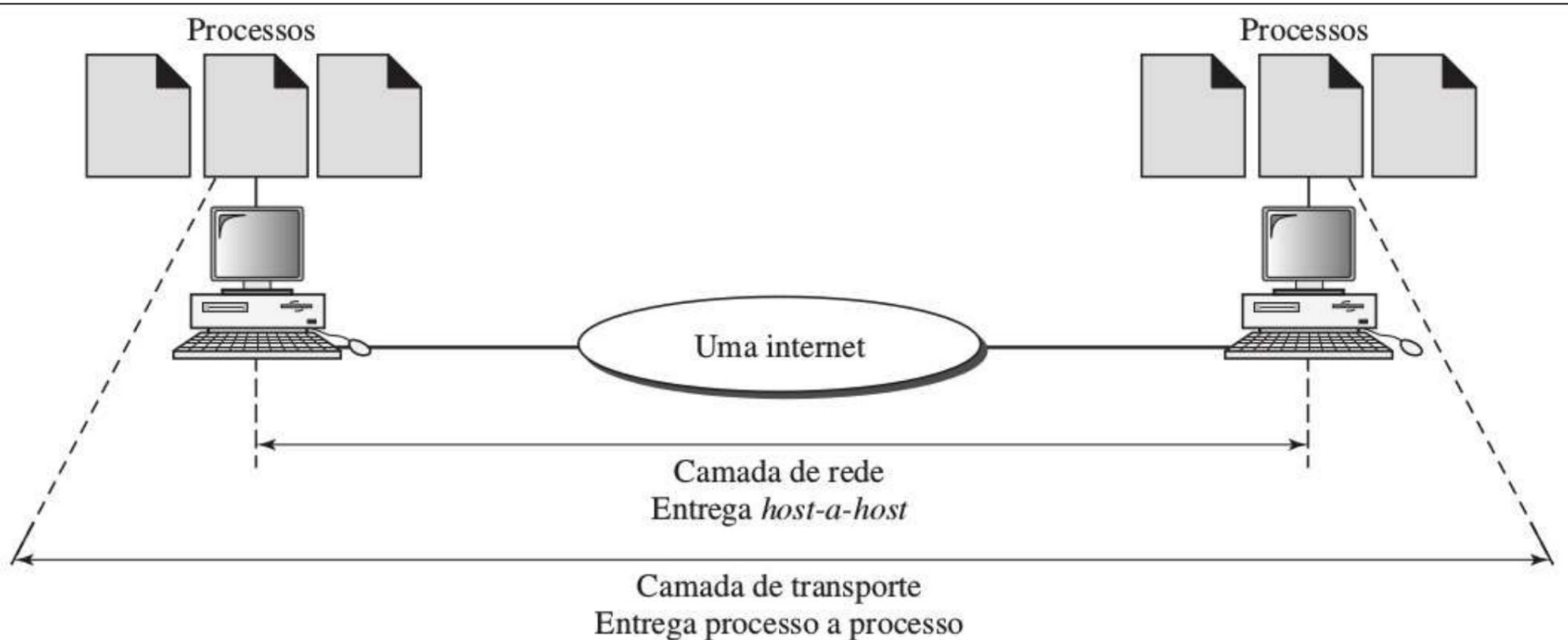
Funções:

- É responsável pela entrega de mensagens, de um programa a outro;
- Endereçamento de portas: utiliza um tipo de endereçamento que especifique o programas que esta utilizando os recursos da rede
- Segmentação e reagrupamento: permite dividir uma mensagem em vários segmentos de tamanhos variáveis, onde cada segmento contém um número de identificação. Com este número é possível o receptor remontar, identificar e/ou substituir pacotes extraviados;
- Controle do enlace: para garantir a integridade dos dados, a camada de transporte permite a orientação à conexão, estabelecendo conectividade fim-a-fim entre aplicações.
- Controle de fluxo: realiza um controle de fluxo fim a fim;
- Controle de erros: realiza um controle de erro fim a fim. Assegura que toda a mensagem chegue ao destino final livre de erros. A correção de erros normalmente se faz através de um pedido de retransmissão

MODELO OSI - CAMADA DE TRANSPORTE



MODELO OSI - CAMADA DE TRANSPORTE

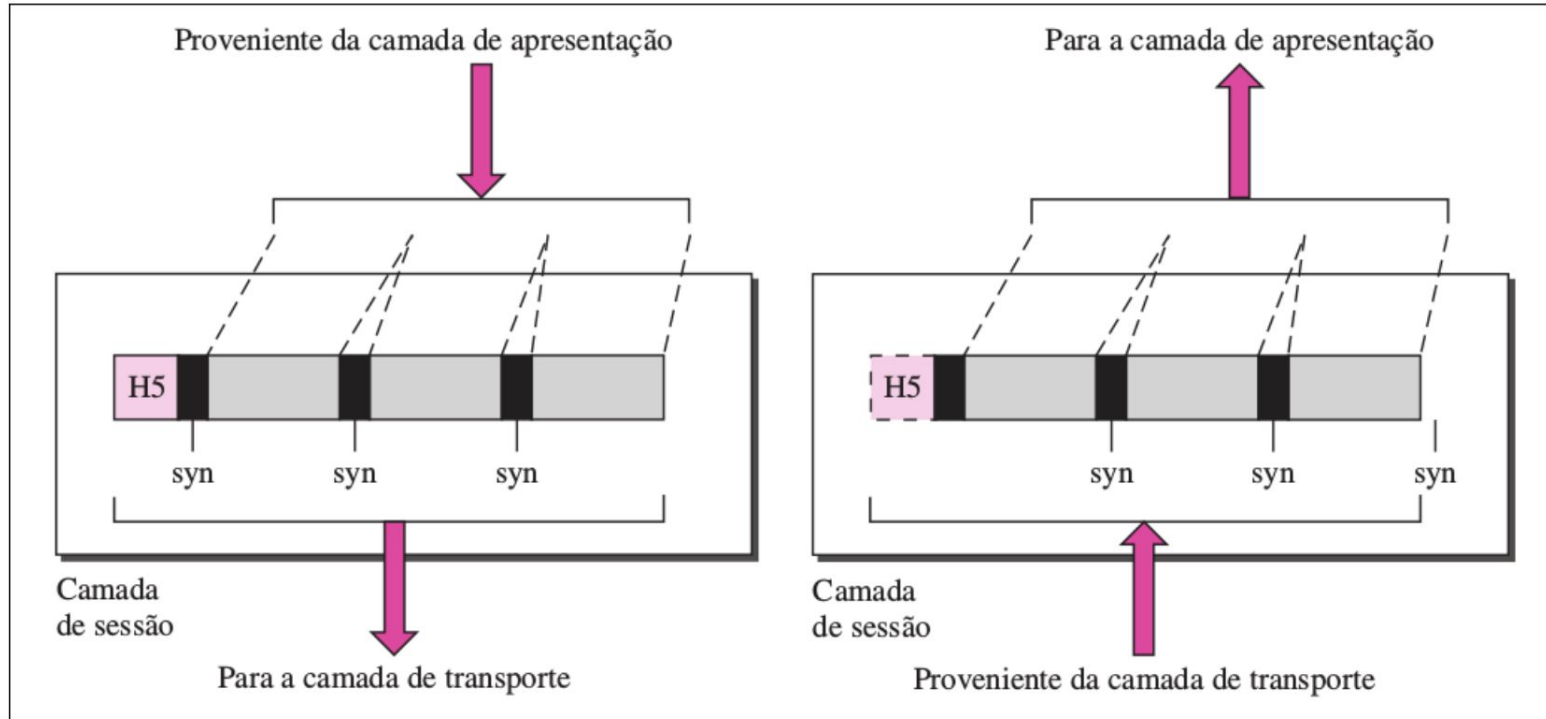


MODELO OSI - CAMADA DE SESSÃO

Funções:

- Controle de diálogo: determina quem deve transmitir em cada momento;
- Sincronização: realizar uma verificação periódica de transmissões longas. Esta verificação permite que retransmissão continuem a partir do ponto em que estavam ao ocorrer uma falha.

