

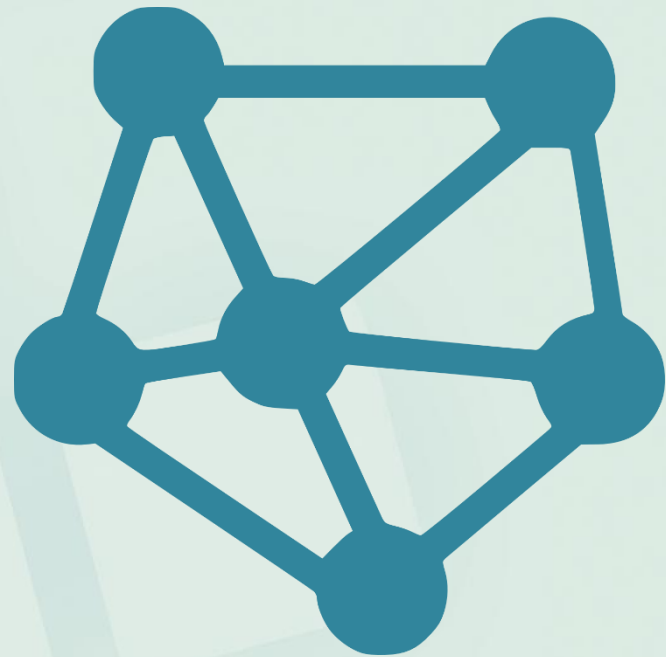


INSTITUTO FEDERAL

Sertão Pernambucano

Sistemas Distribuídos

Comunicação em Sistemas Distribuídos



Prof. Heraldo Gonçalves Lima Junior



1. Introdução

1. Introdução

- A comunicação entre processos distribuídos é feita pela **troca de mensagens**, essa é a forma mais simples de comunicação entre processos de um sistema distribuído.
- A comunicação entre processos na Internet fornece tanto comunicação por **datagrama** como por **fluxo (stream)**.

1. Introdução

- Os protocolos garantem que processos consigam estabelecer a comunicação mesmo em plataformas diferentes ou tecnologias diferentes.



2. Protocolos

2. Protocolos de camadas

- Como não existe memória compartilhada, toda a comunicação em SDs acontece através de troca de mensagens.
- Qual o significado dos bits enviados? Qual a voltagem usada para sinalizar 0 e 1? Como se detecta o bit final da mensagem, ou que uma mensagem foi danificada ou perdida?

2.1. Modelo OSI de camadas

- A International Standards Organization (ISO) desenvolveu um modelo de referência para interconexão de sistemas abertos (OSI).
- Um sistema aberto pode se comunicar com qualquer outro sistema aberto utilizando os protocolos do modelo OSI em 1983.

2.1. Modelo OSI de camadas

- A International Standards Organization (ISO) desenvolveu um modelo de referência para interconexão de sistemas abertos (OSI).
- Um sistema aberto pode se comunicar com qualquer outro sistema aberto utilizando os protocolos do modelo OSI em 1983.

2.1. Modelo OSI de camadas

- Modelo abstrato de redes
- Para que um grupo de computadores se comuniquem em uma rede, todos devem usar os mesmos protocolos de comunicação.



2.1. Modelo OSI de camadas

- Processo cria mensagem que ao passar pelas várias camadas de protocolos é partida e tem cabeçalhos adicionados a ela.



2.1. Modelo OSI de camadas

- Divisão em camadas torna sua implementação mais flexível, facilitando atualizações e correções.
- Redes não necessitam implementar todas as camadas

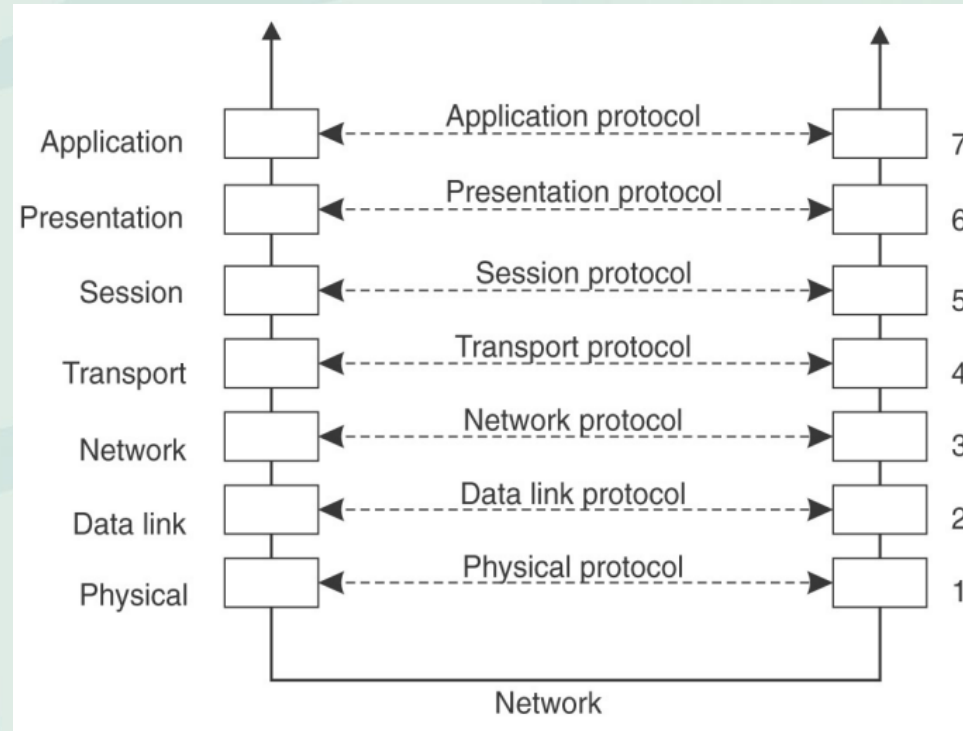


2.1. Modelo OSI de camadas

- Cada camada efetua função bem definida, e elas são definidas para minimizar comunicação entre elas.
- Não detalha serviços.



2.1. Modelo OSI de camadas



2.1.2. Modelo OSI: Estrutura

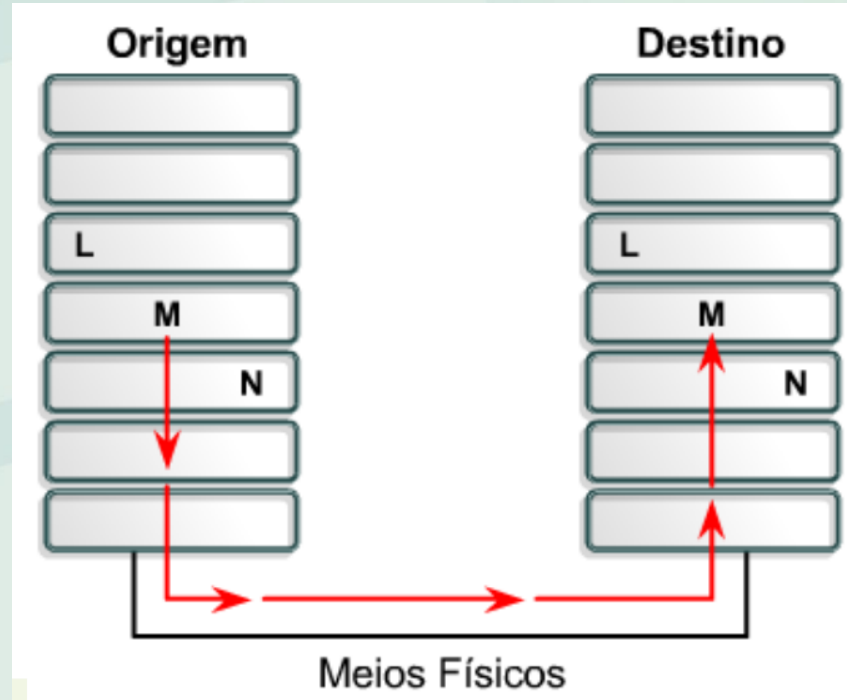
- Camada presta serviços para camada superior.
- Camada usa serviços da camada inferior.
- Camadas de mesmo nível “comunicam-se”
- Uma camada apenas toma conhecimento da camada inferior
- Interação entre camadas feita por serviços
- Divisão de tarefas

2.1.2. Modelo OSI: Estrutura

- Camada presta serviços para camada superior
- Camada usa serviços da camada inferior
- Camadas de mesma função
- Uma camada não requer conhecimento da camada inferior
- Interação entre camadas feita por serviços
- Divisão de tarefas

Facilita abstração

2.1.2. Modelo OSI: Estrutura

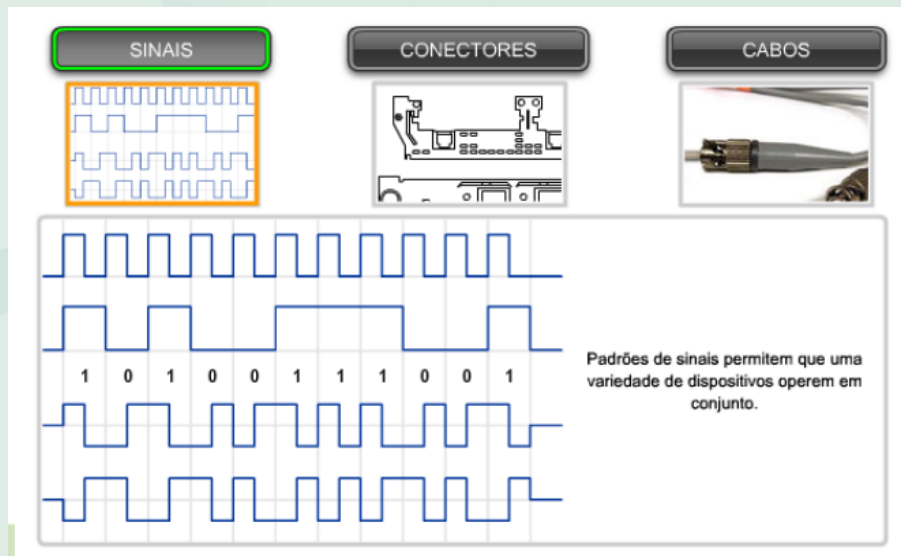


2.1.3. Modelo OSI: Camada Física

- Transmissão de sequências de bits sobre meio físico.
- Especifica:
 - voltagens e correntes
 - tempos
 - conectores e pinagens
 - meio físico utilizado
 - aspectos eletrônicos e mecânicos

2.1.3. Modelo OSI: Camada Física

- Domínio da engenharia eletrônica.
- Não trata de correção de erros na transmissão.



