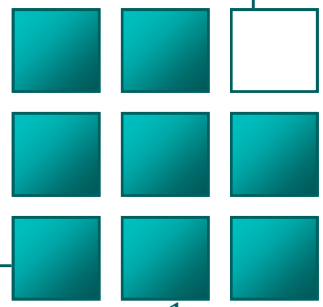
Programa de Pós-graduação em Informática

Tópicos em Sistemas de Computação – Computação em Nuvem

Aletéia Patrícia Favacho de Araújo

Aula 2 – Comunicação

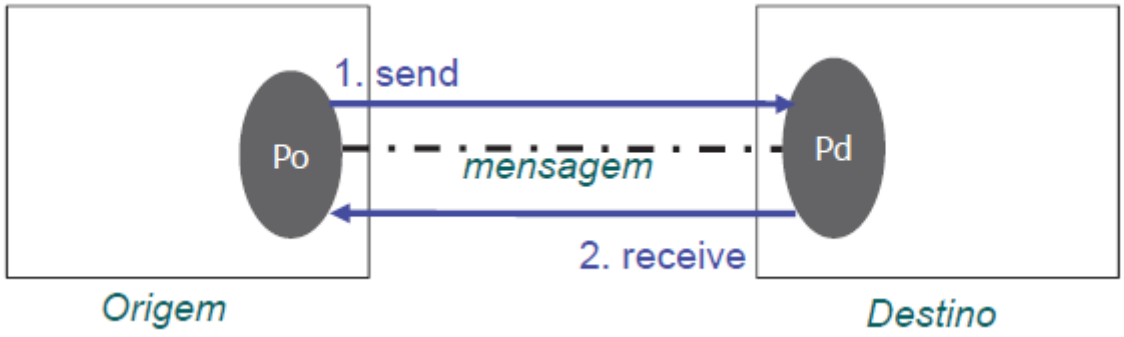


Comunicação em Sistemas Distribuídos

- “Coração” de qualquer Sistema Distribuído.
- Como processos em diferentes máquinas trocam informações?
  - **Não é uma tarefa trivial!**
- Desejável obter modelos onde a complexidade da comunicação seja transparente para o desenvolvedor.

Comunicação em Sistemas Distribuídos

- A passagem de mensagem entre um par de processos pode ser suportada por duas operações: *send e receive;*
- Para ter comunicação, um processo envia (*send)* uma msg para um destino, e o outro processo recebe (*receive).*
- Os remetentes fazem as mensagens serem adicionadas em filas remotas (*buffers)* e os processos destino removem mensagens de suas filas locais.



Comunicação Bloqueantes e Não-Bloqueantes

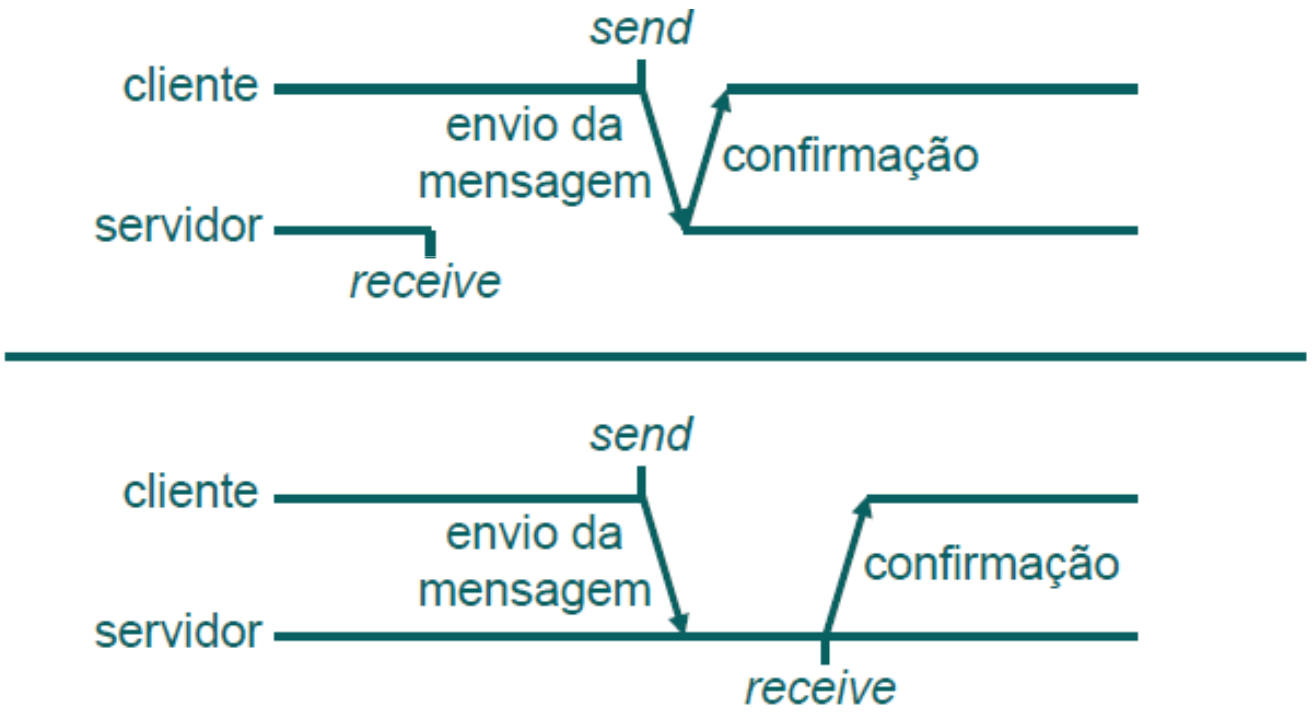
- As primitivas de troca de mensagens podem ser classificadas em:

Primitivas Bloqueantes (Síncronas)	O processo que envia a mensagem é bloqueado até que o receptor tenha aceito a mensagem e a confirmação tenha sido retornada
Primitivas Não-Bloqueantes (Assíncronas)	O processo que envia a mensagem não é bloqueado e pode continuar seu processamento em paralelo com o envio da mensagem

Comunicação Síncrona

- O remetente é bloqueado até saber que sua requisição foi aceita.
- Há, na prática, três maneiras de fazer isso:
  - Primeiro: o remetente pode ser bloqueado até que o *middleware* avise que se encarregará da transmissão da requisição;
  - Segundo: o remetente pode sincronizar até que sua requisição seja entregue ao receptor;
  - Terceiro: a sincronização pode ocorrer permitindo que o remetente espere até que sua requisição tenha sido totalmente processada, isto é, até o instante em que o receptor retornar uma resposta.

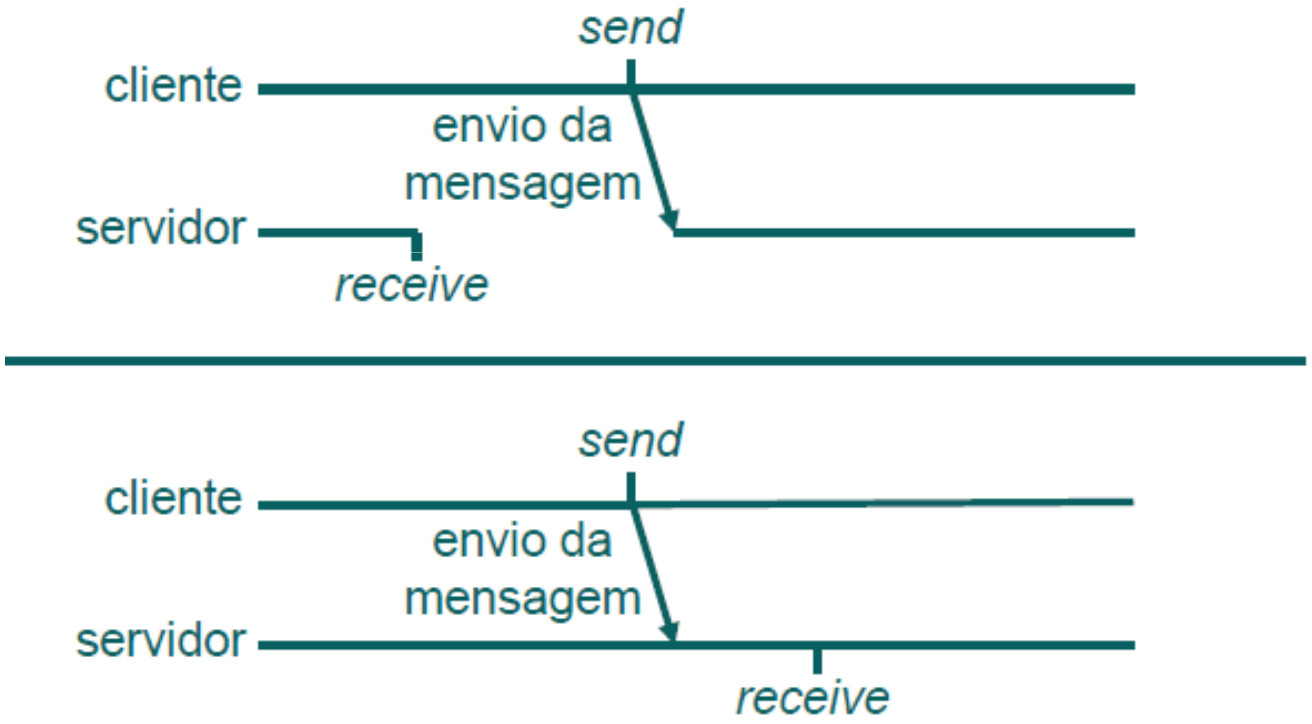
Comunicação Síncrona



Comunicação Assíncrona

- Como a operação *send* é não-bloqueante, o remetente continua sua execução imediatamente após ter apresentado para o *middleware* sua msg.
- Assim, é devolvido o controle ao processo que chamou imediatamente, antes do envio da msg.
- A vantagem é que o processo que envia a mensagem pode continuar processando em paralelo com a transmissão da mensagem.
- Todavia, o transmissor não pode modificar o *buffer* até que a mensagem tenha sido enviada.

Comunicação Assíncrona



Comunicação Persistente e Transiente

- Com **comunicação persistente**, uma mensagem que foi apresentada para transmissão é armazenada durante todo o tempo que for necessário para entregá-la ao receptor.
- Por consequência, não é necessário que a aplicação remetente continue em execução após apresentar a mensagem.
- Da mesma maneira, a aplicação receptora não precisa estar em execução no momento em que a mensagem é apresentada.

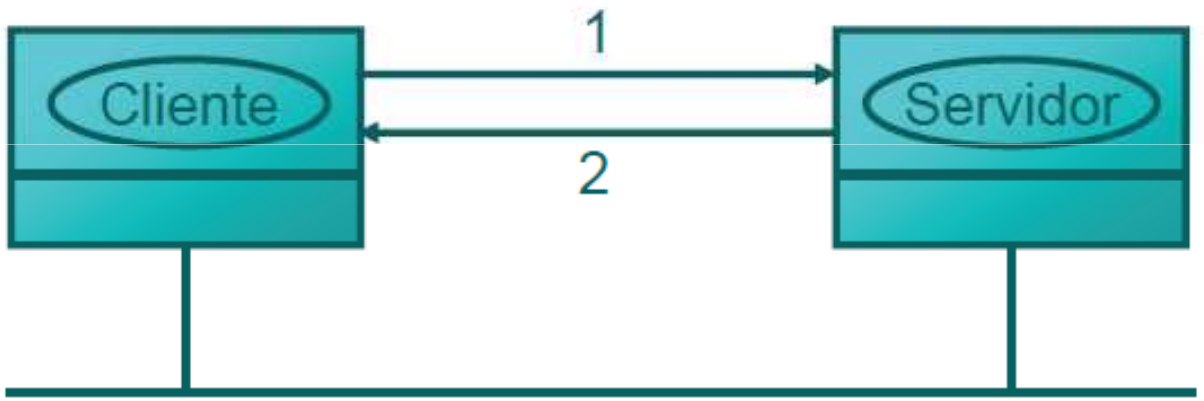
Comunicação Persistente e Transiente

- Com a **comunicação transiente**, uma mensagem é armazenada pelo sistema de comunicação somente durante o tempo em que a aplicação remetente e a aplicação receptora estiverem executando.
- Assim, se a mensagem não puder ser entregue devido a uma interrupção de transmissão, ou se o receptor não estiver ativo no momento considerado, a mensagem será simplesmente descartada.
- Neste caso, a comunicação se baseia em repassadores tradicionais do tipo armazena-e-reenvia.