MODELO OSI - CAMADA DE SESSÃO



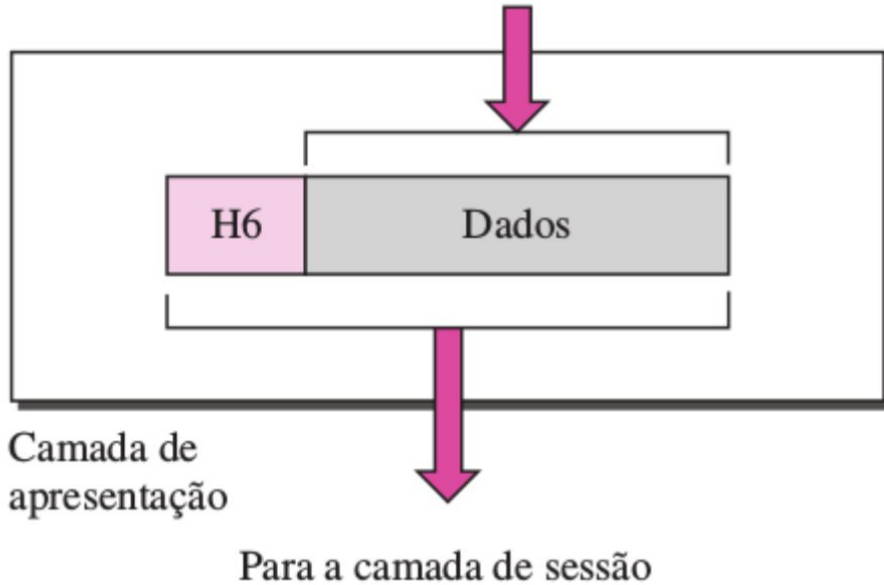
MODELO OSI - CAMADA DE APRESENTAÇÃO

Funções:

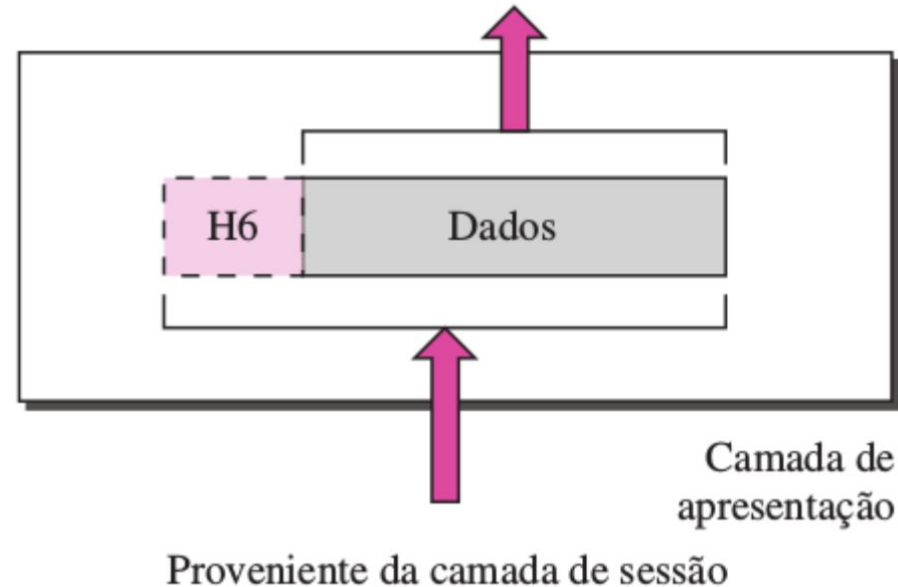
- Tradução: Como diferentes programas utilizam sistemas de codificação diferentes, a camada de apresentação é responsável pela interoperabilidade entre esses métodos de codificação diferentes.
- O transmissor traduz as informações para um formato padrão.
- O receptor traduz o formato padrão num formato específico do receptor;
- Compressão: reduz o número de bits contidos nas informações;
- Criptografia: o emissor converte as informações originais em um outro formato codificado e envia a mensagem resultante pela rede.
- O receptor reverte o processo original convertendo a mensagem de volta ao seu formato original.

