ACH2147 - Desenvolvimento de Sistemas de Informação Distribuídos

Aula 09 – Comunicação

Norton Trevisan Roman

9 de maio de 2022

Comunicação

- Fundamentos
  - Protocolos em Camadas
  - Tipos de Comunicação
- Remote Procedure Call

Comunicação

- **Fundamentos**
  - **Protocolos em Camadas**
  - Tipos de Comunicação
- Remote Procedure Call

Fundamentos

Comunicação entre sistemas distribuídos

- SDs não compartilham memória
  - Toda comunicação se baseia em mensagens de baixo nível
- Se o processo P deseja se comunicar com Q
  - Ele constrói a mensagem em seu próprio espaço de endereçamento
  - Faz então uma chamada ao sistema, para que o SO envie a mensagem pela rede a Q

Fundamentos

Protocolos

- Tratam-se de regras padrão que processos precisam seguir para se comunicar
  - Regras que governam o formato, conteúdo, e significado das mensagens

Fundamentos

Protocolos

- Tratam-se de regras padrão que processos precisam seguir para se comunicar
  - Regras que governam o formato, conteúdo, e significado das mensagens
- Protocolos fornecem um **serviço de comunicação**
  - Serviço orientado à conexão: antes de trocar dados, emissor e receptor primeiro estabelecem explicitamente uma conexão, liberando-a quando terminarem (ex: telefone)
  - Serviço sem conexão: não precisa do estabelecimento ou término de conexão. O emissor transmite a mensagem quando estiver pronto, sem aviso prévio (ex: e-mail)

O modelo OSI

- *Open Systems Interconnection Reference Model*
  - Modelo de referência ISO OSI

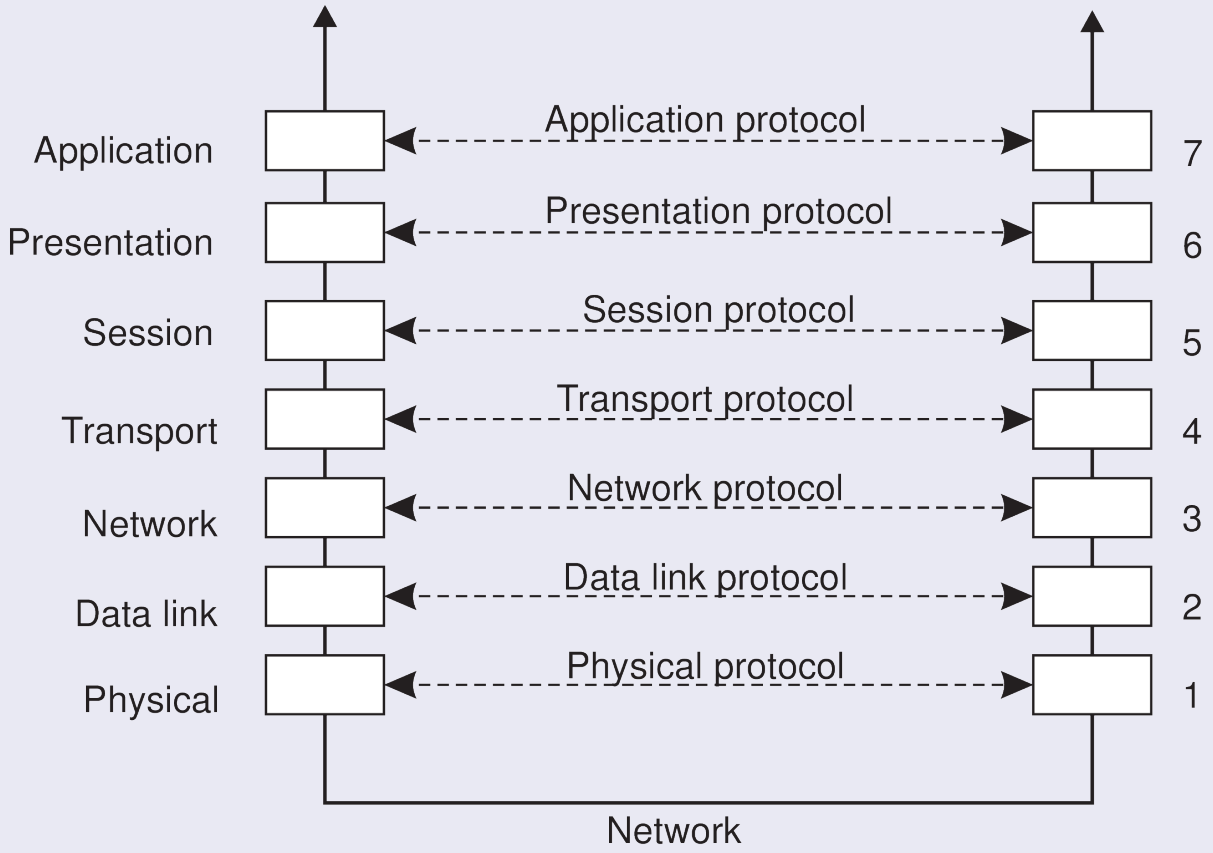
Fundamentos – Protocolos em camadas

O modelo OSI

- *Open Systems Interconnection Reference Model*
  - Modelo de referência ISO OSI
- Projetado para permitir que sistemas abertos se comuniquem
  - Um sistema aberto é um sistema que está preparado para se comunicar com qualquer outro sistema aberto, através de protocolos de comunicação padrão

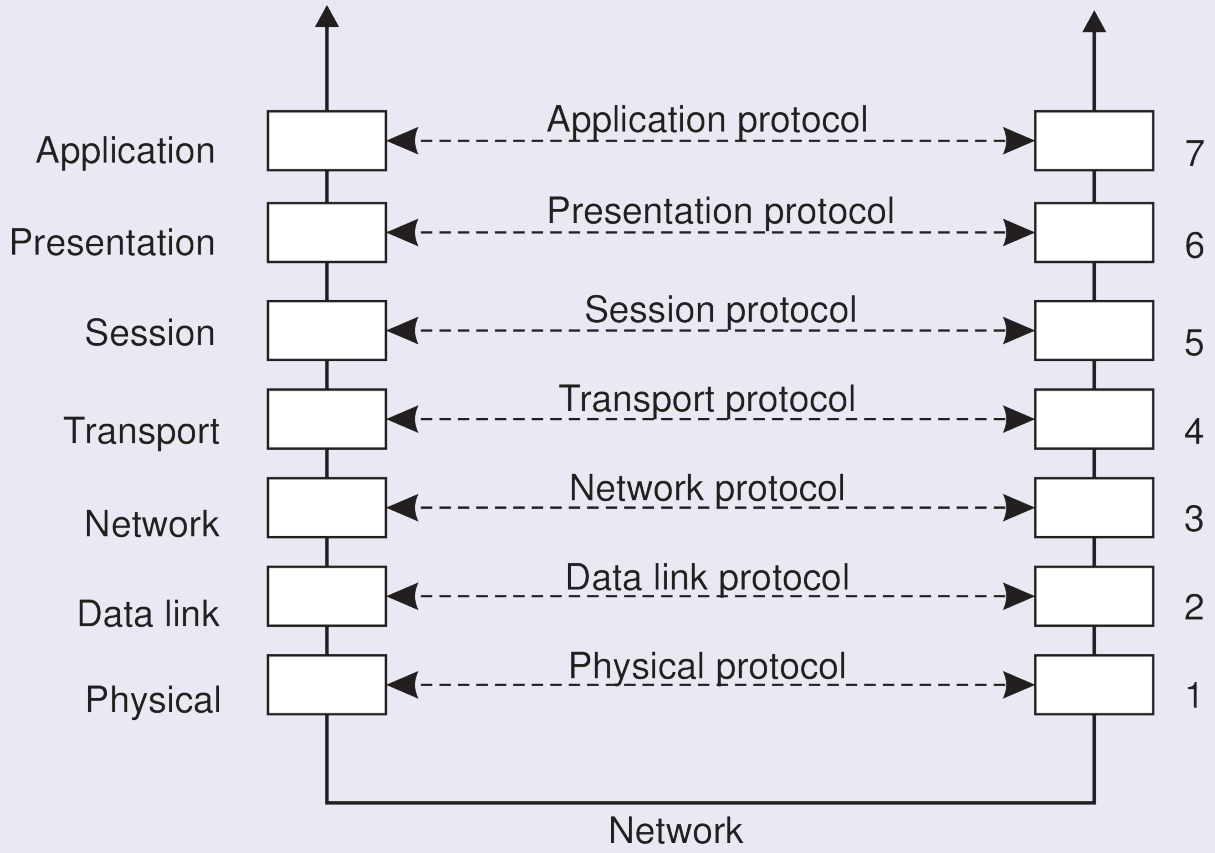
O modelo OSI

No modelo OSI, a comunicação se divide em 7 camadas



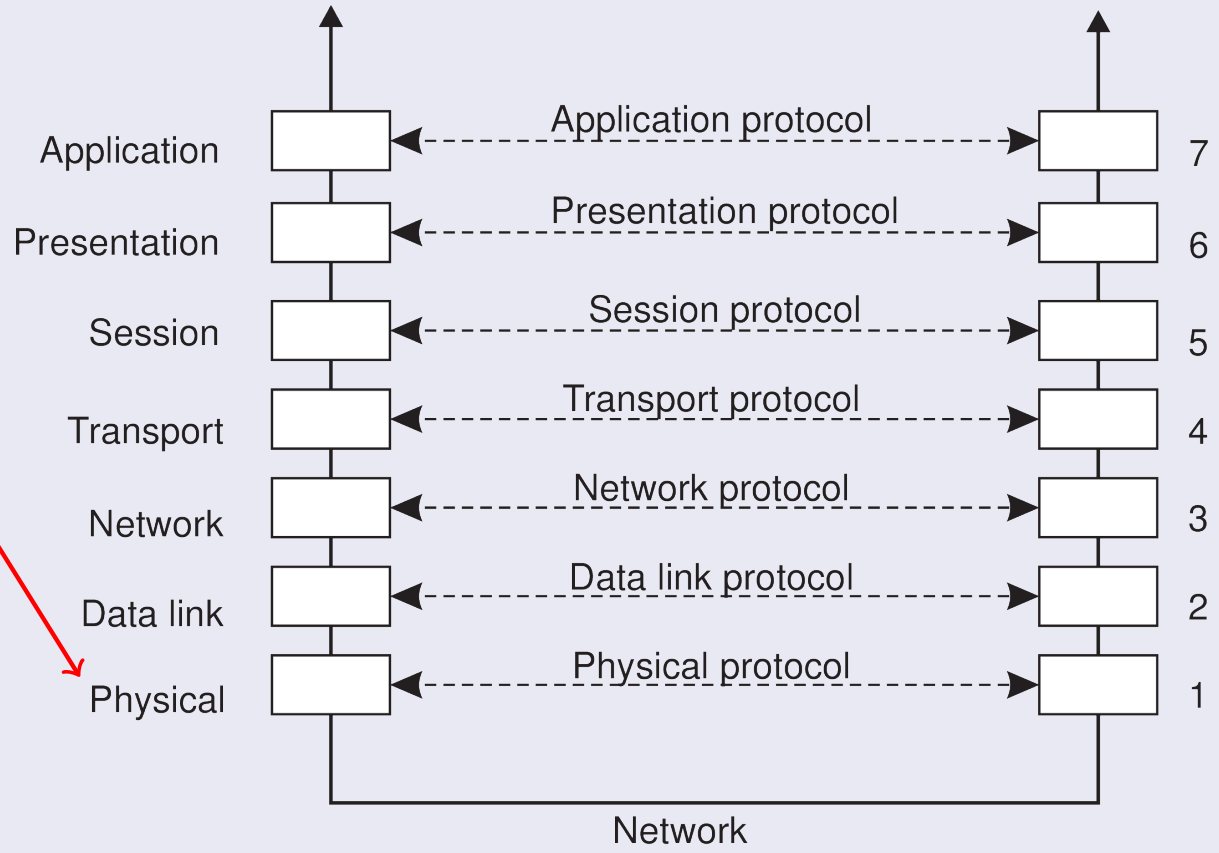
O modelo OSI

Cada camada oferece um ou mais serviços de comunicação específicos à camada acima, por meio de uma interface



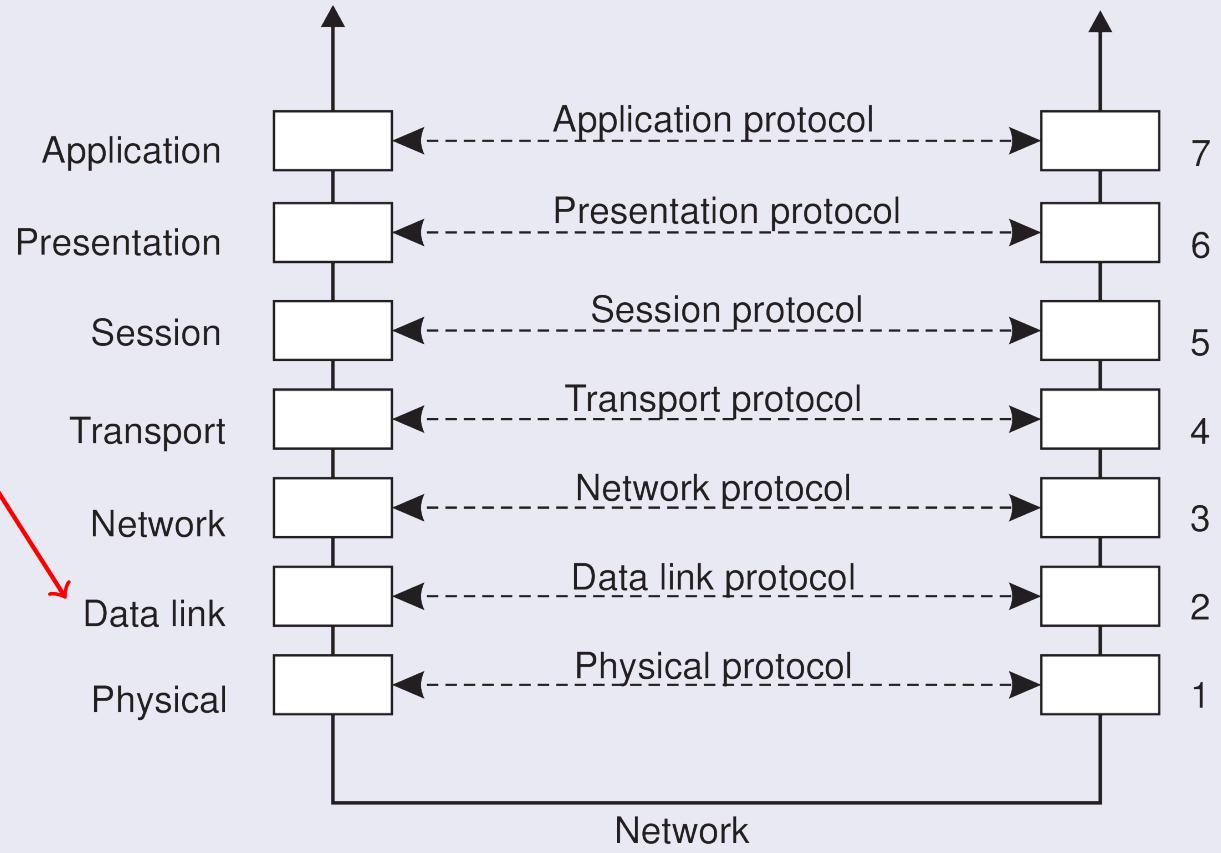
O modelo OSI

Camada física: como transmitir os 0s e 1s (quantos volts usar para cada, quantos bits/segundo enviar etc.). Apenas envia bits



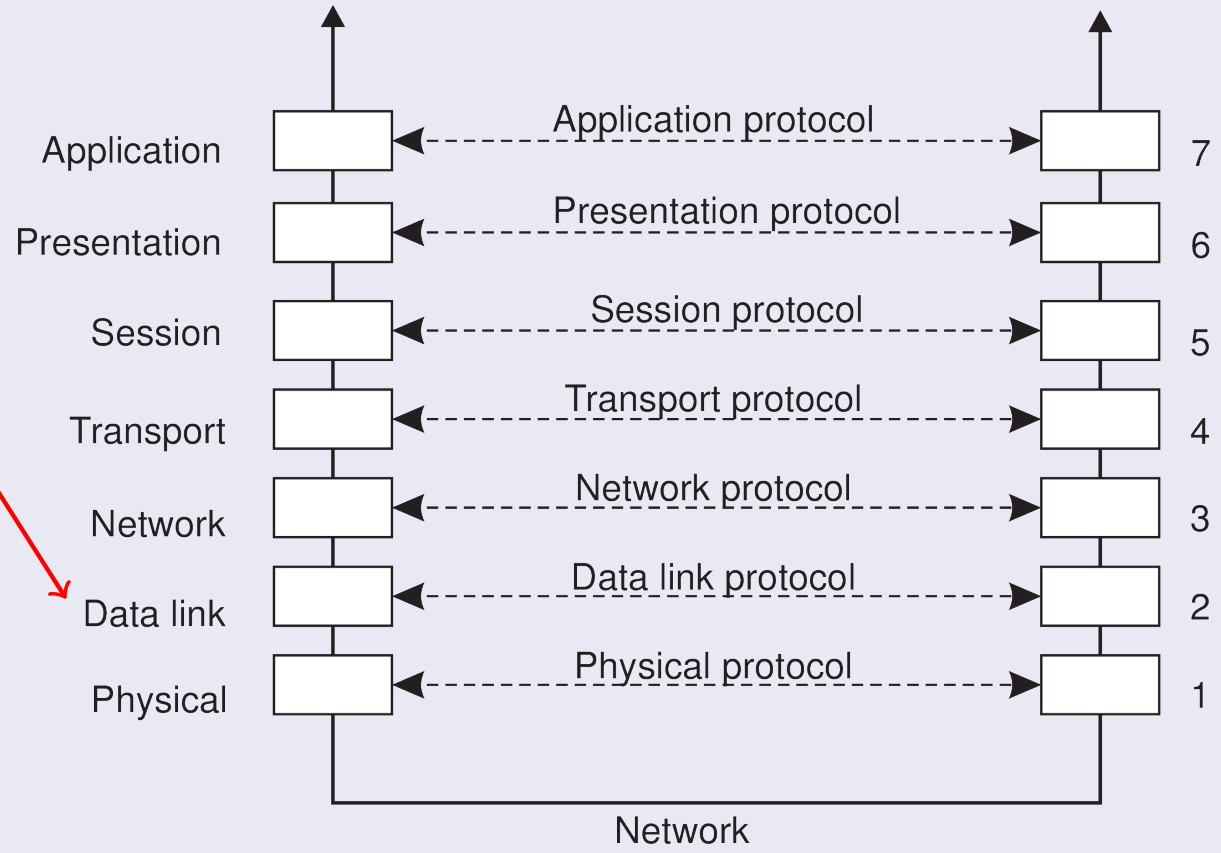
O modelo OSI

Camada de enlace: agrupa os bits em unidades (*frames*), verificando se cada *frame* é corretamente recebida



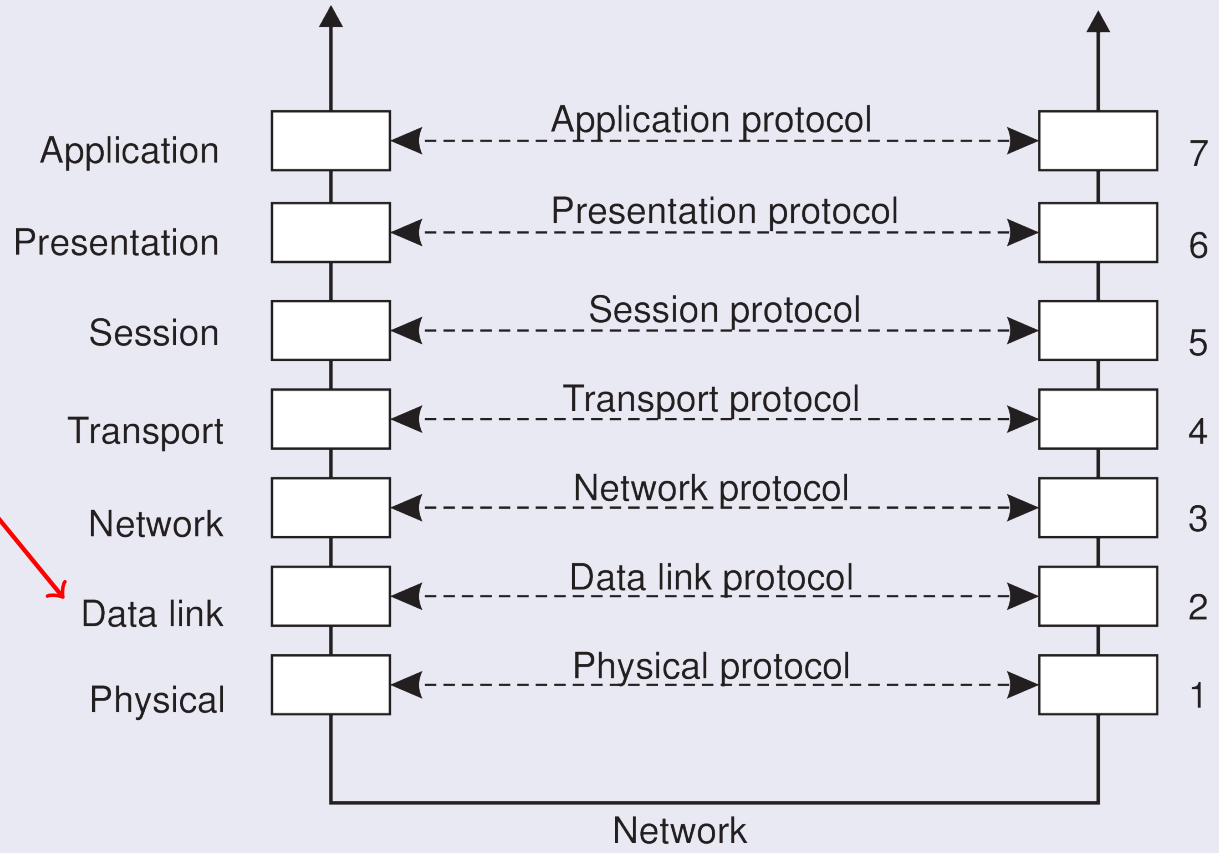
O modelo OSI

Faz isso colocando um conjunto de bits no final e início de cada *frame*, para marcá-los, além de um *checksum*, para detecção de erros



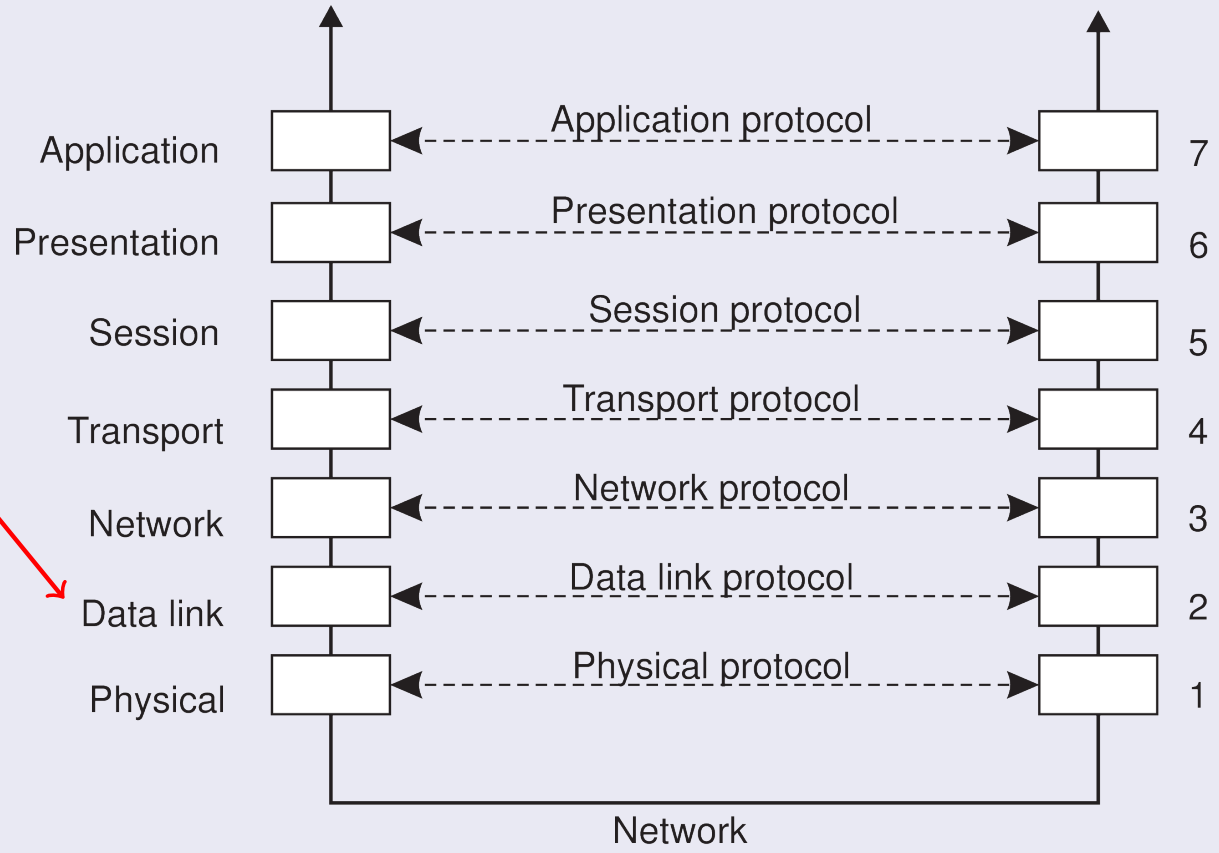
O modelo OSI

Quando uma *frame* chega, o receptor calcula o *checksum* e compara ao que está na *frame*. Se eles não batem, ele pede ao emissor para retransmitir a *frame*