Características da Comunicação - *Destino da Mensagem*

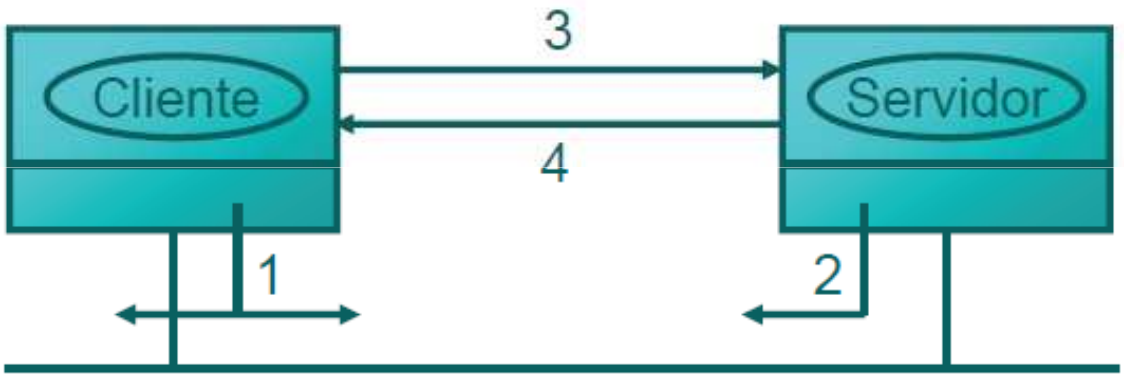
- Um cliente deve conhecer o endereço de um servidor para enviar-lhe uma mensagem.
- Existem diversas formas de endereçamento de processos:
  - Indicar a máquina e o número do processo
  - Deixar o processo escolher um número e localizá-lo através de um *broadcast*
  - Procurar o endereço do processo por meio de um servidor de nomes

Características da Comunicação - *Endereçamento*



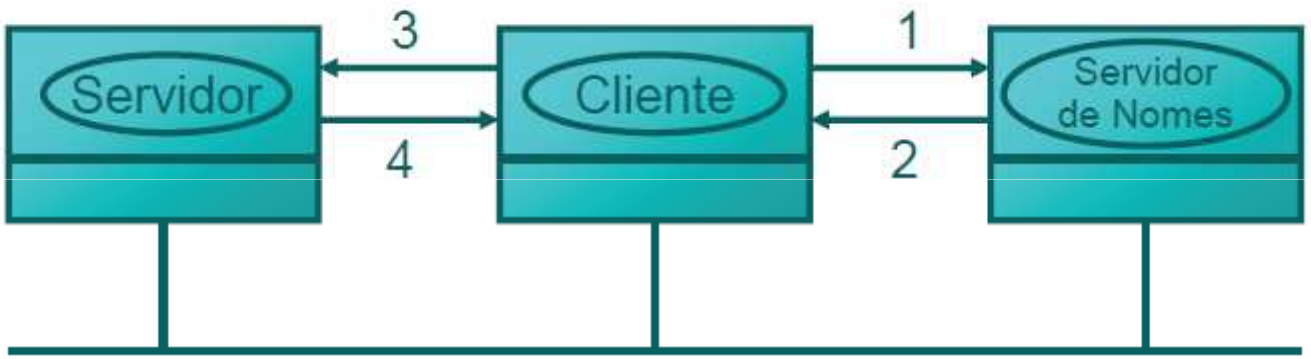
1: Requisição à máquina.número
2: Resposta à máquina.número

Características da Comunicação - *Endereçamento*



1: *Broadcast*
2: Estou aqui
3: Requisição
4: Resposta

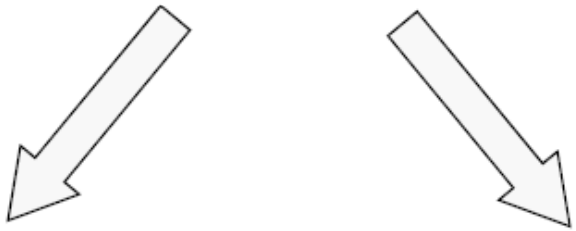
Características da Comunicação - *Endereçamento*



1: Procura
2: Endereço
3: Requisição
4: Resposta

Para Implementar a Comunicação

Quais são as "alternativas de comunicação" entre processos executando na mesma máquina ou máquinas diferentes?



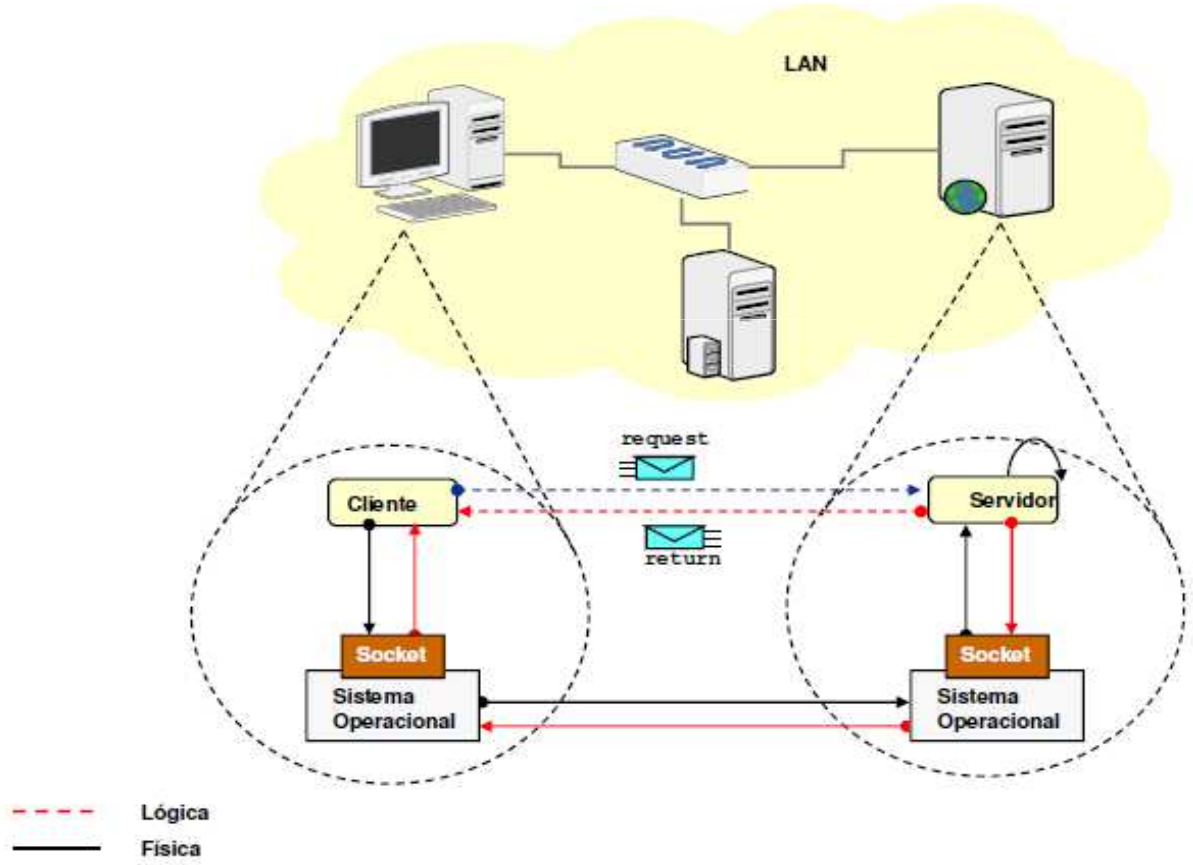
IPC "tradicionais"
(e.g., pipe, socket, memória compartilhada, fila de mensagens, etc)

Middleware

Vamos começar com *Socket*

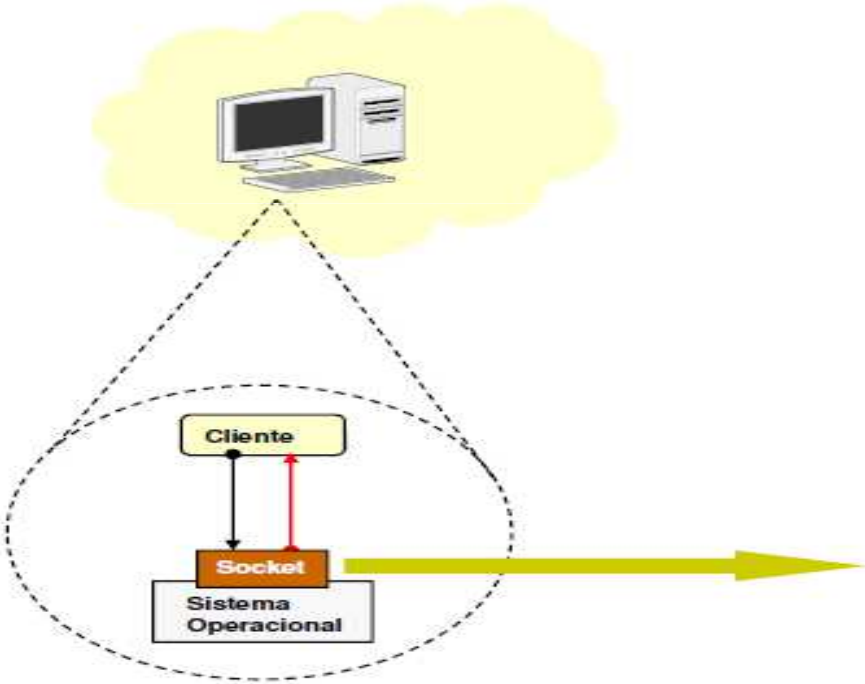
- *Socket* não é *middleware*, mas pode ser usado para construir aplicação distribuída (haja disposição!)
- Construir aplicação distribuída usando *socket* é equivalente a construir aplicações comuns usando *assembly*
- Vamos assumir o uso do TCP...

Vamos começar com *Socket*



Socket

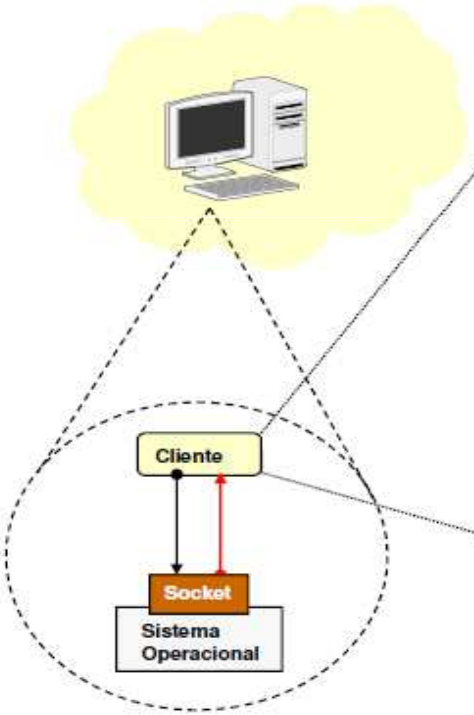
Socket (Unix)



socket ( )
bind ( )
connect ( )
listen ( )
accept ( )
read ( )
write ( )
readv ( )
writev ( )
recv ( )
send ( )
recvfrom ( )
sendto ( )
recvmsg ( )
sendmsg ( )
setsockopt ( )
getsockopt ( )
getpeername ( )
getsockname ( )
gethostbyname ( )
getservbyname ( )

Socket

Socket (Unix)



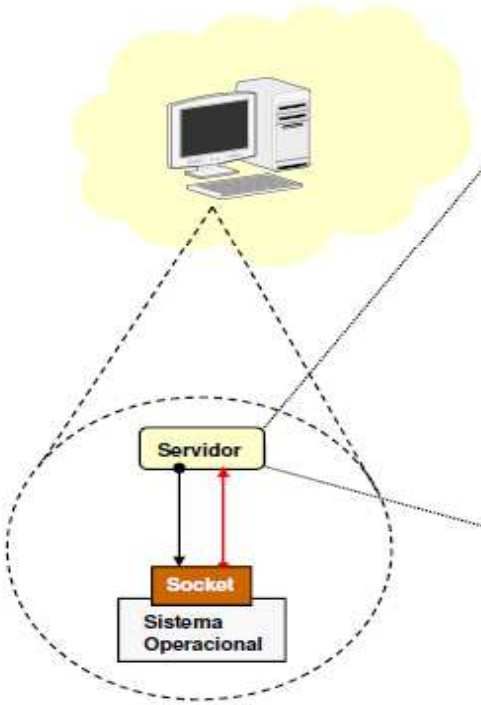
Cliente / TCP

```
...
main(argc, argv)
int argc; char **argv, r;
{
  hp = gethostbyname (hostname);
  ...
  sd = socket (AF_INET, SOCK_STREAM, 0);
  ...
  r = connect (sd,(struct sockaddr *)  &pin, sizeof(pin));
  ...
  r = send (sd, argv[1], strlen(argv[1]), 0);
  ...
  r = recv (sd, dir, DIRSIZE, 0);
  ...
  close (sd);
}
```

Comunicação entre aplicações

Socket

Socket (Unix)