MODELO OSI - CAMADA DE APRESENTAÇÃO

Proveniente da camada de apresentação



Para a camada de apresentação

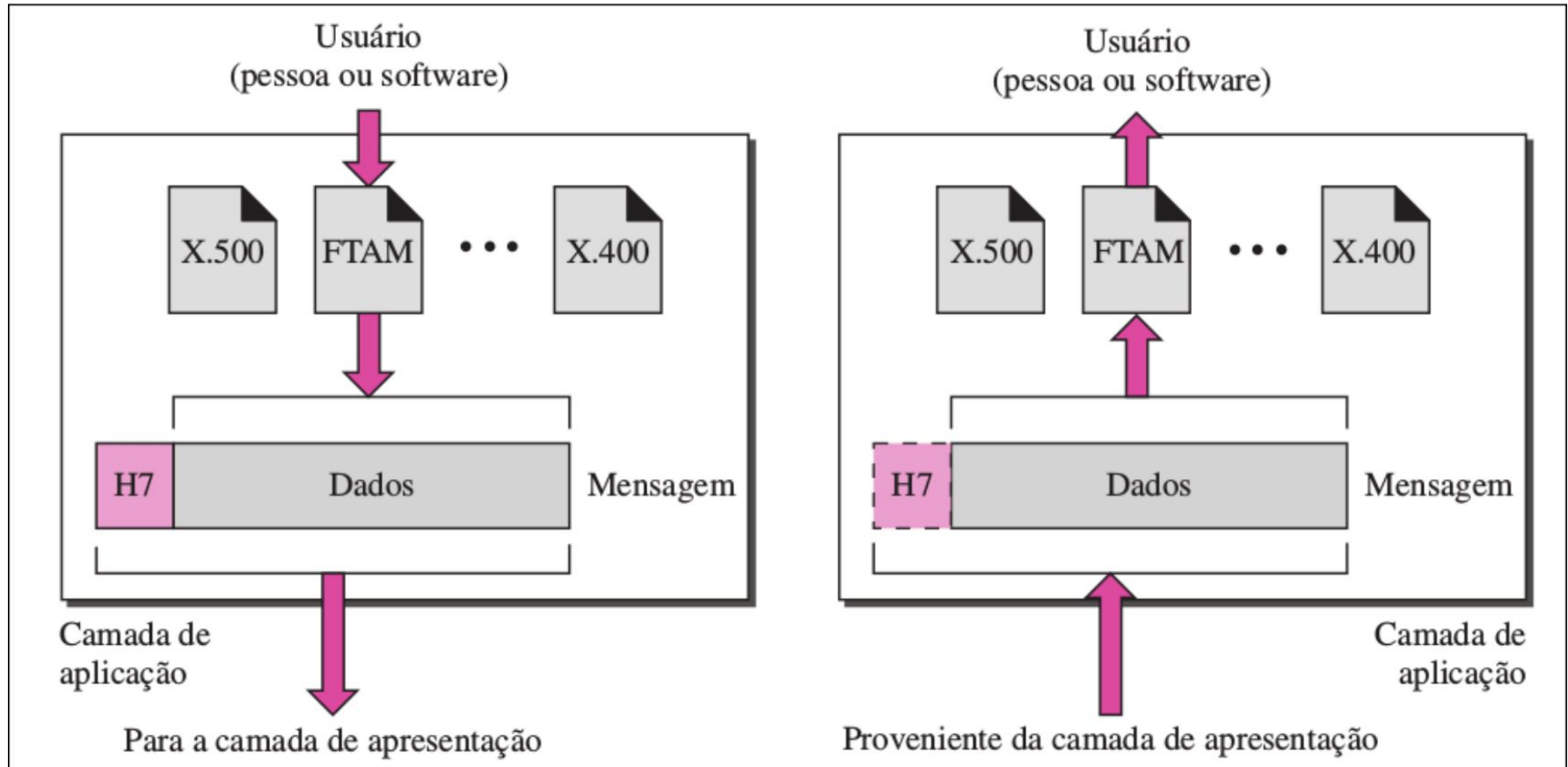


MODELO OSI - CAMADA DE APLICAÇÃO

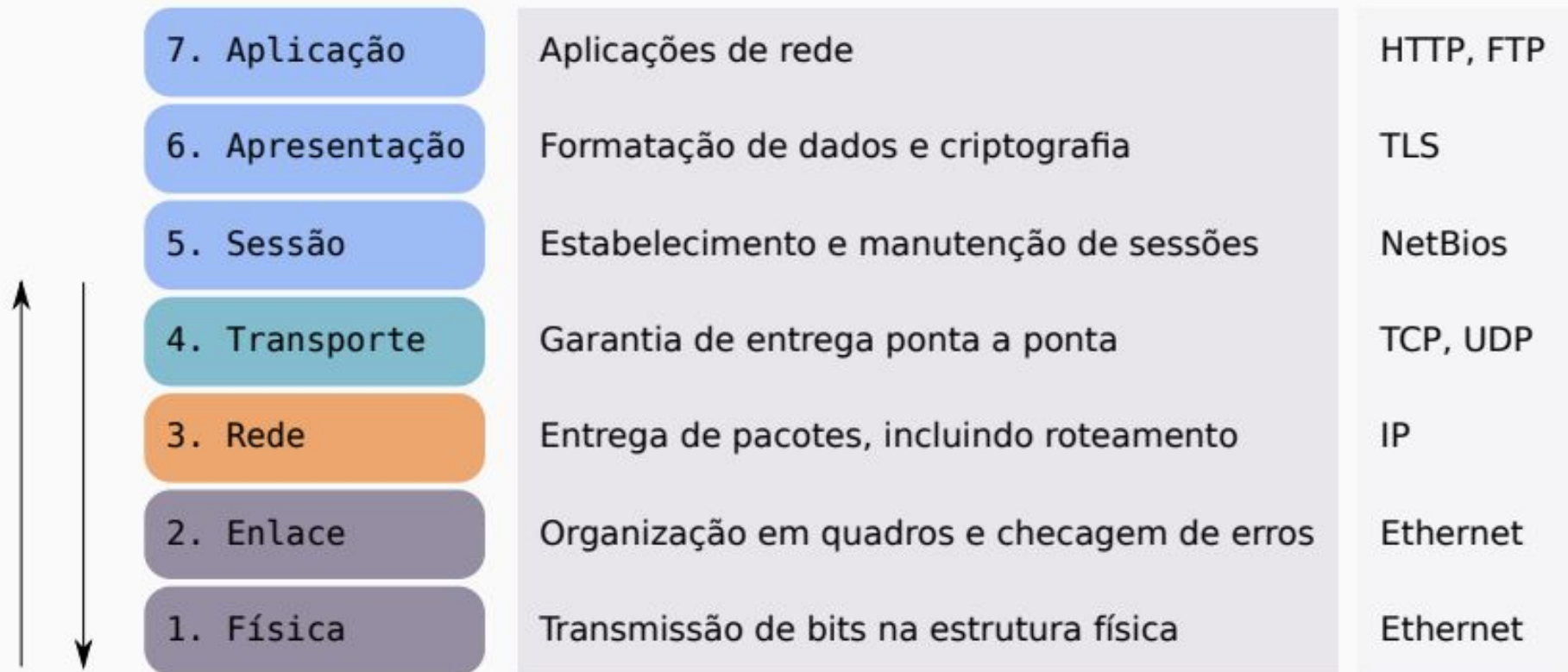
Funções:

- É responsável por prover serviços ao usuário. Provê interfaces e suporta serviços, tais como: Serviço de correio eletrônico (SMTP), Acesso e transferência de arquivos (FTP), Terminal remoto (Telnet), Acesso à World Wide Web (HTTP). Ou seja, Permitir ao usuário final o acesso aos recursos da rede

MODELO OSI - CAMADA DE APLICAÇÃO



- Modelo teórico de referência (definido pela ISO - International Standards Organization) para a criação/análise de modelos de rede.



ARQUITETURAS DE COMUNICAÇÃO

- Cliente-servidor:
 - Servidor: oferece um serviço
 - aguarda conexão de um cliente
 - recebe e processa pedido do cliente
 - retorna resultado
 - Cliente: demanda um serviço
 - se conecta ao servidor
 - envia pedido
 - aguarda resposta
 - Papel cliente/servidor bem distintos