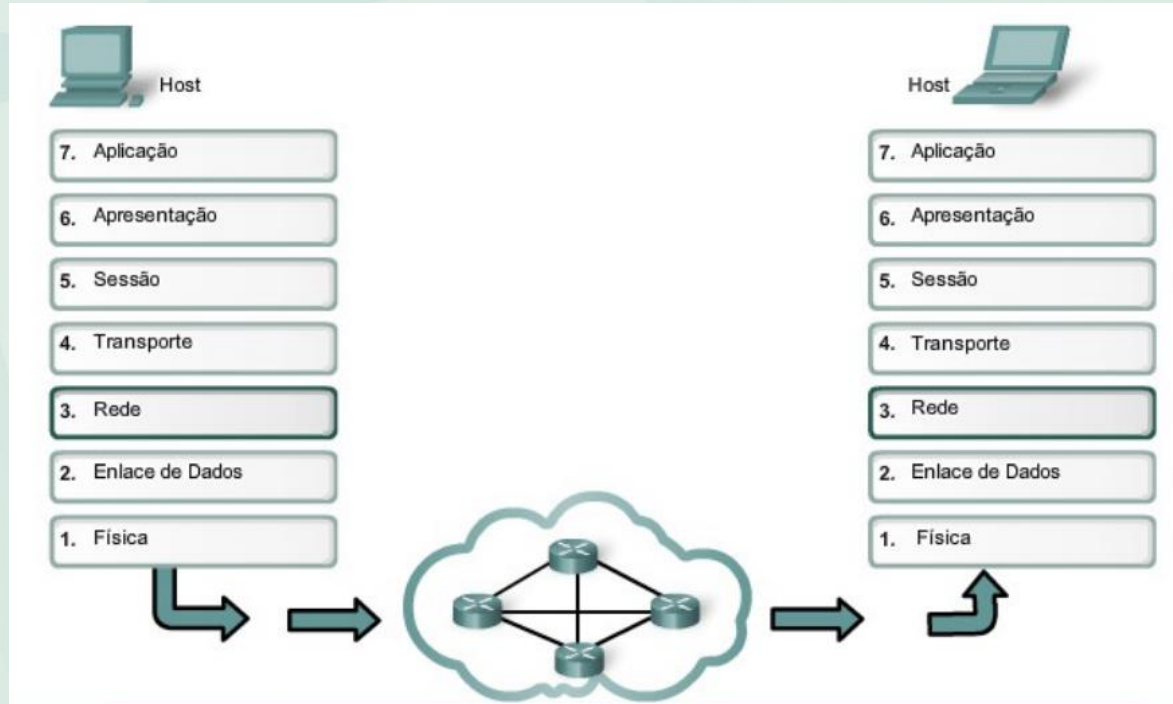
2.1.4. Modelo OSI: Camada de Enlace

- Organiza sequências de bits em conjuntos de bits chamados frames.
- Reconhece início e fim de frames.
- Detecta perdas de frames e requisita retransmissão.

2.1.5. Modelo OSI: Camada de Rede

- Encaminha informação da origem para o destino (roteamento).
- Controla fluxo de transmissão entre sub-redes (controle de congestão).
- Estabelece esquema único de endereçamento independente da sub-rede utilizada.

2.1.5. Modelo OSI: Camada de Rede



2.1.6. Modelo OSI: Camada de Transporte

- Divide e reagrupa a informação binária em **pacotes**.
- **Garante a sequência** dos pacotes.
- Assegura a **conexão confiável** entre origem e destino da comunicação.
- Primeira camada que estabelece comunicação **origem-destino**.

2.1.7. Modelo OSI: Camada de Sessão

- Gerencia **sessões de comunicação**.
- Sessão é uma comunicação que necessita **armazenar estados**.
- Estados são armazenados para permitir **reestabelecimento da comunicação em caso de queda** da comunicação
 - Ex: Retomar transferências de arquivos

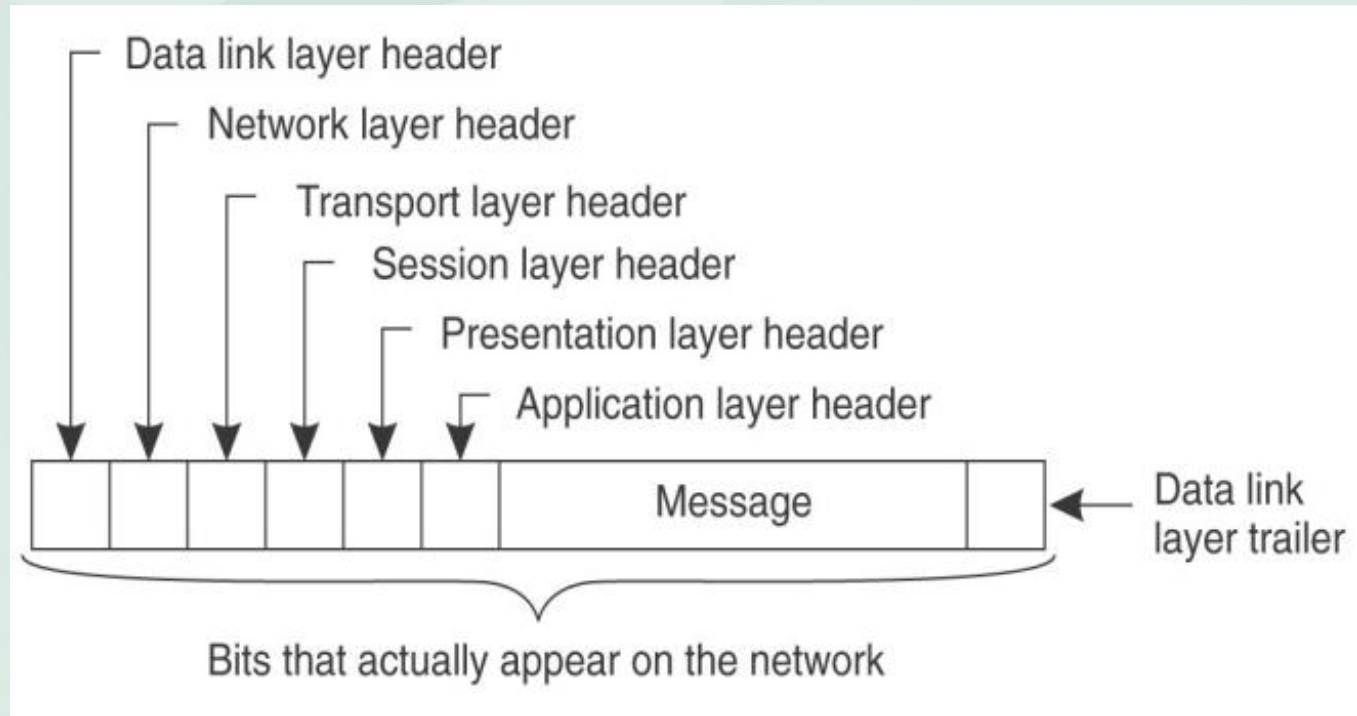
2.1.8. Modelo OSI: Camada de Apresentação

- Trata da representação dos dados em alto nível.
- Adoção de sistema padronizado de representação de caracteres.
- Adoção de códigos de representação numérica padrão.
- Compressão e codificação de dados.

2.1.9. Modelo OSI: Camada de Aplicação

- É onde residem aplicações de rede;
- Permitir ao usuário final o acesso aos recursos da rede;
- Provê interfaces e suporta serviços, tais como:
 - Acesso à Web (HTTP);
 - Acesso e transferência de arquivos (FTP);
 - Serviço de nomes (DNS);
 - Serviço de correio eletrônico (SMTP).

2.1.10. Mensagem na rede





3. Troca de Mensagens

3.1. Conceito

- As mensagens são objetos de dados cuja estrutura e aplicação são definidas pelas próprias aplicações que a usarão. Sendo a troca de mensagens feita através de primitivas explícitas de comunicação:
 - **send(destino, mensagem):** envio da mensagem para o destino
 - **receive(origem, mensagem):** recebimento da mensagem enviada pela origem

3.2. Forma de Comunicação

- **Direta:**
 - **send:** há indicação do processo receptor.
 - `send(process, msg)`
 - **receive:** há indicação do emissor.
 - `receive(process, msg)`

3.2. Forma de Comunicação

- **Indireta:**
 - **send:** envio para uma porta ou mailbox sem o conhecimento de qual será o receptor.
 - `send(mailbox, msg)`
 - **receive:** obtenção da mensagem guardada no mailbox, desconhecendo a identidade do processo emissor.
 - `receive(mailbox, msg)`

3.3. Forma de Sincronização

- **Síncrono ou Bloqueante:**
 - **Send:** espera até que a mensagem seja recebida pelo receptor.
 - **Receive:** aguarda até a mensagem estar disponível.

3.3. Forma de Sincronização

- **Assíncrona ou Não Bloqueante:**
 - **Send:** envia a mensagem mas não espera até que a mensagem seja recebida pelo receptor.
 - **Receive:** se a mensagem estiver disponível, recebe a mensagem, senão, continua o processamento retornando uma indicação de que a mensagem não estava disponível.

3.3. Forma de Sincronização

- Um sistema pode possuir diferentes combinações desses tipos de primitivas, sendo **que um sistema de comunicação passa a ser conhecido como síncrono se ambas as primitivas (send, receive), forem do tipo bloqueante.** Por outro lado, **um sistema de comunicação é dito assíncrono se pelo menos uma das primitivas for assíncrona.**

3.3. Forma de Sincronização

- Um sistema pode possuir diferentes combinações desses tipos de primitivas, sendo **que um sistema de comunicação passa a ser conhecido como síncrono se ambas as primitivas (send, receive), forem do tipo bloqueante.** Por outro lado, **um sistema de comunicação é dito assíncrono se pelo menos uma das primitivas for assíncrona.**



4. Modelo Cliente-Servidor

4.1. Cliente-Servidor: Características

- É um paradigma de programação que representa as interações entre os processos e as estruturas do sistema. Neste tipo de paradigma existem dois de processos:
 - **clientes:** processos que requisitam serviços;
 - **servidores:** processos que recebem requisitos , realizam uma operação e retornam serviços

4.1. Cliente-Servidor: Características

- Em um modelo cliente/servidor, o processo cliente **necessita de um serviço** (ex. Ler dados de um arquivo), então ele envia uma mensagem para o servidor e espera pela mensagem de resposta.



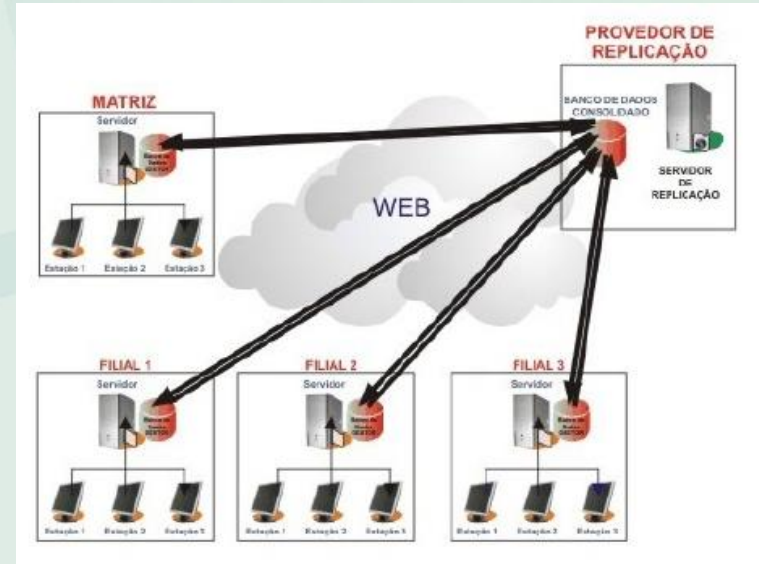
4.1. Cliente-Servidor: Características

- O processo servidor, após realizar a tarefa requisitada, envia o resultado na forma de uma mensagem de **resposta** ao processo cliente.