Servidor / TCP

```
...
main()
{
sd = socket (AF_INET, SOCK_STREAM, 0));
...
r = bind (sd, (struct sockaddr *) &sin, sizeof(sin));
...
r = listen (sd, 5) == -1);
...
sd_current = accept (sd, (struct sockaddr *)  &pin, &addrlen));

r = recv (sd_current, dir, sizeof(dir), 0);
...
r  = send (sd_current, dir, strlen(dir), 0);
...
close(sd_current);
close(sd);
...
}
```

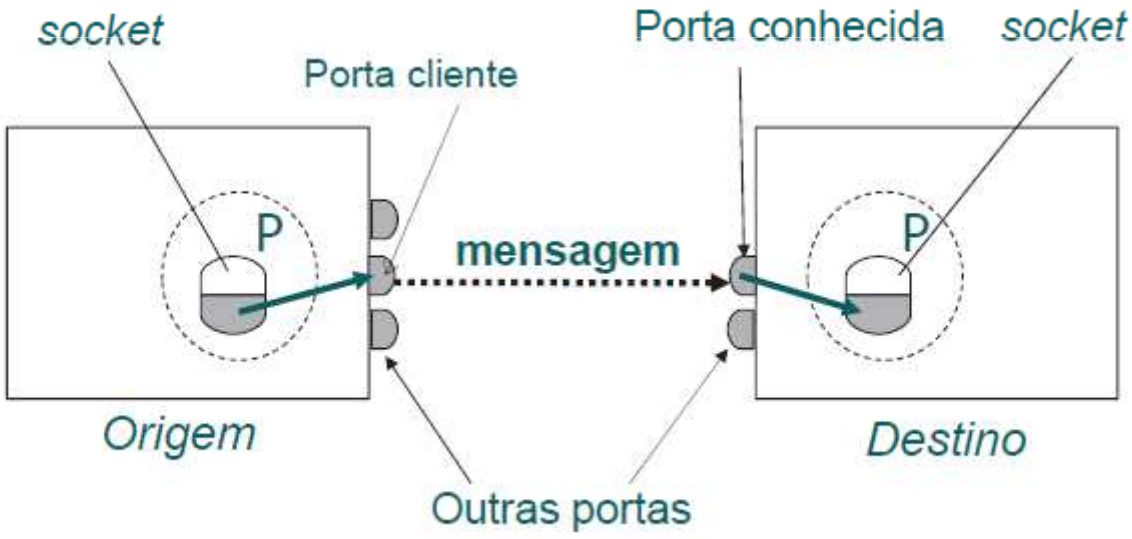
Comunicação entre aplicações

Socket

- *Socket* é uma abstração que representa um ponto de destino para a comunicação entre processos.
  - Identificado pelo par (endereço IP, porta local)
- *Socket* é a porta entre o processo da aplicação e o protocolo de transporte. A comunicação consiste em transmitir uma mensagem entre um *socket* de um processo e um *socket* de outro processo.
  - Para que um processo receba mensagem, seu *socket* deve estar vinculado a uma porta local e a um endereço IP do computador em que é executado.

Socket

- As mensagens enviadas para um *socket* só podem ser recebidas por um processo.



Socket

- Cada computador tem 2^{16} (65.536) números de portas disponíveis para serem usados pelos processos para o envio e recepção de mensagens.
- Qualquer processo pode fazer uso de várias portas para receber mensagens.
- Mas, um processo não pode compartilhar portas com outros processos no mesmo computador.
- A programação por portas se utiliza dos serviços de redes, sejam eles orientados (TCP) ou não orientados (UDP) a conexão.

Socket

- Não existe transparência, pois toda a comunicação está explicita, por meio dos procedimentos *send* e *receive.*
  - Funções mais sofisticadas devem ser feitas na camada de aplicação.

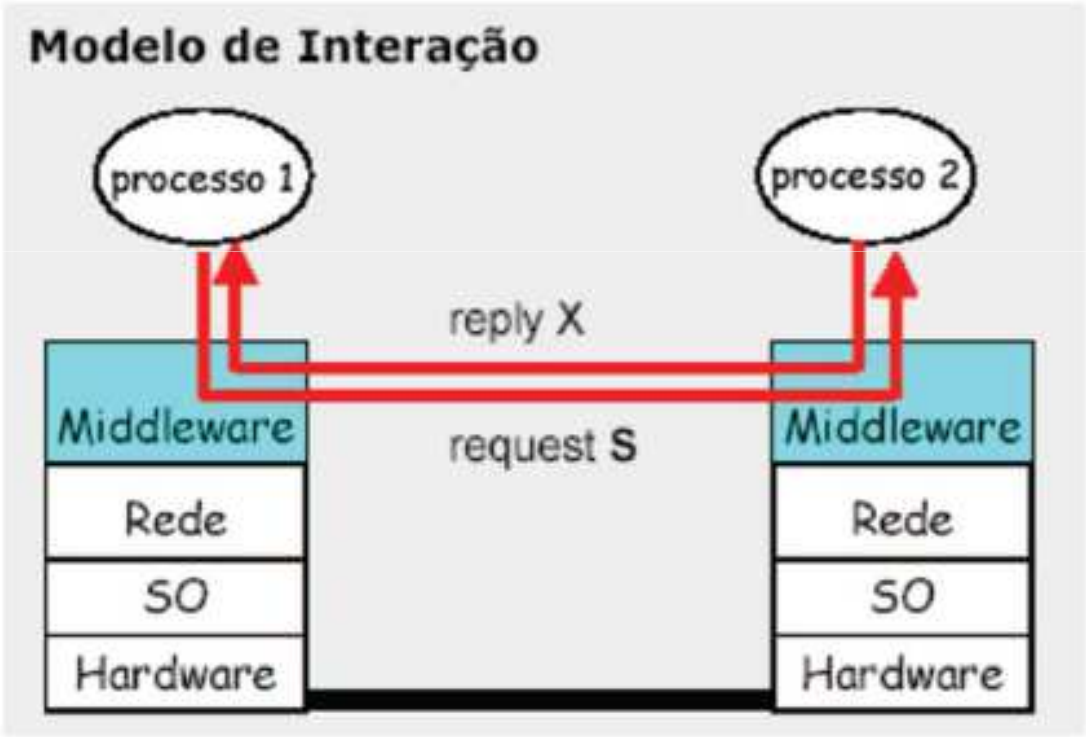
Por que não oferecer comunicação de alto nível, independente da aplicação?

Solução...

Middleware de Comunicação!

- Caracteriza uma camada de software que possibilita comunicação entre aplicações distribuídas, tendo por objetivo diminuir a complexidade por meio de serviços que realizem a comunicação de forma transparente.

Middleware de Comunicação



Middleware

- Camada de software que tem por finalidade:
  - Mascarar a heterogeneidade da plataforma subjacente (hardware, SO, linguagem);
  - Resolver demais problemas oriundos da distribuição de forma transparente;
  - Prover um modelo de programação conveniente para o programador de aplicações;
  - Prover serviços de infraestrutura padronizados para uso no desenvolvimento de aplicações.
    - Ex.: resolução de nomes, segurança, transações etc.

Formação de um *Middleware*

- Processos, objetos ou componentes localizados nos computadores do sistema distribuído.
  - os quais interagem e cooperam entre si para prover o suporte de comunicação e compartilhamento de recursos necessário às aplicações;
- Primitivas básicas para a construção de componentes de software que funcionam cooperativamente em um sistema distribuído.
- Plataforma de alto nível para o desenvolvimento de aplicações.

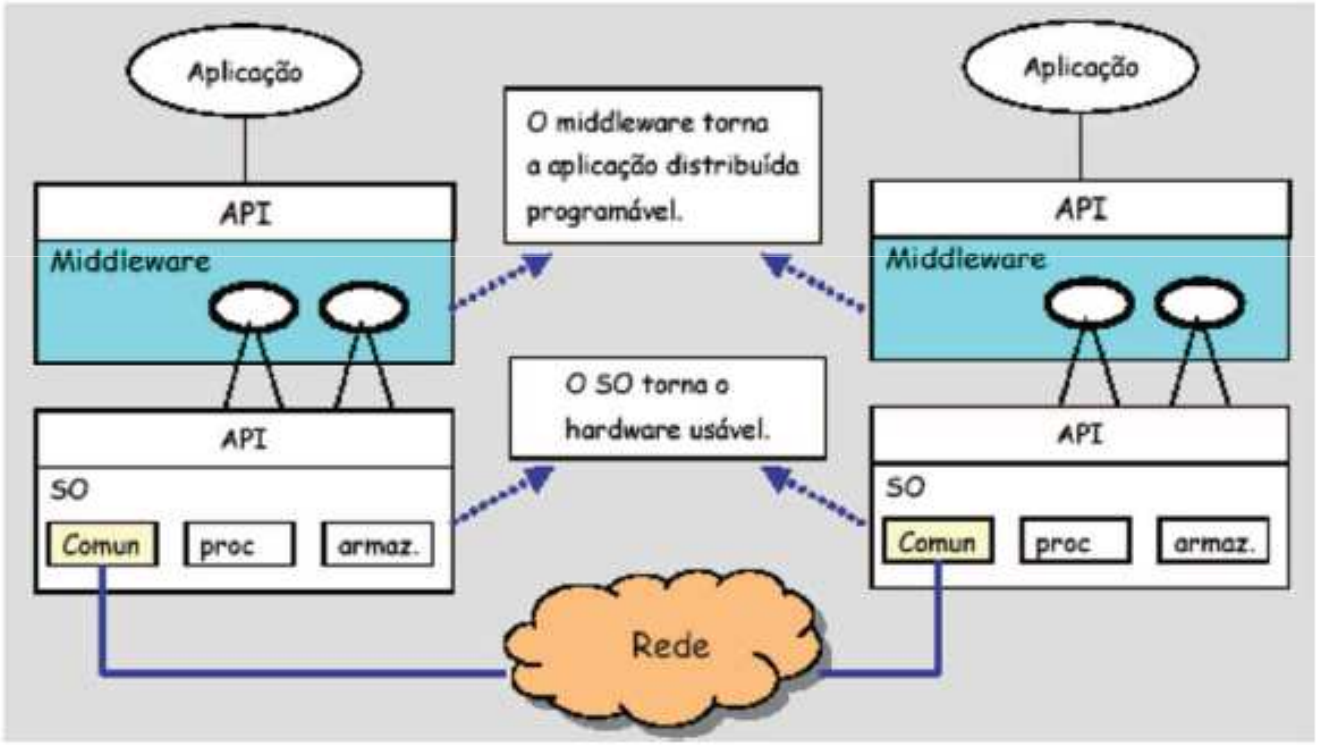
Serviços de um *Middleware*

- Os *middlewares* comumente fornecem os seguintes serviços:
  - Ciclo de vida: gerenciamento do ciclo de vida dos objetos;
  - Serviço de nomes: permite referenciar objetos pelo nome;
  - Relacionamento: cria associações dinamicamente entre objetos;
  - Transação: faz gerenciamento de transações;

Serviços de um *Middleware*

- Cont.:
  - Negócio: permite localização de serviços;
  - Segurança: oferece funcionalidades de segurança;
  - Tempo: serviço de sincronização de relógios;
  - Evento: registra interesse de componentes por eventos.

Serviços de um *Middleware*



Representação Externa de Dados

- Informações armazenadas nos programas em execução são representadas como estruturas de dados. E as informações presentes nas msgs são sequências de bytes.
- Problema:
  - Diferentes sistemas = diferentes estruturas de dados.
  - Ex.: ASCII versus Unicode, pontos flutuantes, etc...
- Solução:
  - Conversão para um formato de dados acordado (*external data representation)*
  - Enviar no formato do emissor e incluir informação sobre o formato empregado.

(Un)Marshalling – (Des)Empacotamento

- Empacotamento (*Marshalling)*
  - Processo de pegar um conjunto de itens de dados e montá-los em uma forma conveniente para transmissão em uma mensagem.
  - Compreende a tradução de itens de estruturas de dados e valores primitivos para uma representação externa de dados.
- Desempacotamento (*Unmarshalling)*
  - É o processo contrário.
- A idéia é que seja feito sem envolvimento explicito da aplicação.
  - Responsabilidade do *middleware.*

Interfaces em Sistemas Distribuídos

- Para controlar as interações possíveis entre os módulos, uma interface deve ser definida para cada módulo.
- Interface de um módulo determina os procedimentos e variáveis que podem ser acessadas a partir de outros módulos.
- Se a interface do módulo permanecer a mesma, a sua implementação pode ser alterada sem afetar os usuários do módulo.

Interface de Serviço

- Interface de serviço
  - O termo **interface de serviço** é usado para se referir à especificação dos procedimentos oferecidos por um servidor, definindo os tipos de argumentos de entrada e saída de cada um dos procedimentos.
  - No modelo cliente-servidor, cada servidor fornece um conjunto de procedimentos remotos que estão disponíveis para o cliente.
  - Exemplo: servidor de arquivos oferece procedimentos para leitura/escrita de arquivos.