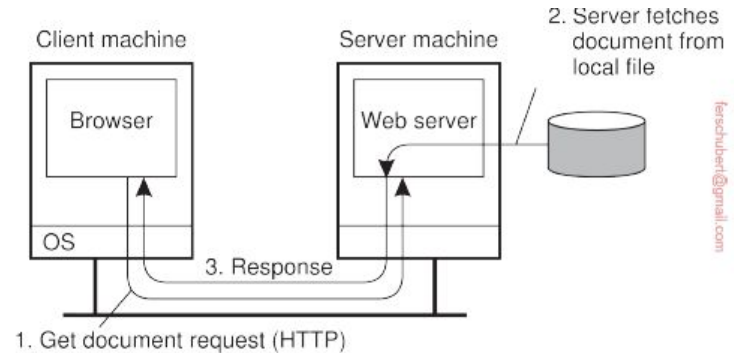
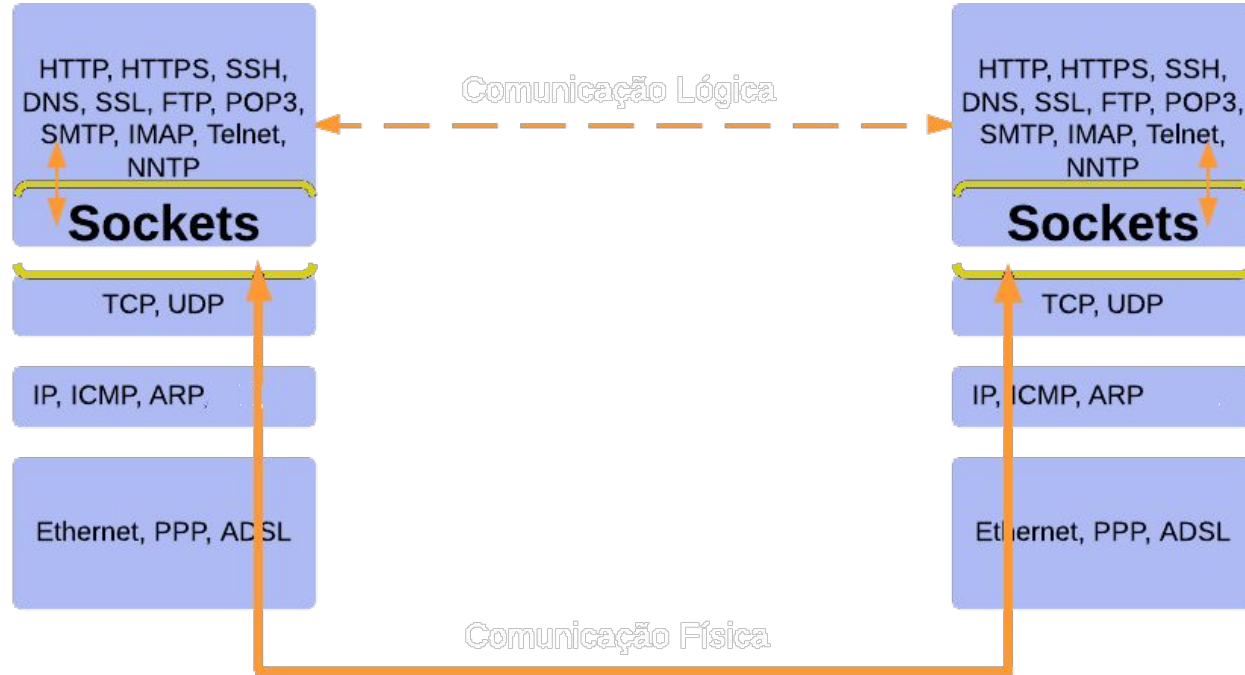


Figure 2.21: The overall organization of a traditional Web site.

ARQUITETURAS DE COMUNICAÇÃO



ARQUITETURAS DE COMUNICAÇÃO

Cliente estabelece conexão
(via sockets)

Servidor aceita conexão
(via sockets), dedica thread
para atender cliente

Cliente envia pedido
usando HTTP (objeto)

Servidor envia objeto
requisitado, usando HTTP

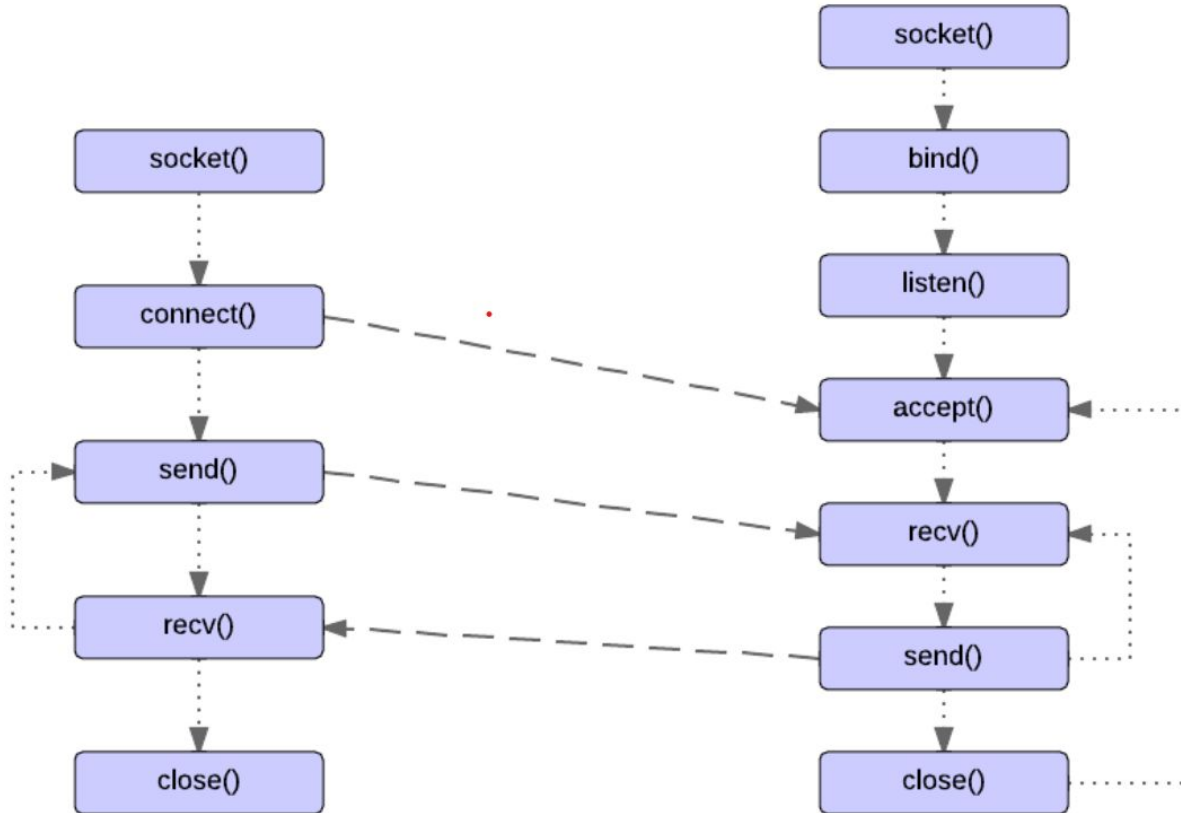
Cliente pode solicitar
outros objetos



ARQUITETURAS DE COMUNICAÇÃO

```
/**  
 * Principais funções para escrever programas com sockets  
 */  
  
getaddrinfo() // Traduz nomes para endereços sockets  
socket()      // Cria um socket e retorna o descritor de arquivo  
bind()        // Associa o socket a um endereço socket e uma porta  
connect()     // Tenta estabelecer uma conexão com um socket  
listen()      // Coloca o socket para aguardar conexões  
accept()      // Aceita uma nova conexão e cria um socket  
send()        // caso conectado, transmite mensagens ao socket  
recv()        // recebe as mensagens através do socket  
close()       // desaloca o descritor de arquivo  
shutdown()    // desabilita a comunicação do socket
```