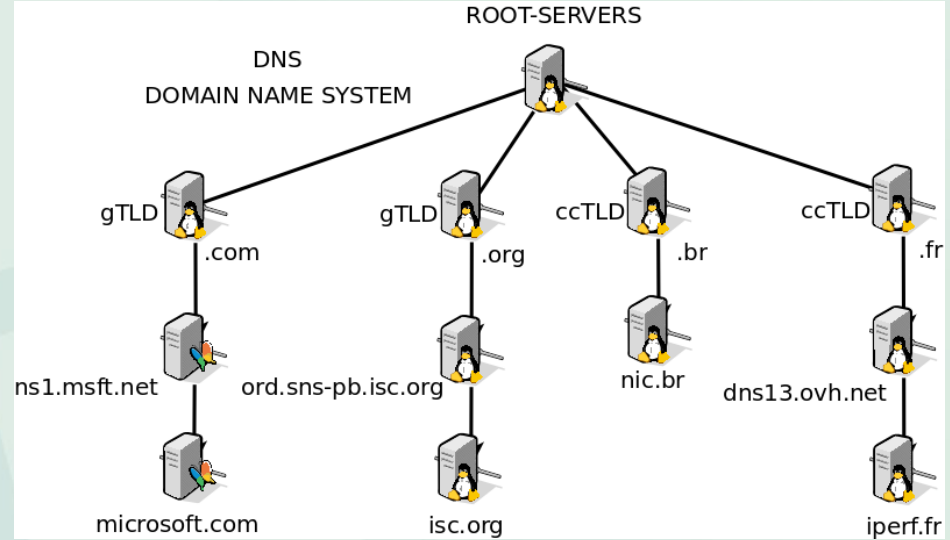
4.1. Cliente-Servidor: Características

- No modelo cliente servidor é possível a existência de vários servidores, onde os mesmos podem ser replicados, isto é, quando há **várias instâncias do mesmo servidor**.



4.1. Cliente-Servidor: Características

- Também é possível termos servidores hierárquicos, isto é, servidores que usam o(s) serviço(s) de outros servidores.



4.2. Cliente-Servidor: Endereçamento

- Como enviar a mensagem a máquina servidor e depois ao processo servidor?
- Para que um cliente envie mensagens a um servidor, primeiro o cliente, deve necessariamente conhecer o endereço do servidor, desse modo, é preciso estabelecer um esquema de identificação.

4.2. Cliente-Servidor: Endereçamento

- **Um identificador único de processo quando na mesma máquina:** com um único processo por máquina basta indicar o endereço da máquina pois o kernel consegue determinar qual é o processo servidor único.

4.2. Cliente-Servidor: Endereçamento

- **Endereçamento indicando o processo e a máquina:**
quando se permite mais de um processo servidor por máquina, deve-se endereçar a mensagem a esse servidor específico. Um esquema comum é o uso de um nome composto por duas partes.

4.2. Cliente-Servidor: Endereçamento

- **Processos escolhem endereços que são detectados por broadcast:** Cada processo deve receber um endereço único que não envolva o número da sua máquina. Este endereço pode ser atribuído de duas formas:
 - através de um escalonador de endereços de processo centralizado;
 - cada processo escolhe um valor aleatoriamente a partir de um espaço de endereçamento grande

4.2. Cliente-Servidor: Endereçamento

- **O kernel emissor de uma requisição pode localizar para qual maquina enviar através do seguinte procedimento:**
 - 1) o emissor envia a mensagem para todas as maquinas (broadcast) com um pacote especial de localização contendo o endereço do processo destino;

4.2. Cliente-Servidor: Endereçamento

- 2) em cada máquina da rede o kernel verifica se o processo está na máquina. Quem localizar envia mensagem indicando seu endereço na rede;
- o kernel emissor guarda essas informações para requisições futuras.
- Essa abordagem tem como vantagem ser transparente mas tem um custo da mensagem

4.2. Cliente-Servidor: Endereçamento

- **uso de servidor de nomes:** neste esquema os servidores são indicados por um identificador de alto nível (nome ASCII) que não inclui nem identificação da máquina nem do processo. Quando um cliente executa uma requisição a um servidor pela primeira vez, uma mensagem especial é enviada para um servidor de mapeamento ou servidor de nomes solicitando o número da máquina onde está o servidor

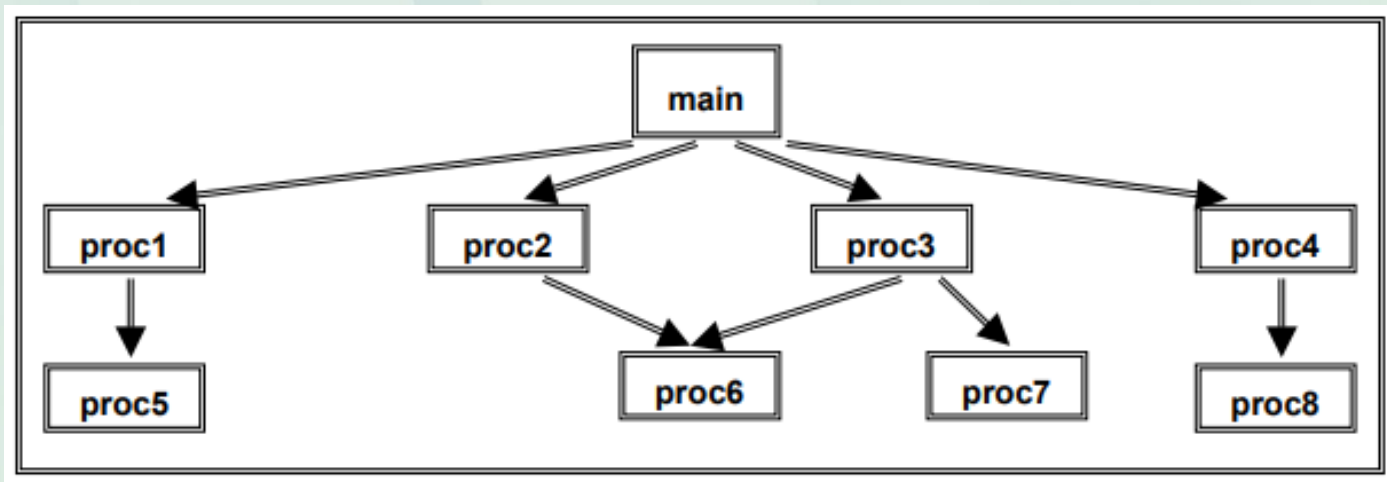
5. Chamada Remota de Procedimento (RPC – Remote Procedure Call)

5.1. RPC: Conceito e Motivação

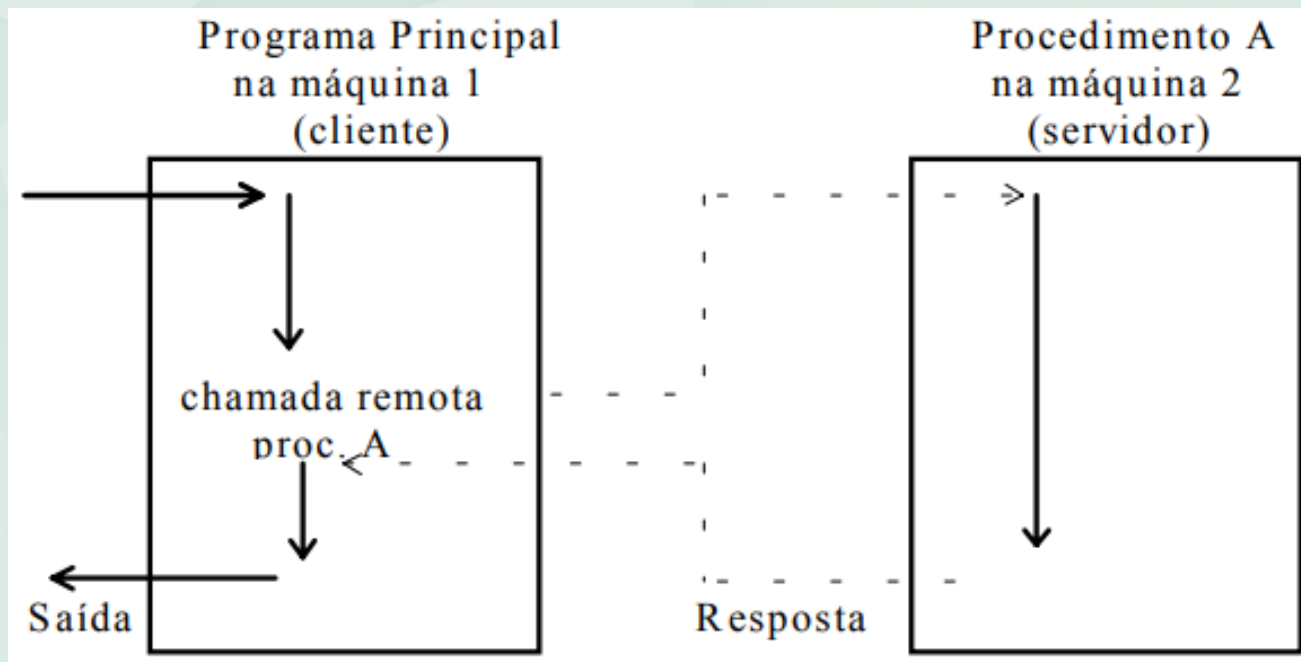
- Chamada remota de procedimento permite que programas invoquem procedimentos ou funções localizadas em outras máquinas como se ele estivesse localmente definidos.
- Surge da necessidade de obter transparência de acesso e sistemas distribuídos o que não ocorre com o uso dos procedimentos send e receive.

5.2. RPC: Conceito de Procedimento

- Procedimentos => permite a divisão do programas em vários pedaços.



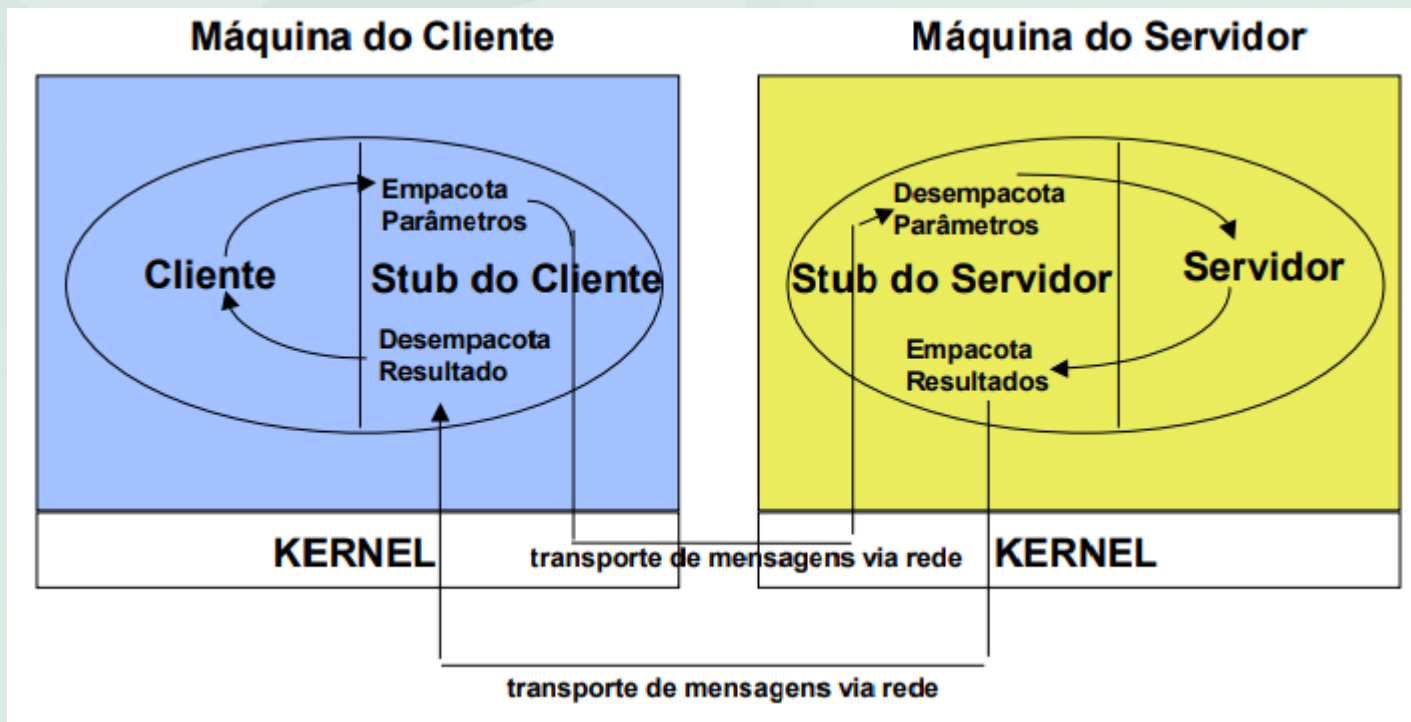
5.3. RPC: Modelo de Execução



5.4. RPC: Objetivo

- Tornar mais fácil a implementação de Aplicações Distribuídas
- Esconde o código de chamadas a rede em procedimentos chamados **stubs**.
- A Ideia do modelo é estender o conceito de chamada de procedimento convencional para ambientes distribuídos.