Interface Remota

- Interface remota
  - Uma interface remota especifica os métodos de um objeto que estão disponíveis para invocação por parte dos objetos de outros processos, definindo os tipos dos argumentos de entrada e saída de cada um deles.
  - Os métodos nas interfaces remotas podem passar objetos como argumentos e resultados de métodos. Além disso, também podem ser passadas referências para objetos remotos.

Middleware na Presença de Falhas

1. O cliente não consegue localizar o servidor.
2. A mensagem de requisição é perdida.
3. A mensagem de resposta é perdida.
4. O servidor sai do ar após receber a requisição.
5. O cliente falha após enviar a requisição.

Semânticas de Invocação

- *Exactly Once* (Exatamente uma vez): *procedimento* remoto executado exatamente uma única vez.
- *At-least-once* (No Mínimo uma vez): *procedimento* remoto é executado até que receba uma resposta.
- *Maybe* (Talvez): *não há controle de retransmissão,* independente se o procedimento remoto foi ou não executado.

Semânticas de Invocação Talvez

- O método remoto pode executar uma vez ou não ser executado.
- A semântica talvez ocorre quando nenhuma das medidas de tolerância a falhas é aplicada.
- Se a mensagem de resultado não tiver sido recebida após um dado tempo limite, não haverá certeza se o método foi executado.
  - Se a invocação for perdida, então o método não será executado.
  - Mas, o método pode ter sido executado e o resultado perdido.
- Esta semântica é útil apenas para aplicações nas quais são aceitáveis invocações mal-sucedidas ocasionalmente.

Semântica de Invocação pelo Menos uma Vez

- O invocador recebe um resultado quando o método foi executado pelo menos uma vez, ou recebe uma exceção, informando-o que nenhum resultado foi obtido.
- Essa semântica pode ser obtida pela retransmissão das mensagens de requisição, o que mascara as falhas por omissão da mensagem de invocação ou resultado.
- A semântica pelo menos uma vez é útil quando os objetos em um servidor podem ser projetados de modo que os métodos em suas interfaces remotas sejam operações idempotentes.
  - **Operações Idempotentes** são aquelas que podem ser executadas repetidamente, com o mesmo efeito de que se tivesse sido executada exatamente uma vez.

Semântica de Invocação no Máximo uma vez

- Com essa semântica, ou o ativador recebe um resultado quando o método foi executado exatamente uma vez, ou em caso contrário, uma exceção.
- Como no caso anterior, o emprego de retentativas mascara as falhas por omissão pela perda das mensagens de requisição ou de resultado.
- É necessário ainda que as falhas arbitrárias sejam evitadas, garantido que um método nunca seja executado mais de uma vez.
  - Para isso, é necessário implementar controle de mensagens duplicadas.
  - Armazenamento de histórico das respostas.

Semânticas de Invocação

Retransmitir Mensagem de Pedido	*Filtragem de Duplicatas*	*Re-executar Procedimento ou Retransmitir Resposta*	*Semântica de Invocação*
Não	Não aplicável	Não aplicável	*Maybe* (talvez) *Cliente não sabe se o método foi executado*
Sim	Não	Re-executar procedimento	***At-least-once* (pelo menos uma vez)** *Cliente sabe que o método foi executado pelo menos uma vez (SUN RPC)*
Sim	Sim	Retransmitir resposta	***At-most-once* (no máximo uma vez)** *Cliente sabe que o método foi executado exatamente uma vez (Java RMI)*

Modelos de Programação para Aplicativos Distribuídos

- Aplicativos distribuídos são compostos de programas que estão em cooperação, executados em vários processos diferentes.
- Esses programas precisam executar (invocar) operações em outros processos.
- Esses processos, frequentemente, são executados em diferentes computadores.

Modelos de Programação para Aplicativos Distribuídos

- Modelos de programação para aplicações distribuídas:
  - **RPC (*Remote Procedure Call)*** permite que programas clientes possam chamar procedimentos em programas servidores remotos.
  - **RMI (*Remote Method Invocation)*** permite a um objeto local invocar métodos de objetos remotos.
  - **Orientada a mensagem** permite aos processos trocarem informações ainda que a outra parte não esteja executando no momento em que a comunicação é ativada.
  - **Orientada a fluxo** permite trocar informações dependentes de tempo como fluxos de áudio e vídeo.

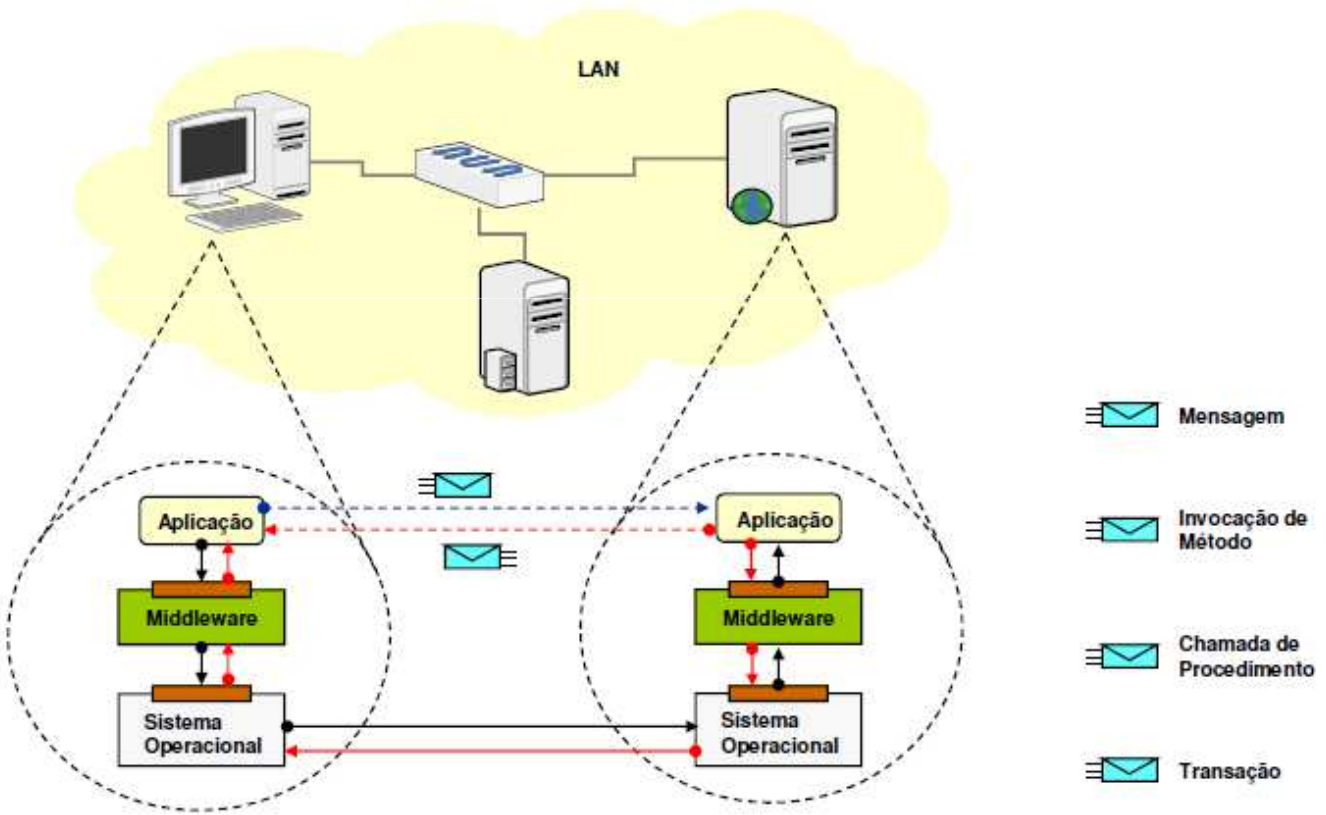
Relembrando...

- Aplicações distribuídas podem ser construídas com mecanismos de IPC, mas é muito mais fácil usar *middleware*.
- *Middleware* é uma camada de software localizada entre o sistema operacional e a aplicação.

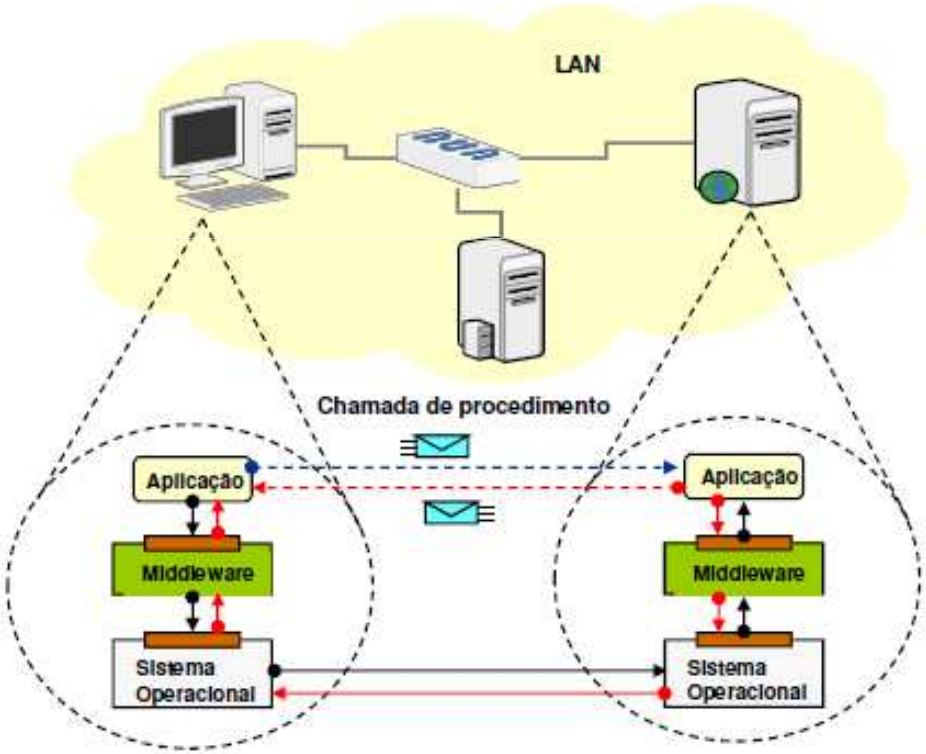
Tipos de *Middleware*

- Não existe uma classificação única. Há diversas classificações baseadas em diferentes características do *middleware.*
- Os *middlewares* são classificados em:
  - Orientado à Chamada Remota de Procedimento (RPC);
  - Orientado a Objetos (ORB);
  - Orientado à Mensagem (MOM);
  - Orientado à Transação (ou Transacionais).

Tipos de *Middleware*

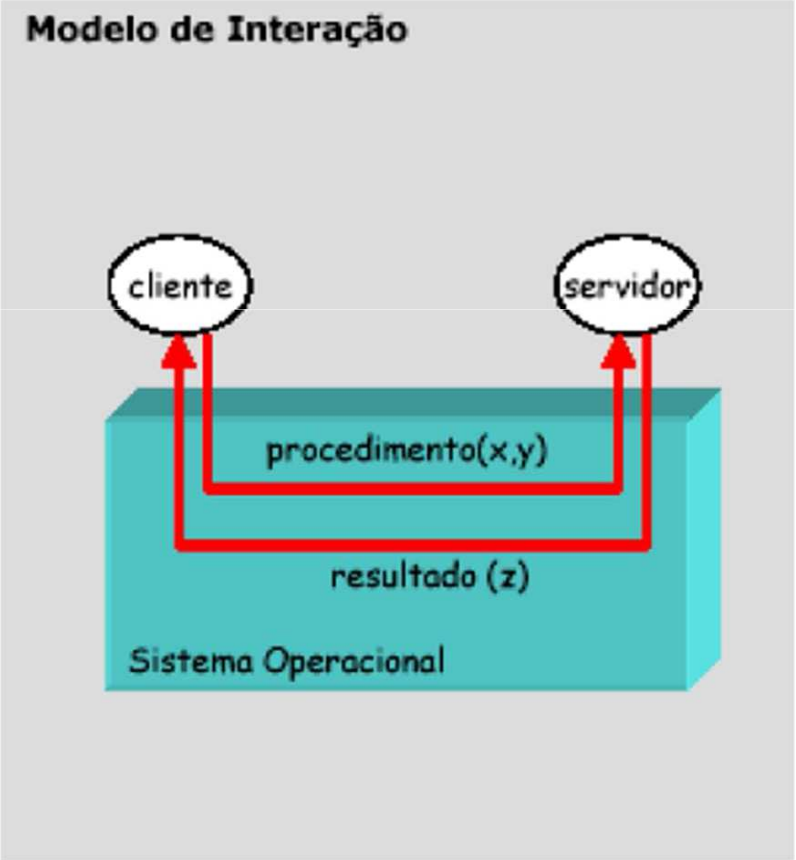


Middleware Orientado à Chamada Remota de Procedimentos (RPC)