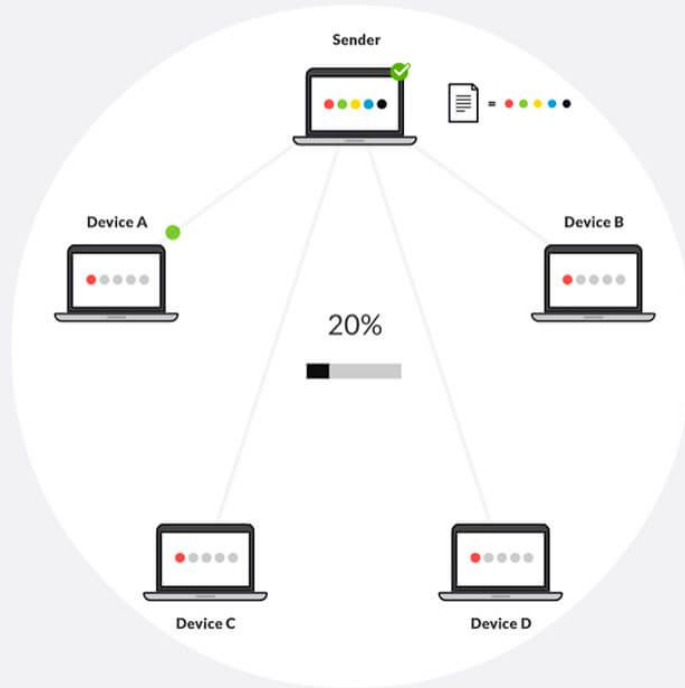
ARQUITETURAS DE COMUNICAÇÃO



ARQUITETURAS DE COMUNICAÇÃO

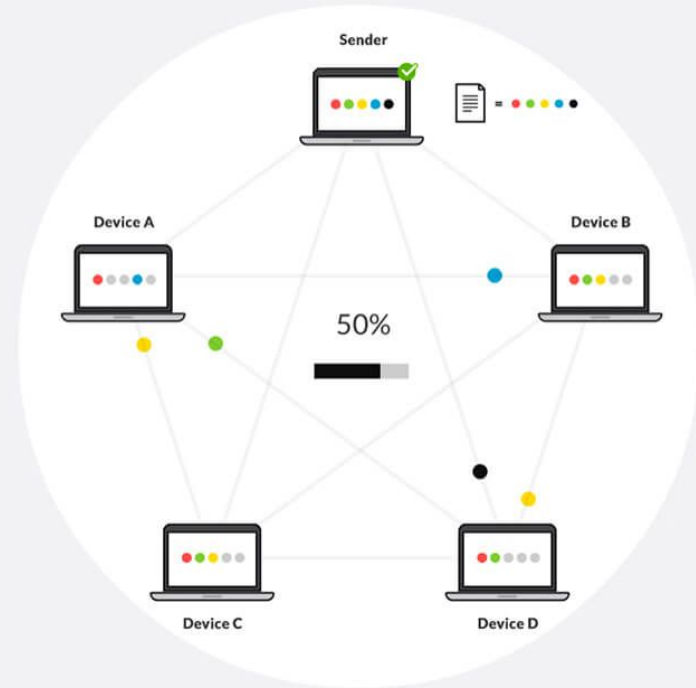
- **Peer-to-Peer (ponto a ponto):**
 - **Igualdade entre pares (Peers):** Todos os nós ou "pares" em um sistema P2P têm capacidades e responsabilidades semelhantes. Não há distinção hierárquica entre eles.
 - **Auto-organização:** Os pares em um sistema P2P são capazes de se auto-organizar e coordenar suas atividades sem a necessidade de um controle centralizado. Isso permite a escalabilidade e a resistência a falhas.
 - **Recursos compartilhados:** Os recursos, como armazenamento, largura de banda e poder computacional, são compartilhados entre os pares na rede. Cada nó contribui com seus próprios recursos e também pode acessar os recursos dos outros pares.
 - **Descentralização:** A arquitetura do sistema é descentralizada, o que significa que não há servidores centrais ou autoridades de controle. As decisões são tomadas pelos próprios pares na rede.
 - **Autonomia e Independência:** Cada par em um sistema P2P é autônomo e independente, o que significa que pode se juntar ou deixar a rede a qualquer momento sem afetar a operação global do sistema.
 - **Escalabilidade:** Os sistemas P2P são conhecidos por sua capacidade de escalar eficientemente à medida que mais pares são adicionados à rede. Isso é possível devido à natureza distribuída e auto-organizadora do sistema.
 - **Redundância e tolerância a falhas:** Devido à sua arquitetura distribuída, os sistemas P2P geralmente são resilientes a falhas. Se um nó falhar, os outros nós ainda podem continuar operando e fornecendo serviços.
 - **Segurança:** Como os dados podem ser armazenados e transmitidos entre os pares na rede, a segurança é uma consideração importante em sistemas P2P. Estratégias de criptografia e autenticação são frequentemente implementadas para proteger a integridade e a privacidade dos dados.

Client-Server Architecture



Vs.

Peer-to-Peer Architecture



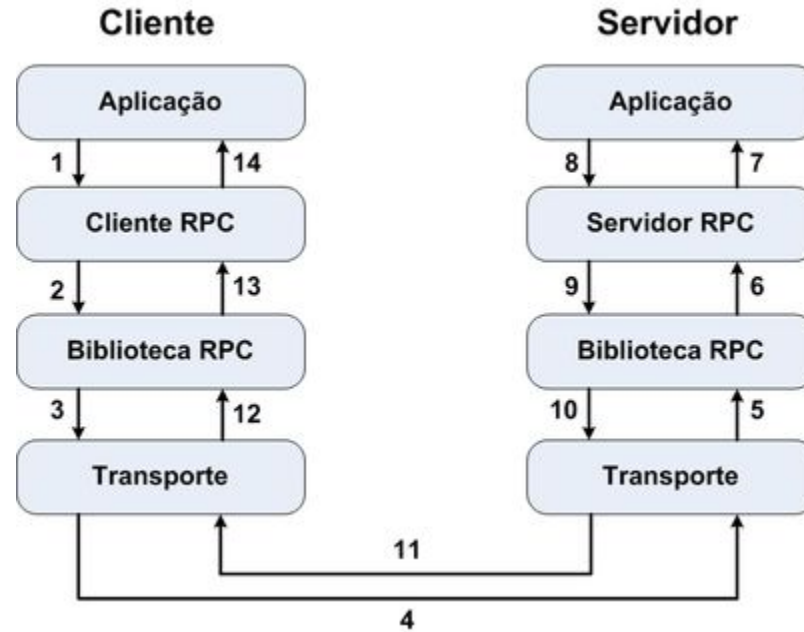
RPC - Remote Procedure Call

- RPC: distribuição da computação através de chamadas de função
- Nível de abstração mais elevado
- Esconde a complexidade
 - nada de sockets, ou protocolos para o programador
- Mais transparente
 - independente da máquina, SO, etc
- Modelo compreendido por programadores (chamada de função)
 - função é um serviço