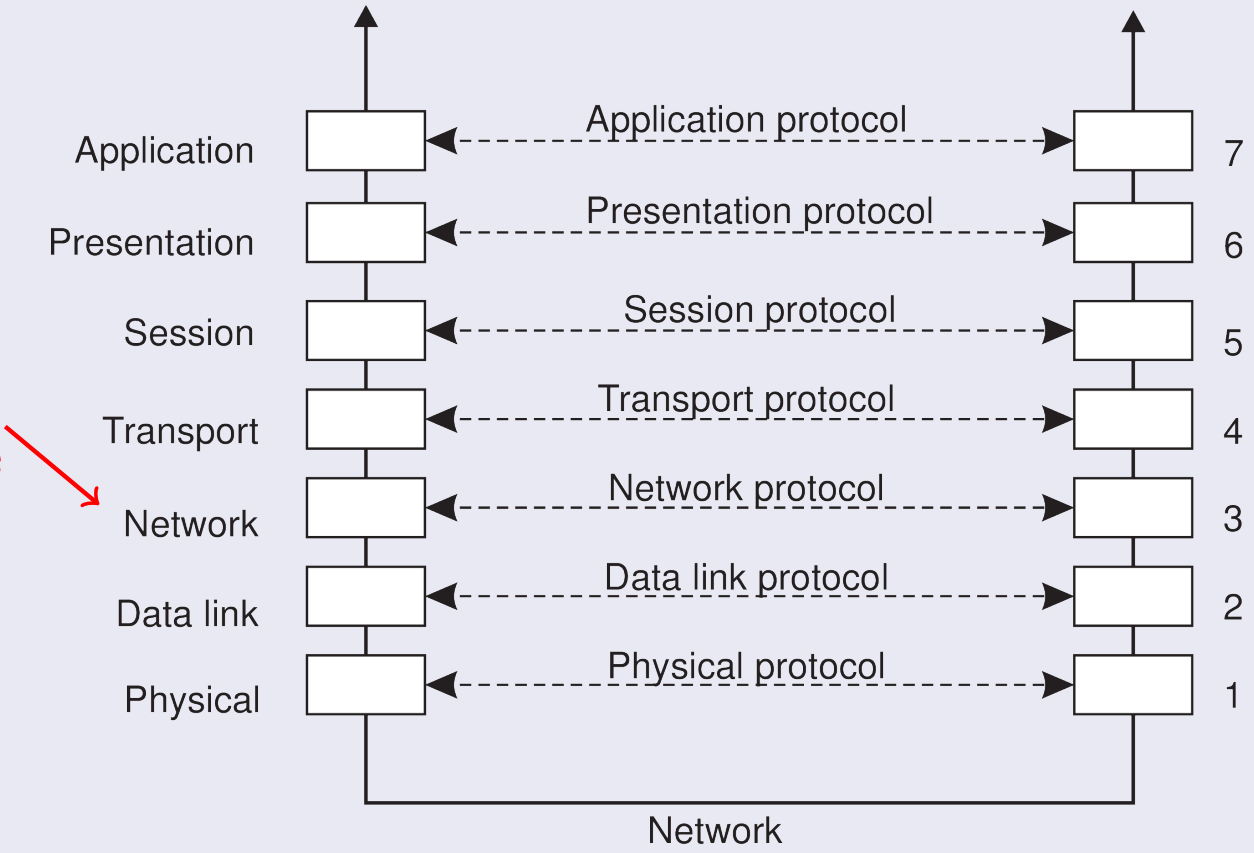
O modelo OSI

Frames recebem números sequenciais em seu cabeçalho, para identificação



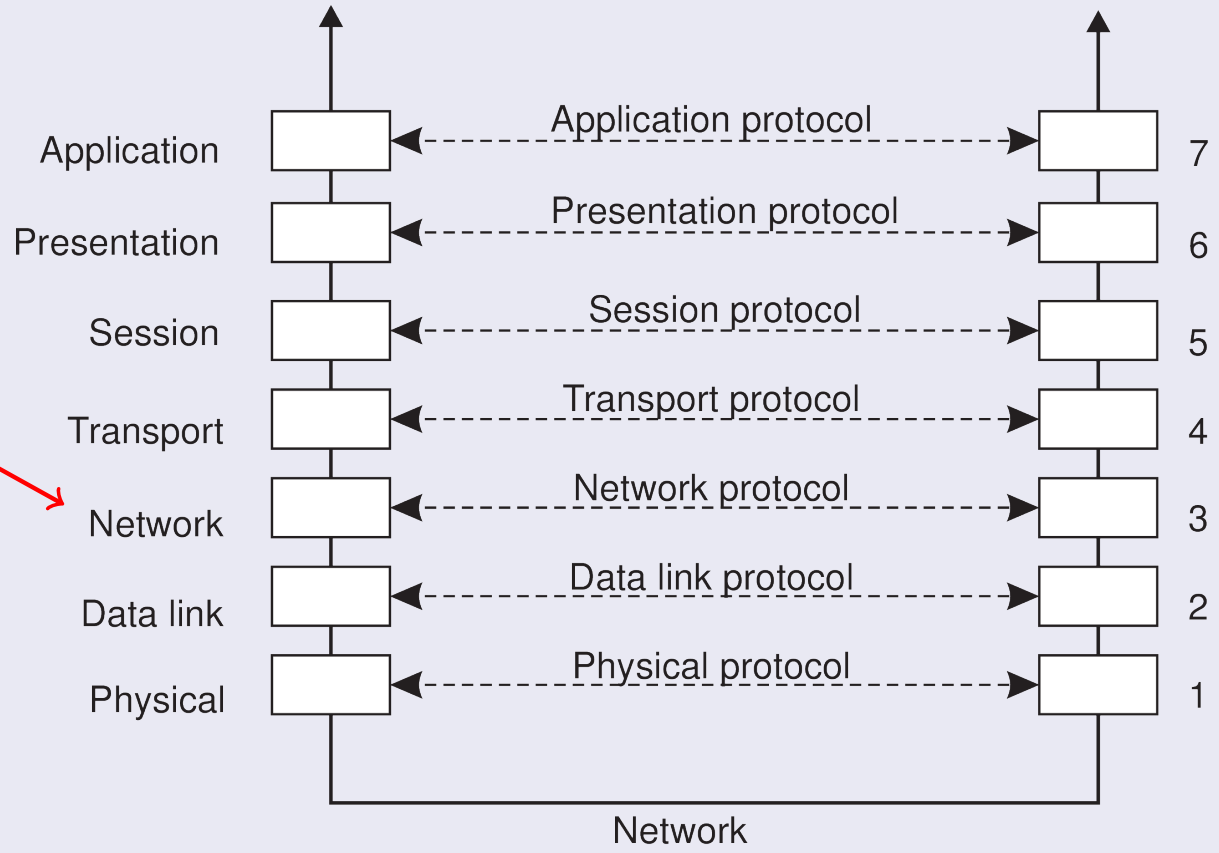
O modelo OSI

Camada de rede: responsável pelo roteamento (como escolher o melhor caminho entre emissor e receptor). Ex: pacotes IP



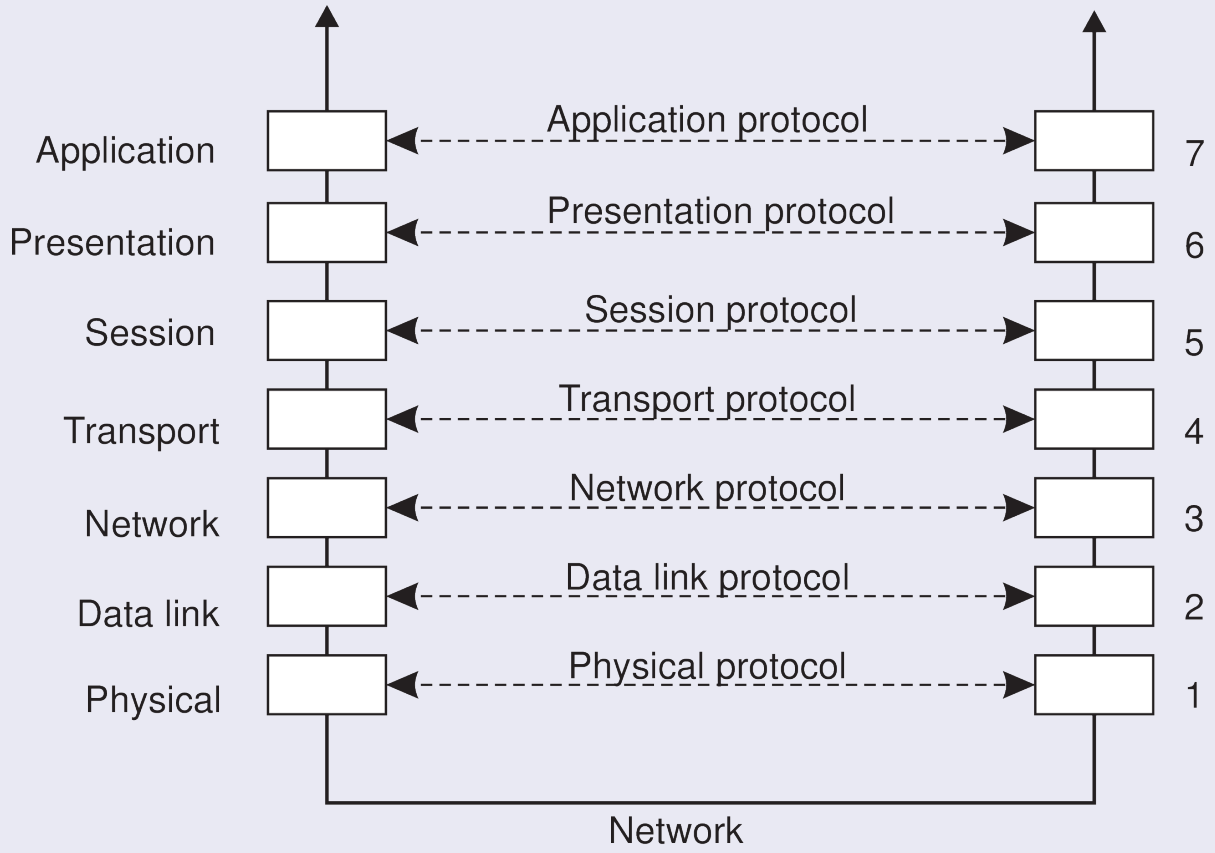
O modelo OSI

Em muitos sistemas distribuídos, a interface de mais baixo nível é a interface de rede



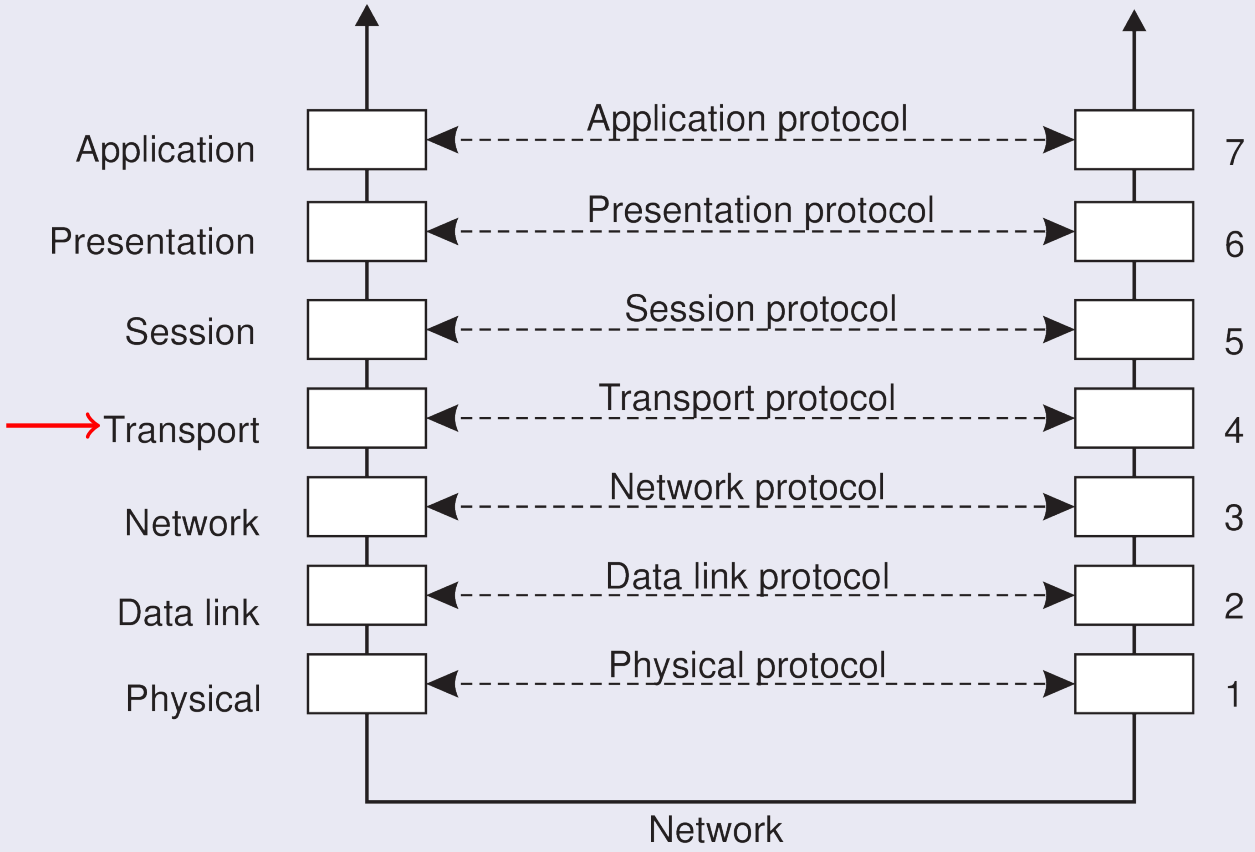
O modelo OSI

Juntas, essas 3 camadas de baixo nível implementam as funções básicas de uma rede



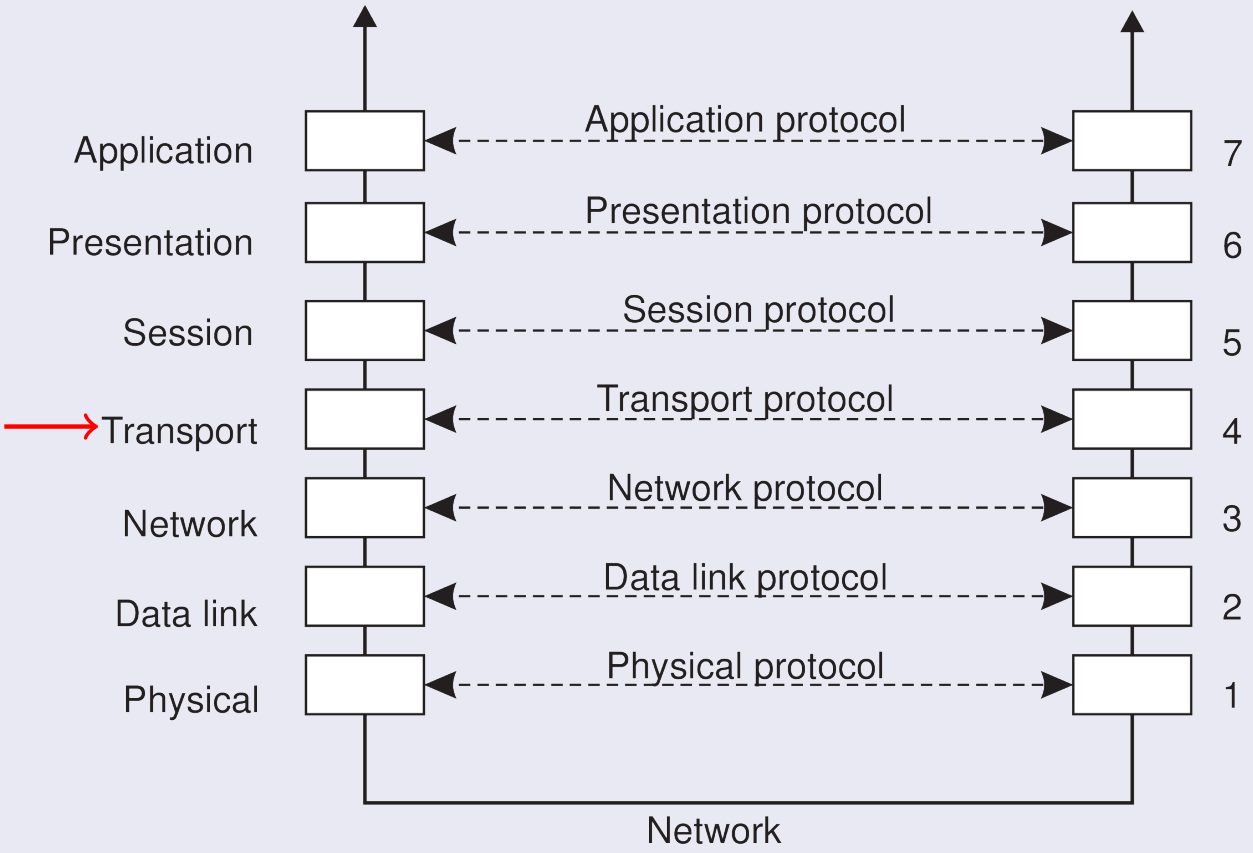
O modelo OSI

Camada de transporte: serviços para estabelecimento de comunicação confiável (entrega da mensagem sem perda). Ex: TCP, UDP



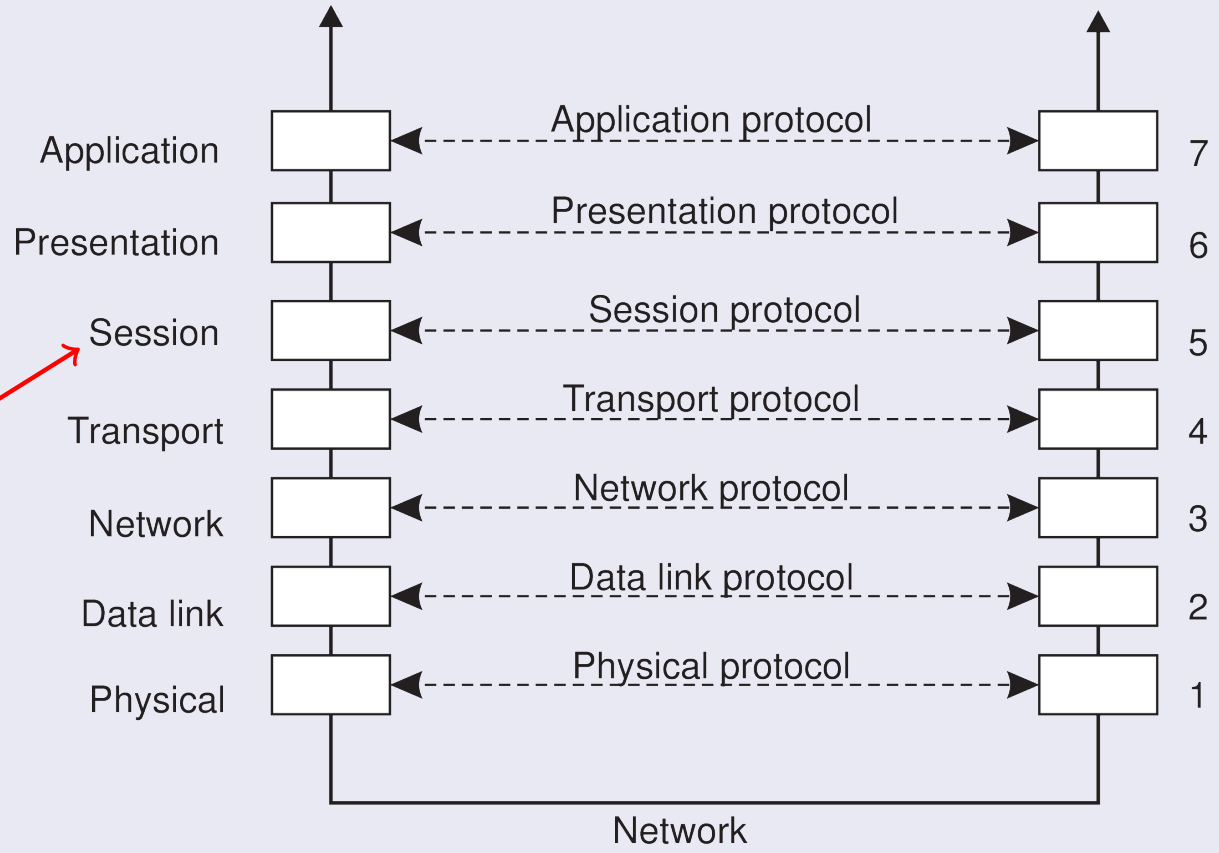
O modelo OSI

Quebra a mensagem da camada de aplicação em pedaços para transmissão, numerando-os e então os enviando



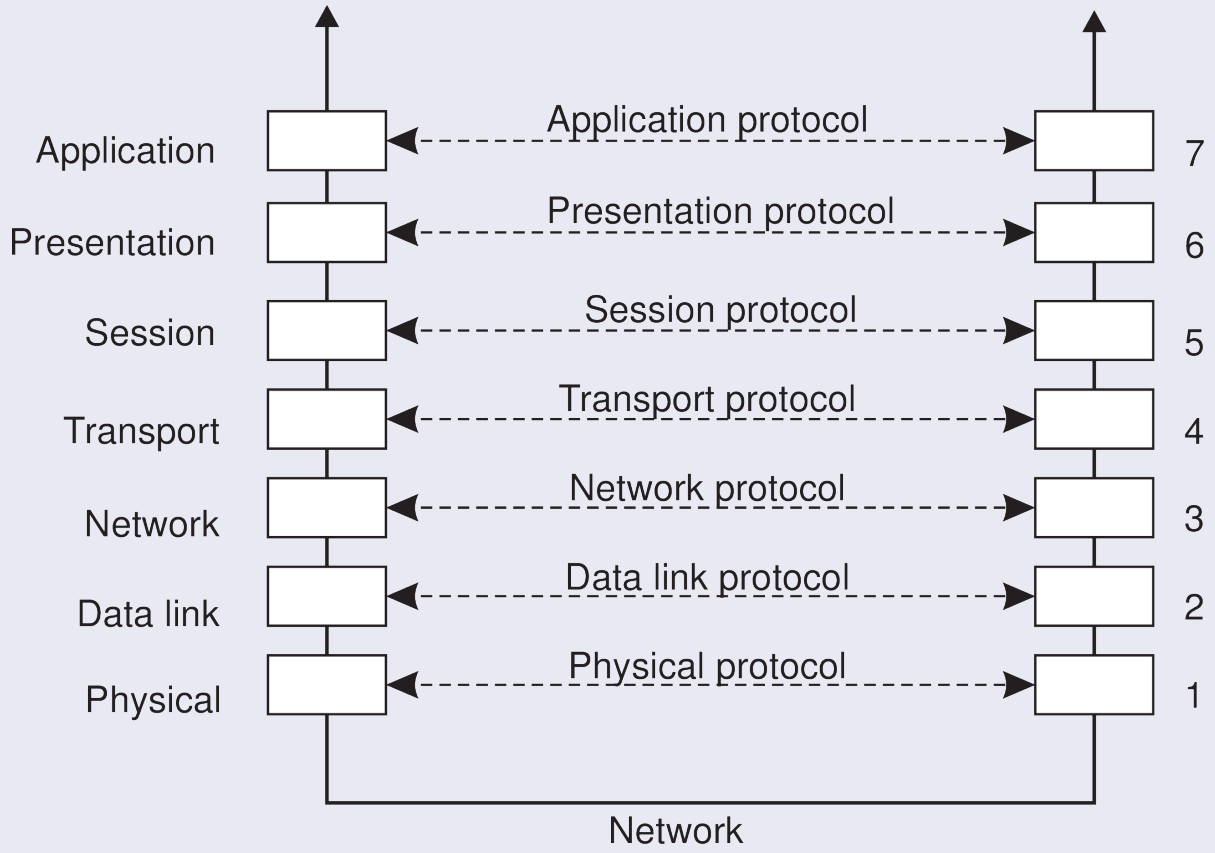
O modelo OSI

Camada de seção: controle de diálogo, identificando qual parte está falando, e fornecendo sincronização. Sequer presente na suíte do TCP/IP