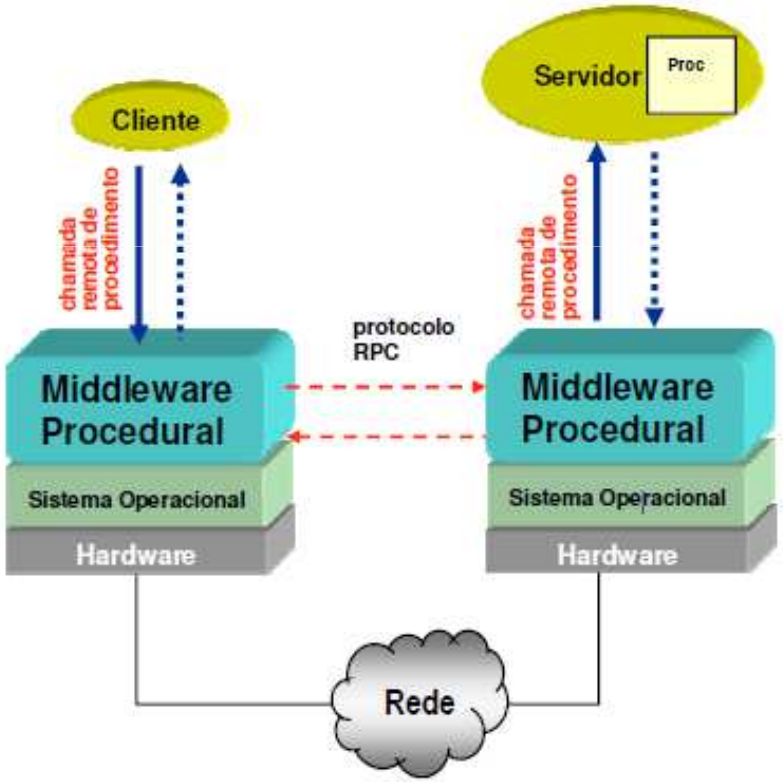
Middleware Orientado à Chamada Remota de Procedimentos (RPC)

- Também chamado de *middleware* procedural.
- Uma das primeiras formas de comunicação entre processos remotos.
- Primitiva de interação: chamada remota de procedimento (1-1).
- Comunicação tipicamente síncrona.

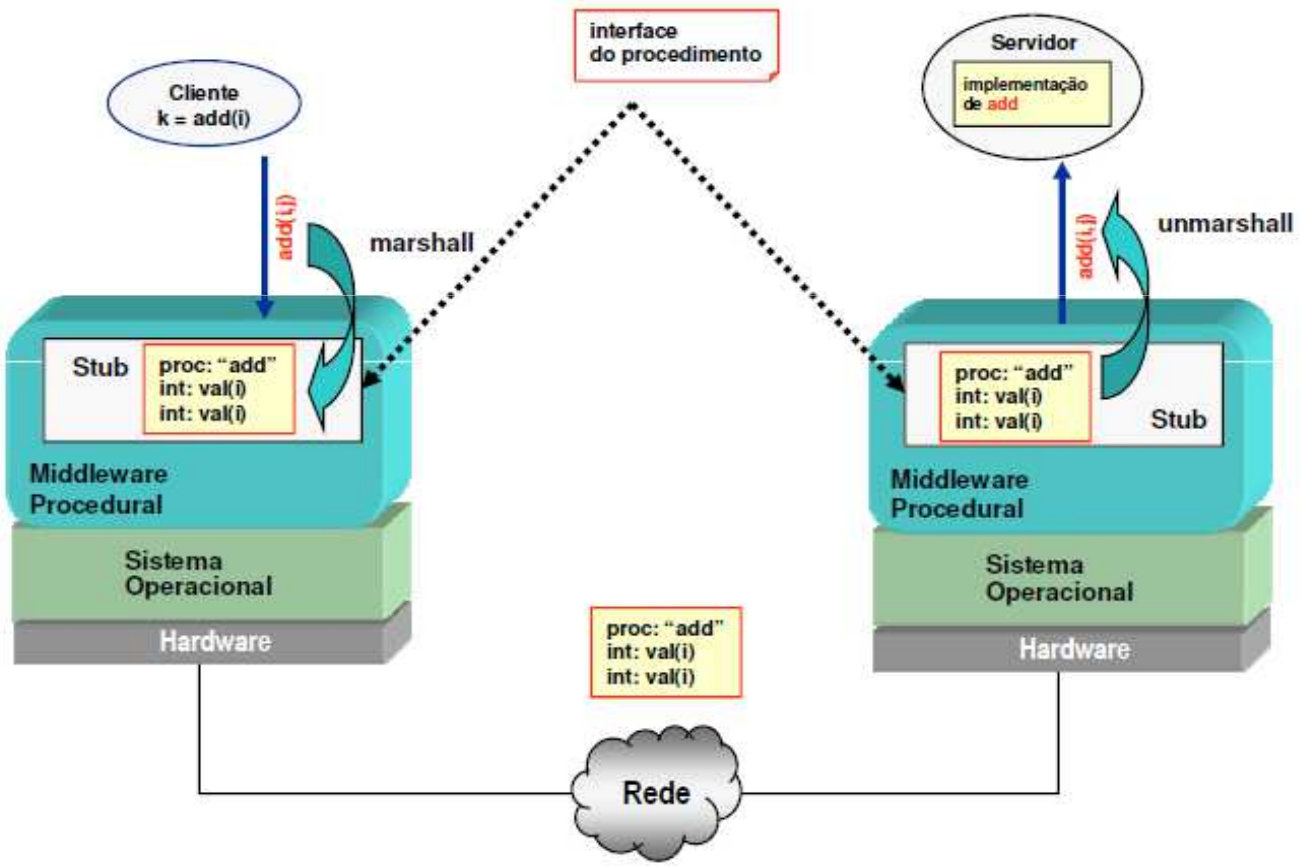


Middleware Orientado à Chamada Remota de Procedimentos (RPC)

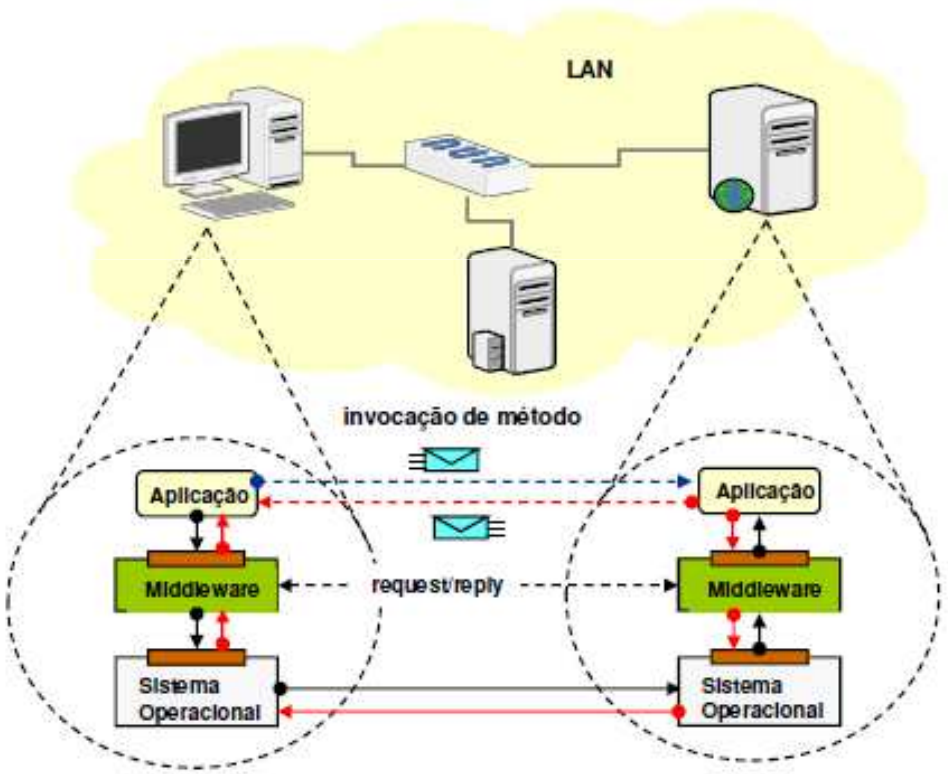
- Protocolo *request/wait-for-reply*.
- Os tipos usados como parâmetro são padronizados.
- Há IDLs para descrever as interfaces.
- Ex.: DCE (*Distributed Computing Environment*) e RPC da SUN.



Middleware Orientado à Chamada Remota de Procedimentos (RPC)

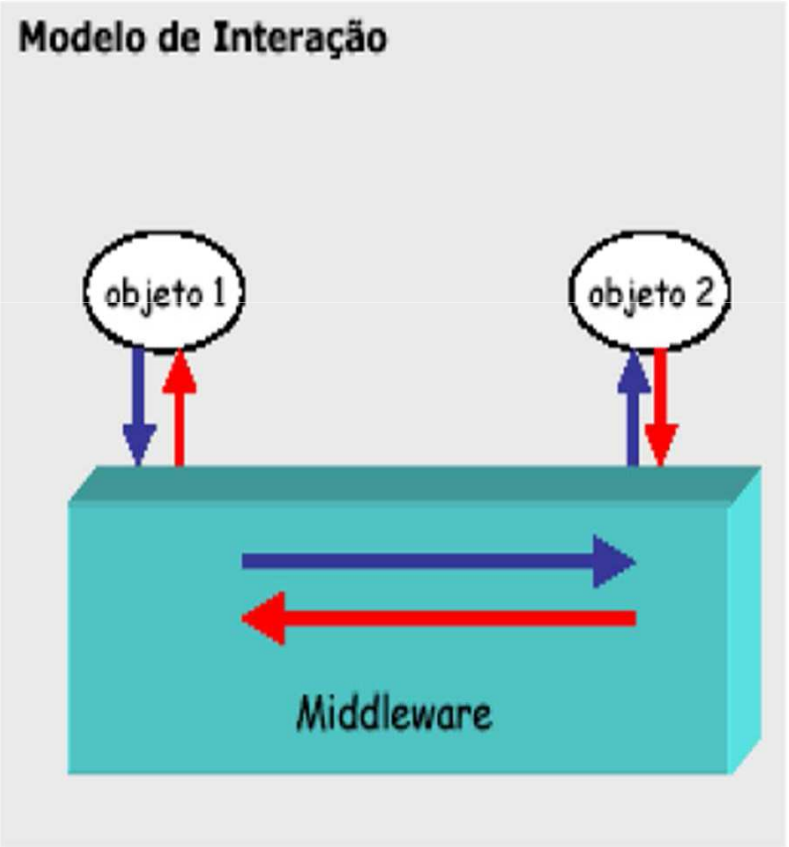


Middleware Orientado a Objetos



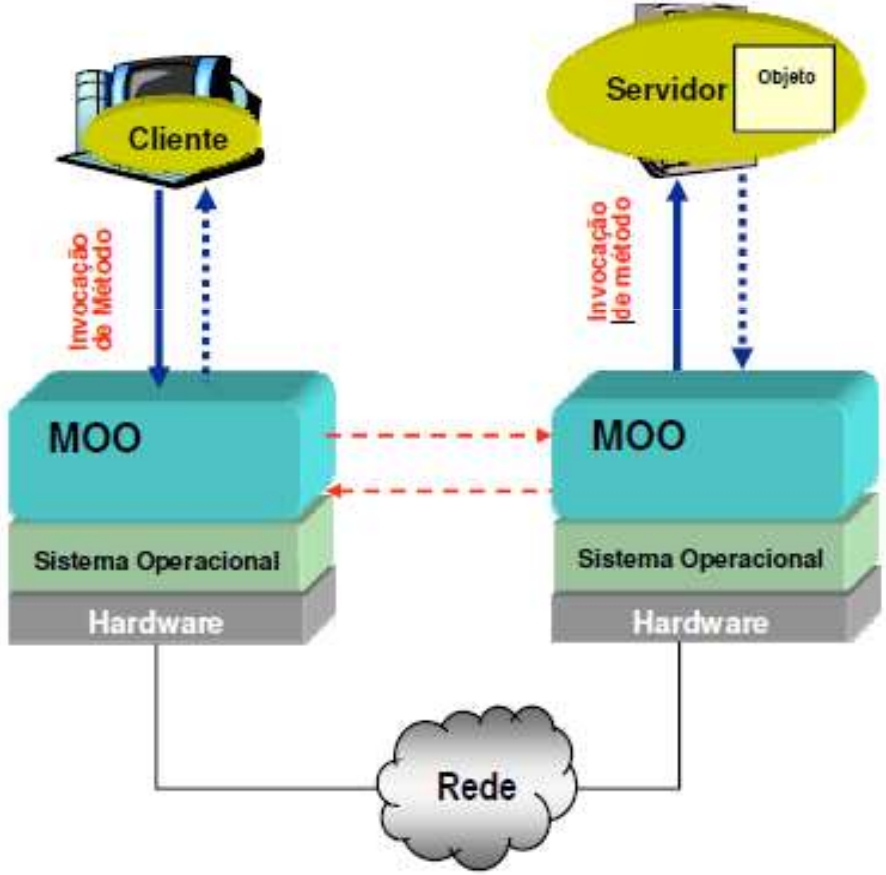
Middleware Orientado a Objetos

- Pode ser considerado um RPC orientado a objetos.
- Evolução do *middleware* procedural.
- Interação por invocação de métodos (RMI).
- Comunicação tipicamente síncrona entre os objetos.