RPC - Desafios

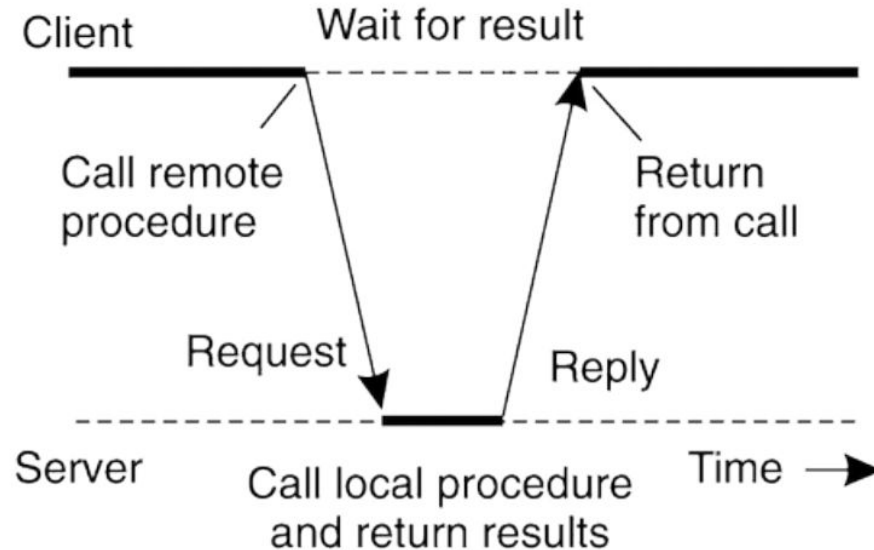
- Processo que chama função e que executa função estão em máquinas diferentes
 - espaço de endereçamento diferentes
- SO diferente, HW diferente (possivelmente)
- Representação dos dados possivelmente diferentes
- Redes e máquinas podem falhar durante a chamada de função

RPC



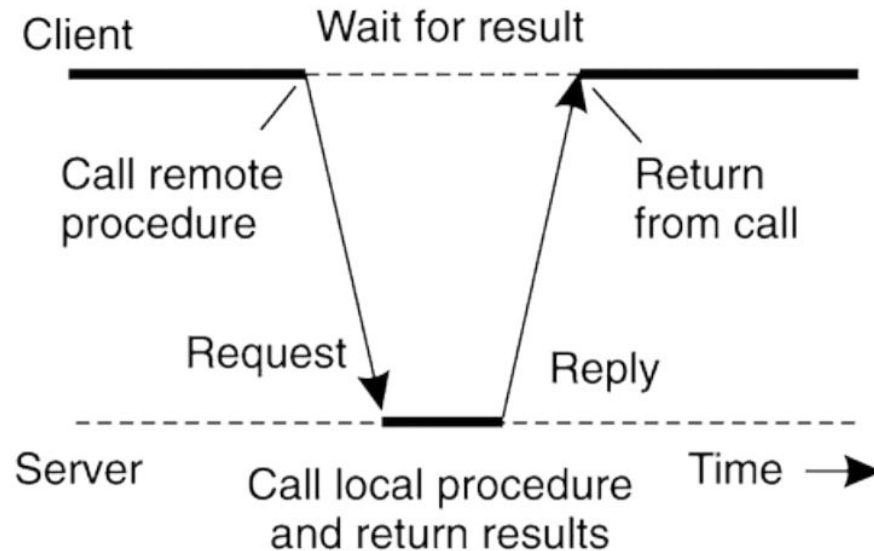
RPC - Modelo síncrono

- Síncrono: função retorna quando retorna!



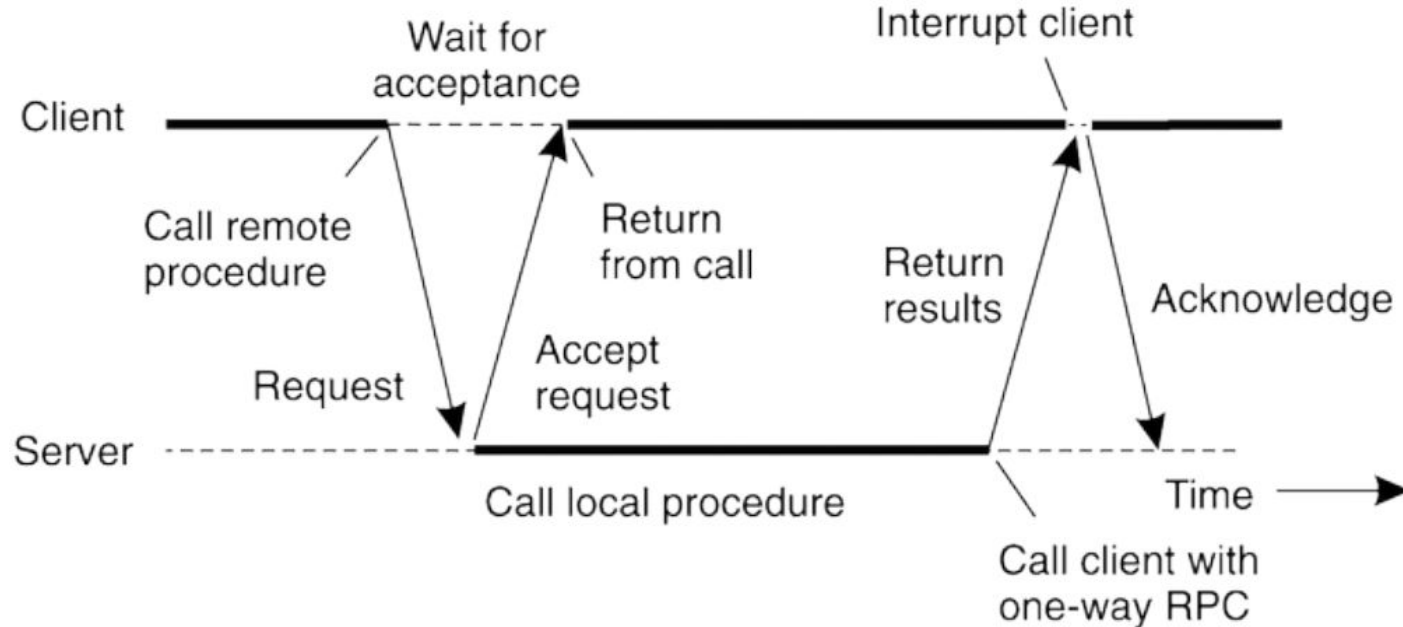
RPC - Modelo síncrono

- Síncrono: função retorna quando retorna!
- Função bloqueante, mais natural para programador retorna quando resultado está pronto



RPC - Modelo assíncrono

- Assíncrono: função “retorna” duas vezes (quando aceito, quando pronto)
- A aplicação pode continuar sua execução



RPC - Modelo multicast

- A mensagem é enviada para vários servidores remotos de RPC
- O cliente precisa definir como a resposta será processada