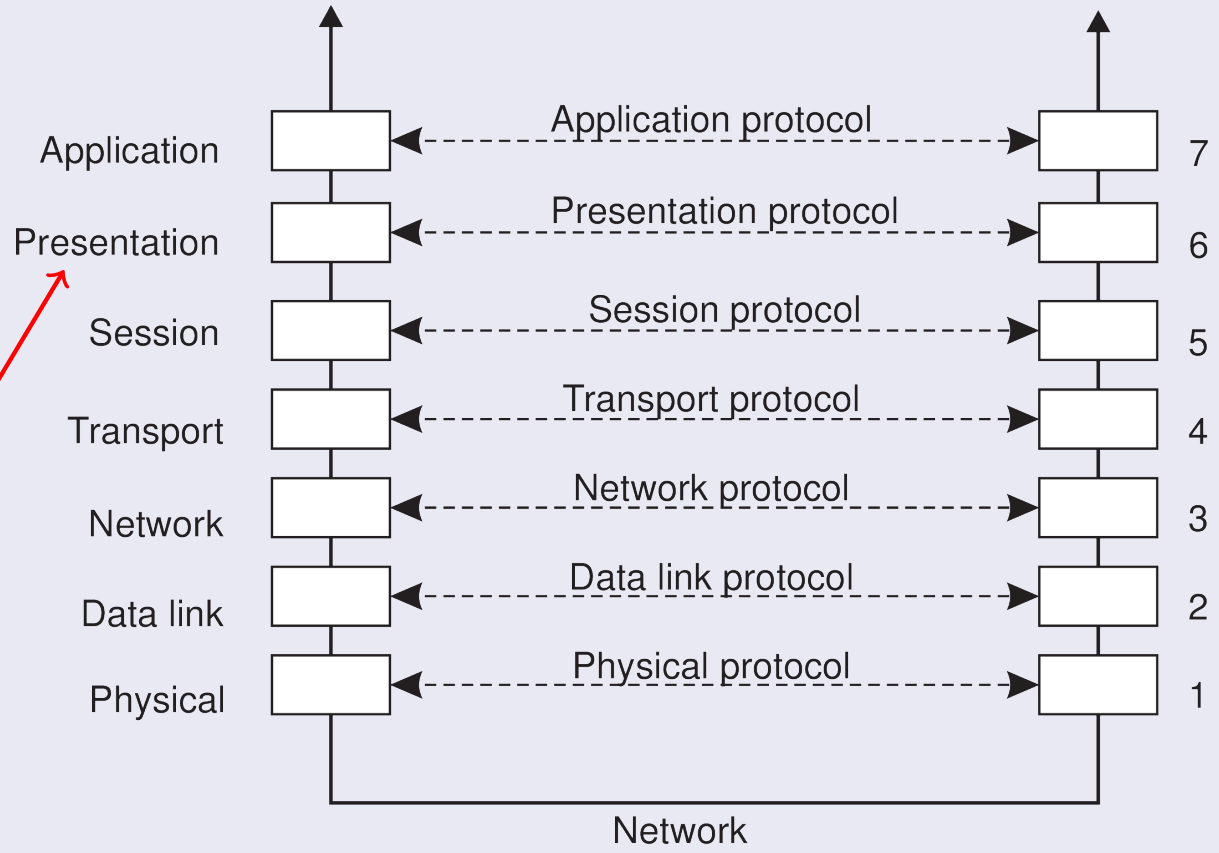
O modelo OSI

Uma **suíte de protocolo** (ou **pilha de protocolo**) é a coleção de protocolos usadas em um sistema em particular



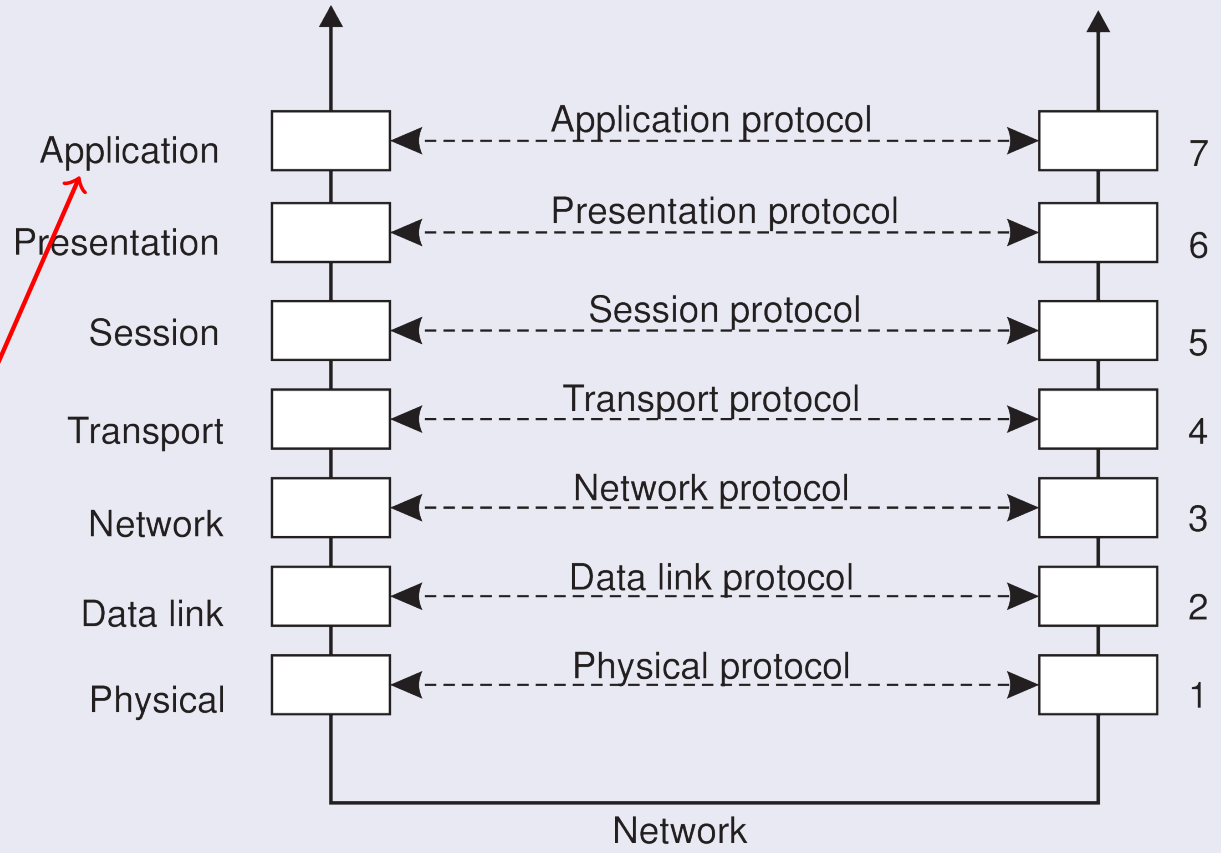
O modelo OSI

Camada de apresentação: trata do significado dos bits, definindo registros e seus campos, e permitindo que o emissor notifique o receptor do formato de cada registro



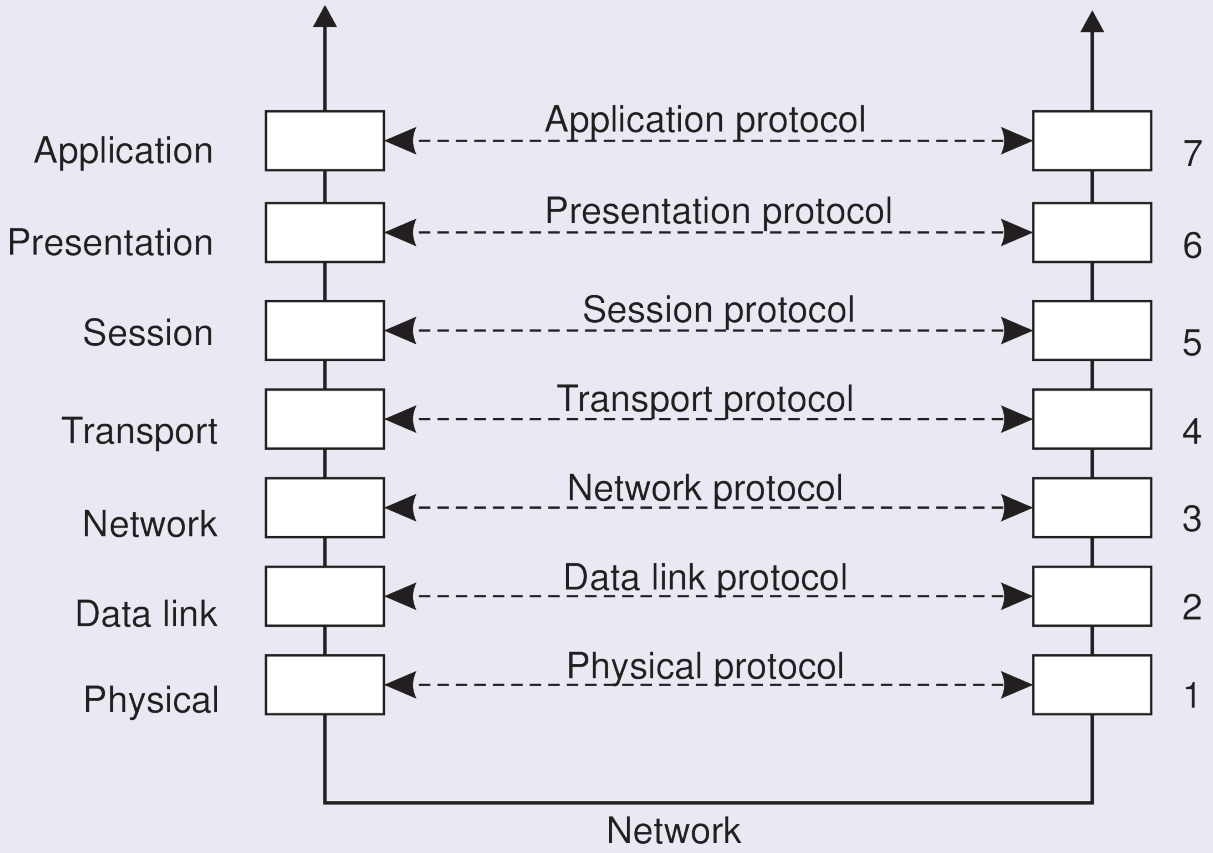
O modelo OSI

Camada de aplicação: tornou-se o lugar para aplicações e protocolos que não se encaixam nas camadas abaixo (email, transferência de arquivos etc.)



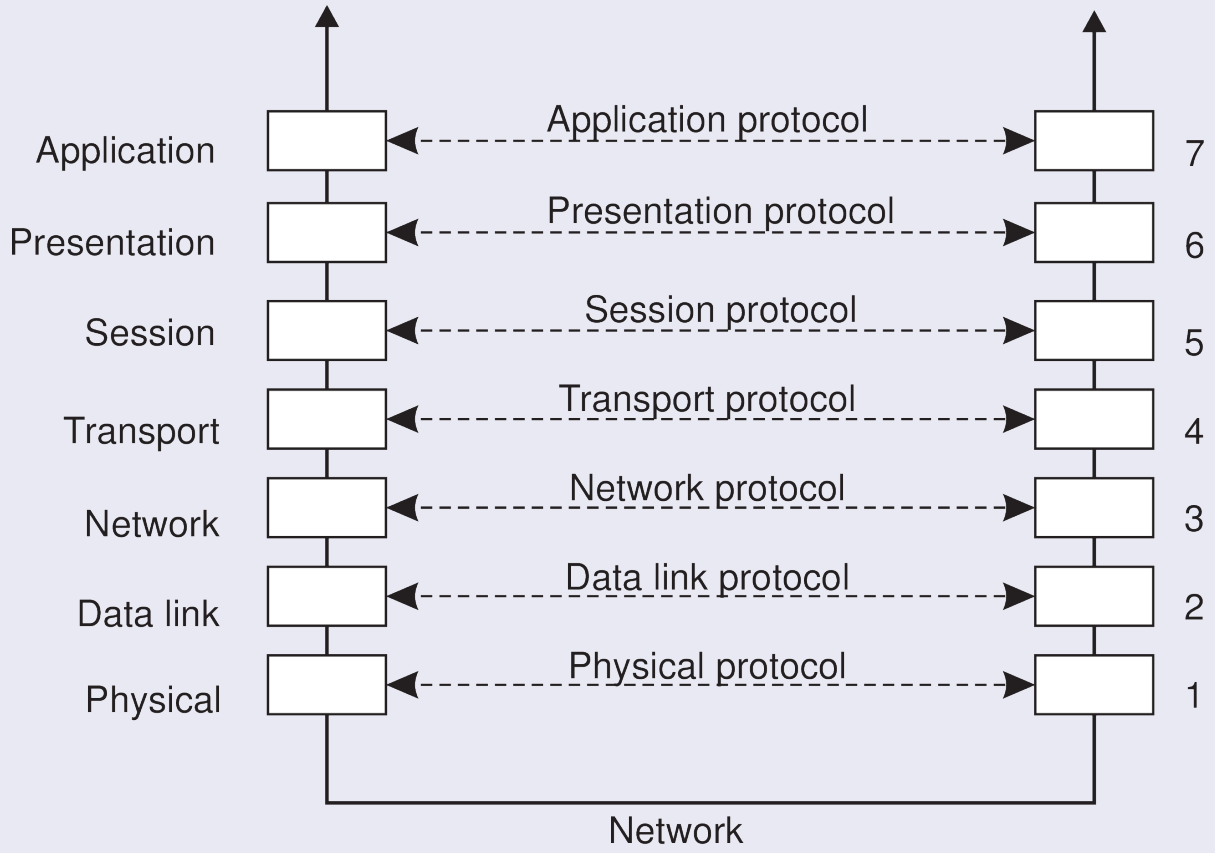
O modelo OSI

Acima da camada de transporte, o OSI define as 3 camadas de alto nível. Na prática, somente a camada de aplicação é usada



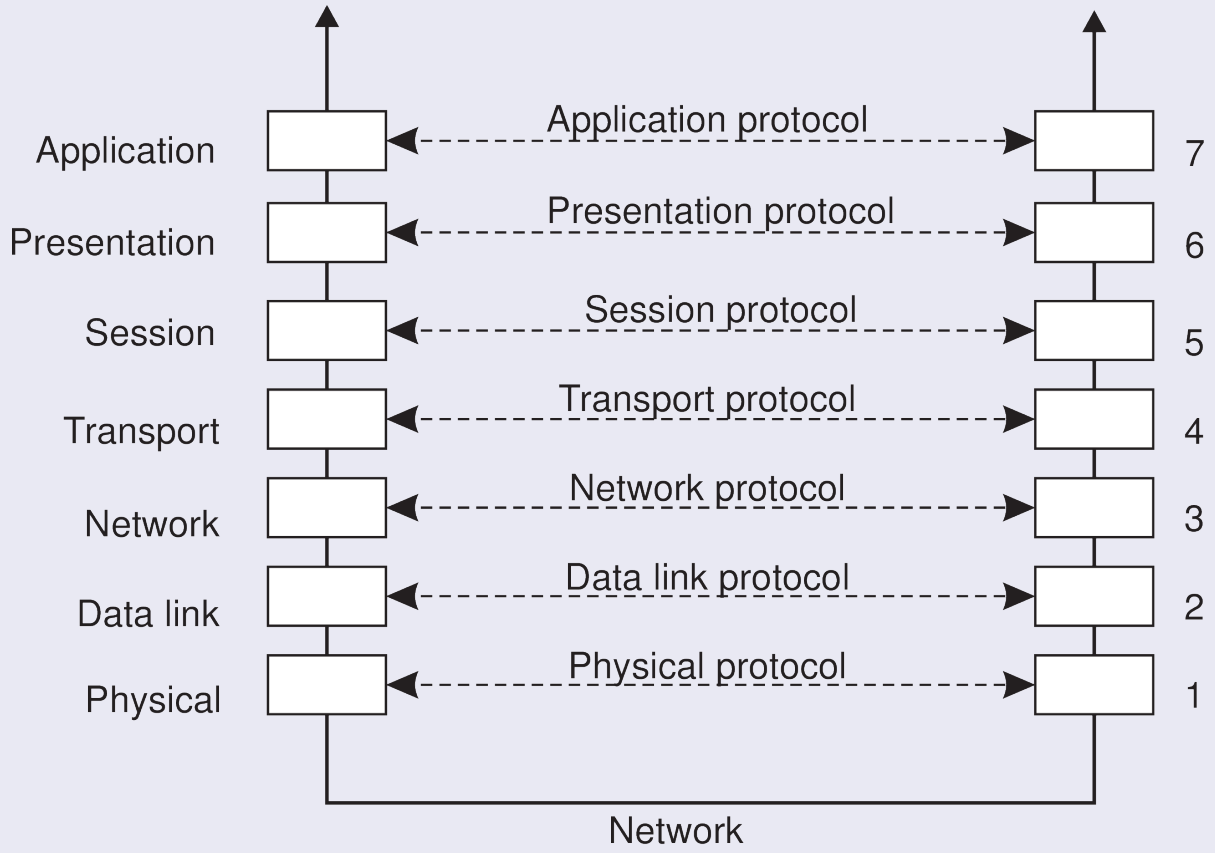
O modelo OSI

E tudo acima da camada de transporte é agrupado

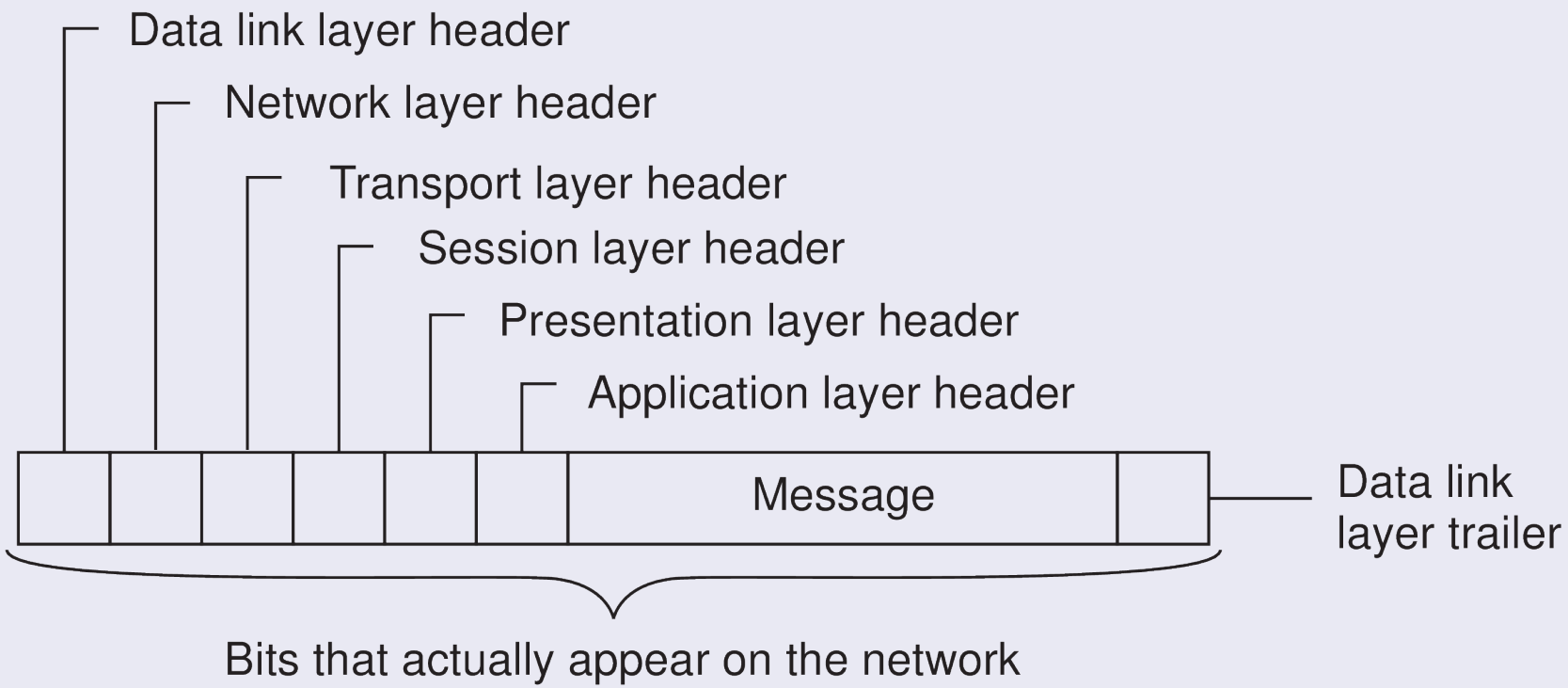


O modelo OSI

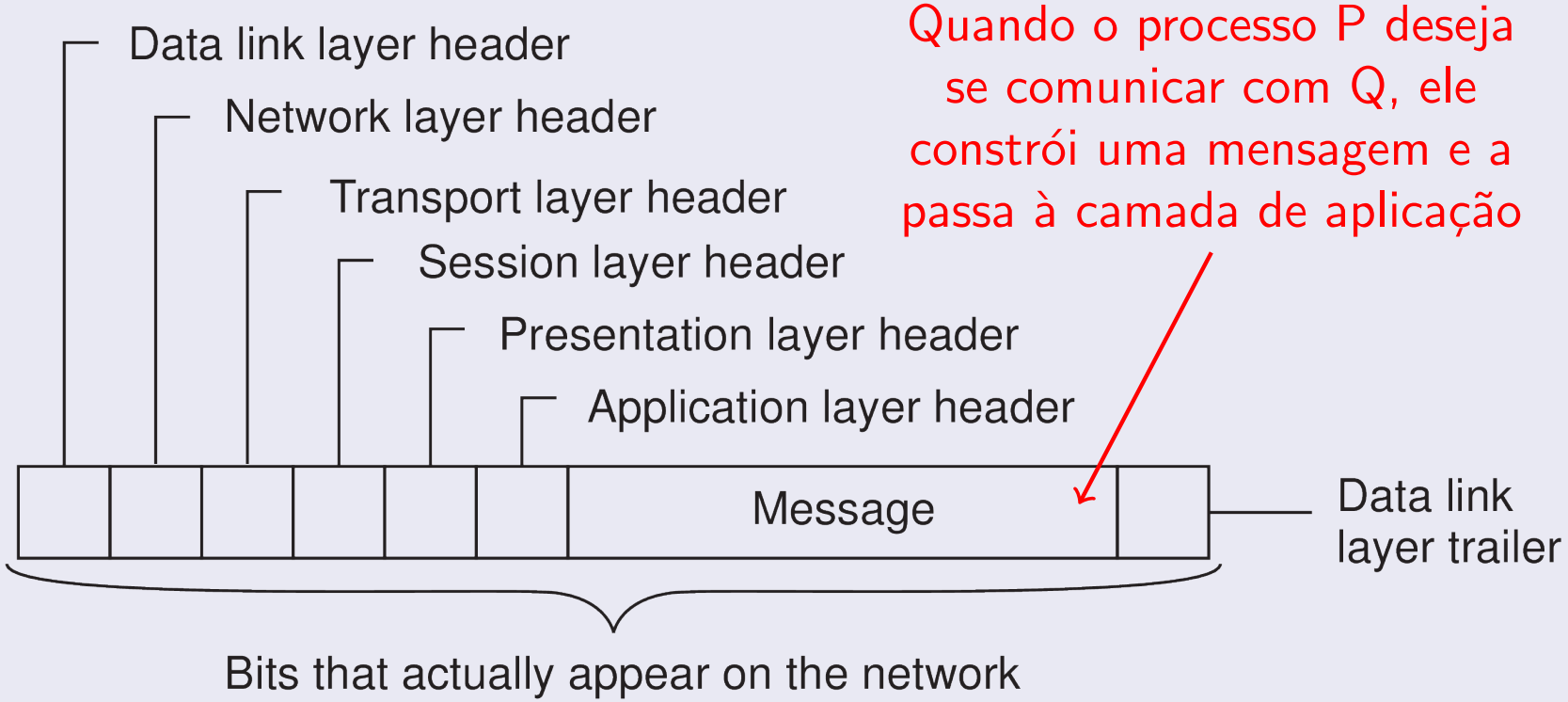
Os protocolos OSI nunca foram populares, dando lugar a protocolos desenvolvidos para a internet, como TCP/IP