5.5. RPC: Passos de uma chamada remota

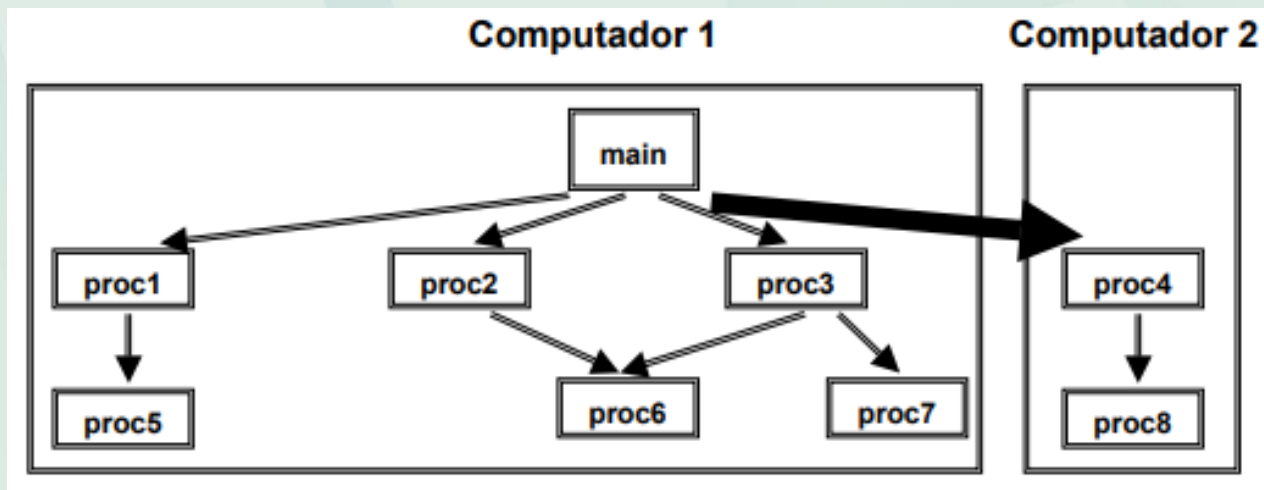


5.5. RPC: Programa Distribuído

- Permite ao programador projetar um programa convencional que solucione o problema, e então dividir o programa em procedimentos que podem ser executadas em vários computadores.

5.2. RPC: Programa Distribuído

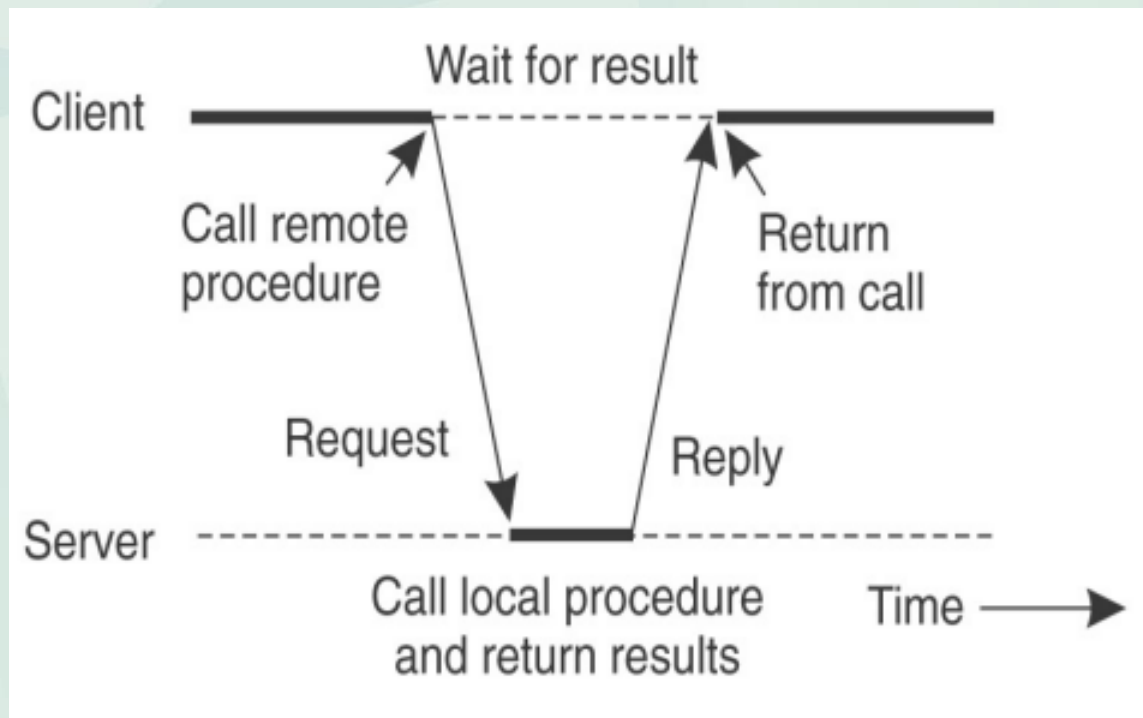
- Nenhum aspecto da troca de mensagens é visível para o programador!



5.5. RPC: Problemas

- Chamador e chamado executam em espaços de endereçamento diferentes;
- Como fazer a passagem de parâmetros e resultados quando as máquinas envolvidas têm arquiteturas distintas;
- Possibilidades de falhas.

5.5. RPC: Operação RPC Básica



5.5. RPC: Operação RPC Básica

- A RPC segue os seguintes passos:
 1. O procedimento chama o stub cliente de forma normal, isto é, como se fosse um procedimento local qualquer;
 2. O stub cliente constrói mensagens e passa ao kernel;

5.5. RPC: Operação RPC Básica

3. O kernel remoto envia a mensagem ao stub servidor;
4. O kernel remoto entrega a mensagem ao stub servidor;
5. O stub servidor desempacota os parâmetros e chama o servidor;
6. O servidor realiza o trabalho solicitante e envia o resultado ao stub servidor;

5.5. RPC: Operação RPC Básica

7. O stub servidor empacota o resultado em uma mensagem e passa para o kernel;
8. O kernel remoto envia mensagem ao kernel cliente;
9. O kernel cliente entrega a mensagem ao stub cliente;
10. O stub desempacota o resultado e retorna ao stub cliente.

5.6. RPC: Passagem de Parâmetros

- É preciso tratar a passagem de parâmetros e a conversão de dados. Isso é feito pela operação de “empacotamento de parâmetros” (parameter **marshalling**). Essa operação trata problemas de representação distinta de dados (números/caracteres) para máquinas heterogêneas.

5.6. RPC: Passagem de Parâmetros

- Tanto o cliente quanto o servidor sabem os tipos de parâmetros passados (identificador do procedimento + parâmetros) e devem conseguir obter a representação correta dos dados.
 - **Opção 1: uso de um padrão de representação de dados, como por exemplo o XDR do RPC UNIX.**

5.6. RPC: Passagem de Parâmetros

- **Opção 2: a mensagem é enviada no formato nativo com indicação de qual o formato utilizado. O receptor deve fazer a conversão quando for o caso.**

5.7. RPC: Passagem de Ponteiros

- Um ponteiro representa um endereço de memória que não tem nenhum significado na máquina remota.
 - **Opção 1:** proibir a passagem de ponteiros, essa obviamente não é uma alternativa desejável pelo programador;

5.7. RPC: Passagem de Ponteiros

- **Opção 2:** Copiar o valor apontado pela variável ponteiros. As alterações são feitas remotamente e no retorno as alterações são atualizadas no chamador (semântica cópia/restauração)

5.8. RPC: Falhas

- Na execução de uma chamada remota de procedimentos, existem cinco classes diferentes de falhas possíveis.

1. O cliente não consegue localizar o servidor;

5.8. RPC: Falhas

- Na execução de uma chamada remota de procedimentos, existem cinco classes diferentes de falhas possíveis.
 1. **O cliente não consegue localizar o servidor;**
 2. **Mensagem de requisição do cliente para o servidor é perdida;**

5.8. RPC: Falhas

- Na execução de uma chamada remota de procedimentos, existem cinco classes diferentes de falhas possíveis.
 1. **O cliente não consegue localizar o servidor;**
 2. **Mensagem de requisição do cliente para o servidor é perdida;**
 3. **Mensagem de resposta perdida;**

5.8. RPC: Falhas

- Na execução de uma chamada remota de procedimentos, existem cinco classes diferentes de falhas possíveis.

4. O servidor cai após receber uma requisição;

5.8. RPC: Falhas

- Na execução de uma chamada remota de procedimentos, existem cinco classes diferentes de falhas possíveis.
 4. **O servidor cai após receber uma requisição;**
 5. **O cliente falha após enviar uma requisição (falha do cliente).**

5.9. RPC: Questões de implementação

- As questões de implementação analisadas nesta seção estão relacionadas as questões de desempenho.
- **Protocolo:** Em princípio qualquer mecanismo de troca de mensagem poderia dar suporte a implementação de RPC. Uma das questões a serem decididas na implementação é se o protocolo será orientado a conexão ou sem conexão.

5.9. RPC: Questões de implementação

- **Confirmação:** As mensagens de confirmação são usadas para detecção de falhas. Na implementação, é preciso decidir se os pacotes serão confirmados individualmente ou não. Assim temos dois tipos de protocolos: **stop-and-wait** (no momento em que um pacote for danificado ou perdido, o cliente não recebendo o ACK providência o reenvio), e o **blast** (o servidor envia mensagem ACK apenas no final).



6. Invocação de Método Remoto(RMI – Remote Method Invocation)

6.1. RMI: O que é?

- É uma extensão natural do RPC para POO.
- Em java, permite que objetos em uma JVM invoque métodos de objetos em outra JVM.
- As aplicações são conectadas remotamente usando objetos chamados **stub** e **skeleton**.
- Os objetos são conectados por meio de uma interface.

6.2. RMI: Elementos

- **Objeto Remoto (OR):** Objeto de uma JVM, que possui métodos expostos, passíveis de serem invocados por objetos de outra JVM.
- **Interface Remota:** Uma interface java que define os métodos que existem em um OR.

6.2. RMI: Elementos

- **RMI:**
 - Invocação de um objeto remoto usando uma interface remota.
 - Possui sintaxe similar a invocações locais.

6.2. RMI: Elementos

- **RMI REGISTRY:** É um registro de objetos remotos que é usado pelo RMI Server no mesmo host para vincular objetos remotos a nomes.
- **Stub:**
 - Recebe requisições do cliente e roteia para o OR.
 - Representa o identificador do OR na JVM do cliente.

6.2. RMI: Elementos

- **Skeleton:**
 - Ponto de entrada das requisições do cliente no lado servidor.
 - Atua como o OR que interage com o cliente.
- **RMI Client:** Procuram pelos objetos remotos e fazem requisições pelo STUB.