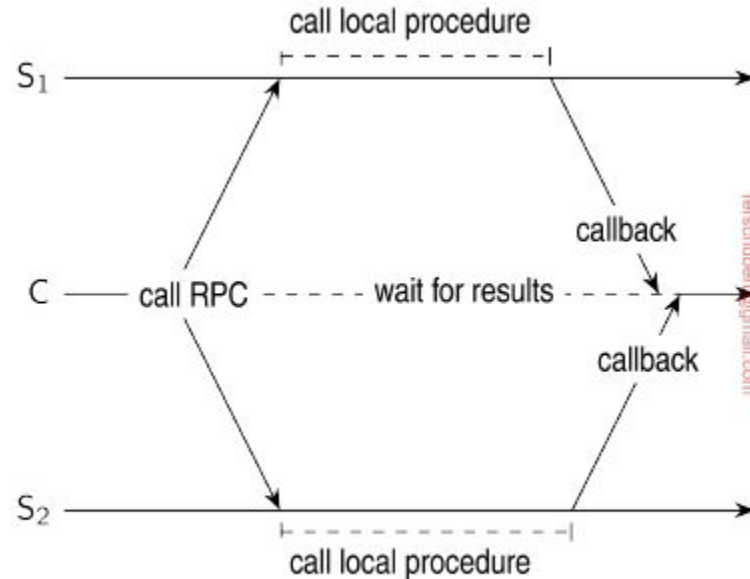


Figure 4.16: The principle of a multicast RPC.

RPC - Marshalling e Stubs

- Ideia: Converter dados para representação que não depende da máquina
 - de comum acordo entre os dois lados;
 - cada lado faz sua parte encapsulado “fora” do programa
- Usar também para strings, floats, tipos mais complicados (structs), etc.
- Stubs: código responsável por marshalling e unmarshalling dos parâmetros
 - oferecido pela biblioteca, de uso obrigatório, mas transparente (programador não sabe)
 - descrevem os parâmetros da função através de algum padrão, como XML

RPC - Marshalling e Stubs

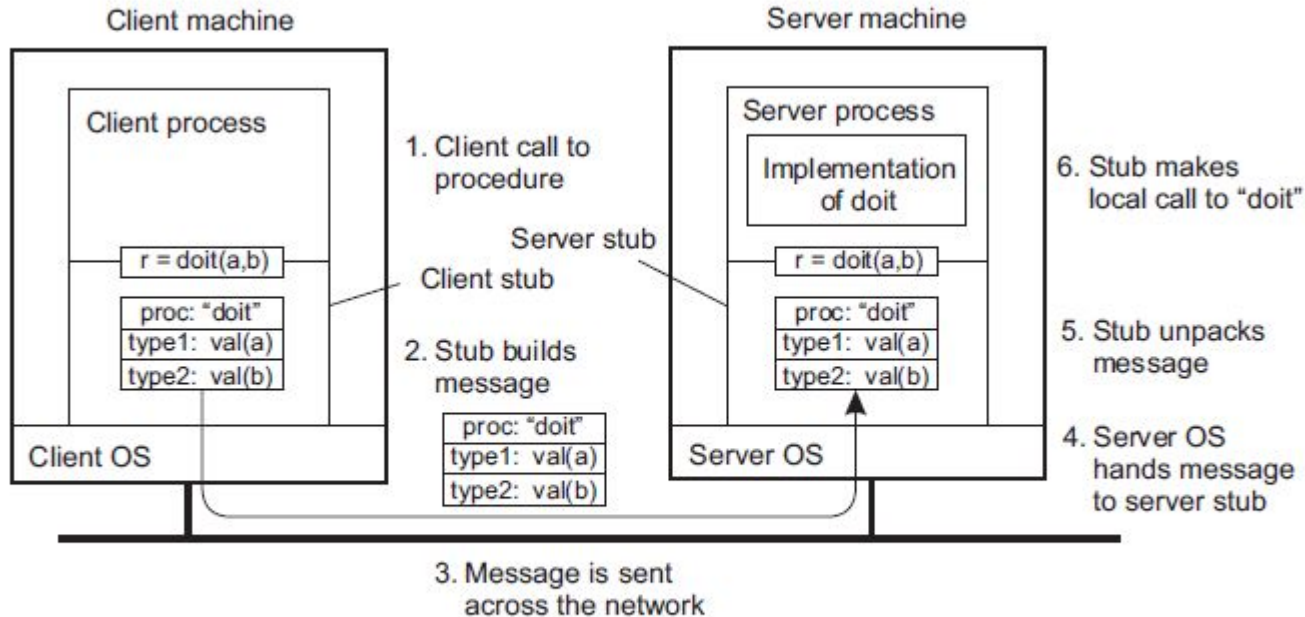


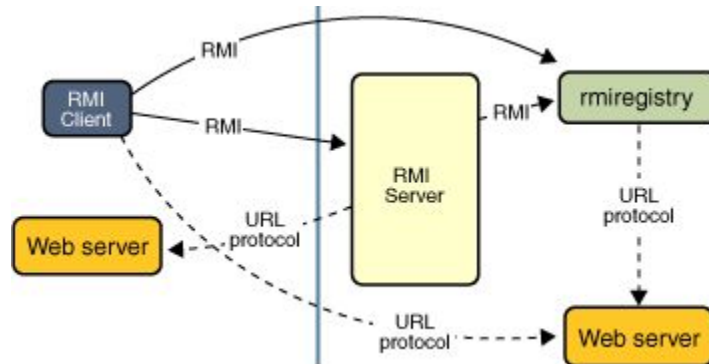
Figure 4.7: The steps involved in calling a remote procedure `doit(a,b)`. The return path for the result is not shown.

RPC - Parâmetros por Referência

- Como passar os parâmetros por referência (ponteiros)?
- Ideia: Pass by copy/restore criar estrutura como parâmetro, copiar os dados, passar como valor, copiar na volta
 - transparente para programador

RMI - Remote Method Invocation

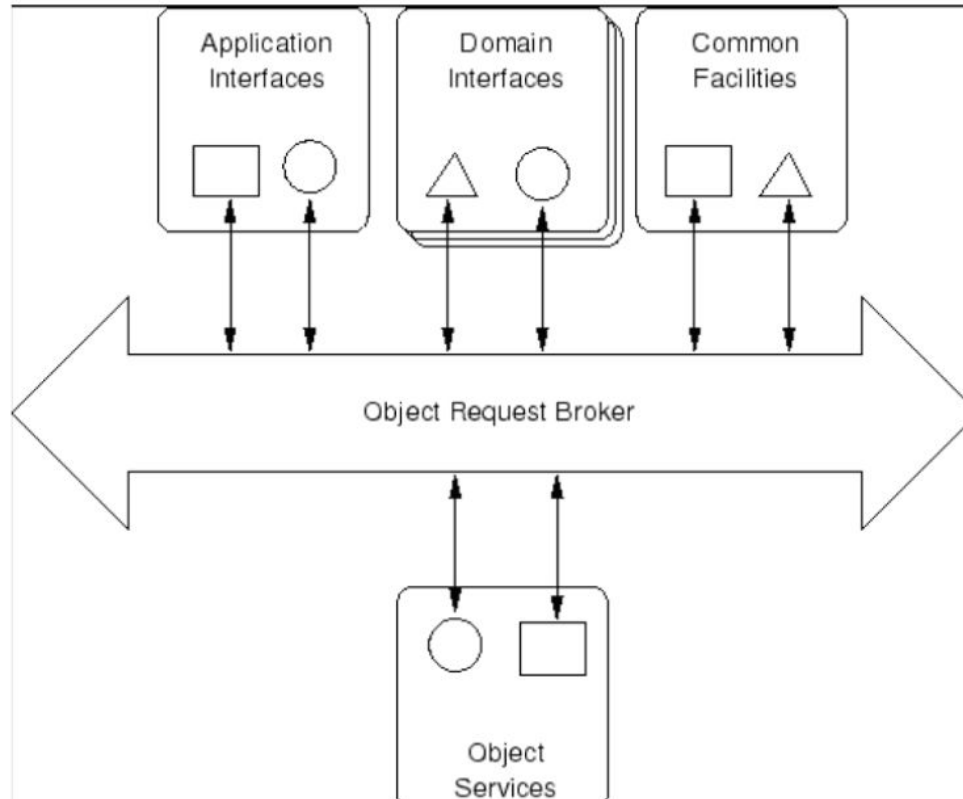
- RMI: RPC aplicado a orientação a objetos sai função entra método
- Objetos instanciados do lado do servidor (remote) e do lado do cliente (local) objetos podem persistir, mantém seu estado
- Facilita a distribuição do sistema programação ainda mais transparente
- Precisa resolver mesmos problemas que RPC, e alguns outros como persistência e sincronização