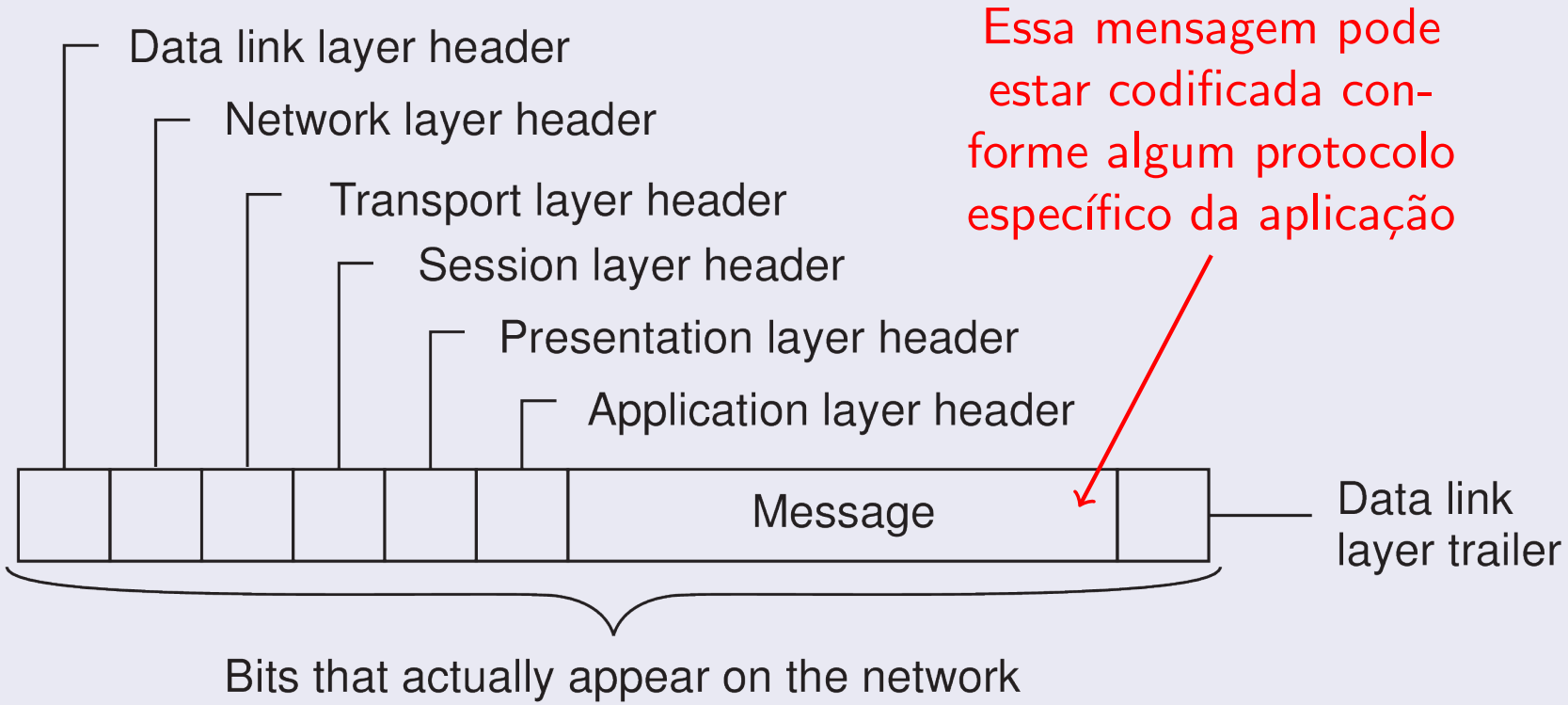
O modelo OSI – Encapsulamento



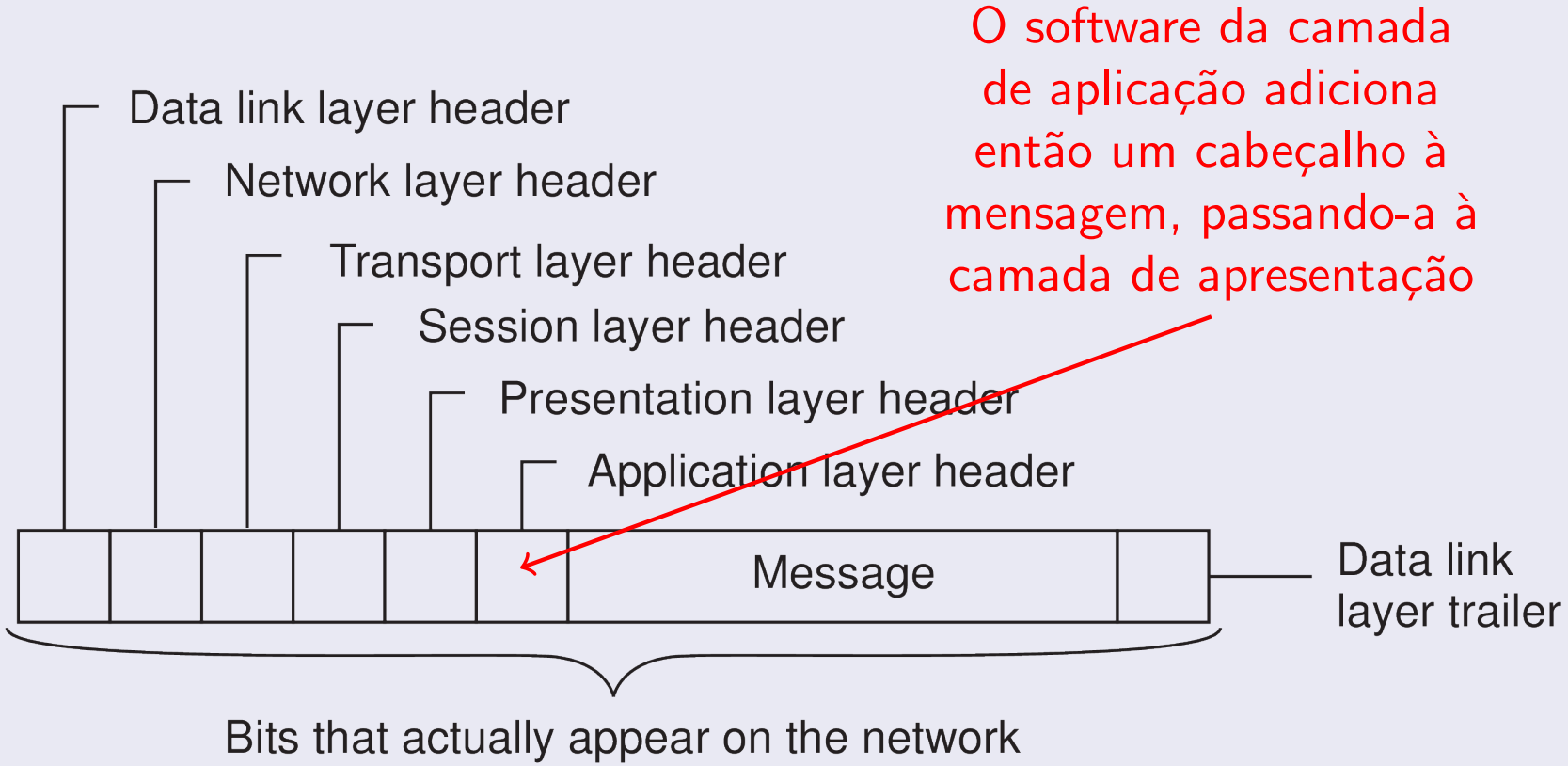
O modelo OSI – Encapsulamento



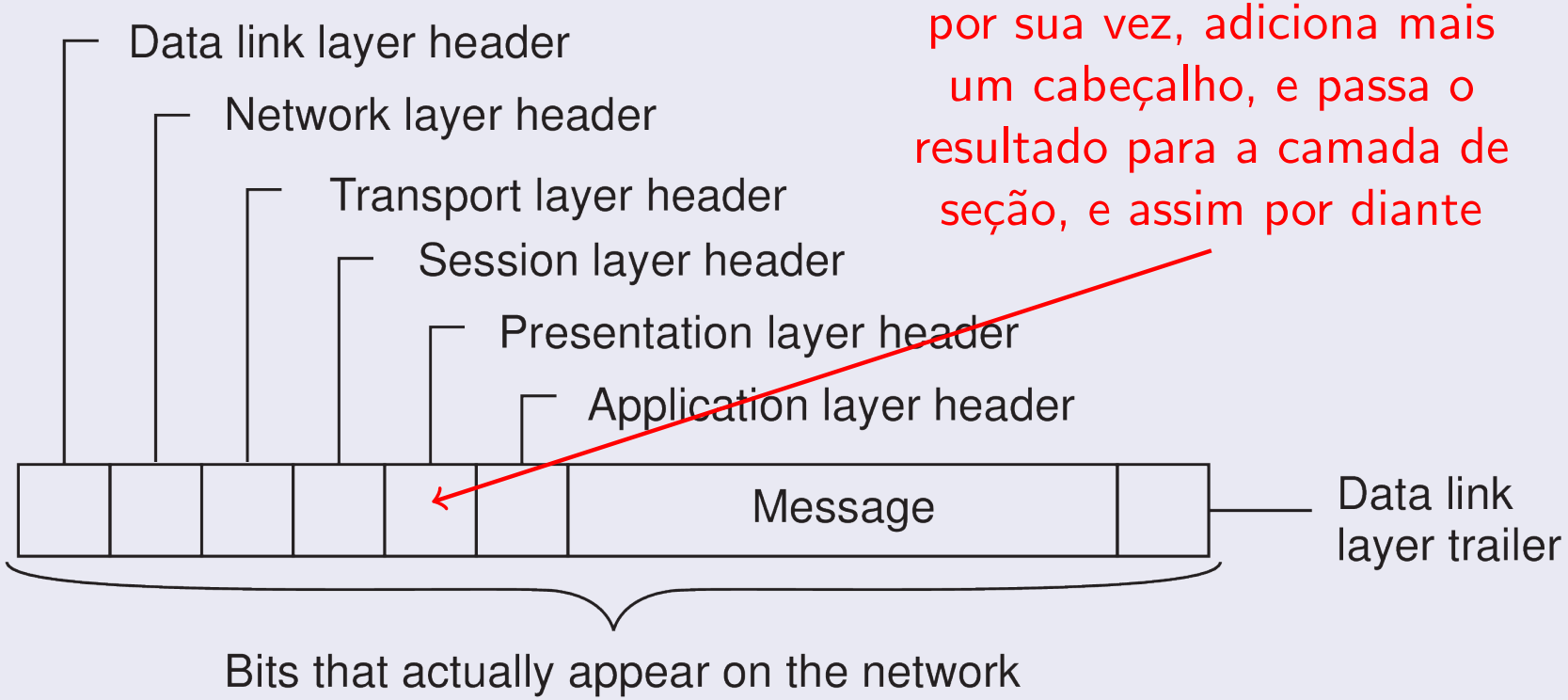
O modelo OSI – Encapsulamento



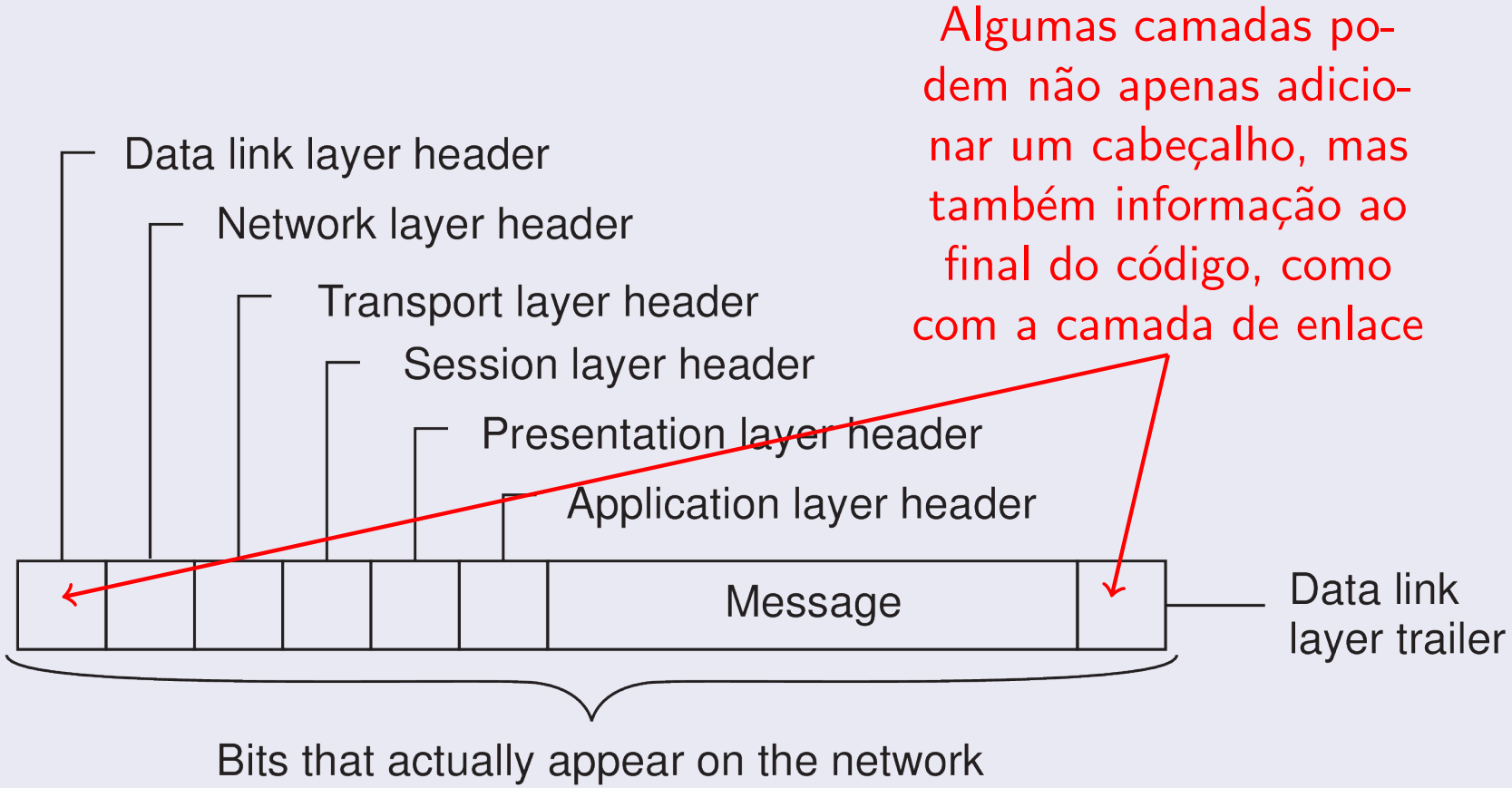
O modelo OSI – Encapsulamento



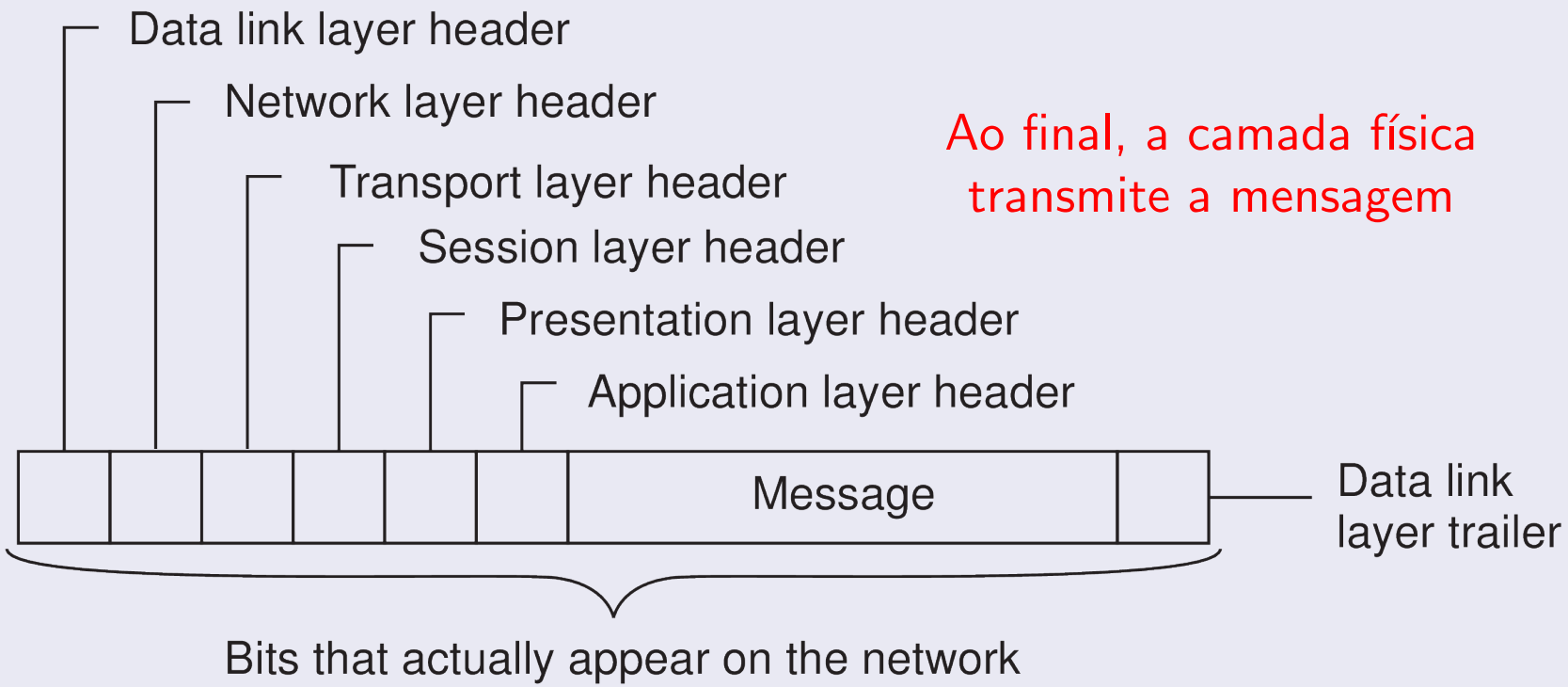
O modelo OSI – Encapsulamento



O modelo OSI – Encapsulamento

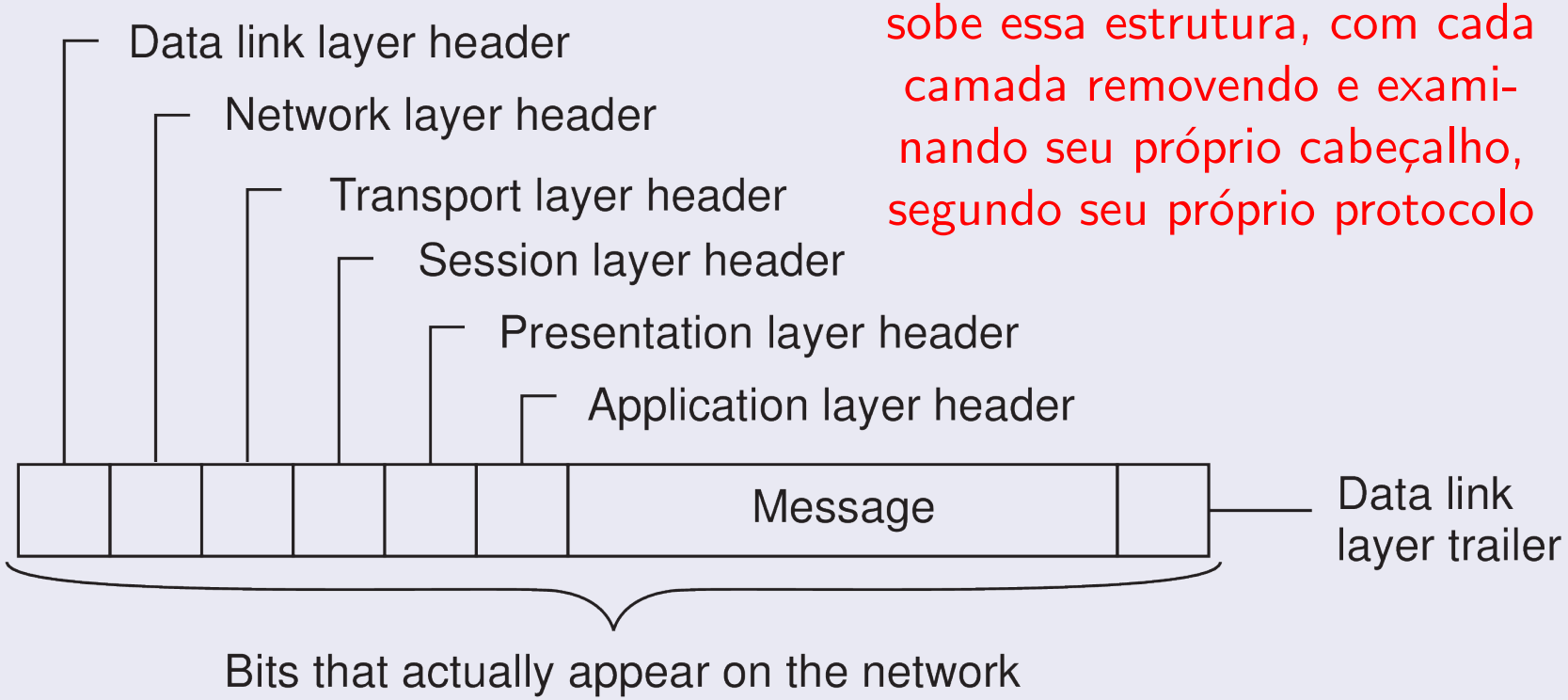


O modelo OSI – Encapsulamento



O modelo OSI – Encapsulamento

Quando a mensagem chega à máquina contendo Q, ela sobe essa estrutura, com cada camada removendo e examinando seu próprio cabeçalho, segundo seu próprio protocolo

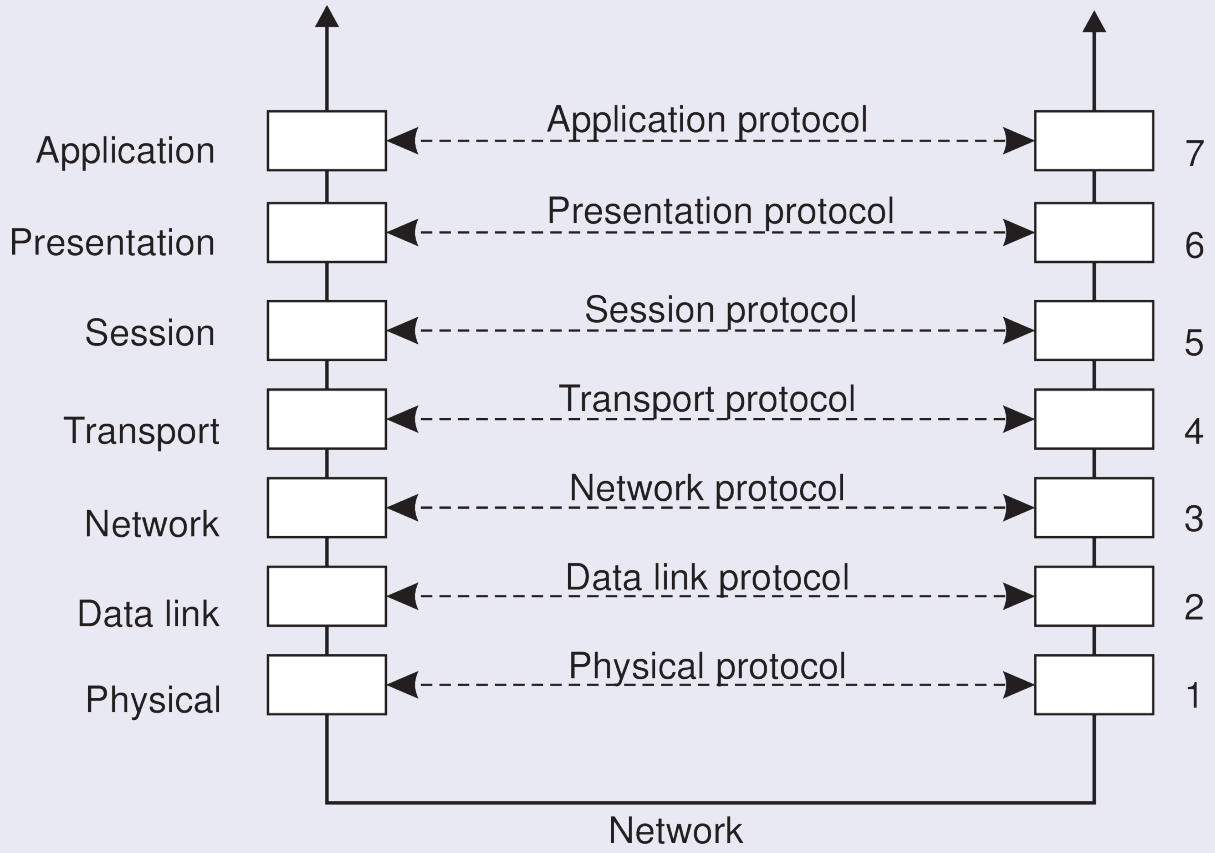


O modelo OSI

- Desvantagens
  - Foca apenas em passagem de mensagens
  - Frequentemente possui funcionalidades desnecessárias
  - Viola a transparência de acesso (esconder diferenças entre as representações de dados e como um objeto é acessado)

Middleware e o modelo OSI

Middleware é uma aplicação que vive logicamente (em sua maior parte) na camada de aplicação, mas que contém muitos protocolos de uso geral