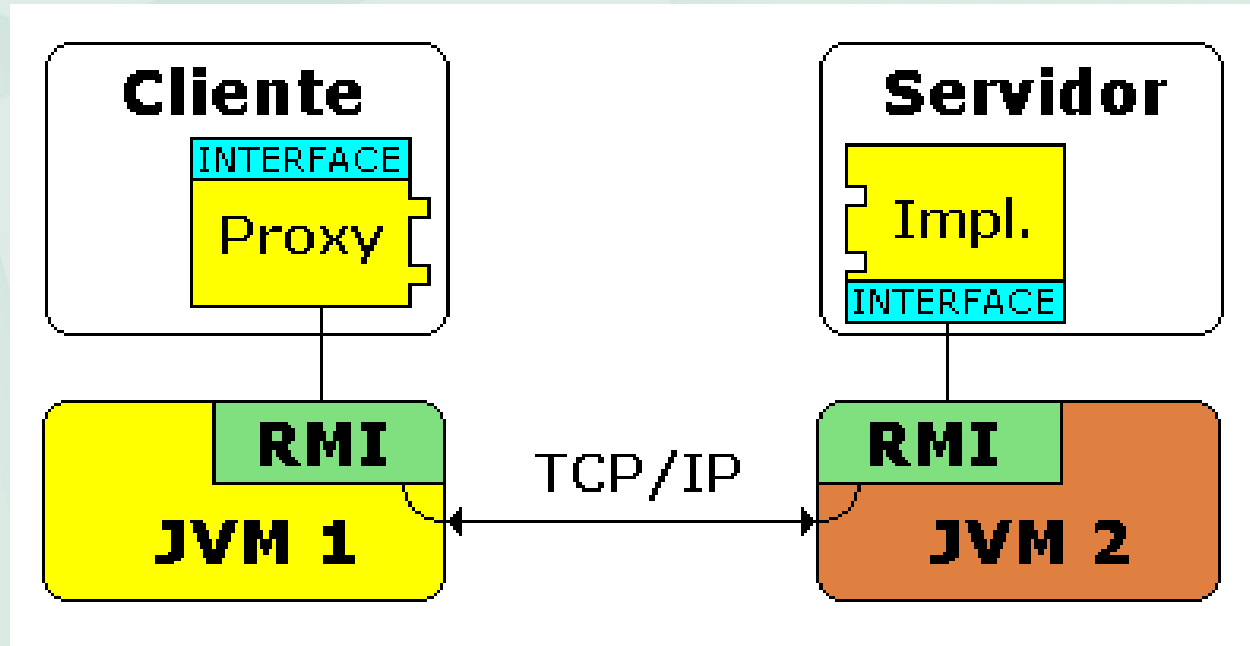
6.2. RMI: Elementos

- **RMI Server:**
 - Contém os métodos remotos;
 - Recebe as requisições pelo skeleton, executa as operações e devolve resultados.

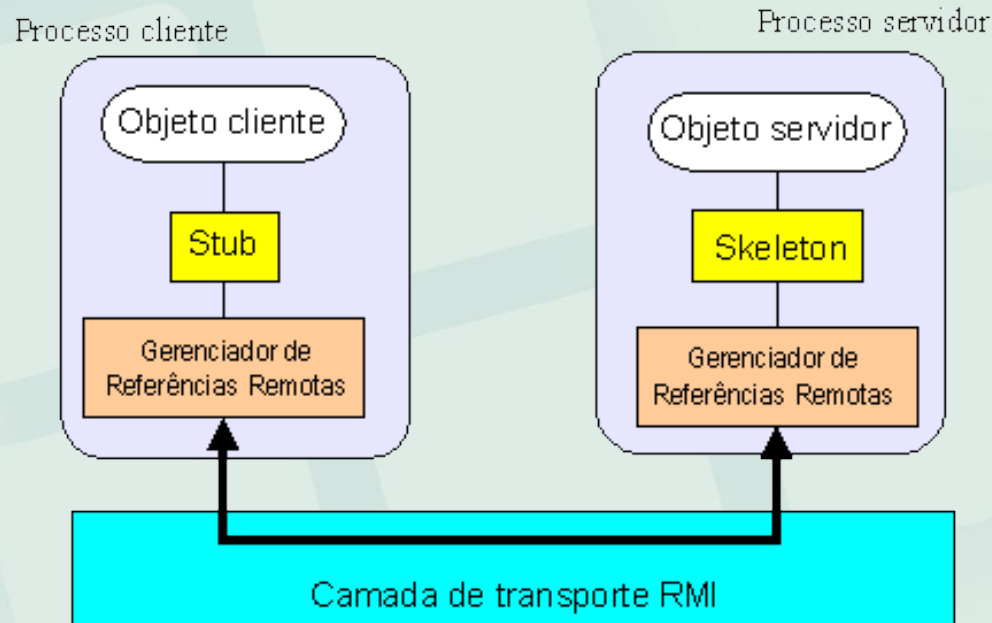
6.3. RMI: Programação com Interfaces

- Objetos remotos são como objetos locais que tem métodos invocados e retornam resultados, mas...
- Objetos remotos só podem ser acessados por meio de interfaces.

6.4. RMI: Modelo de Execução



6.4. RMI: Modelo de Execução



6.5. RMI: Funcionamento do Stub

- Objeto no cliente invoca o método no stub.
- Stub faz a RMI:
 1. Inicia a conexão com a JVM remota;
 2. Serializa (marshalls) os parâmetros passados para ele;
 3. Envia requisição;

6.5. RMI: Funcionamento do Stub

4. Espera pela resposta de um OR e desserializa (unmarshalls) o valor retornado ou a execução;
5. Responde ao objeto invocador com o resultado ou exceção.

6.6. RMI: Funcionamento do Skeleton

1. Espera pela requisição do cliente;
2. Recebe a RMI;
3. Lê (unmarshalls) os parâmetros enviados para o método remoto;
4. Invoca o método remoto correspondente;

6.6. RMI: Funcionamento do Skeleton

5. Serializa(marshalls) e envia o resultado de volta para o stub do cliente;



7. Comunicação em Grupos

7.1. Comunicação em grupos: Introdução

- Coordenação é necessária em sistemas distribuídos mas é difícil de se obter, porque:
 - Eventos ocorrem concorrentemente;
 - Linhas de comunicação não são totalmente confiáveis;
 - Computadores podem falhar ou quebrar;
 - Novas máquinas podem ser adicionadas ao sistema;
 - etc.

7.1. Comunicação em grupos: Introdução

- Comunicação em RPC envolve somente dois processos, gerando problemas quando se quer enviar mensagens para mais de um servidor;
- Em comunicação de grupos, com uma única operação pode-se enviar uma mensagem á vários destinos;
- Permitem que processos em um grupo sejam tratados como uma única abstração.

7.2. Comunicação em grupos: Conceitos

- **O que é um grupo?**
 - Um conjunto de processos que cooperam entre si para prover um serviço
 - Uma entidade abstrata que nomeia um conjunto de processos
- Comunicação de grupos coordena a troca de mensagens entre os membros do grupo e dos processos externos com o grupo;

7.3. Comunicação em grupos: Aplicações

- Servidores altamente disponíveis e confiáveis;
- Replicação de bancos de dados;
- Conferências multimídia;
- Jogos distribuídos;
- Aplicações que em geral necessitem de uma alta taxa de disponibilidade, confiabilidade, tolerância a falhas.

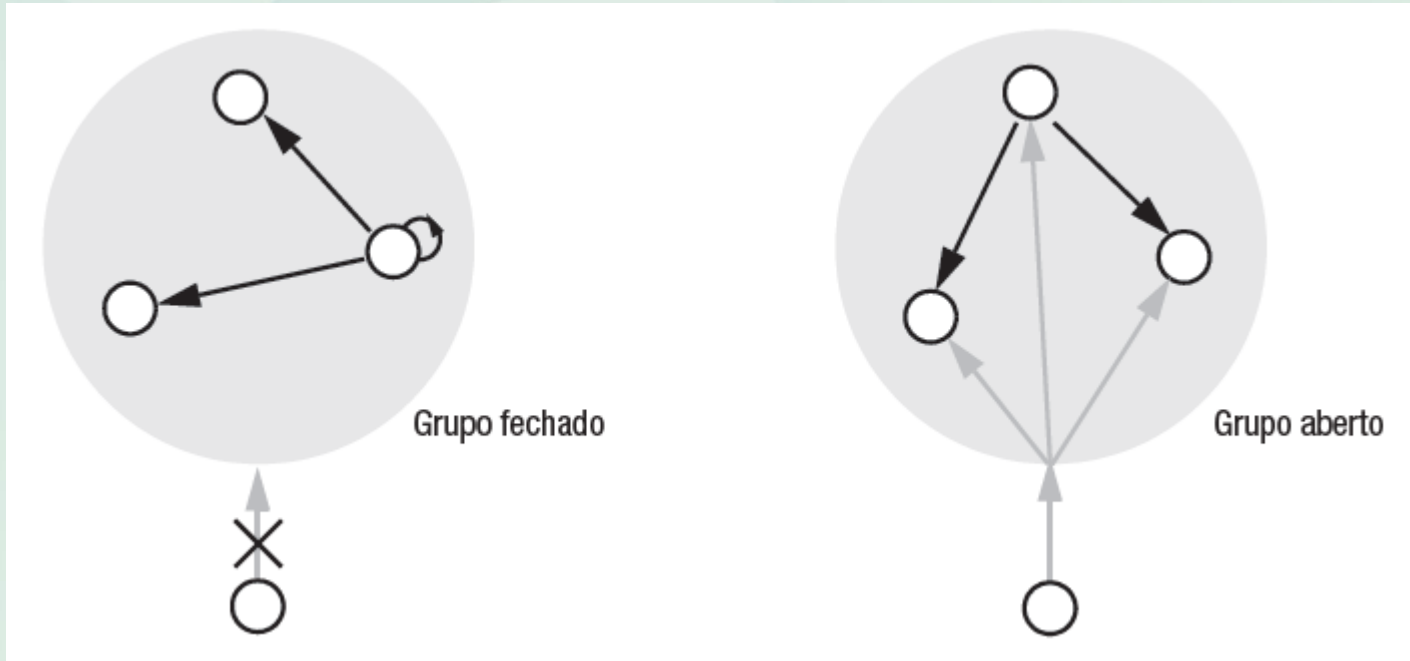
7.4. Comunicação em grupos: Comunicação

- **Multicast:** Pacotes são enviados de uma só vez para todos os processos de um grupo;
- **Broadcast:** Pacotes são enviados para todas as máquinas e somente os processos que fazem parte do grupo não os descartam;
- **Unicast:** Transmissão ponto-a-ponto, processo tem que enviar mensagem para cada membro do grupo

7.5. Comunicação em grupos: Tipos

- **Grupos fechados:** Nos grupos fechados, somente membros do grupo podem mandar mensagens para os outros membros;
- **Grupos abertos:** Qualquer processo pode enviar uma mensagem ao grupo;

7.5. Comunicação em grupos: Tipos



7.5. Comunicação em grupos: Tipos

- **Grupos sobrepostos:** as entidades (processos ou objetos) podem ser membros de vários grupos;
- **Grupos não sobrepostos:** a participação como membro não deve se sobrepor (isto é, qualquer processo pertence, no máximo, a um grupo).

7.6. Comunicação em grupos: Organização

- **Grupos Pares**
 - Todos os processos são tratados como iguais (pares);
 - Decisões são tomadas coletivamente.
- **Grupos hierárquicos**
 - Um processo é o coordenador e os outros são subordinados a ele.