CORBA

- CORBA (Common Object Request Broker Architecture) é uma arquitetura de software para facilitar a comunicação entre objetos distribuídos em redes de computadores.
- Utiliza um middleware chamado Object Request Broker (ORB) para gerenciar a comunicação entre os objetos, permitindo interoperabilidade entre diferentes linguagens e plataformas.
- Define uma linguagem de definição de interface (IDL) para descrever a interface dos objetos distribuídos e protocolos de comunicação para garantir interoperabilidade.
- Foi amplamente utilizada na década de 1990 e início dos anos 2000 para integrar sistemas heterogêneos.
- Com o surgimento de novas tecnologias, como serviços da web e APIs RESTful, a popularidade de CORBA diminuiu.

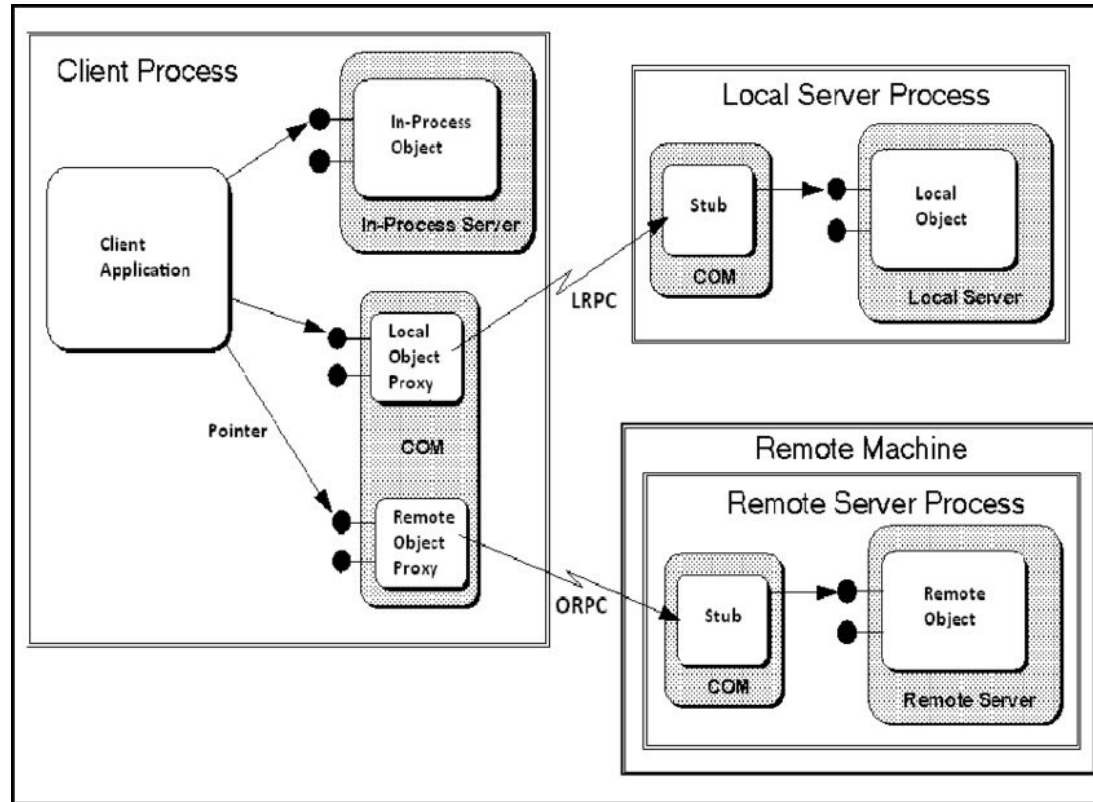
CORBA



DCOM

- DCOM (Distributed Component Object Model) é uma tecnologia da Microsoft para comunicação entre componentes de software distribuídos em uma rede de computadores.
- Permite que objetos escritos em diferentes linguagens de programação e executando em diferentes computadores possam interagir entre si de forma transparente.
- Baseia-se no conceito de Component Object Model (COM), estendendo-o para suportar comunicação entre componentes distribuídos.
- Oferece um modelo de programação baseado em interfaces, onde os objetos expõem suas funcionalidades por meio de interfaces que podem ser acessadas remotamente.
- Foi amplamente utilizado em sistemas distribuídos baseados em tecnologias Microsoft, mas também enfrentou desafios de interoperabilidade com sistemas não-Windows. Com o tempo, tecnologias alternativas como serviços da web ganharam mais popularidade.

DCOM



RPC - Praticando

- Crie um servidor RPC que tenha um método chamado `is_even` que calcula se um número recebido é par ou não. Para inicializar o servidor deve escutar na porta 8000 e o bind deve ser informado como parâmetro para inicializar o servidor, podendo ser localhost ou o IP da máquina

RPC - Praticando

- Crie um cliente RPC chame o método remoto e envie uma série de números distintos
- Conecte o seu cliente em servidores distintos.

RPC - Praticando

- Crie um servidor RPC que tenha um método chamado `is_even` que calcula se um número recebido é par ou não. Para inicializar o servidor deve escutar na porta 8000 e o bind deve ser informado como parâmetro para inicializar o servidor, podendo ser localhost ou o IP da máquina

```
from xmlrpc.server import SimpleXMLRPCServer

def is_even(n):
    print("Requisição recebida com o seguinte argumento: " + str(n))
    return n % 2 == 0

server = SimpleXMLRPCServer(("localhost", 8000))
print("Listening on port 8000...")
server.register_function(is_even, "is_even")
server.serve_forever()
```

RPC - Praticando

- Crie um cliente RPC chame o método remoto e envie uma série de números distintos
- Conecte o seu cliente em servidores distintos.

```
import xmlrpc.client

with xmlrpc.client.ServerProxy("http://localhost:8000/") as proxy:
    print("3 é par: %s" % str(proxy.is_even(3)))
    print("100 é par: %s" % str(proxy.is_even(100)))
```