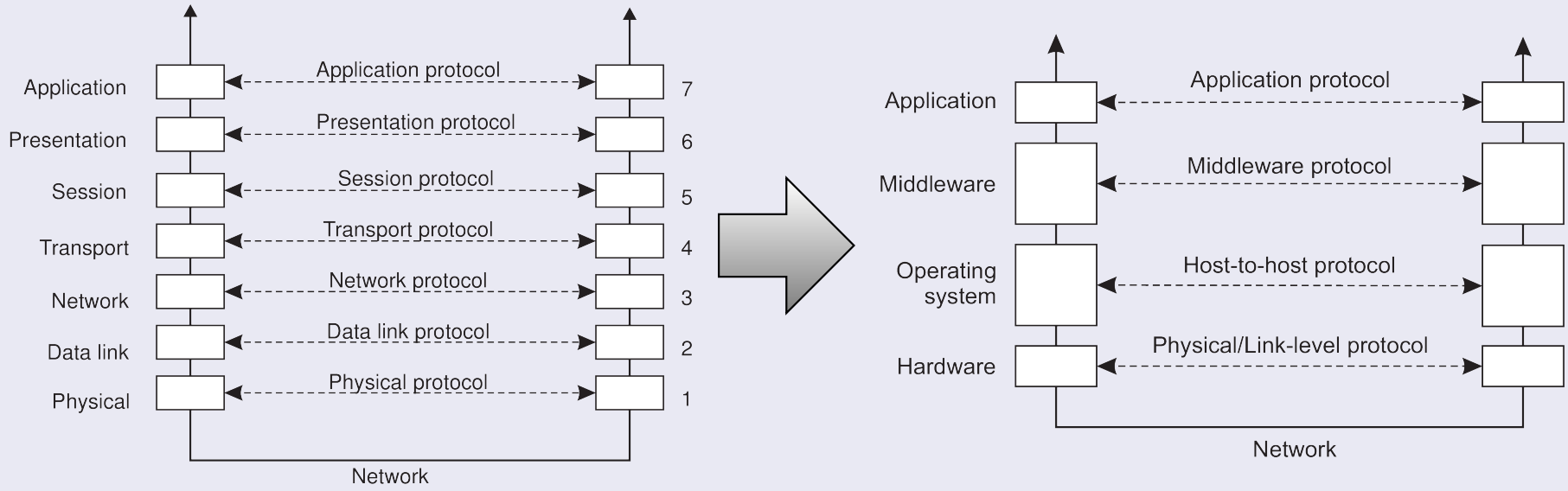
Fundamentos – Protocolos em camadas

Middleware e o modelo OSI

- O *middleware* foi inventado para prover serviços e protocolos em comum que podem ser utilizados por várias aplicações diferentes.
  - Um conjunto rico de protocolos de comunicação
  - (Des)empacotamento [*(un)marshaling*] de dados, necessários para a integração de sistemas
  - Protocolos de gerenciamento de nomes, para auxiliar o compartilhamento de recursos
  - Protocolos de segurança para comunicações seguras
  - Mecanismos de escalabilidade, tais como replicação e *caching*

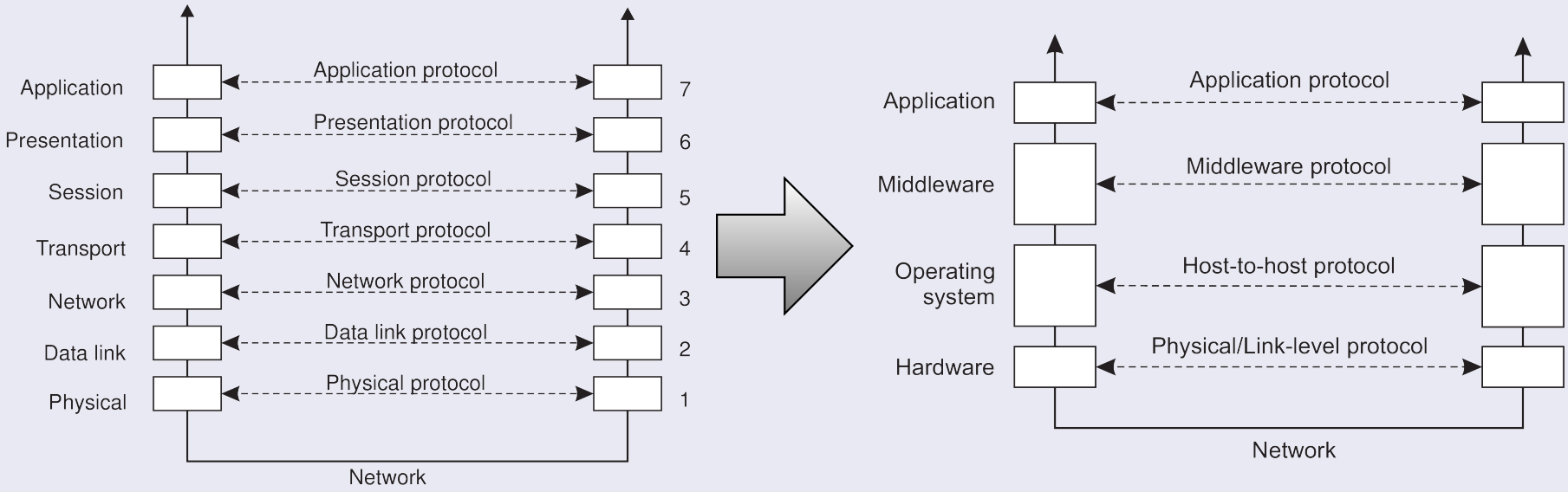
Camada de *middleware*

Isso nos leva a um modelo de comunicação adaptado e simplificado:



Camada de *middleware*

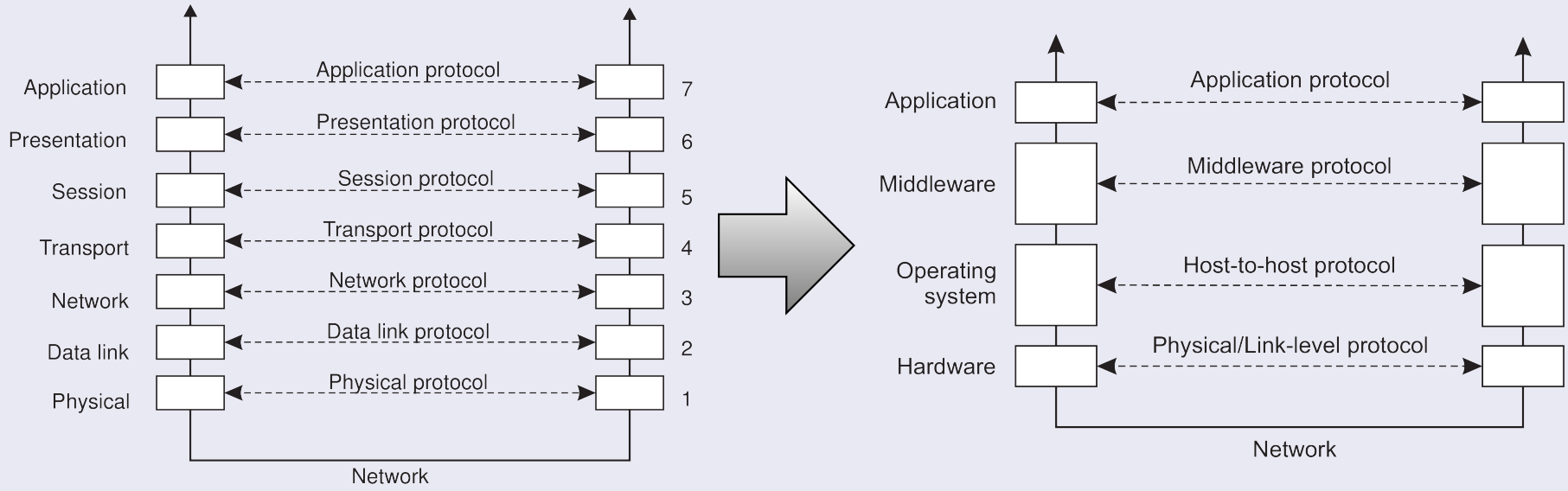
Isso nos leva a um modelo de comunicação adaptado e simplificado:



Substituímos as camadas de seção e apresentação pela de middleware, contendo protocolos independentes de aplicação

Camada de *middleware*

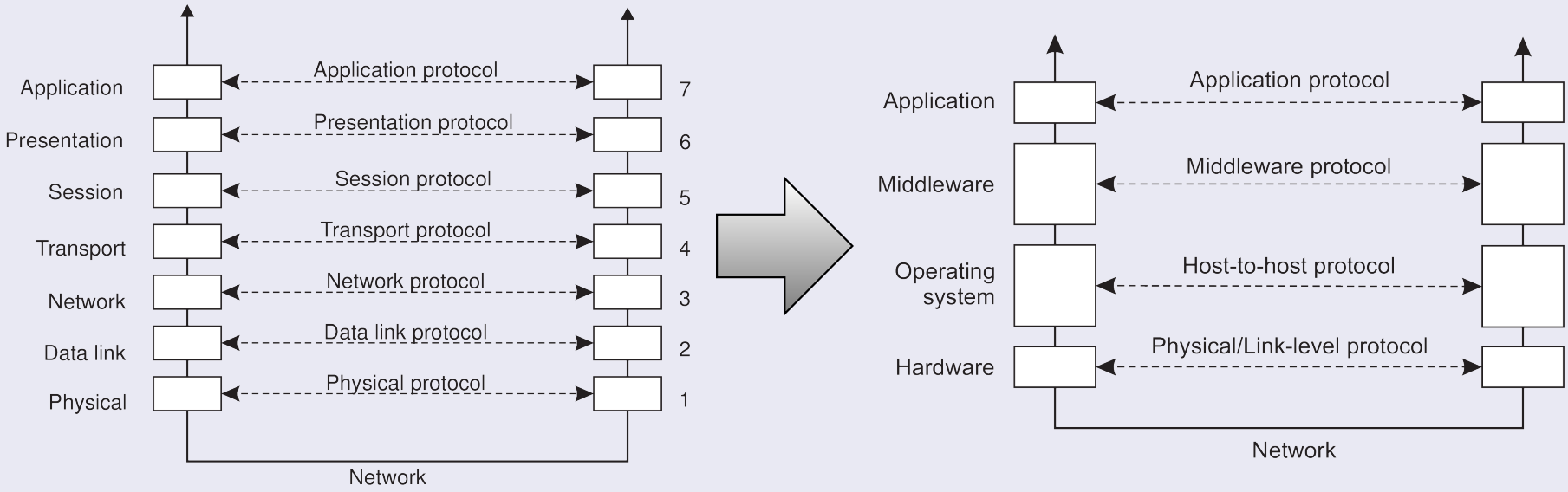
Isso nos leva a um modelo de comunicação adaptado e simplificado:



Serviços de rede e de transporte são agrupados em serviços de comunicação, normalmente oferecidos pelo SO

Camada de *middleware*

Isso nos leva a um modelo de comunicação adaptado e simplificado:



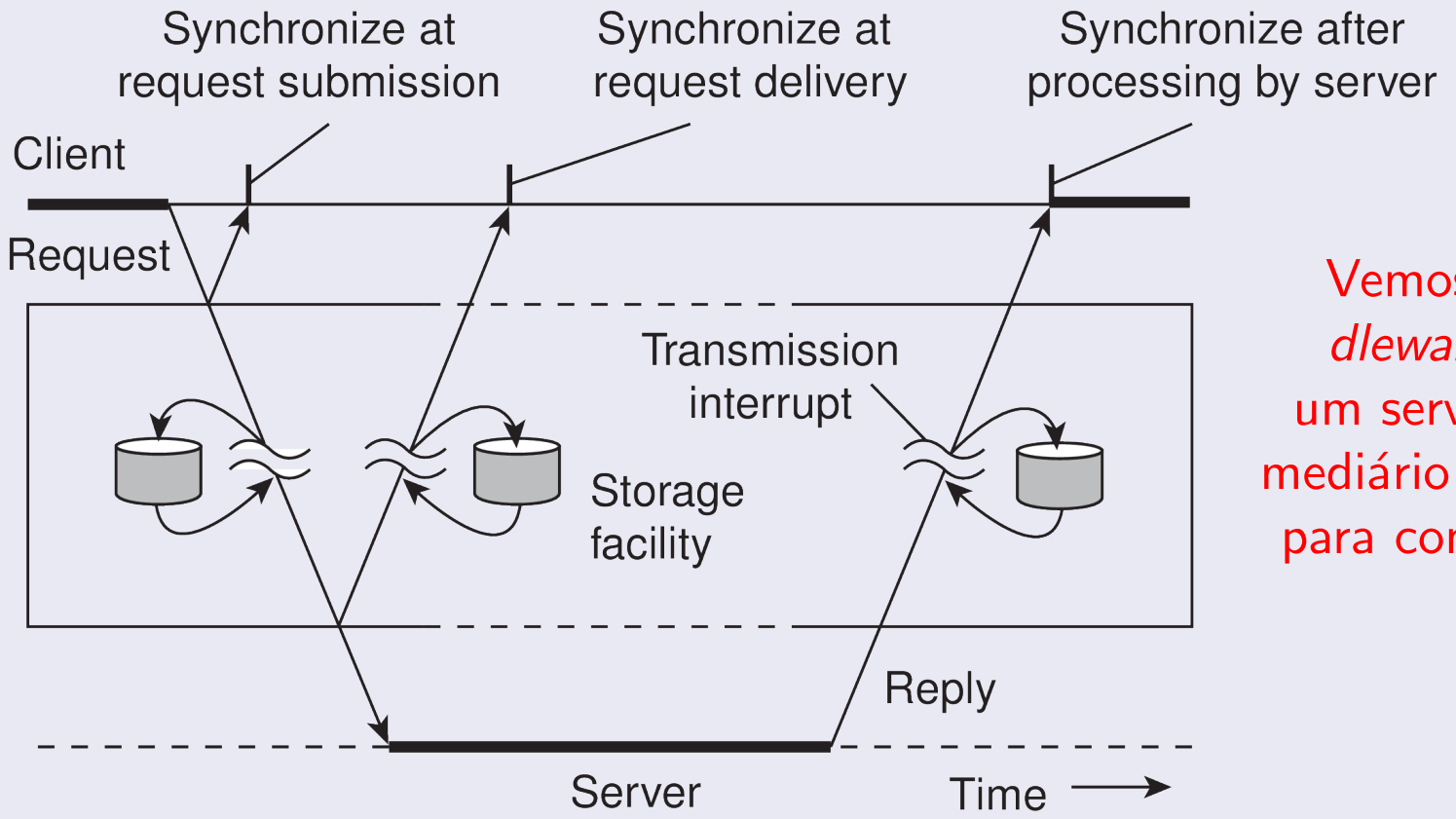
E o SO gerencia o hardware de baixo nível para estabelecer a comunicação

Comunicação

- **Fundamentos**
  - Protocolos em Camadas
  - **Tipos de Comunicação**
- Remote Procedure Call

Fundamentos – Tipos de Comunicação

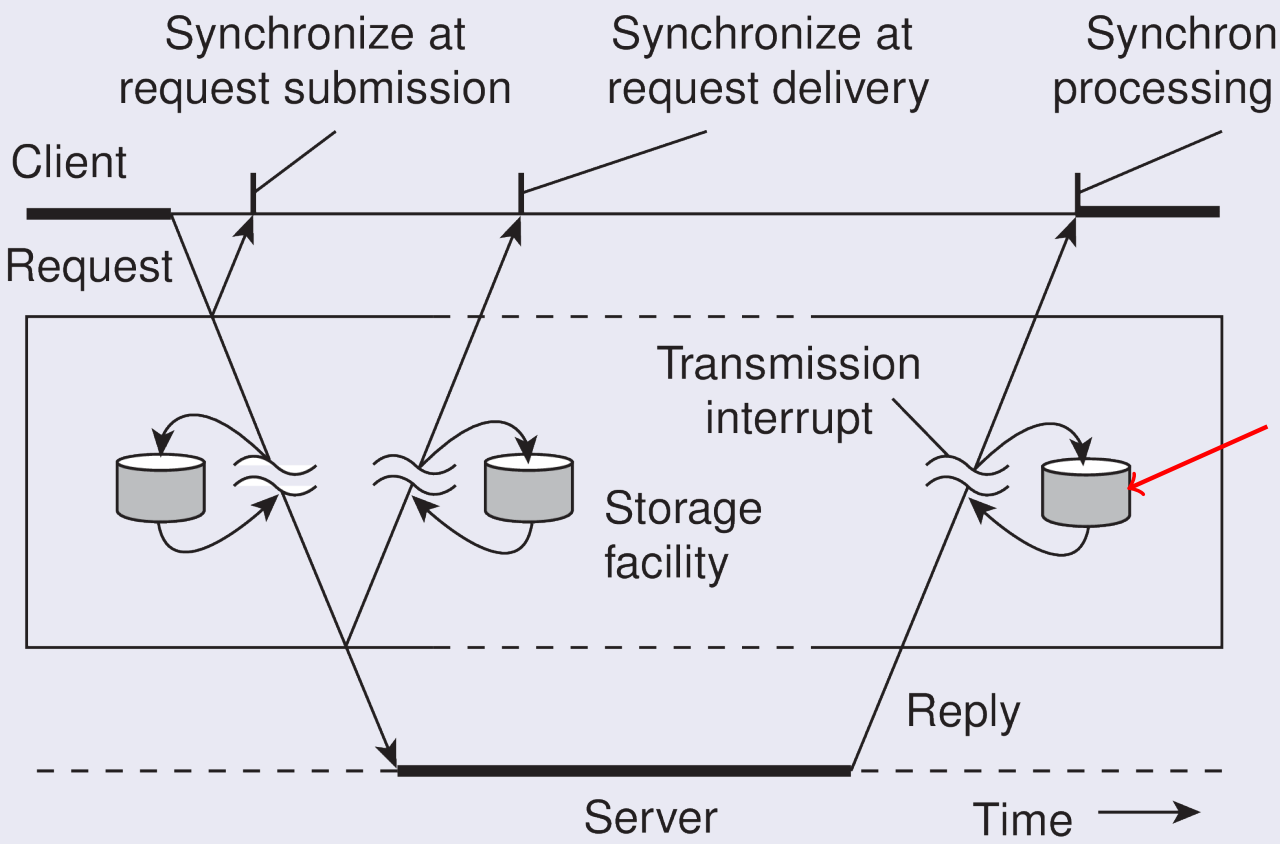
Middleware



Vemos o *middleware* como um serviço intermediário distribuído para comunicação

Fundamentos – Tipos de Comunicação

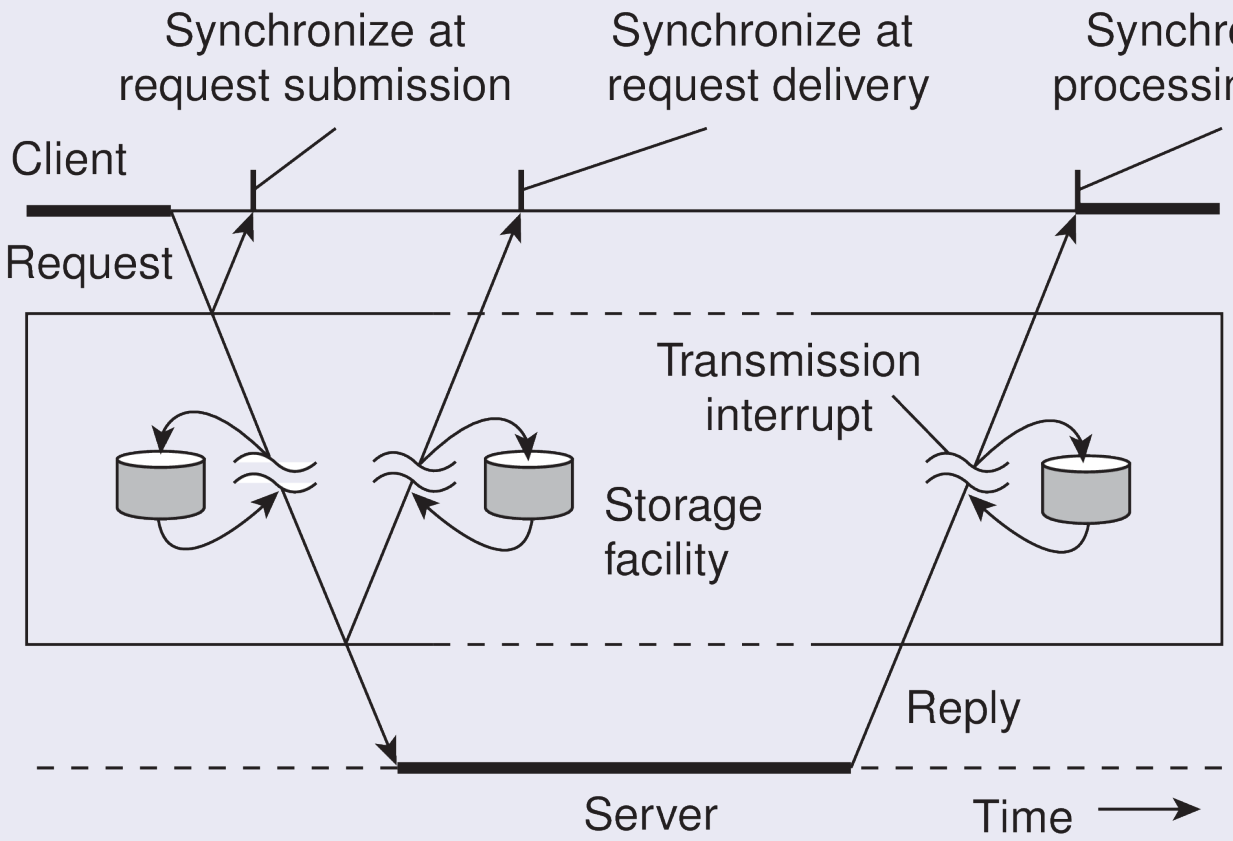
Comunicação transiente × persistente



Persistente: a mensagem é armazenada no *middleware* de comunicação pelo tempo que for necessário para que seja entregue

Fundamentos – Tipos de Comunicação

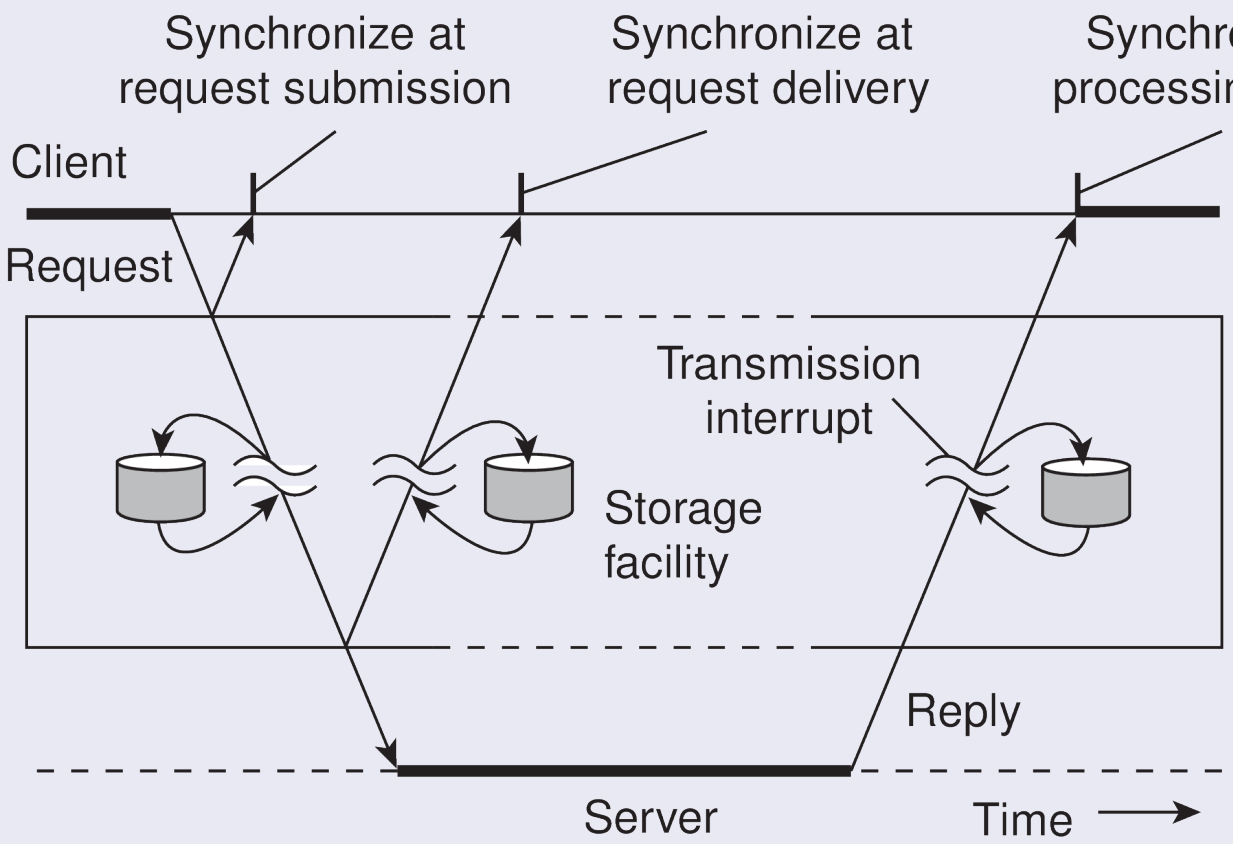
Comunicação transiente × persistente



Assim, a aplicação remetente não precisa continuar sua execução após o envio da mensagem