7.7. Comunicação em grupos: Manutenção

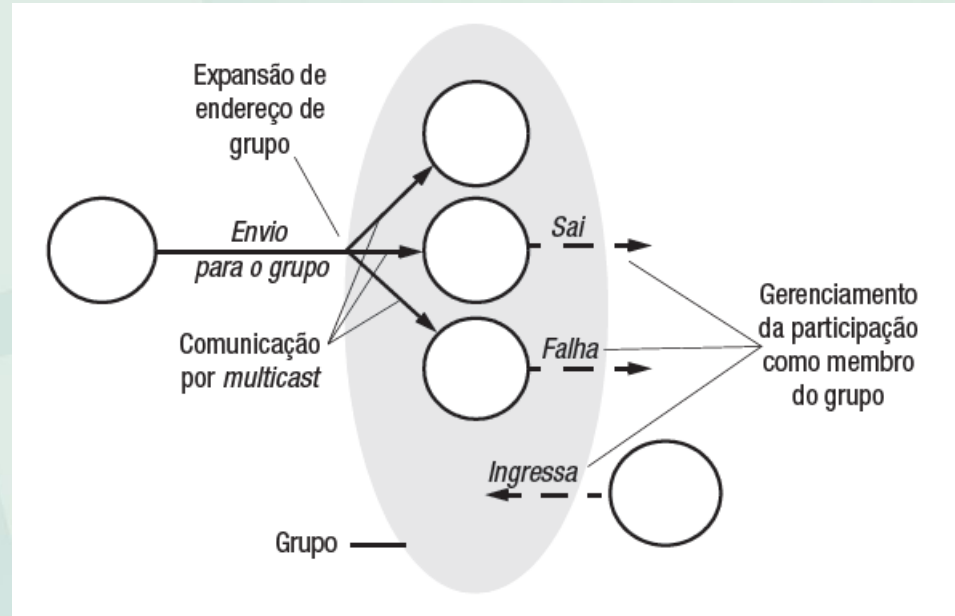
- Manutenção dos grupos pode ser feita:
 - de maneira centralizada, através de um servidor de grupos;
 - ou distribuída, com um processo anunciando a todos os membros de um grupo que ele está se filiando ao mesmo.


7.8. Comunicação em grupos: Ordenação

- **Desordenadas**
- **Ordenação global:** Entrega todas as mensagens exatamente na ordem em que foram criadas;
- **Ordenação consistente:** Sistema decide qual mensagem precede outra quando são enviadas quase ao mesmo tempo e entrega as mensagens nesta ordem

7.9. Comunicação em grupos: Problemas

- Confiabilidade e ordenação em multicast;
- Gerenciamento da participação no grupo.





8. Filas de Mensagens

8.1. Sistema de Filas de Mensagens: Introdução

- Middleware orientado a Mensagem (MOM);
- Proporcionam suporte extensivo para comunicação assíncrona persistente;
- Oferecem capacidade de armazenamento de médio prazo para as mensagens;

8.1. Sistema de Filas de Mensagens: Introdução

- Não exigem que o remetente ou o receptor estejam ativos durante a transmissão da mensagem
- **Exemplo:** Consulta que abranja vários bancos de dados pode ser repartida em subconsultas que são repassadas para bancos de dados individuais.

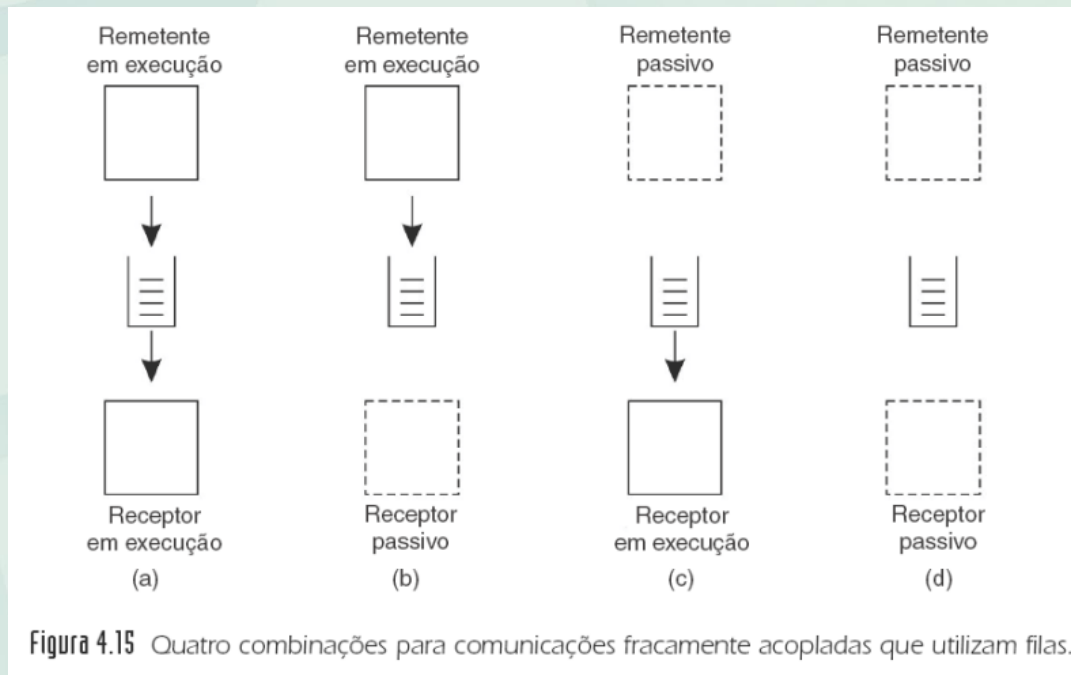
8.1. Sistema de Filas de Mensagens: Introdução

- Aplicações se comunicam inserindo mensagens em filas específicas;
- Mensagens são repassadas por uma série de servidores de comunicação, antes de serem entregues ao destinatário.
 - Na prática, a maioria dos servidores estão diretamente conectados uns aos outros.

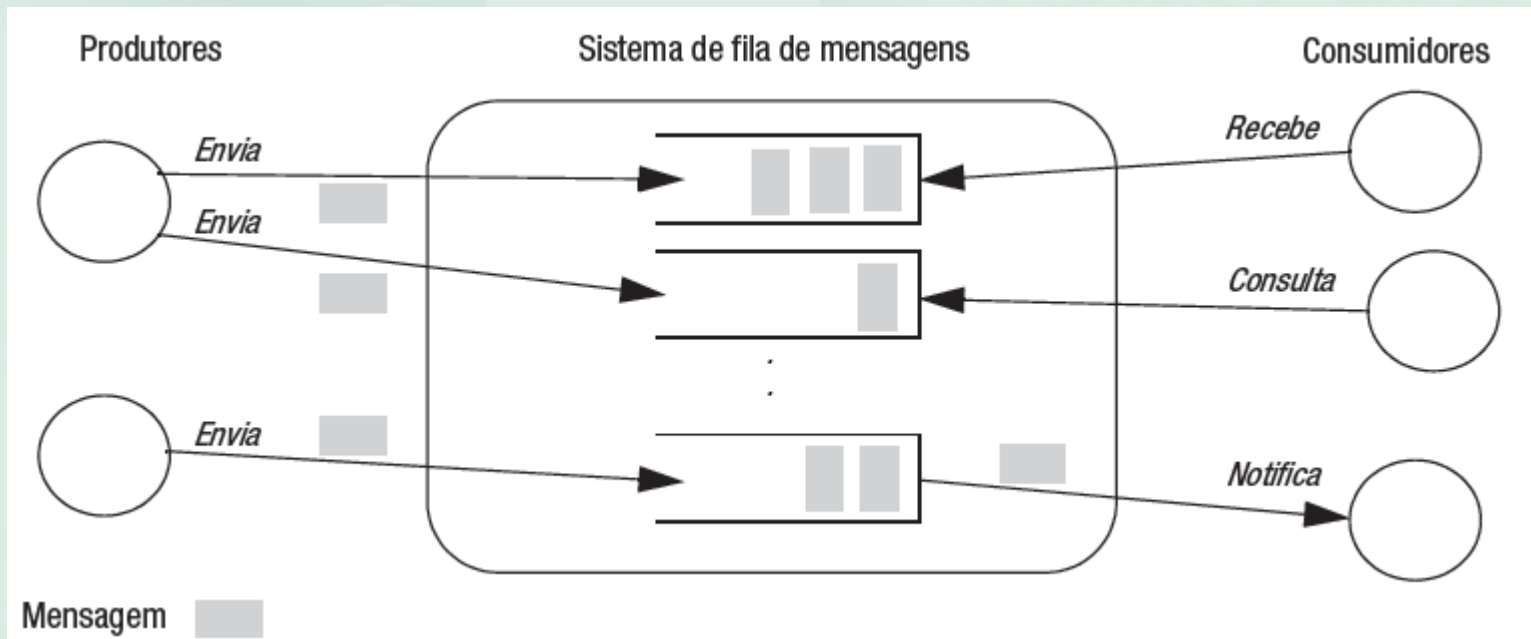
8.1. Sistema de Filas de Mensagens: Introdução

- Remetente só tem a garantia de que, a certa altura, sua mensagem será inserida na fila do receptor;
- Nenhuma garantia é dada sobre quando e nem se a mensagem será realmente lida
 - ações totalmente determinadas pelo comportamento do receptor

8.1. Sistema de Filas de Mensagens: Introdução



8.1. Sistema de Filas de Mensagens: Introdução



8.2. Sistema de Filas de Mensagens: Características das Mensagens

- Mensagens podem conter qualquer tipo de dado;
- Mensagens devem ser adequadamente endereçadas;
- O endereçamento é feito com o fornecimento de um nome exclusivo da fila destinatária no âmbito do sistema.

8.3. Sistema de Filas de Mensagens: Filas

- Mensagens somente podem ser colocadas em filas locais do remetente, na mesma máquina ou em uma máquina na mesma LAN(**Filas de fonte**);
- Mensagem colocada em uma fila contém a especificação de uma **fila de destino**;

8.3. Sistema de Filas de Mensagens: Filas

- Sistema de enfileiramento é responsável por fornecer filas para remetentes e receptores e providenciar para que as mensagens sejam transferidas de sua **fila de fonte** para a **fila de destino**;
- Sistema de enfileiramento deve manter mapeamento de filas para localizações de rede, similar ao serviço de DNS.

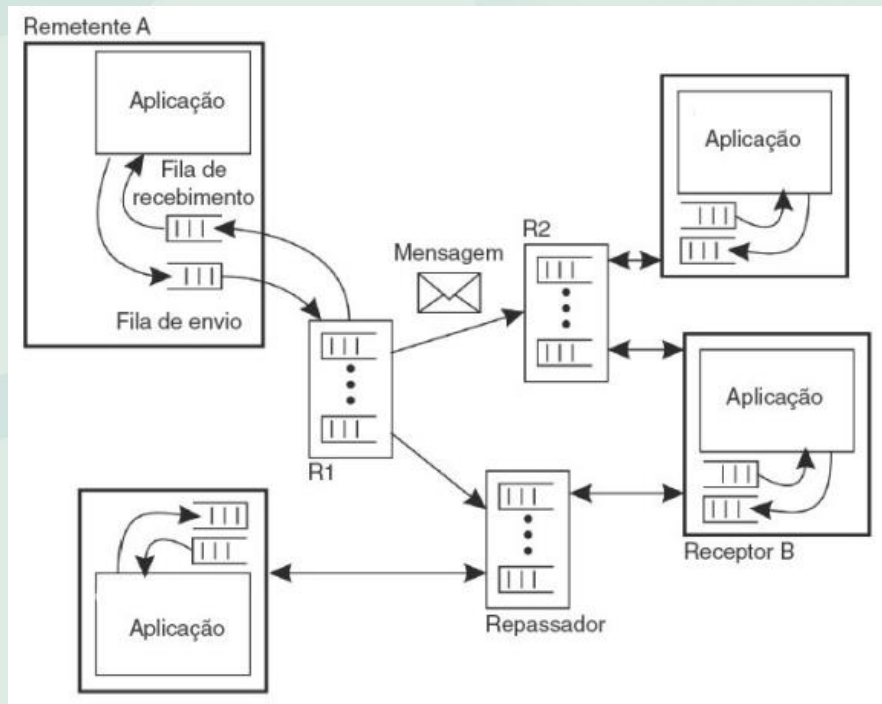
8.4. Sistema de Filas de Mensagens: Arquitetura

- **Gerenciador de Filas:**
 - Interage diretamente com a aplicação que está enviando ou recebendo uma mensagem;

8.4. Sistema de Filas de Mensagens: Arquitetura

- **Repassadores:**
 - Gerenciadores especiais, que funcionam como roteadores;
 - Sistema de enfileiramento pode crescer gradativamente até uma rede de sobreposição de nível de aplicação;
 - Podem ser usados para multicasting.