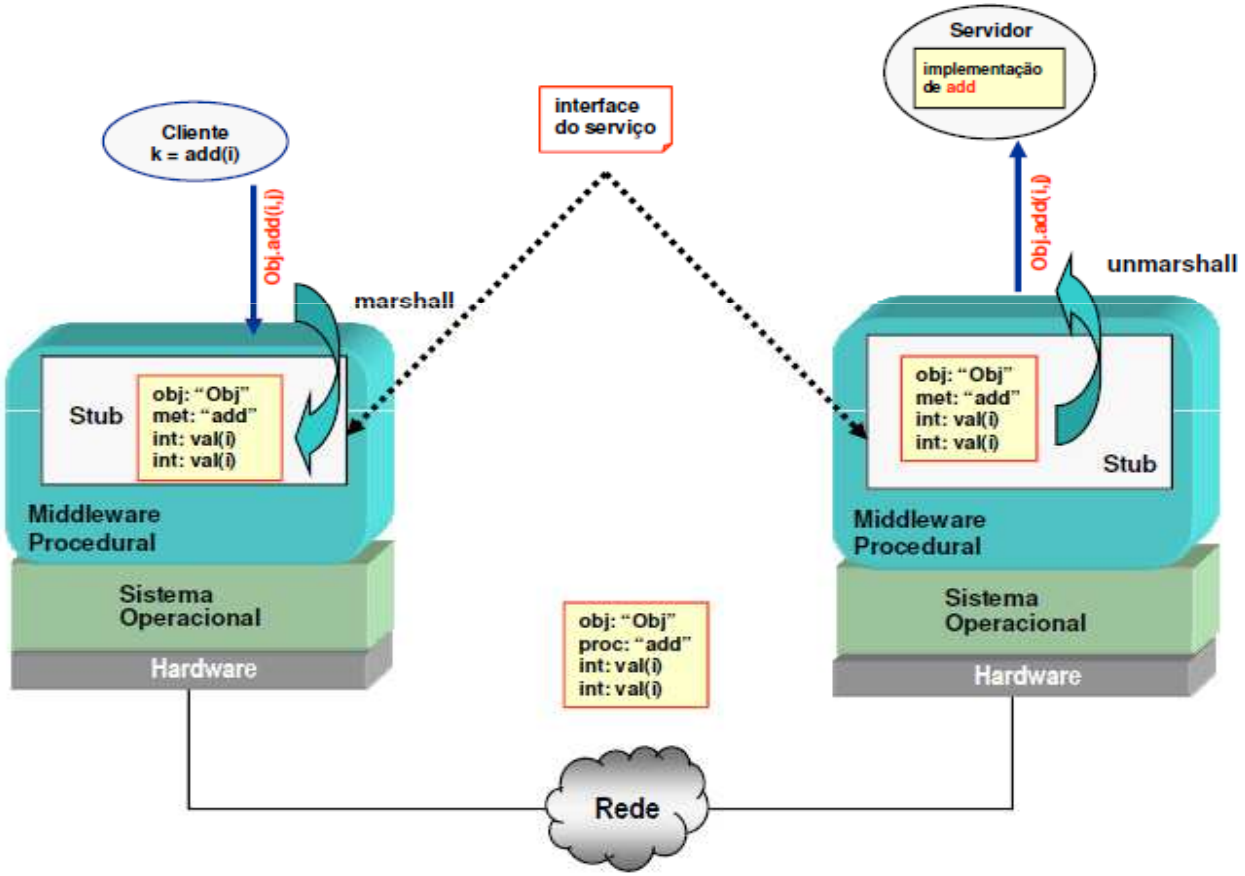


Middleware Orientado a Objetos

- São usadas IDL para descrever os serviços.
- Exemplos: RMI e CORBA.

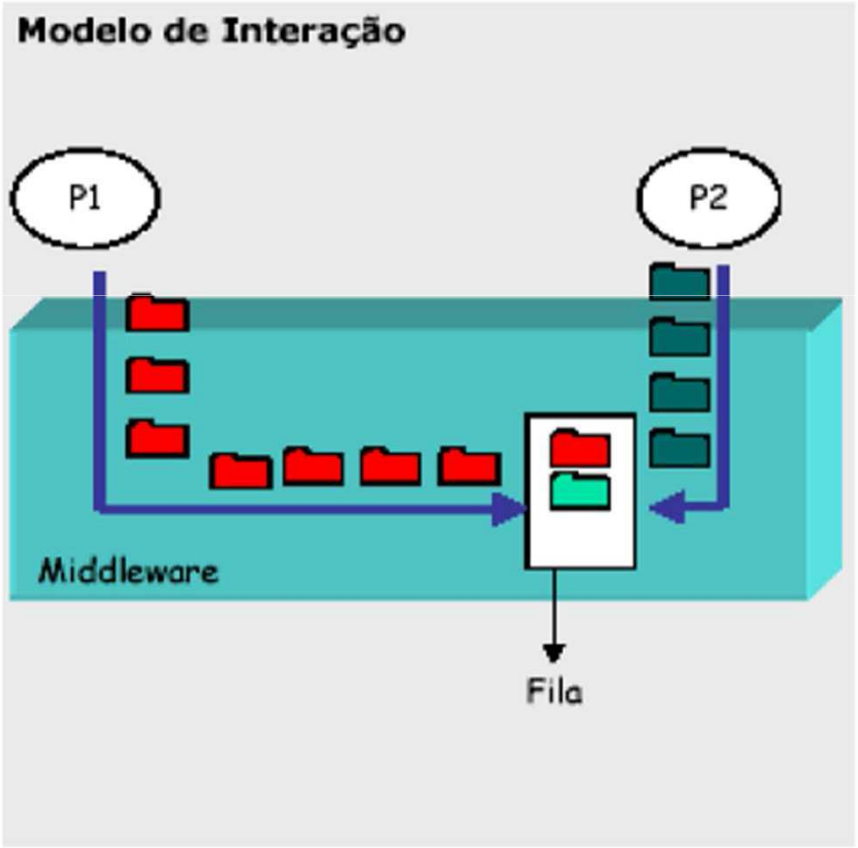


Middleware Orientado a Objetos



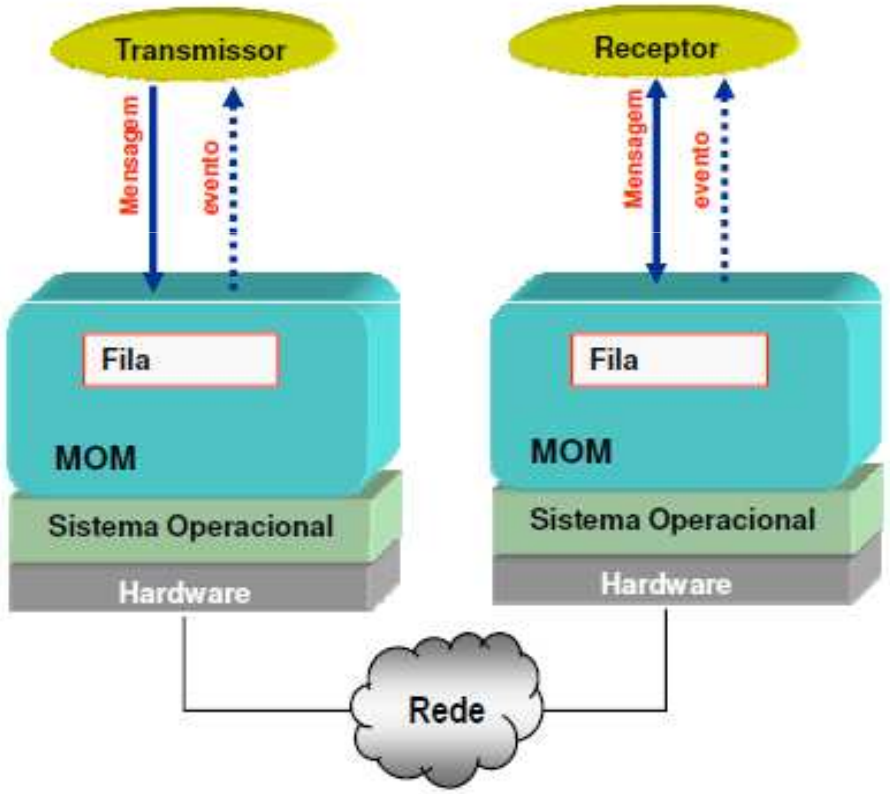
Middleware Orientado à Mensagem (MOM)

- Primitiva de interação é a passagem de mensagem.
- Comunicação assíncrona/em grupo naturalmente implementadas
- Comunicação indireta
  - remetente e receptor não precisam estar ativos ao mesmo tempo.

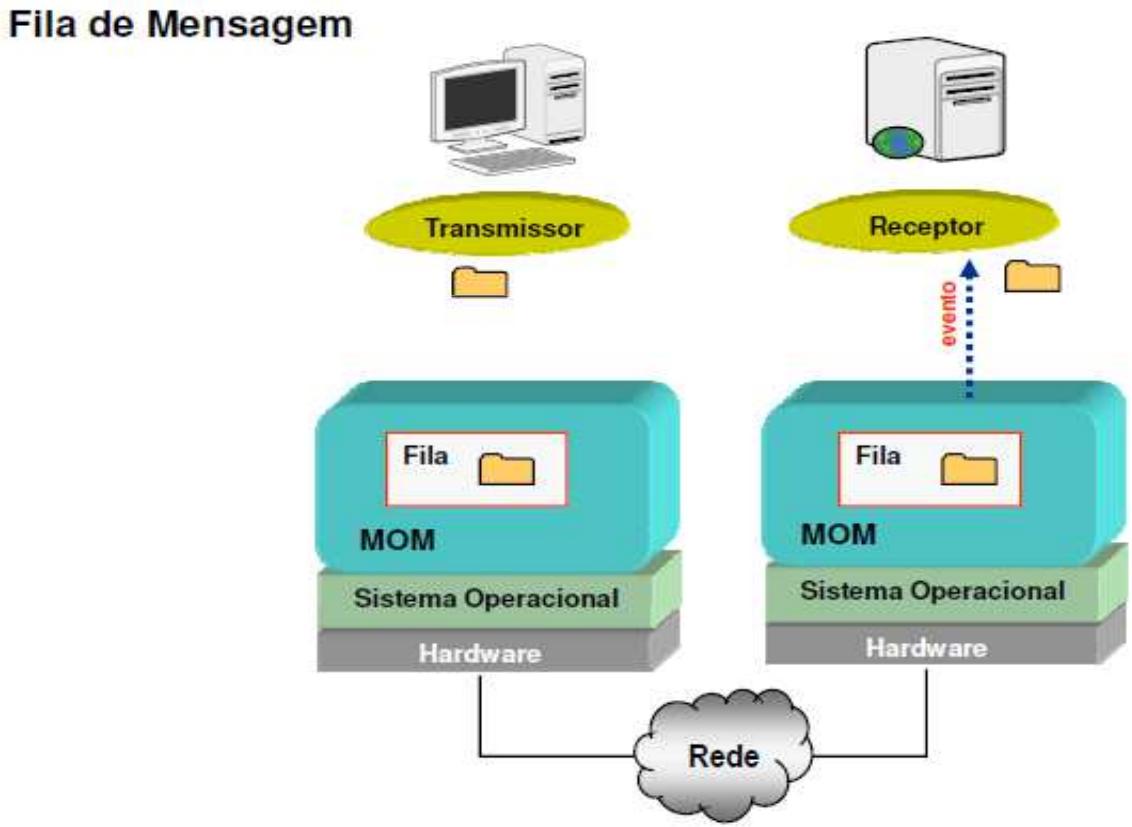


Middleware Orientado a Mensagem (MOM)