Fundamentos – Tipos de Comunicação

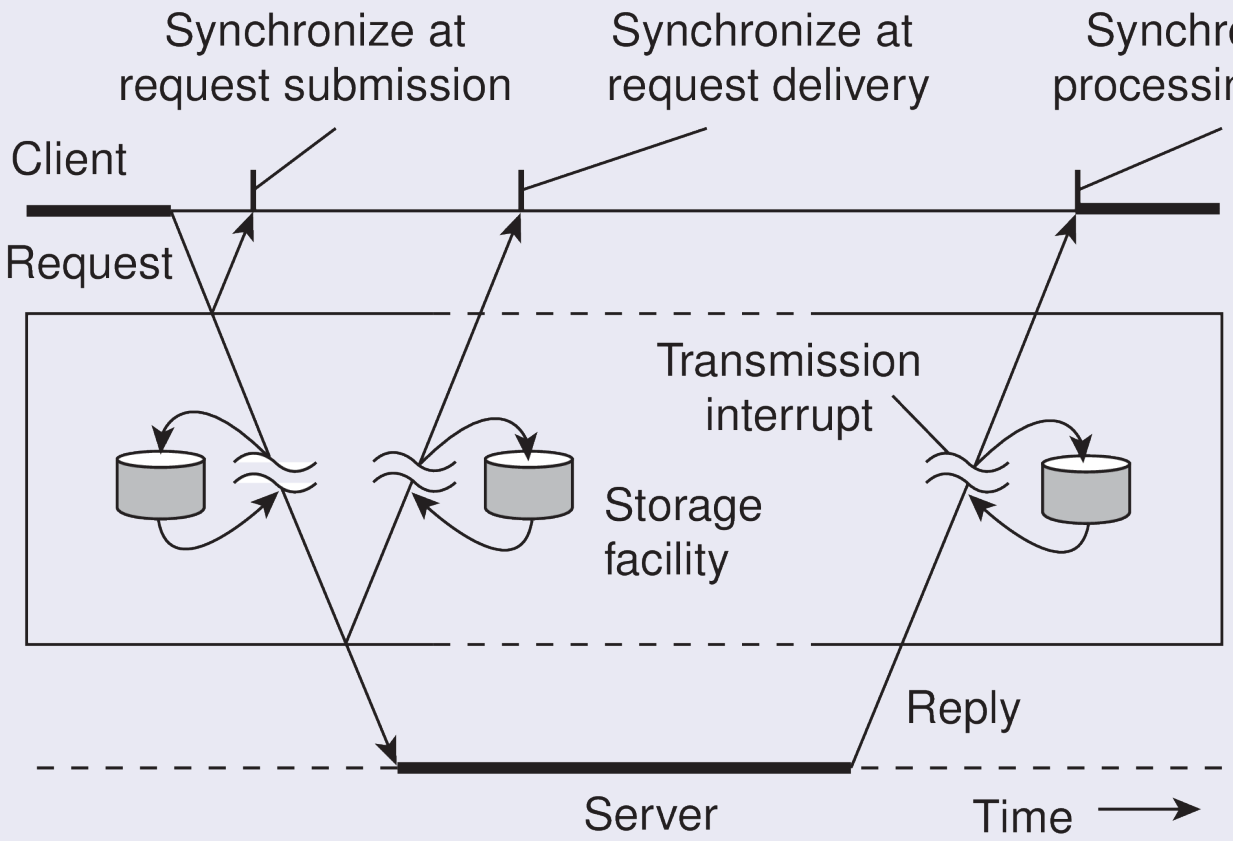
Comunicação transiente × persistente



Nem a aplicação receptora precisa estar rodando quando a mensagem é enviada

Fundamentos – Tipos de Comunicação

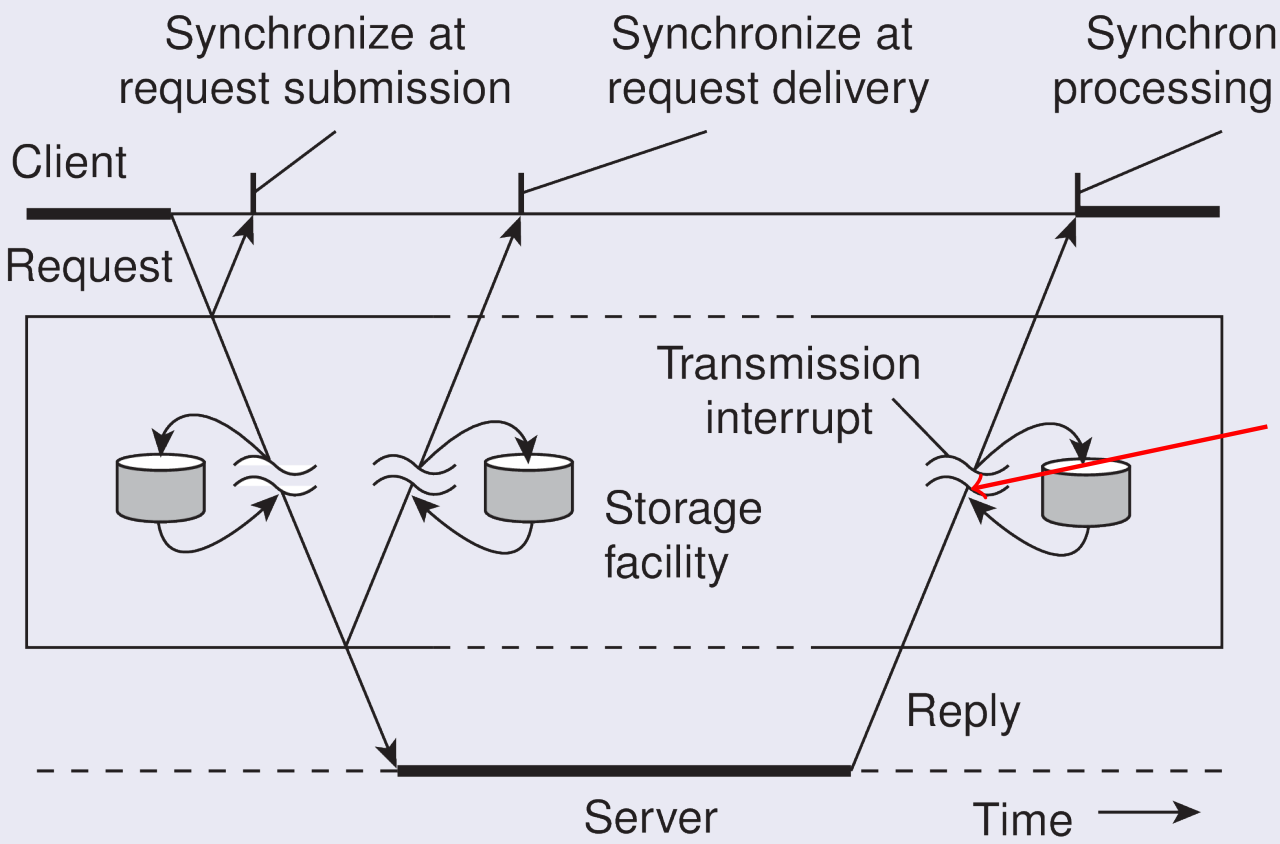
Comunicação transiente × persistente



Transiente: o *middleware* armazena a mensagem somente enquanto o emissor e receptor estiverem executando

Fundamentos – Tipos de Comunicação

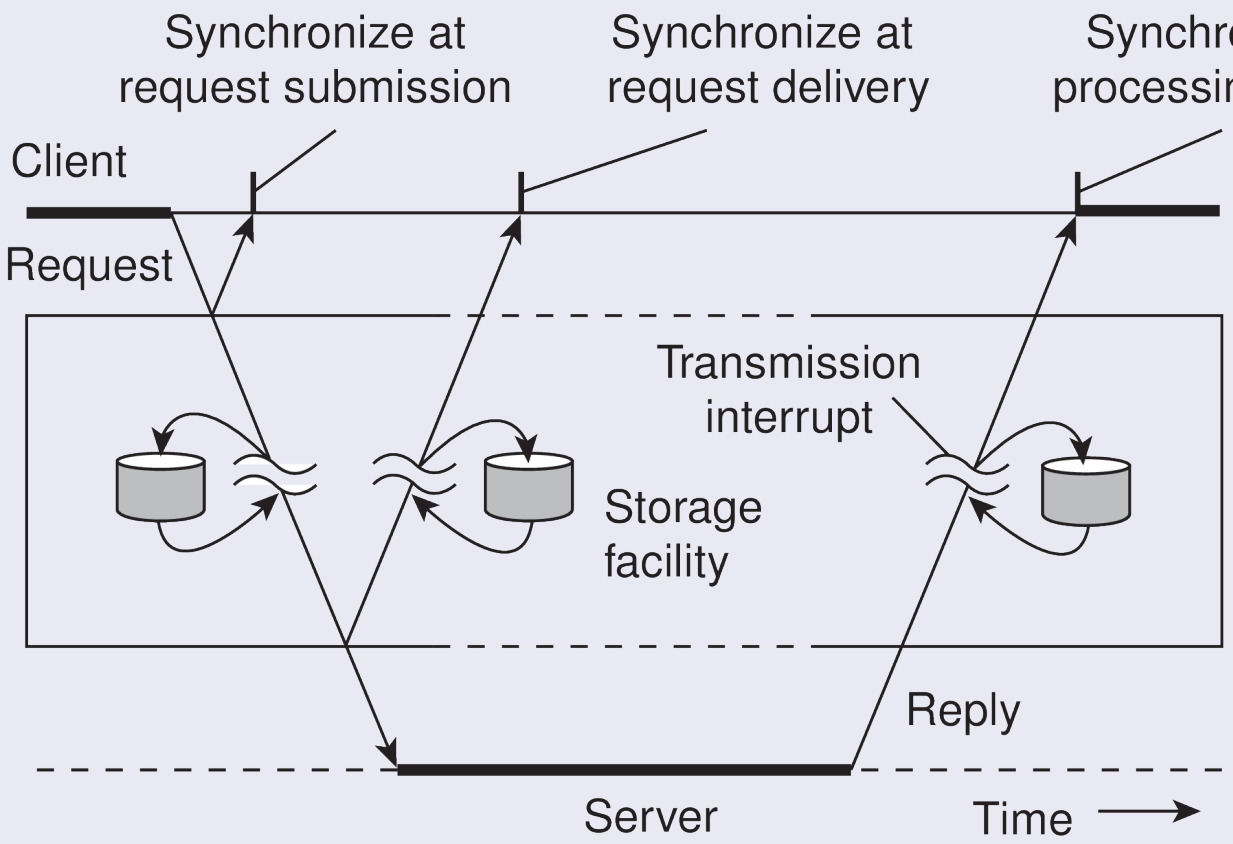
Comunicação transiente × persistente



Ele descarta a mensagem se ela não puder ser encaminhada para o destinatário ou ao próximo servidor