8.4. Sistema de Filas de Mensagens: Arquitetura



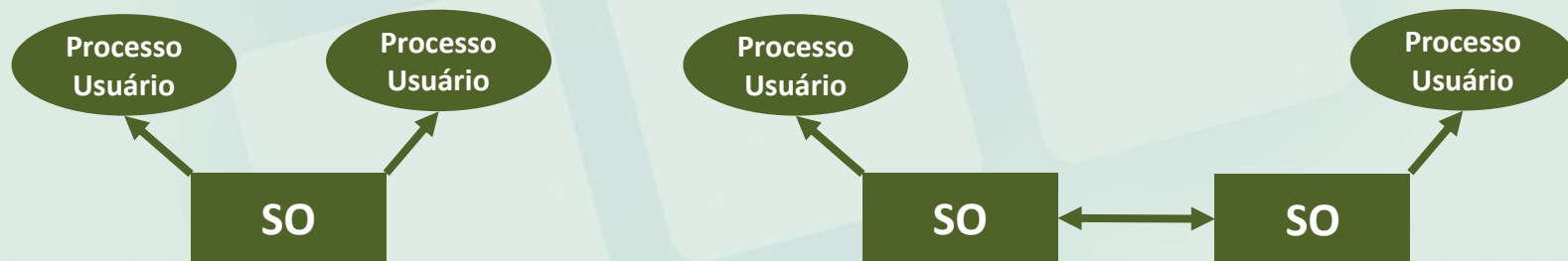


9. Comunicação

IPC - Sockets

9.1. Sockets: Conceitos Básicos

- Sockets são uma forma de IPC (InterProcess Communication) que fornece comunicação entre processos residentes em sistema único ou processos residentes em sistemas remotos.



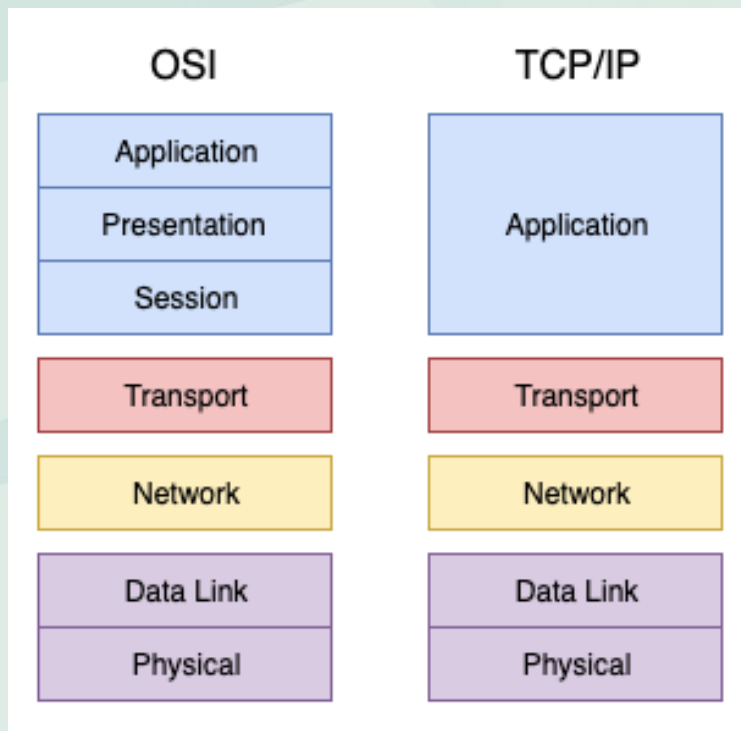
9.2. Sockets: Domínios e Protocolos

- Sockets criados por diferentes programas são referenciados através de nomes;
- Esses nomes devem ser traduzidos em endereços;
- O espaço no qual o endereço é especificado é chamado de domínio;

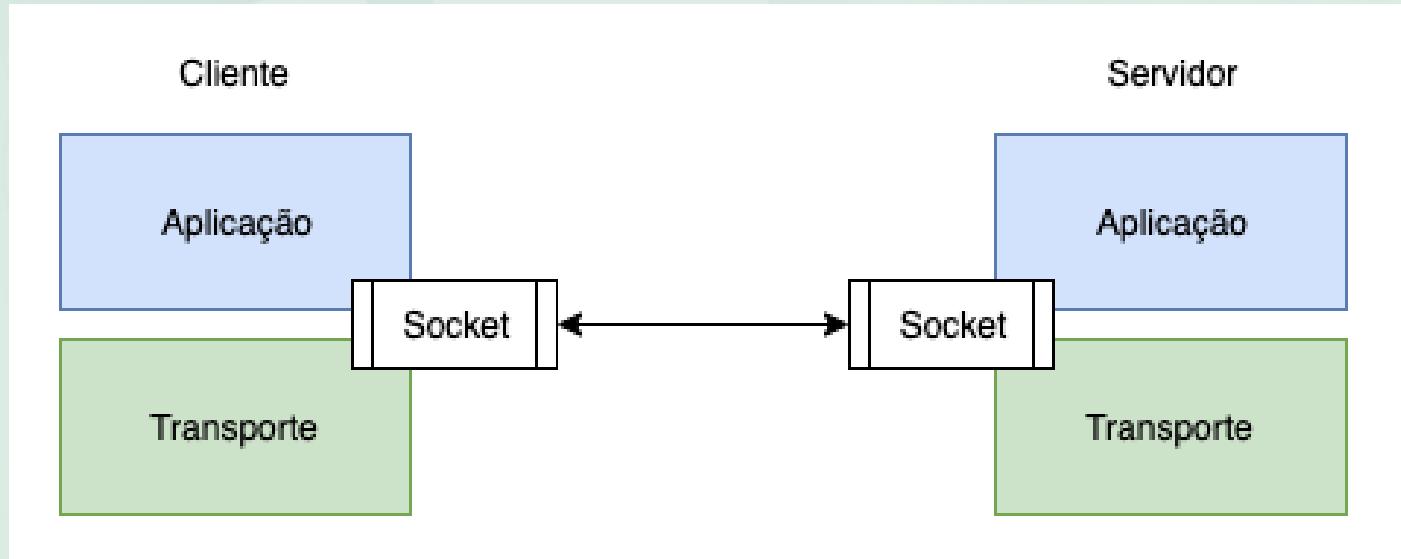
9.2. Sockets: Domínios e Protocolos

- Domínios básicos:
 - **INTERNET (AF_INET)** - os endereços consistem do end. de rede da máquina e da identificação do nº da porta, o que permite a comunicação entre processos de sistemas diferentes;
 - **Unix (AF_UNIX)** - os processos se comunicam referenciando um *pathname*, dentro do espaço de nomes do sistema de arquivos.