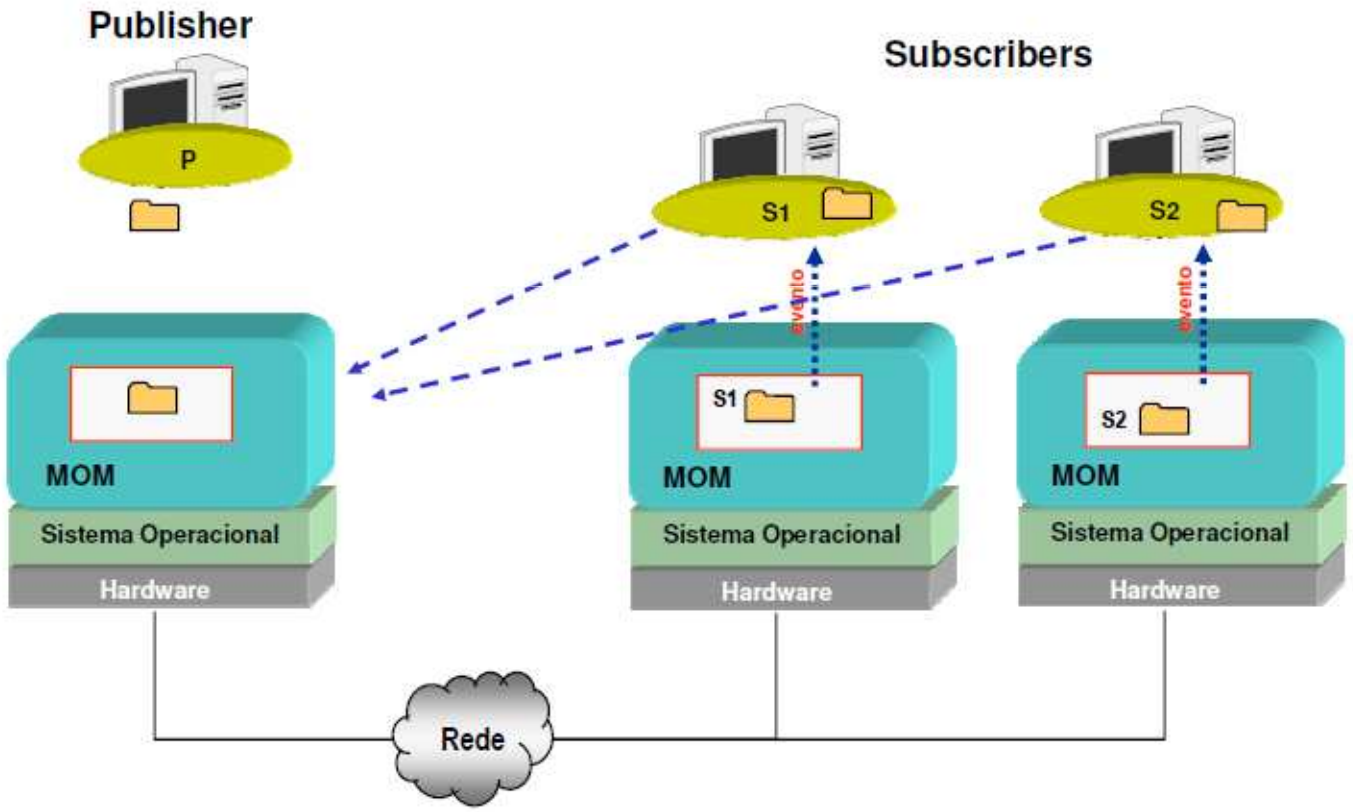
- Em geral, um remetente só tem a garantia de que, a certa altura, sua mensagem será inserida na fila do receptor.
- Nenhuma garantia é dada sobre quando o receptor vai ler a mensagem, pois isso é determinado pelo seu comportamento.
- Comunicação: 1-1, 1-N
- Exemplo: StackSync.



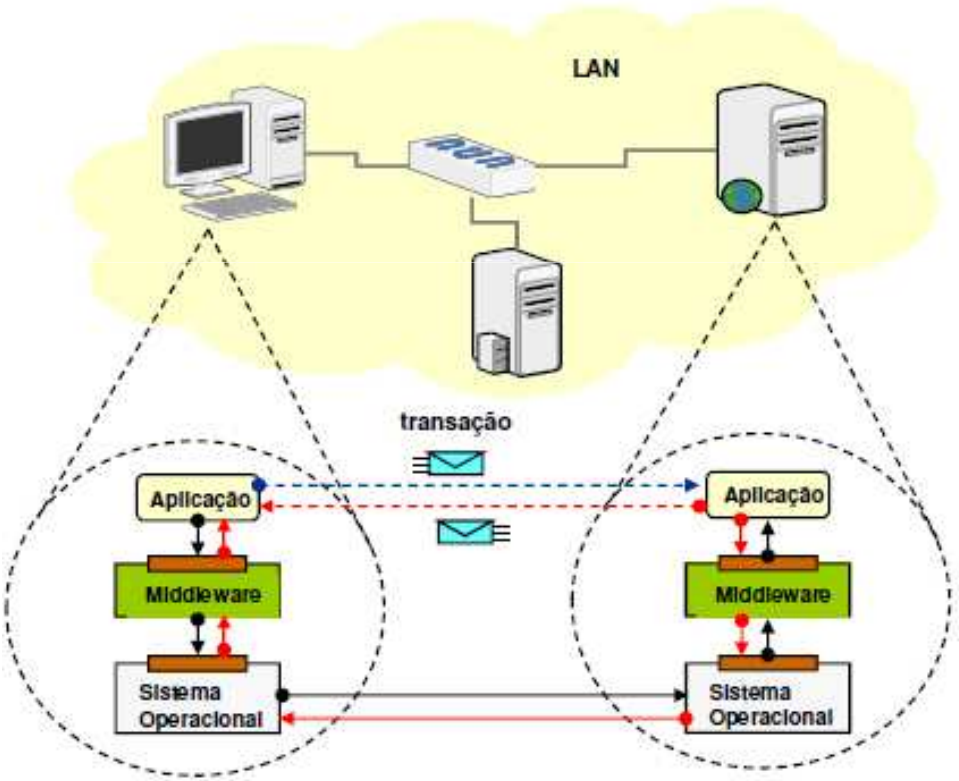
Middleware Orientado a Mensagem (MOM)



Middleware Orientado a Mensagem (MOM)

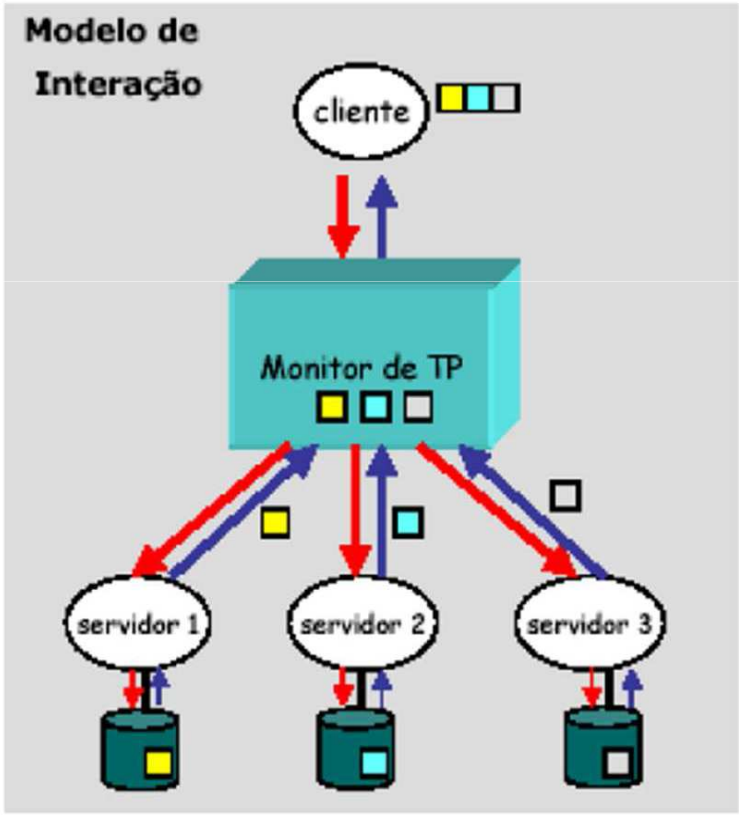


Middleware Orientado à Transação



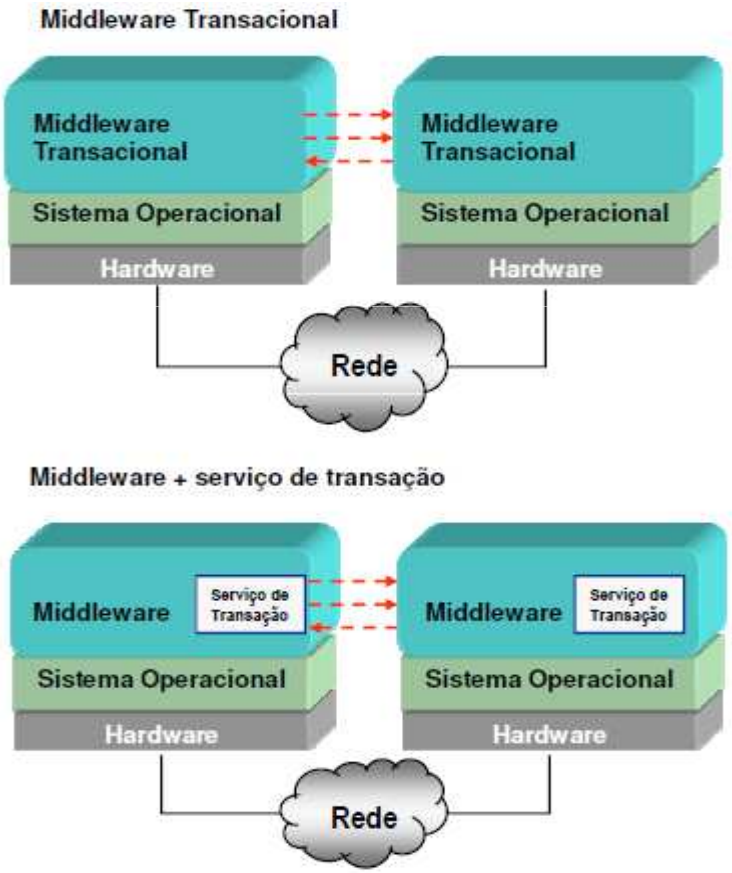
Middleware Orientado à Transação

- Também chamado de *Middleware Transacional.*
- Fornecem suporte (coordenação e sincronização) à execução de transações de acesso à bases de dados.
- Esconde a complexidade da implementação de transações em redes.



Middleware Orientado à Transação

- Primitiva de interação: transação
  - Chamada de procedimento remoto + controle de transações
- protocolo comumente usado: protocolo *two-phase commit*
- comunicação síncrona (1-1)



Middleware Orientado à Transação

- Esconde a complexidade da implementação de transações em redes.
- Tipicamente construído sobre outro *middleware.*
- Usado em sistemas de processamento de transações de alta escala (banco).
- Suporte a transações síncronas.
- EX.: CICS (IBM), Encina (DCE-based), OMG Object Transaction Service (CORBA-based), Microsoft Transaction Server (DCOM-based).

Alguns *Middlewares...*

- SUN RPC
- Java RMI
- OMG CORBA
- Microsoft COM/DCOM
- OSF DCE
- IBM CICS
- Sun ONE, etc...

Web Service

- É uma solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes.
- Interface que descreve uma coleção de operações acessíveis em uma rede através de mensagens XML padronizadas.
- Com essa tecnologia é possível que novas aplicações possam interagir com aquelas que já existem e que sistemas desenvolvidos em plataformas diferentes sejam compatíveis.
- Cada aplicação pode ter a sua própria "linguagem", que é traduzida para uma linguagem universal, o formato *XML*.

Web Service

- Geralmente, uma interface de *web services* consiste em um conjunto de operações que podem ser usadas por um cliente na Internet.
- Os *web services* são projetados para suportar computação distribuída na Internet.
  - Utilizando a tecnologia Web Service, uma aplicação pode invocar outra para efetuar tarefas simples ou complexas mesmo que as duas aplicações estejam em diferentes sistemas e escritas em linguagens diferentes.
- Eles são independentes de qualquer paradigma de programação em particular.