Fundamentos – Tipos de Comunicação

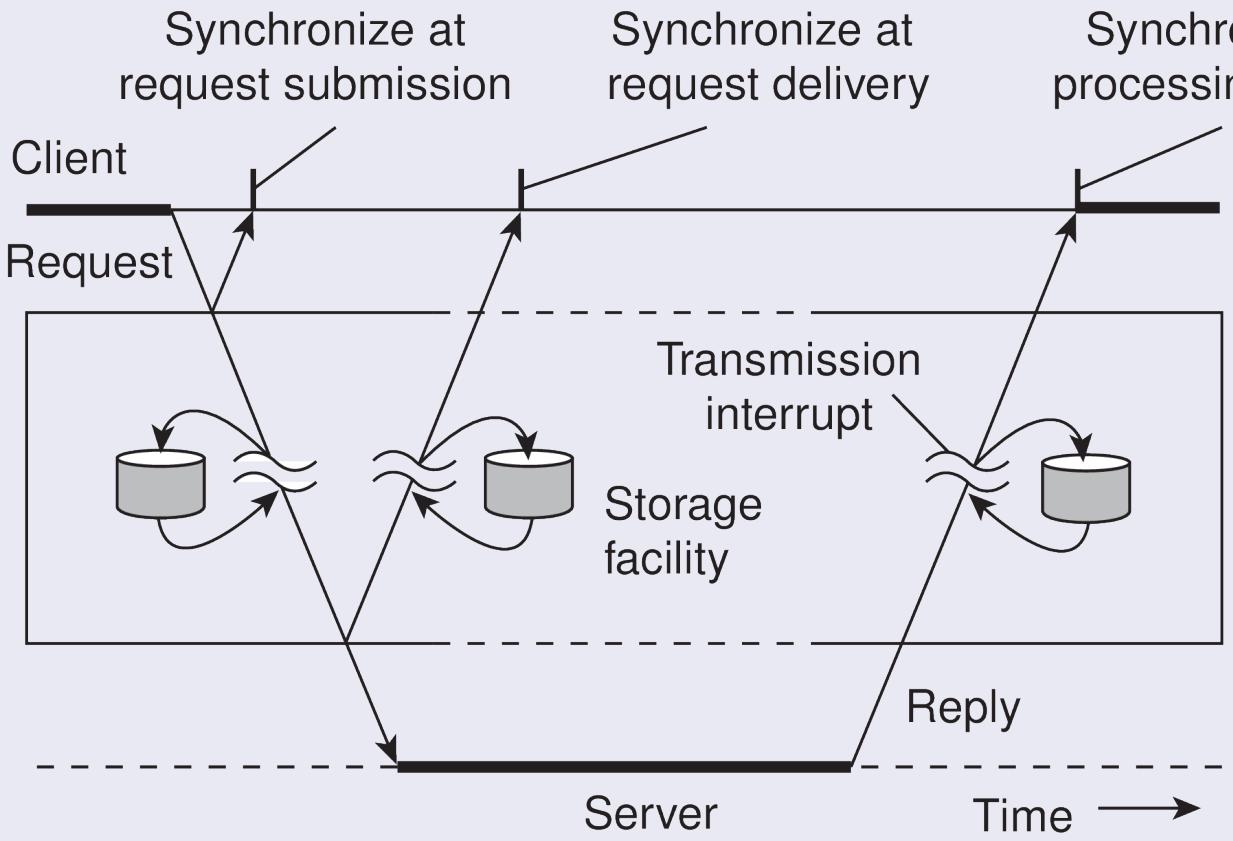
Comunicação assíncrona × síncrona



Assíncrona: o emissor continua seu trabalho imediatamente após enviar a mensagem

Fundamentos – Tipos de Comunicação

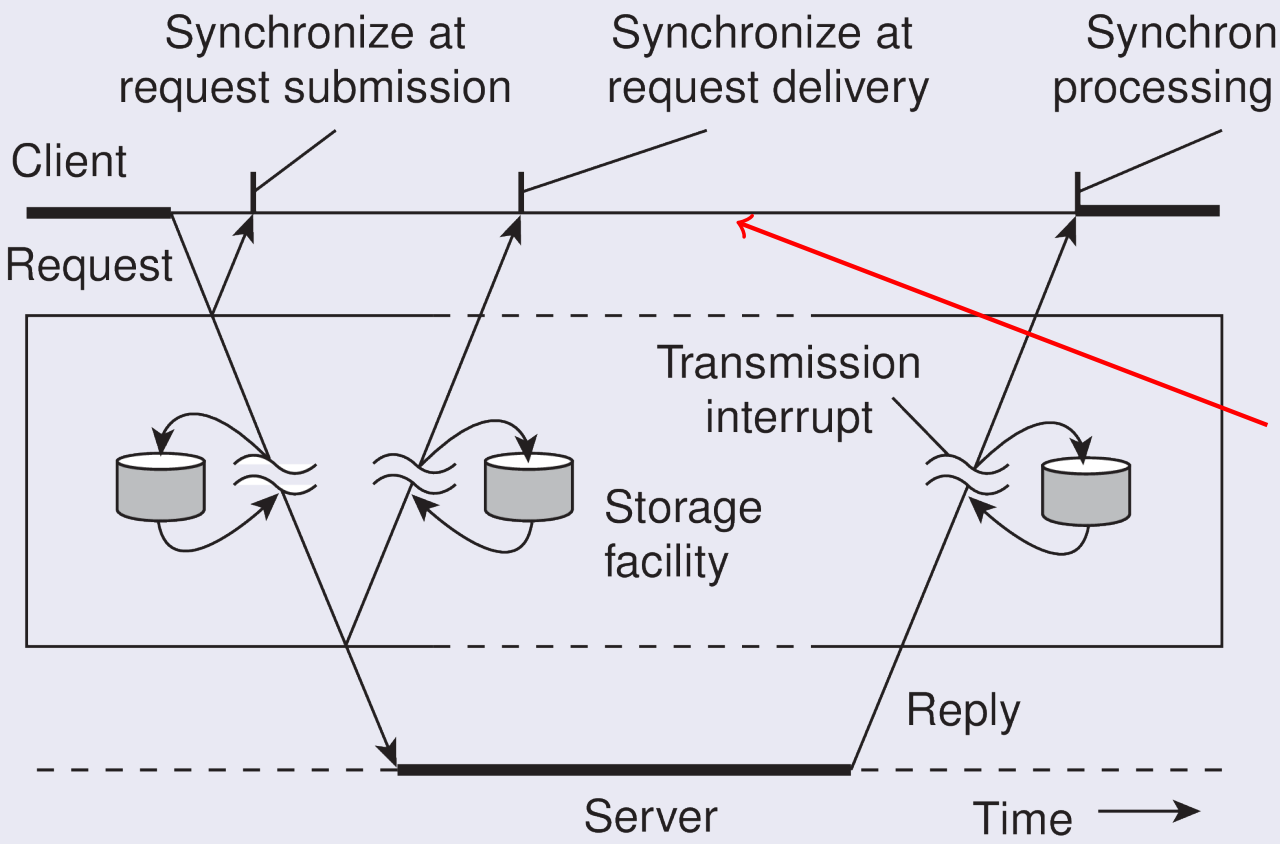
Comunicação assíncrona × síncrona



Esta é temporariamente armazenada no *middleware*

Fundamentos – Tipos de Comunicação

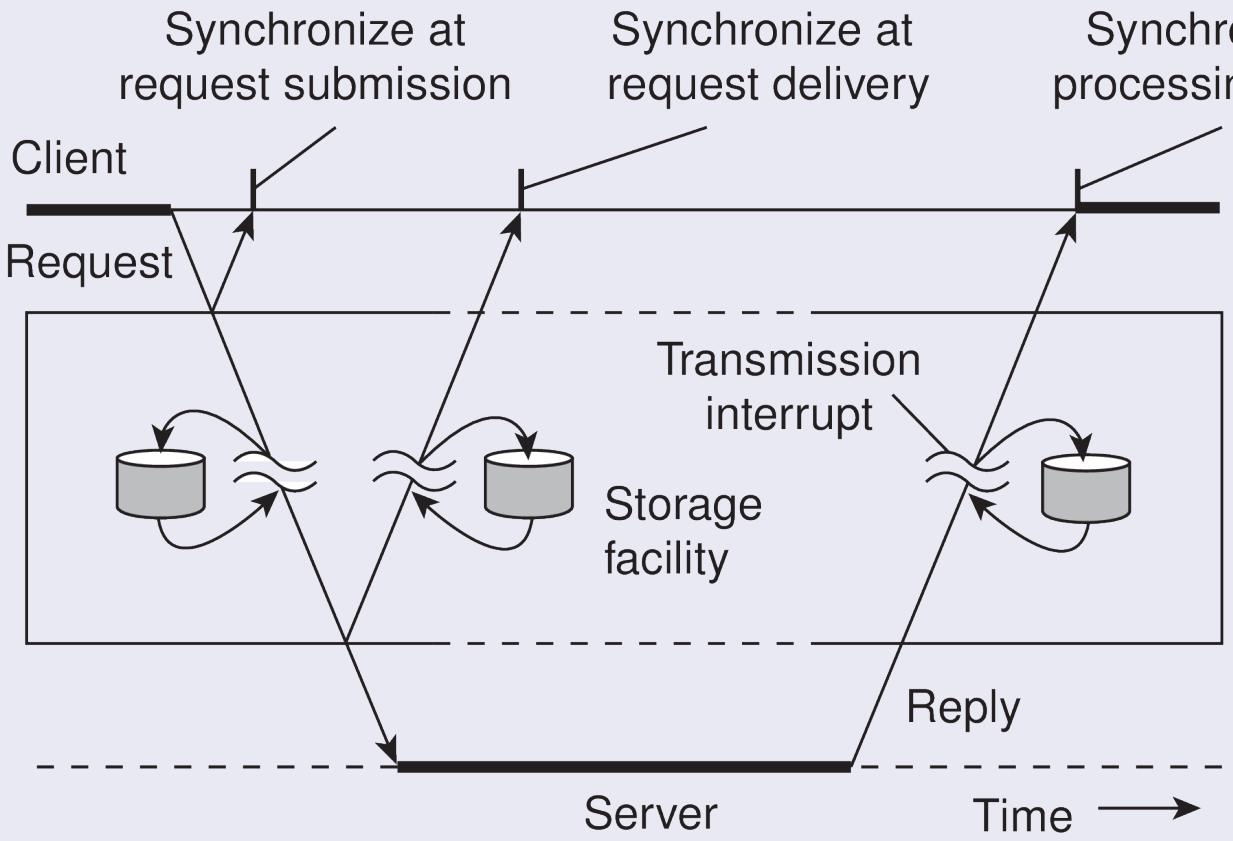
Comunicação assíncrona × síncrona



Síncrona: o emissor é bloqueado até ter certeza de que sua requisição foi aceita

Fundamentos – Tipos de Comunicação

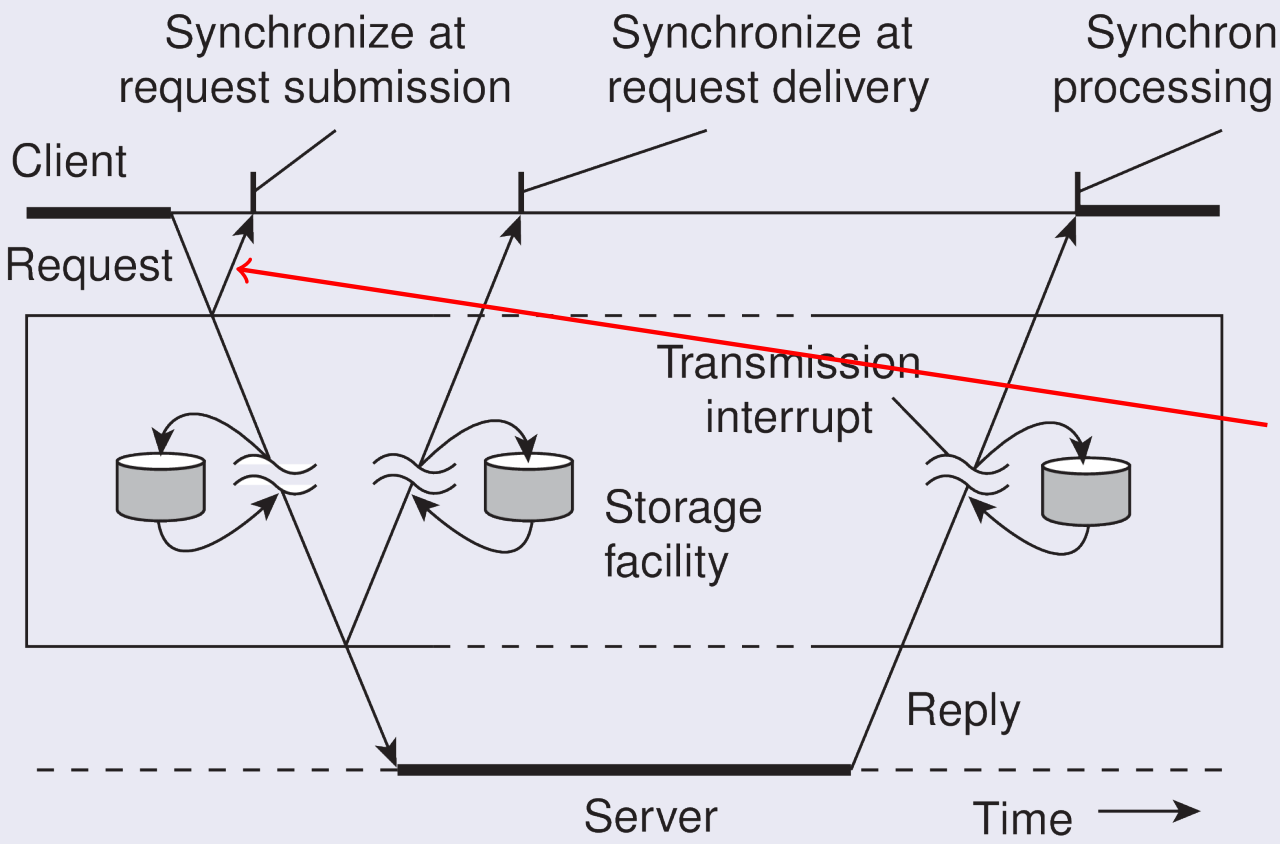
Comunicação assíncrona × síncrona



Há essencialmente 3 pontos para sincronização

Fundamentos – Tipos de Comunicação

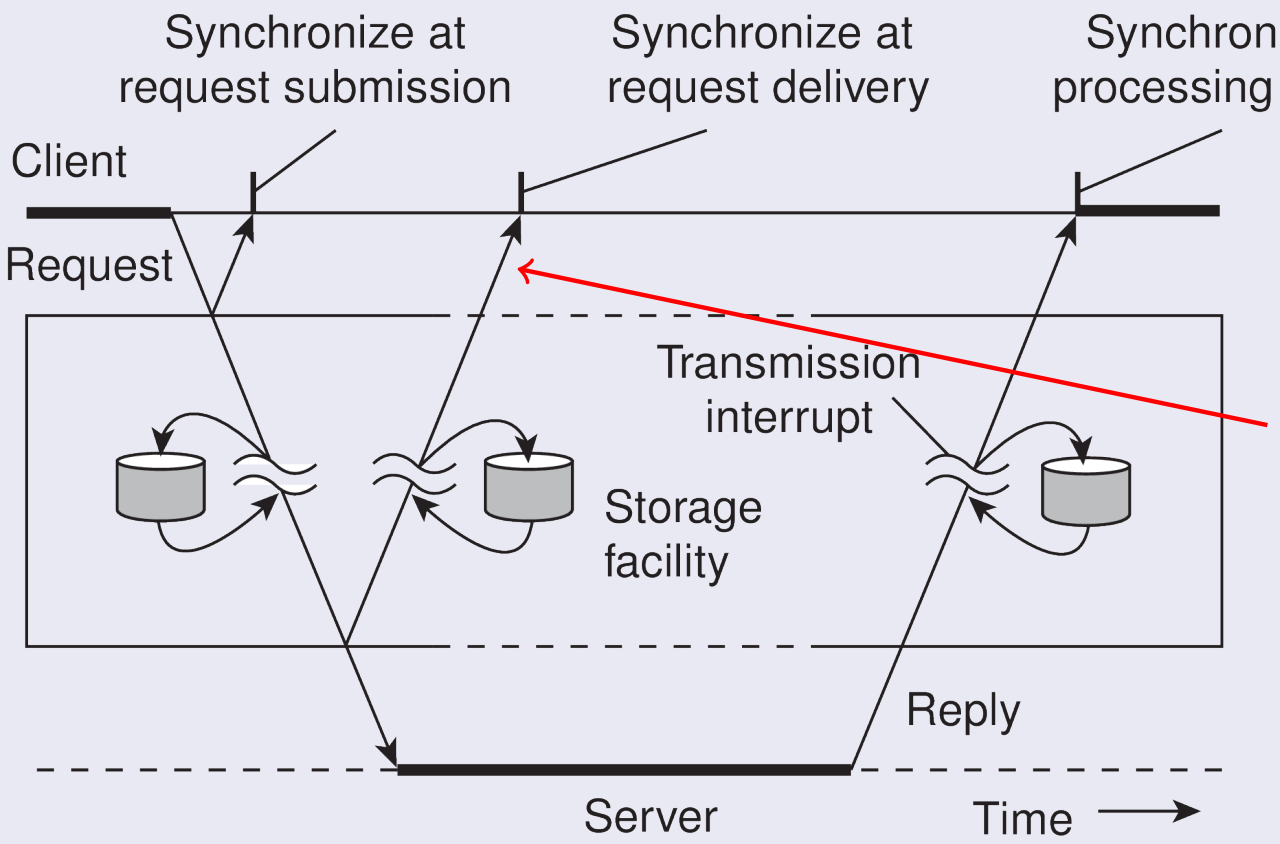
Comunicação assíncrona × síncrona



No envio da requisição: o remetente é bloqueado até que o *middleware* o notifique de que irá transmitir a requisição

Fundamentos – Tipos de Comunicação

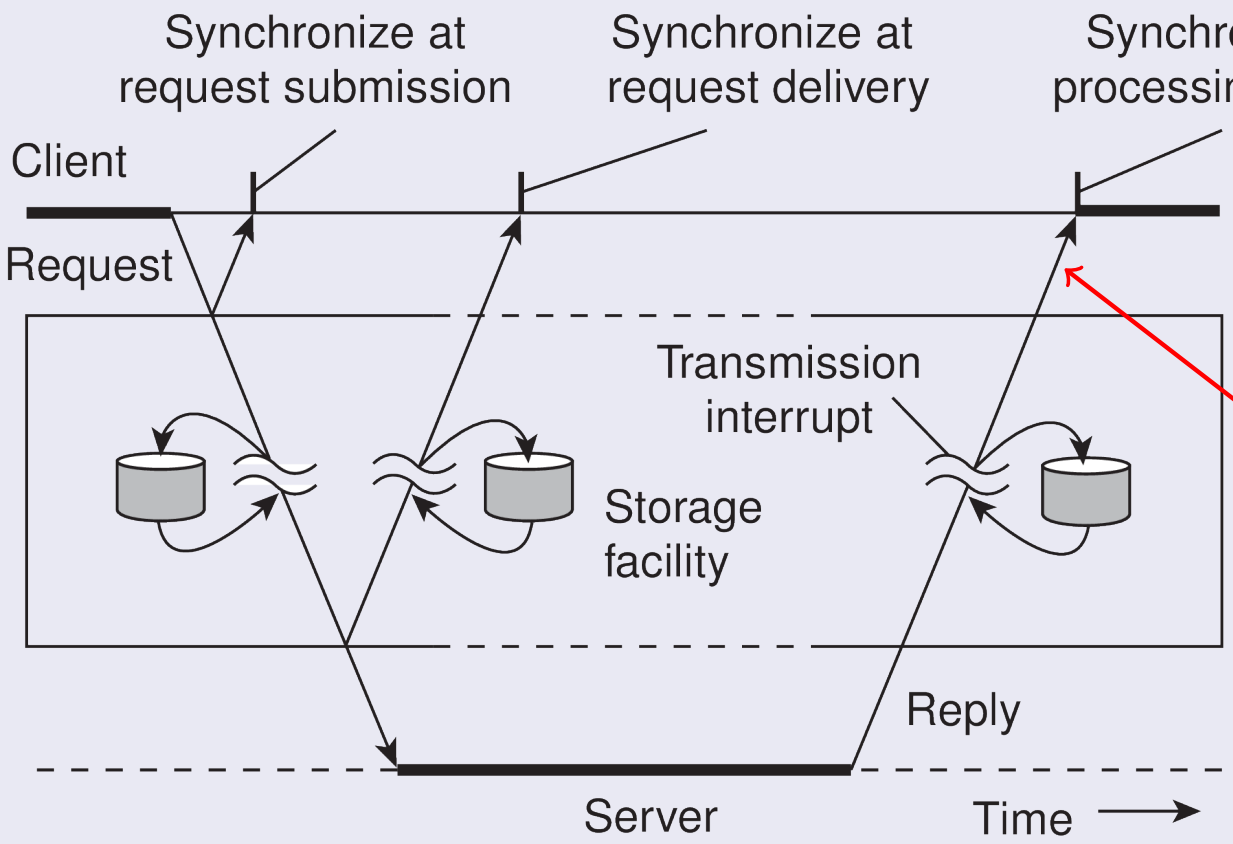
Comunicação assíncrona × síncrona



Na entrega da requisição: o emissor é avisado de que a mensagem foi entregue

Fundamentos – Tipos de Comunicação

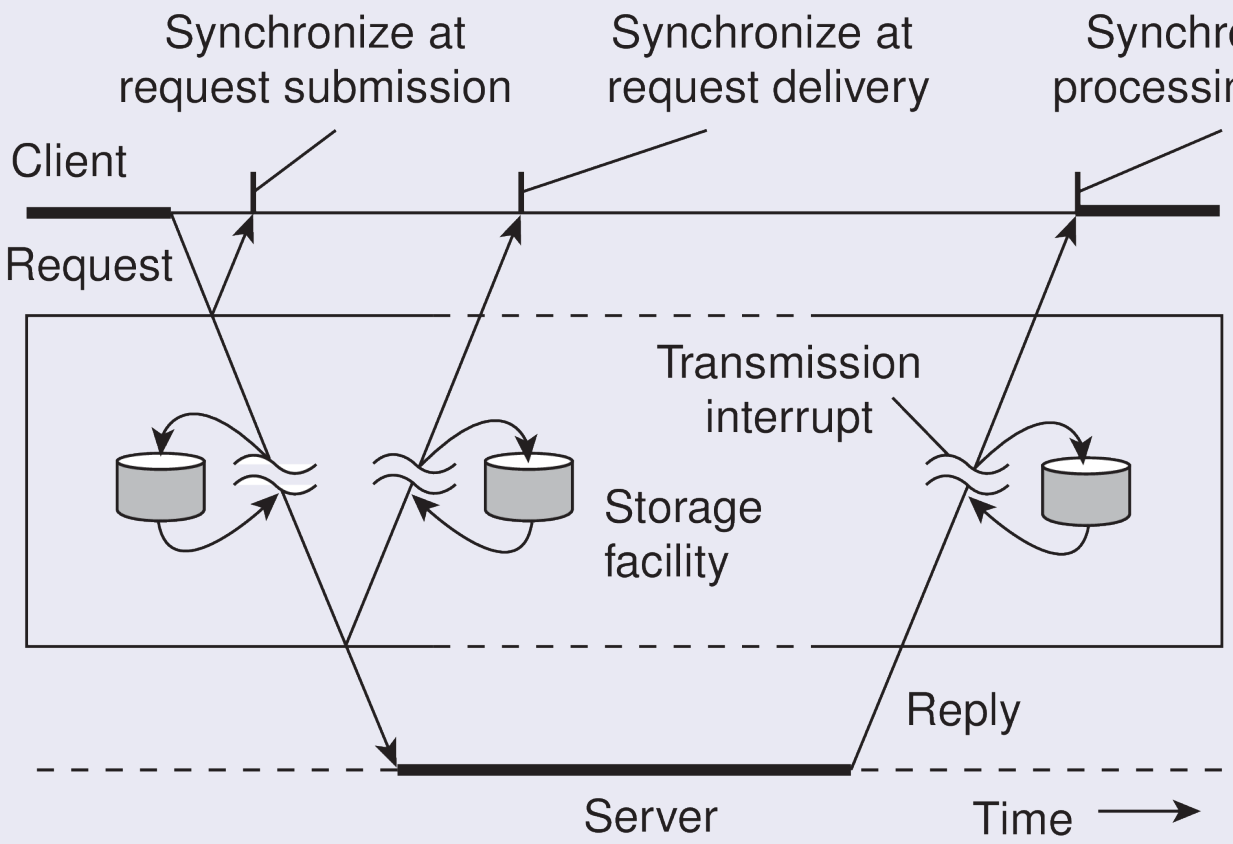
Comunicação assíncrona × síncrona



Após o processamento da requisição: o emissor espera até que o receptor retorne uma resposta

Fundamentos – Tipos de Comunicação

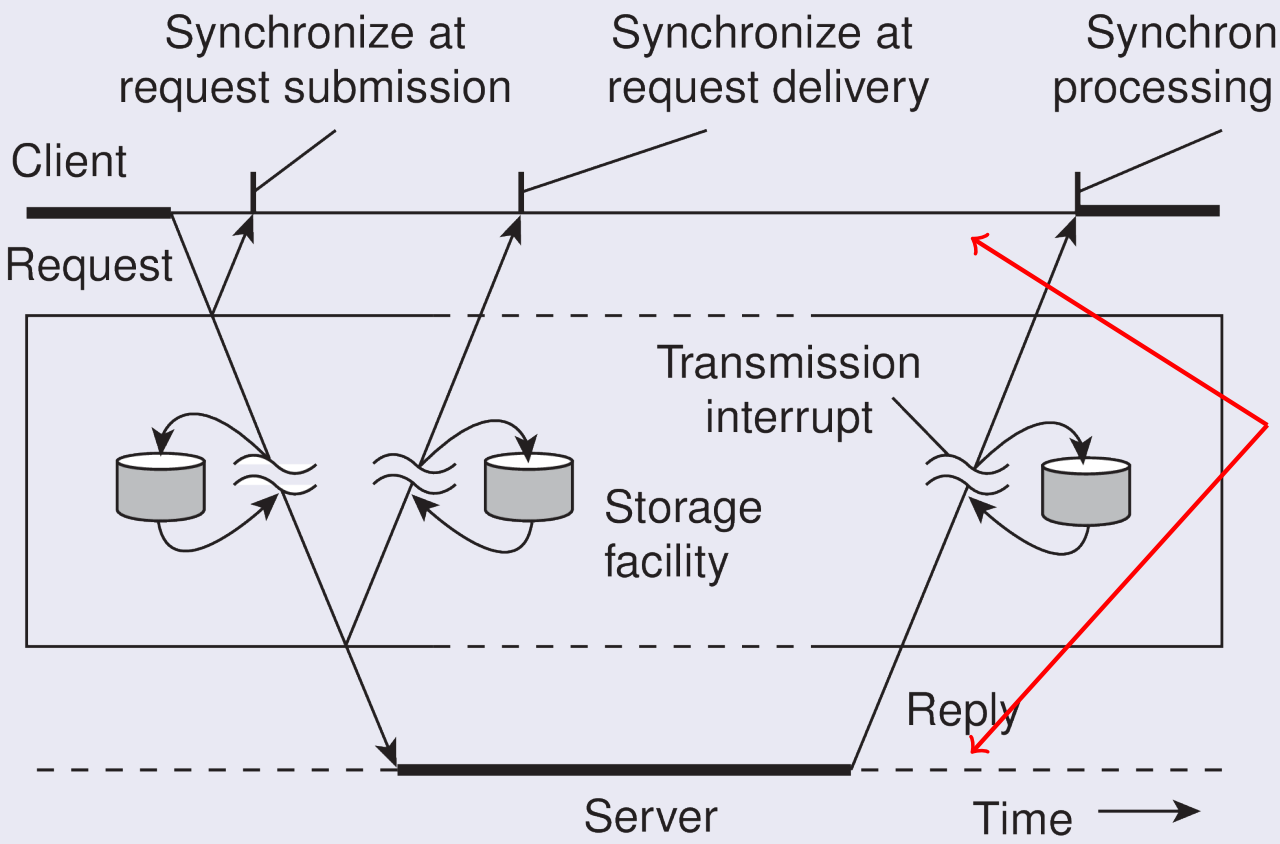
Observações



Computação **Cliente/Servidor** geralmente é baseada em um modelo de comunicação transiente síncrona

Fundamentos – Tipos de Comunicação

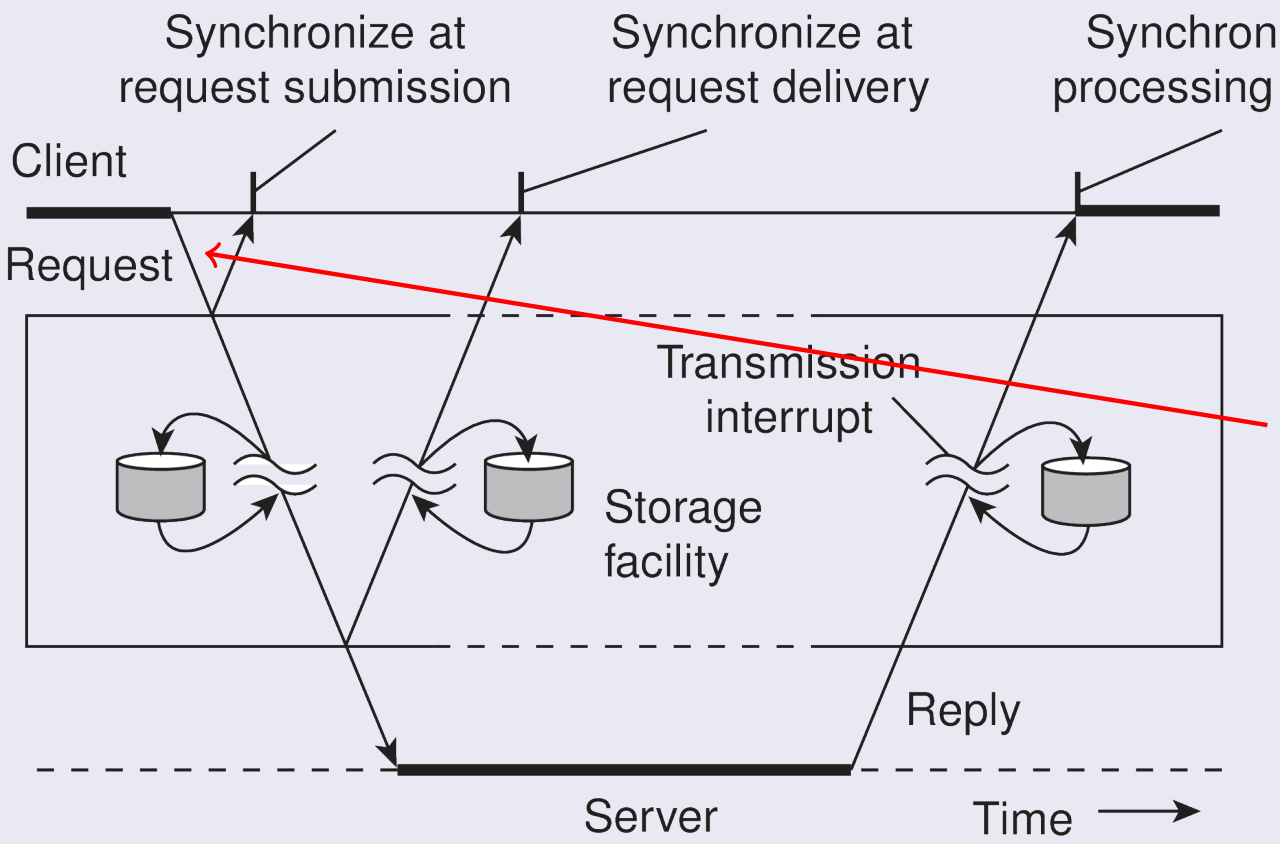
Observações



Cliente e servidor devem estar ativos no momento da comunicação

Fundamentos – Tipos de Comunicação

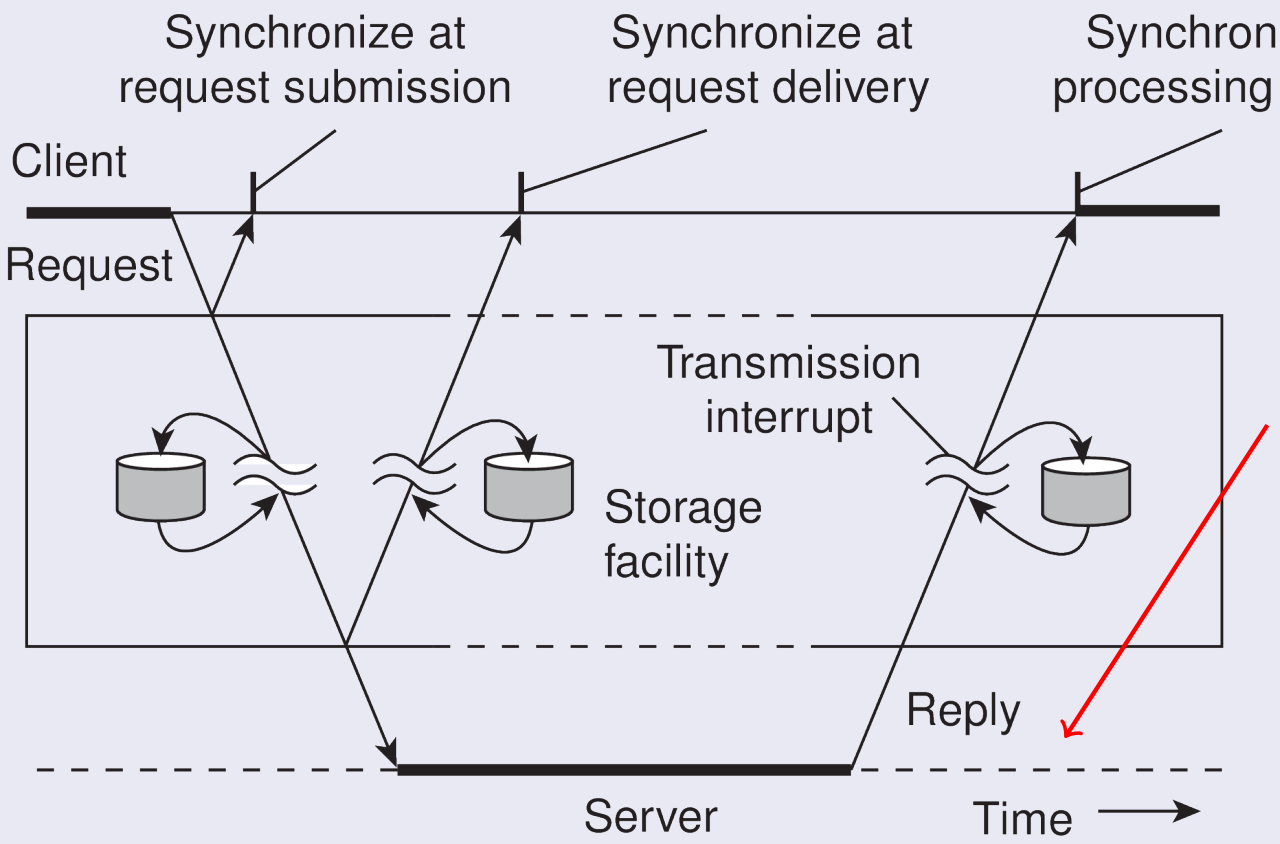
Observações



O cliente envia uma requisição e bloqueia até que receba sua resposta

Fundamentos – Tipos de Comunicação

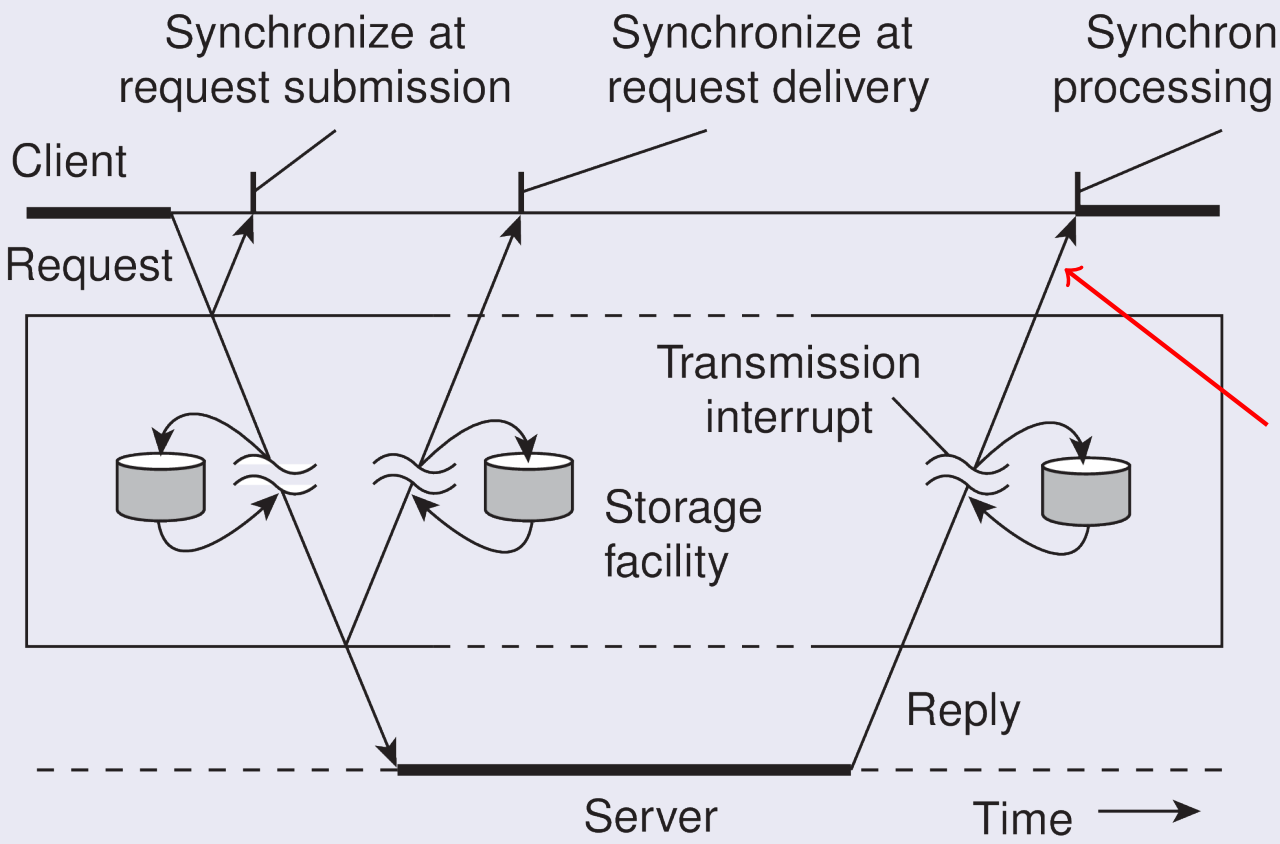
Observações



O servidor essencialmente espera por requisições e as processa

Fundamentos – Tipos de Comunicação

Observações



Comunicação transiente com sincronização após a requisição ser totalmente processada corresponde a **chamadas a procedimentos remotos**

Fundamentos – Tipos de Comunicação

Desvantagens de comunicação síncrona

- O cliente não pode fazer nenhum trabalho enquanto estiver esperando por uma resposta
- Falhas precisam ser tratadas imediatamente (afinal, o cliente está esperando)
- O modelo pode não ser o mais apropriado (mail, news)

Fundamentos – Trocas de mensagens

Middleware orientado a mensagens

- Tem como objetivo prover comunicação persistente assíncrona:
  - Processos trocam mensagens entre si, as quais são armazenadas em uma fila
  - O remetente não precisa esperar por uma resposta imediata, pode fazer outras coisas enquanto espera
  - O *middleware* normalmente assegura tolerância a falhas

Comunicação

- Fundamentos
  - Protocolos em Camadas
  - Tipos de Comunicação
- **Remote Procedure Call**

Chamada a Procedimento Remoto (RPC)

Por que existe?