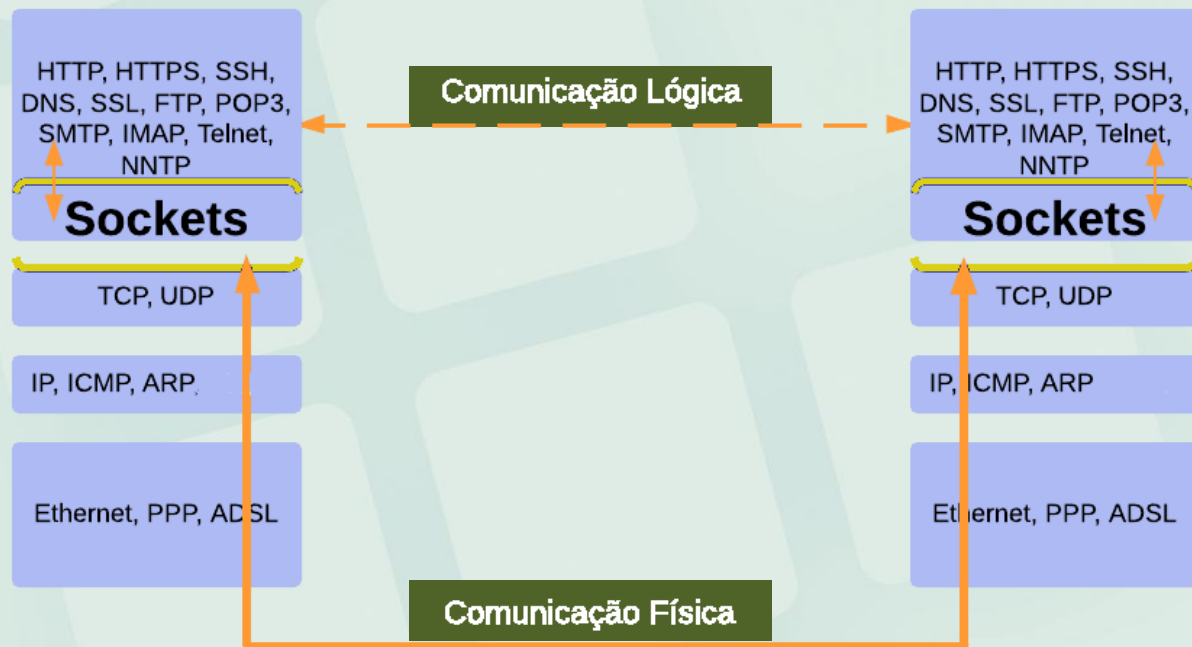
9.2. Sockets: Domínios e Protocolos



9.2. Sockets: Domínios e Protocolos



9.2. Sockets: Domínios e Protocolos



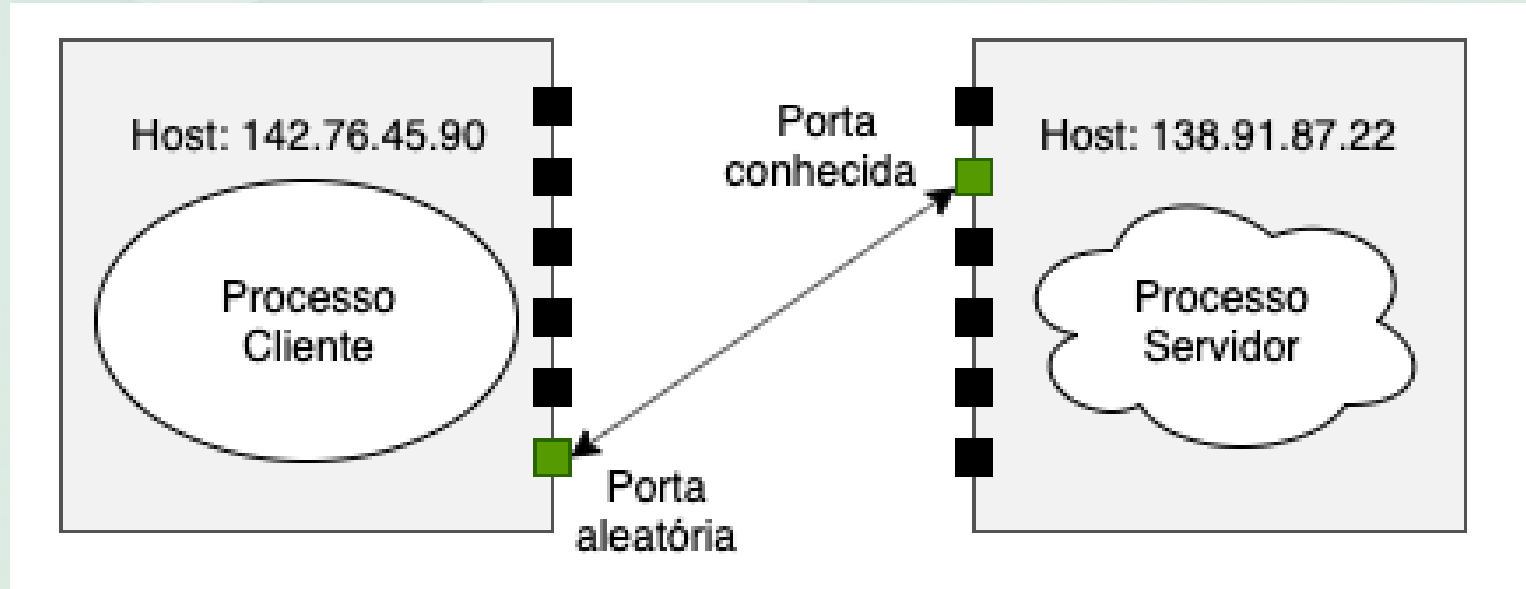
9.2. Sockets: Domínios e Protocolos

- **Domínio Internet**
 - Implementação dos protocolos **TCP** ou **UDP**;
 - Consiste de:
 - endereço de rede da máquina;
 - identificação do número da porta;
 - Permite a comunicação entre máquinas diferentes;

9.2. Sockets: Domínios e Protocolos

- **Domínio Internet**
 - Conexões sob a forma de sockets do tipo *stream* e do tipo datagramas
- **Portas:**
 - “Endereço” para um processo comunicante;
 - Inteiro de 16 bits (definido pelo usuário);
 - Portas 1 a 1023 são do sistema;
 - Portas de TCP independentes das de UDP;

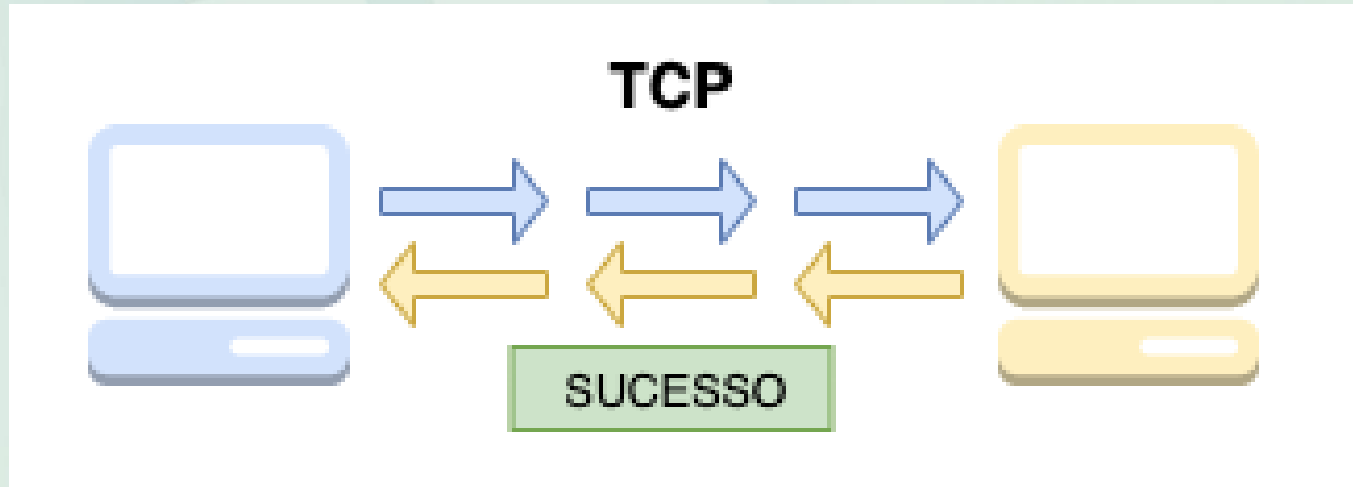
9.2. Sockets: Domínios e Protocolos



9.2. Sockets: Domínios e Protocolos

- **Protocolo TCP**
 - Transmission Control Protocol;
 - Para comunicação longa (conexão);
 - Confiável;
 - Baixo desempenho em comunicações curtas (?);
 - Usos típicos: login remoto, transferência de arquivo...

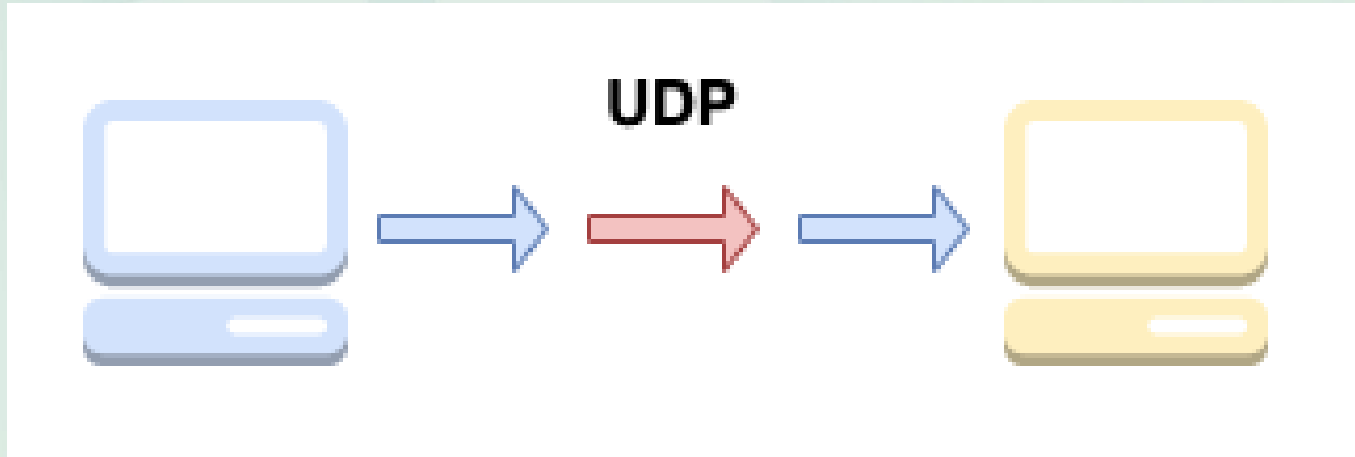
9.2. Sockets: Domínios e Protocolos



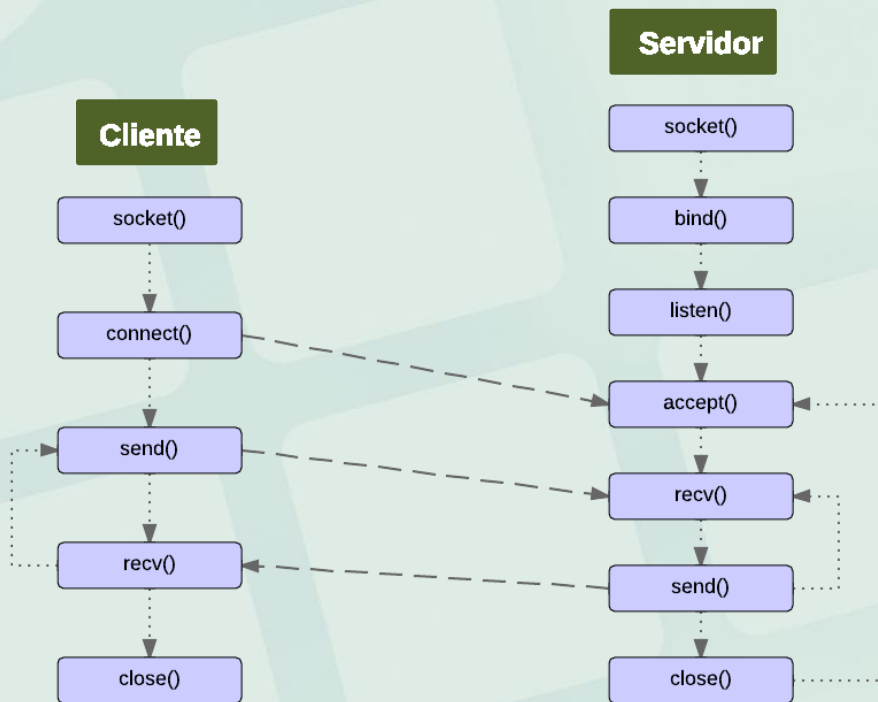
9.2. Sockets: Domínios e Protocolos

- **Protocolo UDP**
 - User Datagram Protocol;
 - Para comunicação curta (sem conexão);
 - Não confiável;
 - Pouco prático para comunicações longas (confiabilidade precisa ser programada);
 - Usos típicos: RPC, Broadcast...

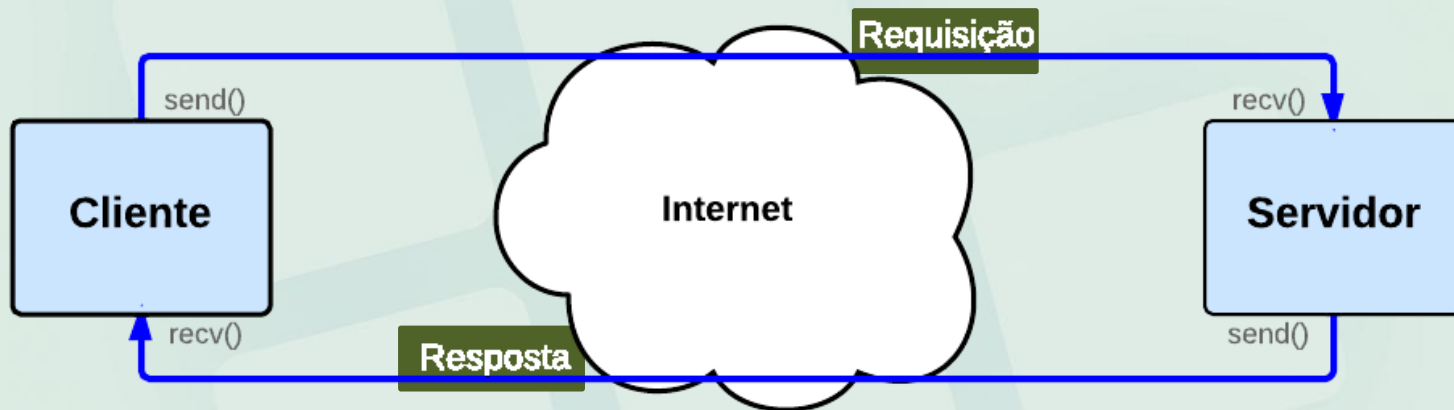
9.2. Sockets: Domínios e Protocolos



9.3. Sockets: Modelo de Comunicação



9.3. Sockets: Modelo de Comunicação



9.4. Sockets: Resumindo

- Os sockets abstraem as camadas de rede para que programadores possam se preocupar com a comunicação de maneira distribuída de seus processos e aplicações.
- A implementação dos sockets foi concebida como uma API com interface para o sistema operacional; que é o responsável por controlar e garantir segurança da criação e destruição desses sockets.

Obrigado!
Vlw! Flw!



INSTITUTO FEDERAL
Sertão Pernambucano