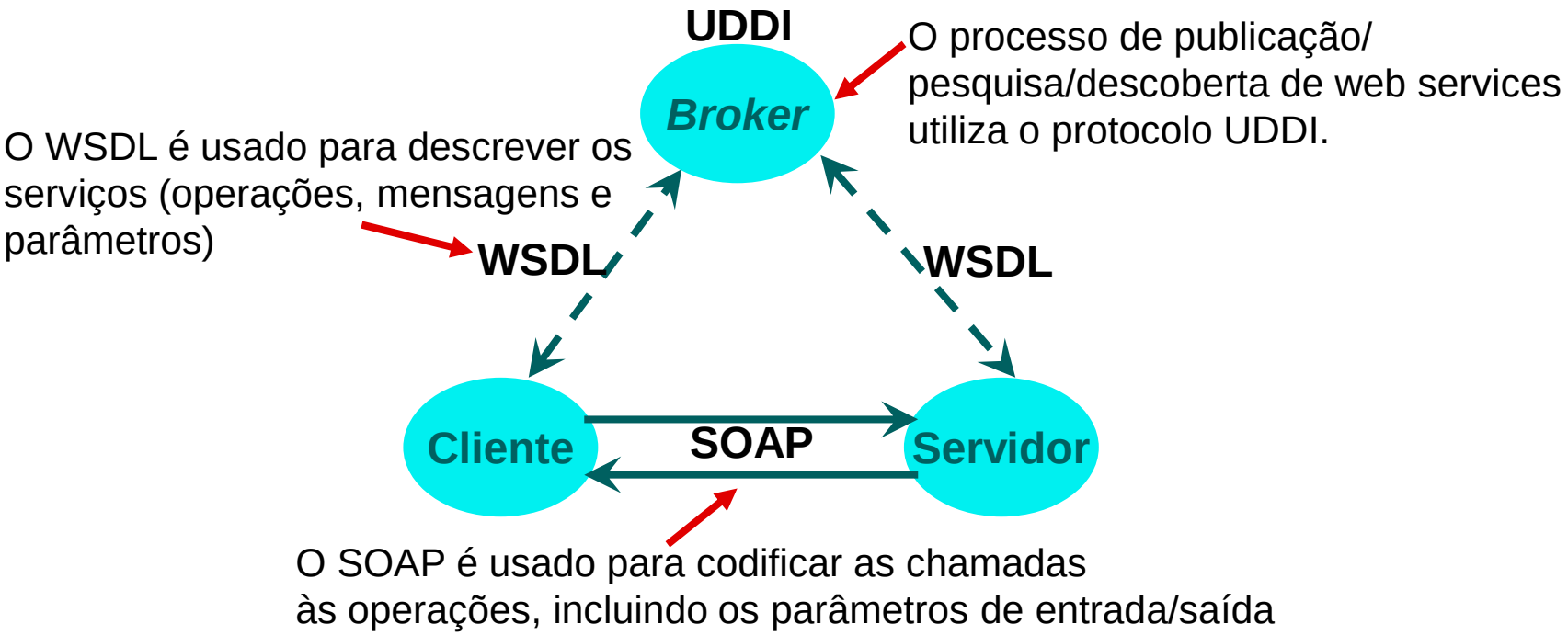
Web Service – Utilização

- Imagine um *site* de vendas pela Internet, que necessita validar o crédito do comprador antes de proceder com a venda
  - O sistema então acessa um serviço (Web Service) que cuida de todos os passos necessários à verificação de crédito: checa o histórico das compras efetuadas pelo consumidor na empresa, checa a situação de crédito do consumidor no sistema público, etc
  - O Web Service obtém esses dados e retorna a situação de crédito deste consumidor para o site.
- Um site que faz cotação de preços
  - Suponha que cada loja virtual possua seu próprio *Web Service*. Este *Web Service* deve receber o pedido para uma determinada cotação e enviar uma resposta contendo o preço do produto.

Padrões dos *Web Service*

- *Web Service* é baseado em padrões abertos já aceitos pelo mercado:
  - HTTP (*Hiper Text Transfer Protocol*)
  - UDDI (*Universal Description, Discovery and Integration*)
  - WSDL (*Web Services Description Language*)
  - SOAP (*Simple Object Access Control*)

Padrões dos *Web Services*



- O XML é usado para representação e estruturação dos dados nas mensagens recebidas/enviadas.

Tecnologias envolvidas

- *Simple Object Access Protocol* (SOAP)
  - Protocolo para troca de mensagens entre clientes e provedores de serviços baseado em XML.
  - Sua especificação define um *framework* que provê maneiras para se construir mensagens que podem trafegar através de diversos protocolos, o qual foi especificado para ser independente de qualquer modelo de programação ou outra implementação específica.
- *Web Services Description Language* (WSDL)
  - Define os serviços externos ou as interfaces que são oferecidos por uma determinada aplicação, independente da sua plataforma ou linguagem de programação.
  - Também é usado para a validação das chamadas dos métodos.
- *Universal Description, Discovery and Integration* (UDDI)
  - ***Binder*** (vinculador – "serviço de nomes") para Web Services.
  - Permitem a publicação e a descoberta de *Web Service*.

SOAP
(*Simple Object Access Protocol*)

- Baseia-se numa invocação remota de um método, e para tal necessita especificar o endereço do componente, o nome do método e os argumentos para esse método.
- Estes dados são formatados em XML com determinadas regras e enviados normalmente por HTTP para esse componente.
- Assim, tem-se a interoperabilidade e intercomunicação entre diferentes sistemas, através da utilização da linguagem XML e do mecanismo de transporte HTTP ou outro como, por exemplo, SMTP.

SOAP (*Simple Object Access Protocol*)

- O SOAP providencia o transporte de dados para os Web Services.
- Em relação a Web, o SOAP é um protocolo de RPC que funciona sobre HTTP (ou SMTP, ou outro).
- Assim, ao invés de usar HTTP para pedir uma página HTML para ser visualizada num *browser*, o SOAP envia uma mensagem de XML através do pedido HTTP e recebe uma resposta, se existir, através da resposta do HTTP.

WSDL
(Web Services Description Language)

- O WSDL é uma especificação desenvolvida para descrever os *Web Services* segundo um formato XML.
- Usado para descrever o serviço remoto disponibilizado no repositório.
- Uma descrição WSDL inclui:
  - Tipos de dados
  - Formato das mensagens
  - Operações suportadas pelo web service
  - Endereço de rede do web service (URL)

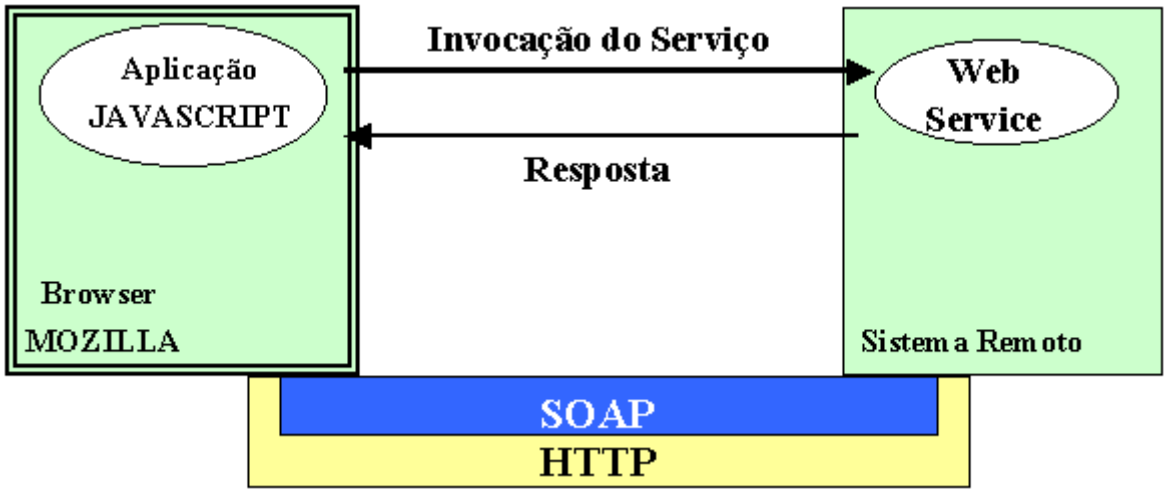
UDDI- (*Universal Description Discovery and Integration*)

- É um serviço de diretório para uso com serviços web, que permite cadastrar e localizar os web services.
- Assim, as descrições de serviço WSDL podem ser pesquisadas pelo nome ou pelo atributo. Elas também podem ser acessadas diretamente por meio de seus URLs.
- É baseado em dados replicados e armazenados em registros.
- Permite que empresas registrem seus serviços e possam interagir com empresas interessadas.

UDDI- (*Universal Description Discovery and Integration*)

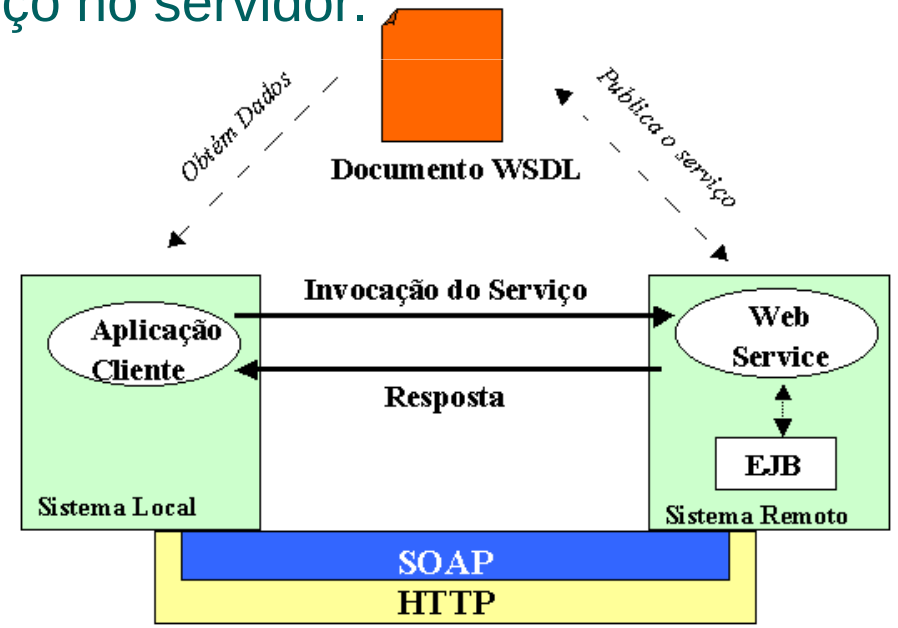
Cenário1: Acesso a Web Services através de Browser

- Uma aplicação rodando no Mozilla, ou no Netscape, ou no IE podem acessar um Web Service, usando chamadas em SOAP, diretamente do navegador, sem a necessidade de fazer uma chamada a uma aplicação no servidor.



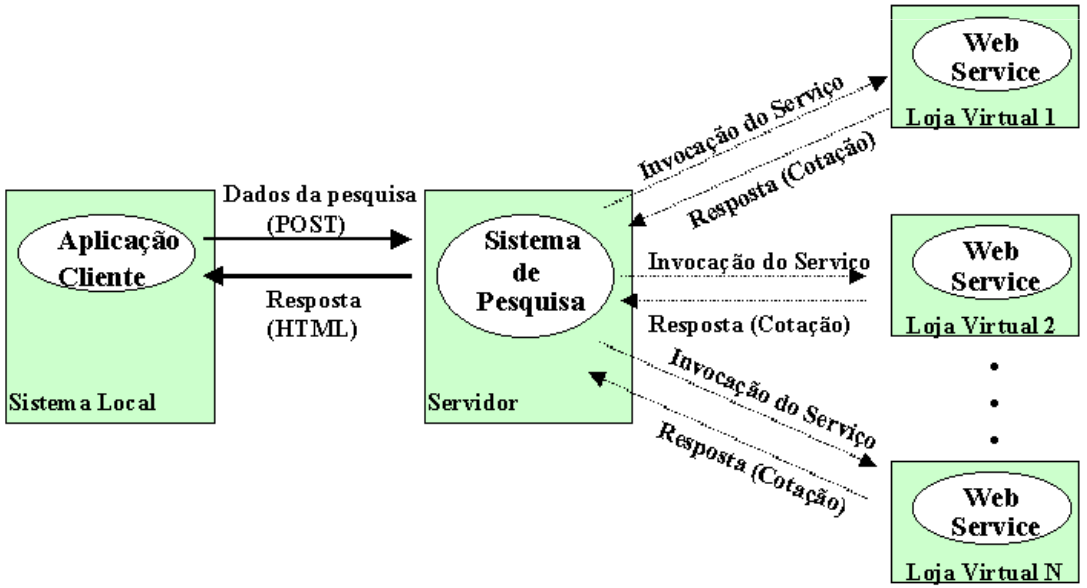
Cenário 2: Cliente acessando Web Services no servidor

- Neste novo modelo, em vez de existir um *browser* realizando chamadas SOAP, tem-se uma aplicação cliente (definida em JAVA, C++, ou outra linguagem) enviando essas chamadas para um serviço no servidor.



Cenário 3: Acessando Web Services no lado servidor

- Neste último cenário tem-se um aplicação cliente acessando uma aplicação no servidor. Esta, por sua vez, faz chamadas a diversos Web Services em diferentes máquinas, processa as respostas destes serviços e, por fim, envia uma resposta ao cliente.



Fontes Bibliográficas

G. Coulouris, J. Dollimore, T. Kindberg. *Sistemas Distribuídos – Conceitos e Projetos*. 4○ edição. Bookman, 2007.

A. Tanenbaum, M. Steen. Sistemas Distribuídos – Princípios e Paradigmas. 2○ edição. Prentice Hall, 2007.