- *Send* e *receive* não escondem a comunicação
  - Violam a transparência de acesso

Chamada a Procedimento Remoto (RPC)

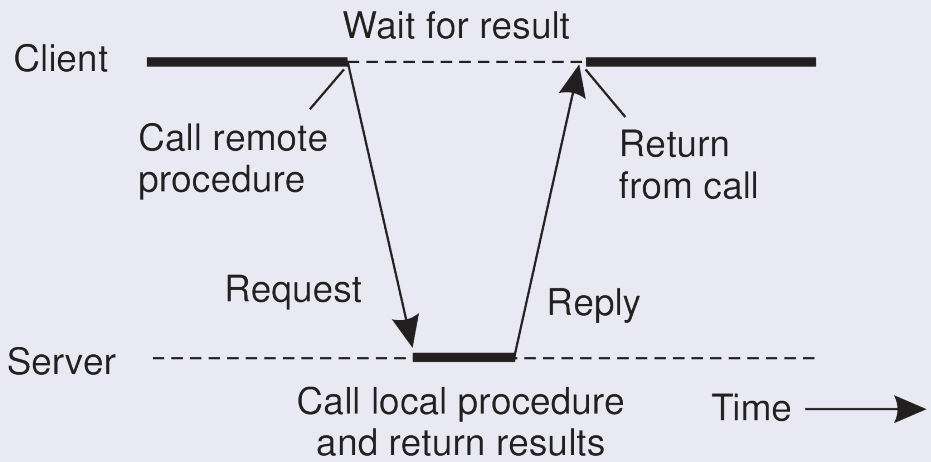
Por que existe?

- *Send* e *receive* não escondem a comunicação
  - Violam a transparência de acesso
- Algumas observações:
  - Os desenvolvedores estão familiarizados com o modelo de procedimentos
  - Procedimentos bem projetados operam isoladamente (*black box*)
  - Então não há razão para não executar esses procedimentos em máquinas separadas

Chamada a Procedimento Remoto (RPC)

Proposta

- Permitir que programas chamem procedimentos localizados em outras máquinas
  - Escondendo assim a comunicação entre chamador & chamado

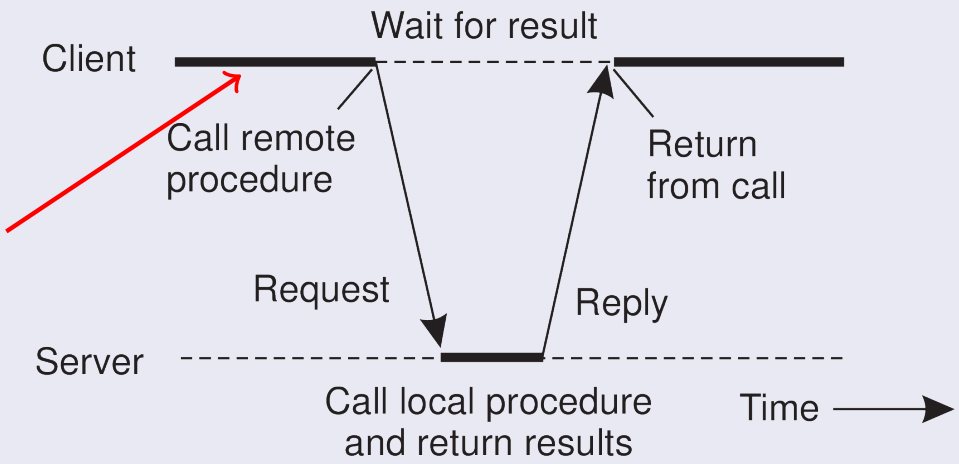


Chamada a Procedimento Remoto (RPC)

Proposta

- Permitir que programas chamem procedimentos localizados em outras máquinas
  - Escondendo assim a comunicação entre chamador & chamado

Quando um processo na máquina A chama um procedimento na máquina B, o processo chamador é suspenso

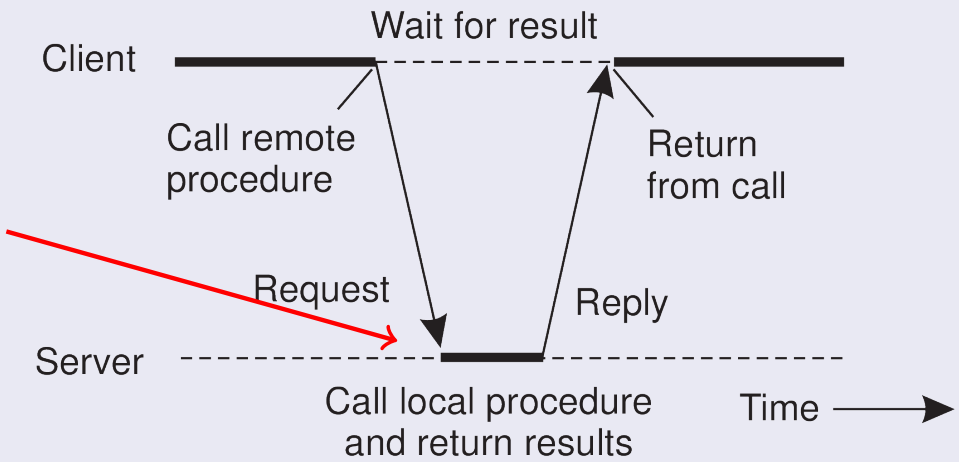


Chamada a Procedimento Remoto (RPC)

Proposta

- Permitir que programas chamem procedimentos localizados em outras máquinas
  - Escondendo assim a comunicação entre chamador & chamado

E a execução do procedimento chamado ocorre em B

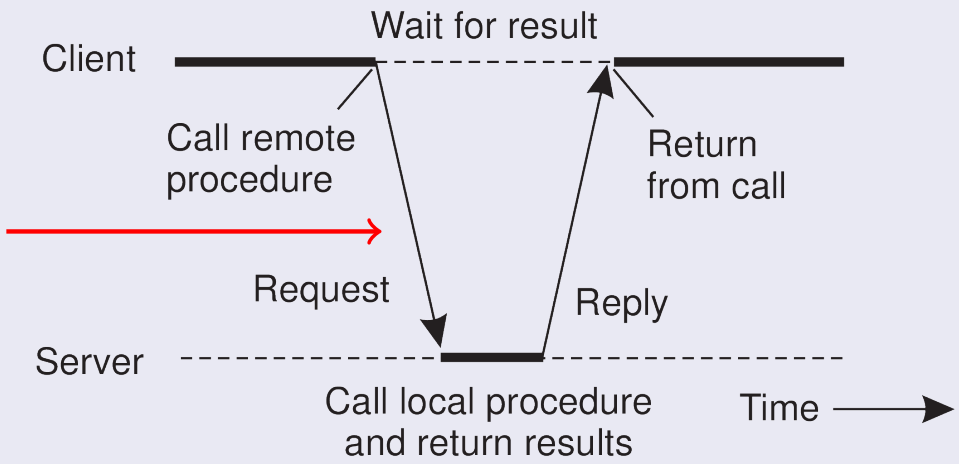


Chamada a Procedimento Remoto (RPC)

Proposta

- Permitir que programas chamem procedimentos localizados em outras máquinas
  - Escondendo assim a comunicação entre chamador & chamado

Informação é transmitida do chamador ao chamado via parâmetros, e retorna ao chamador no resultado do procedimento

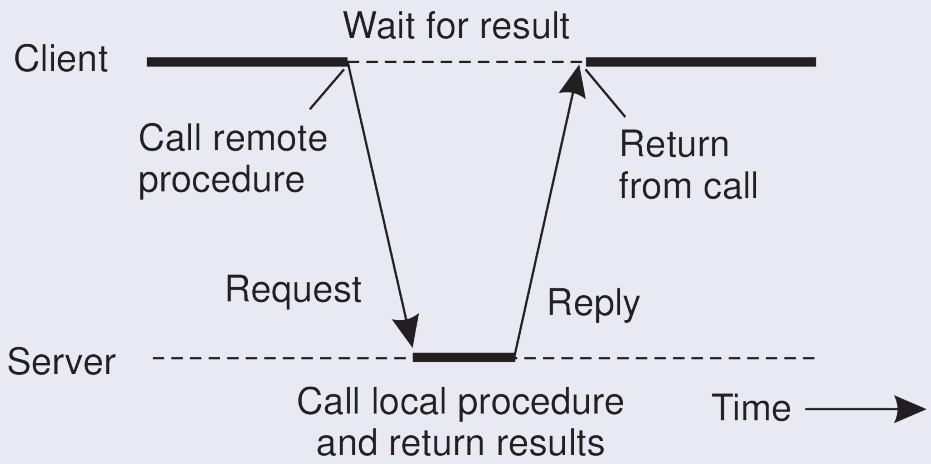


Chamada a Procedimento Remoto (RPC)

Proposta

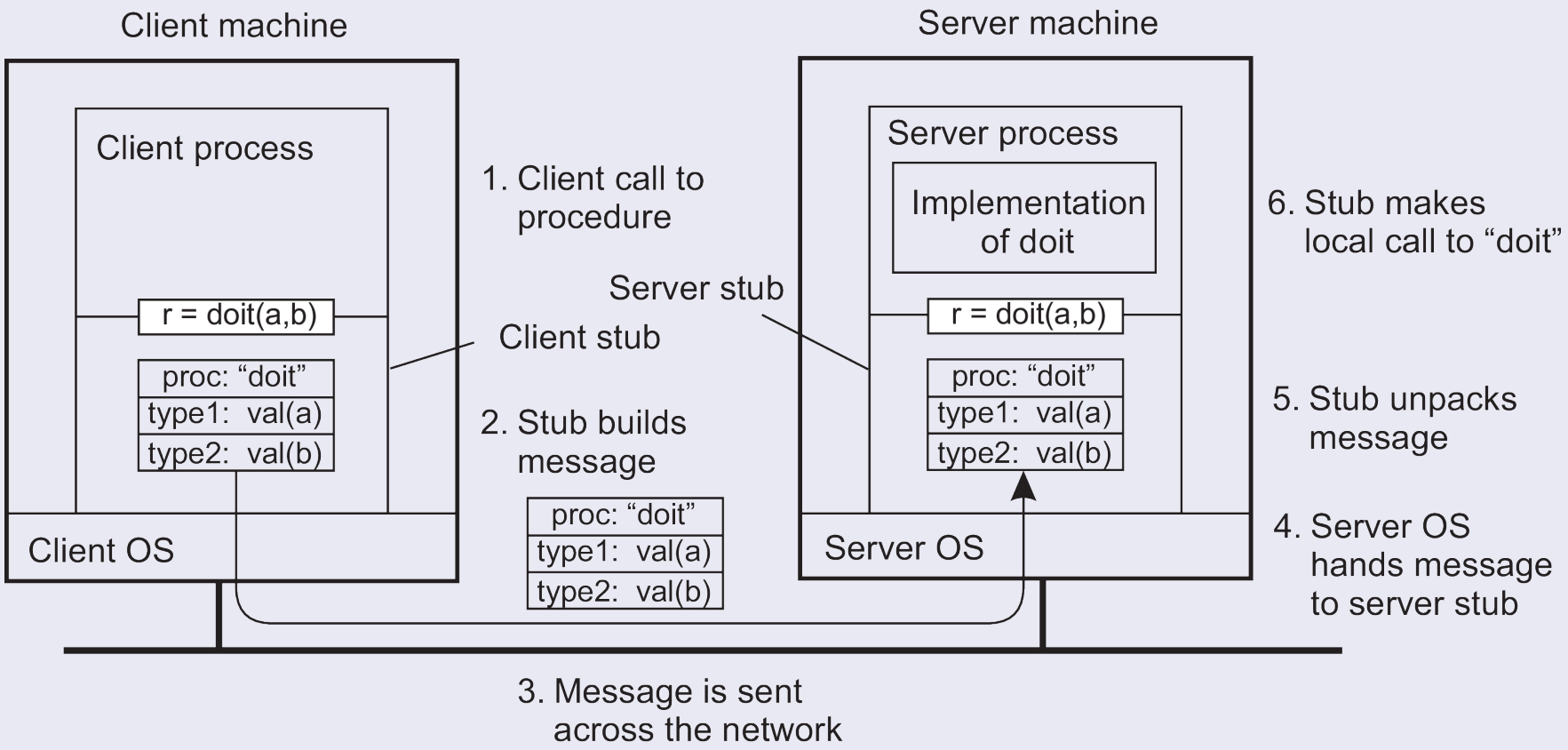
- Permitir que programas chamem procedimentos localizados em outras máquinas
  - Escondendo assim a comunicação entre chamador & chamado

Nenhuma passagem de mensagem é visível ao programador



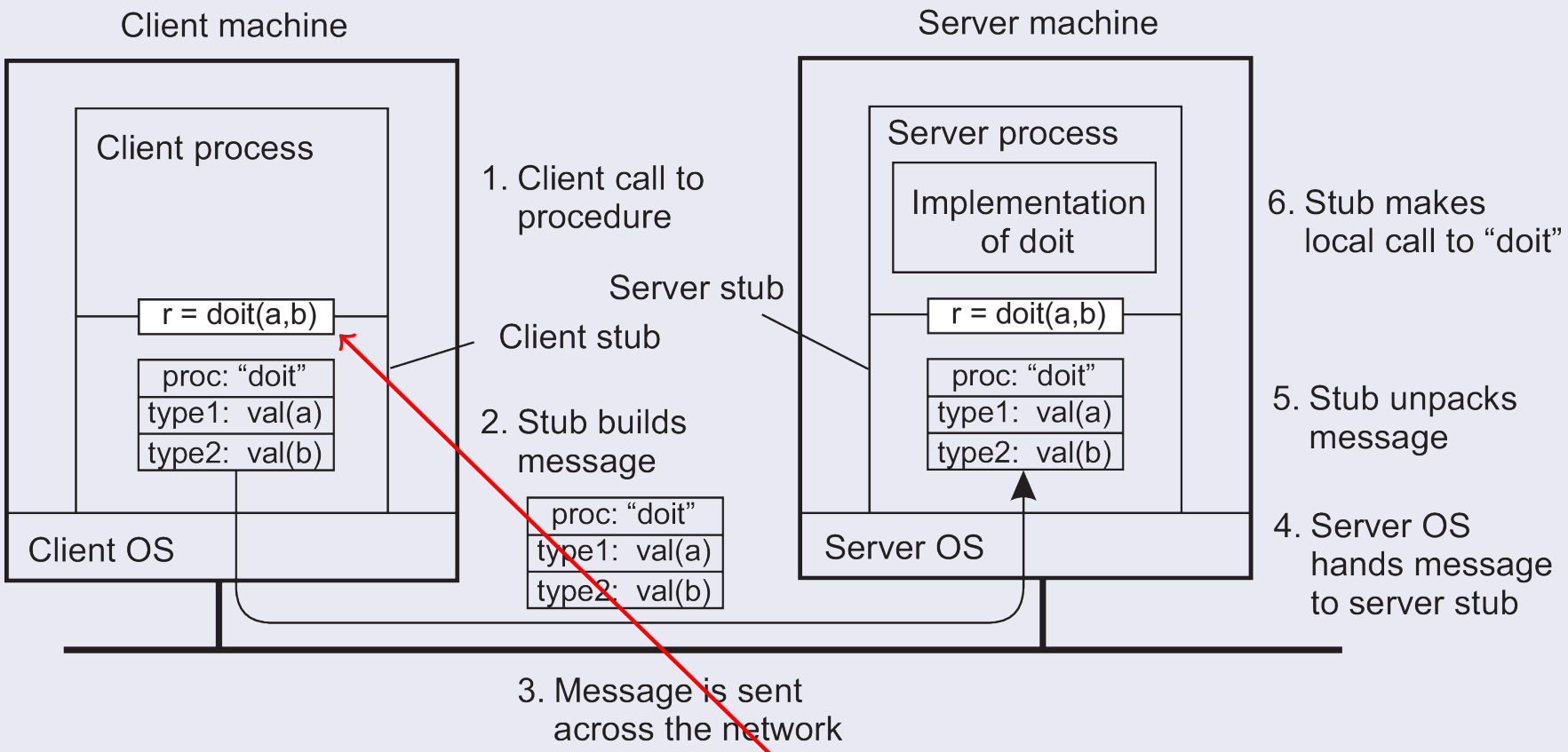
Chamada a Procedimento Remoto (RPC)

Funcionamento básico



Chamada a Procedimento Remoto (RPC)

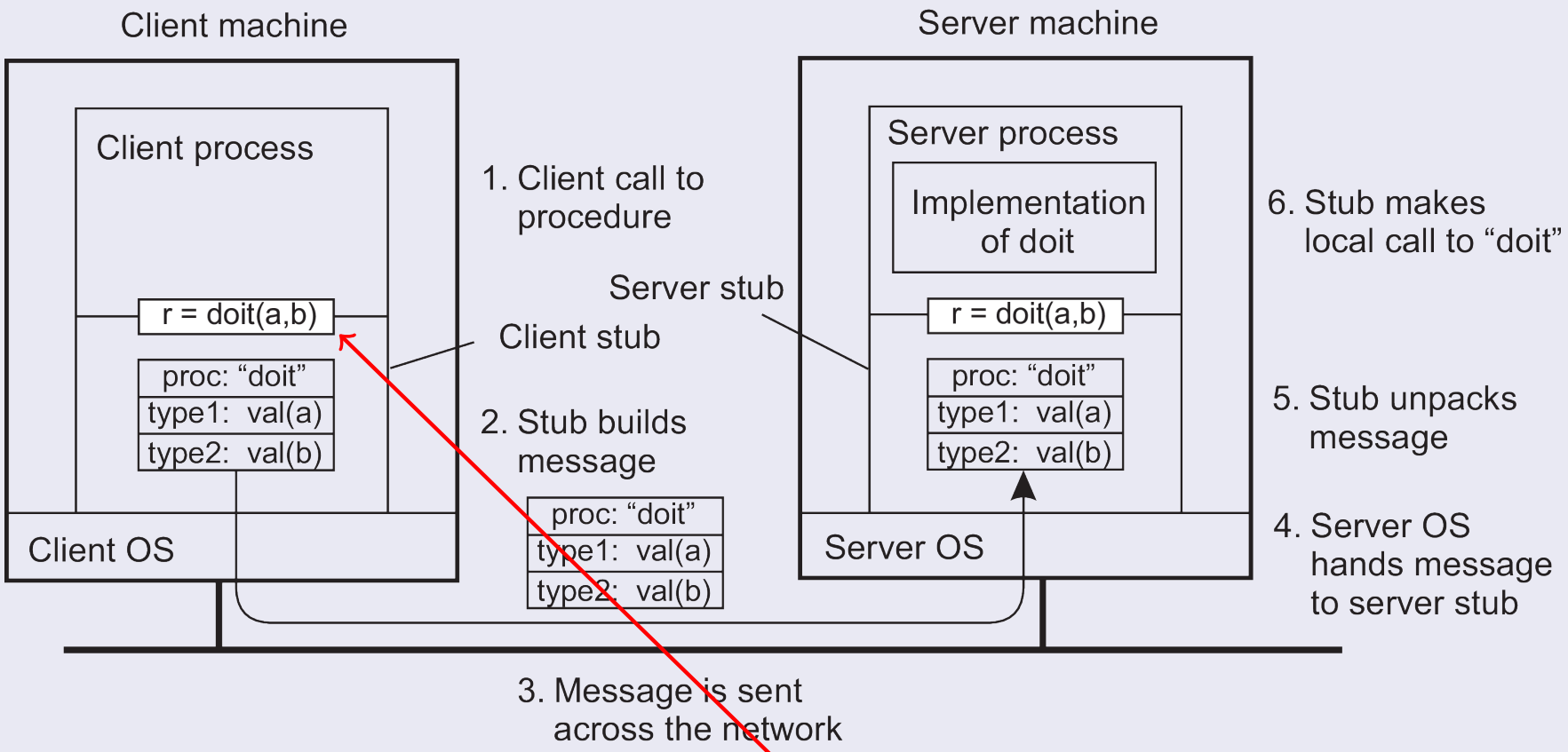
Funcionamento básico



1. Uma versão diferente do procedimento – o **stub do cliente** – é oferecida ao cliente

Chamada a Procedimento Remoto (RPC)

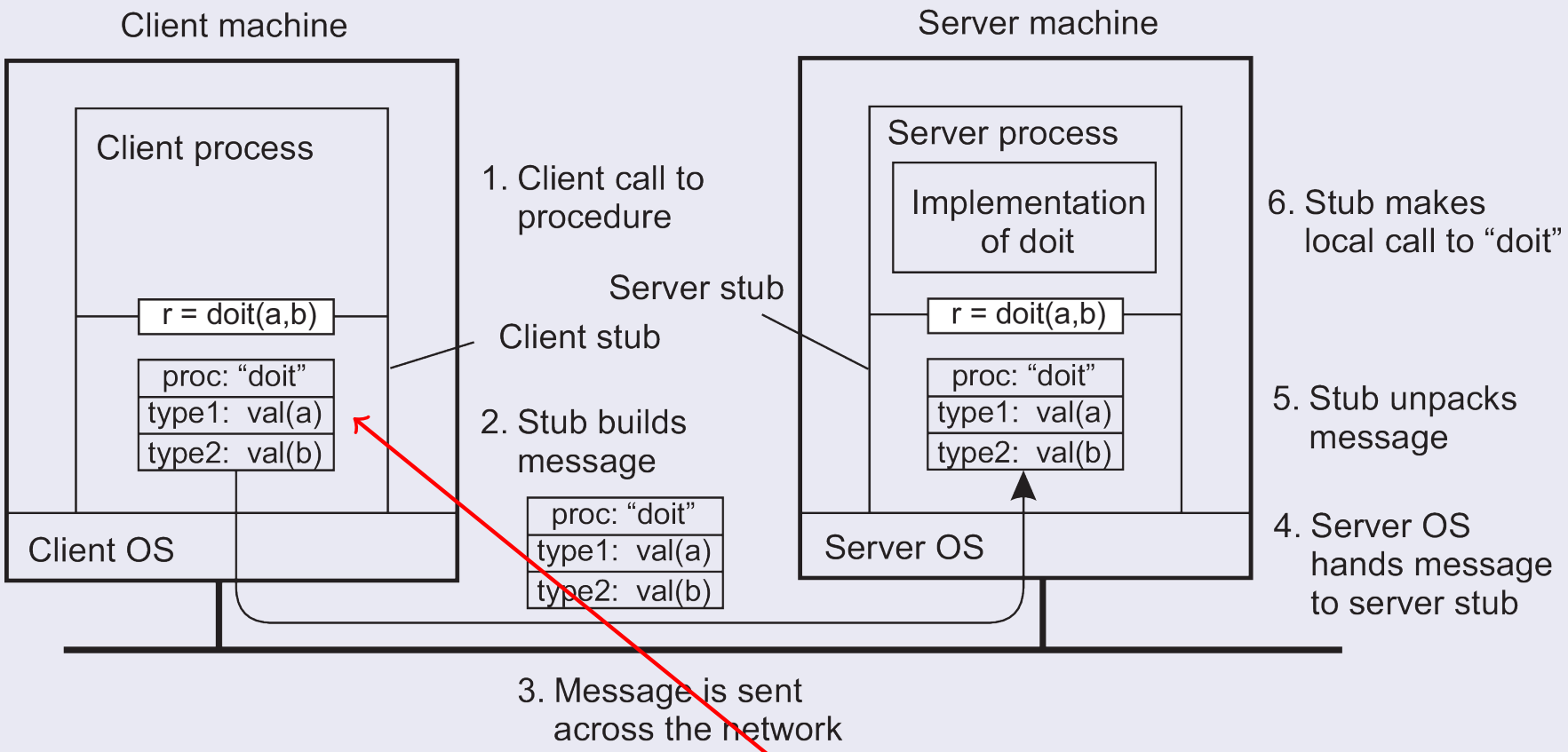
Funcionamento básico



O procedimento no cliente chama então esse *stub*, crendo que está chamando o procedimento

Chamada a Procedimento Remoto (RPC)

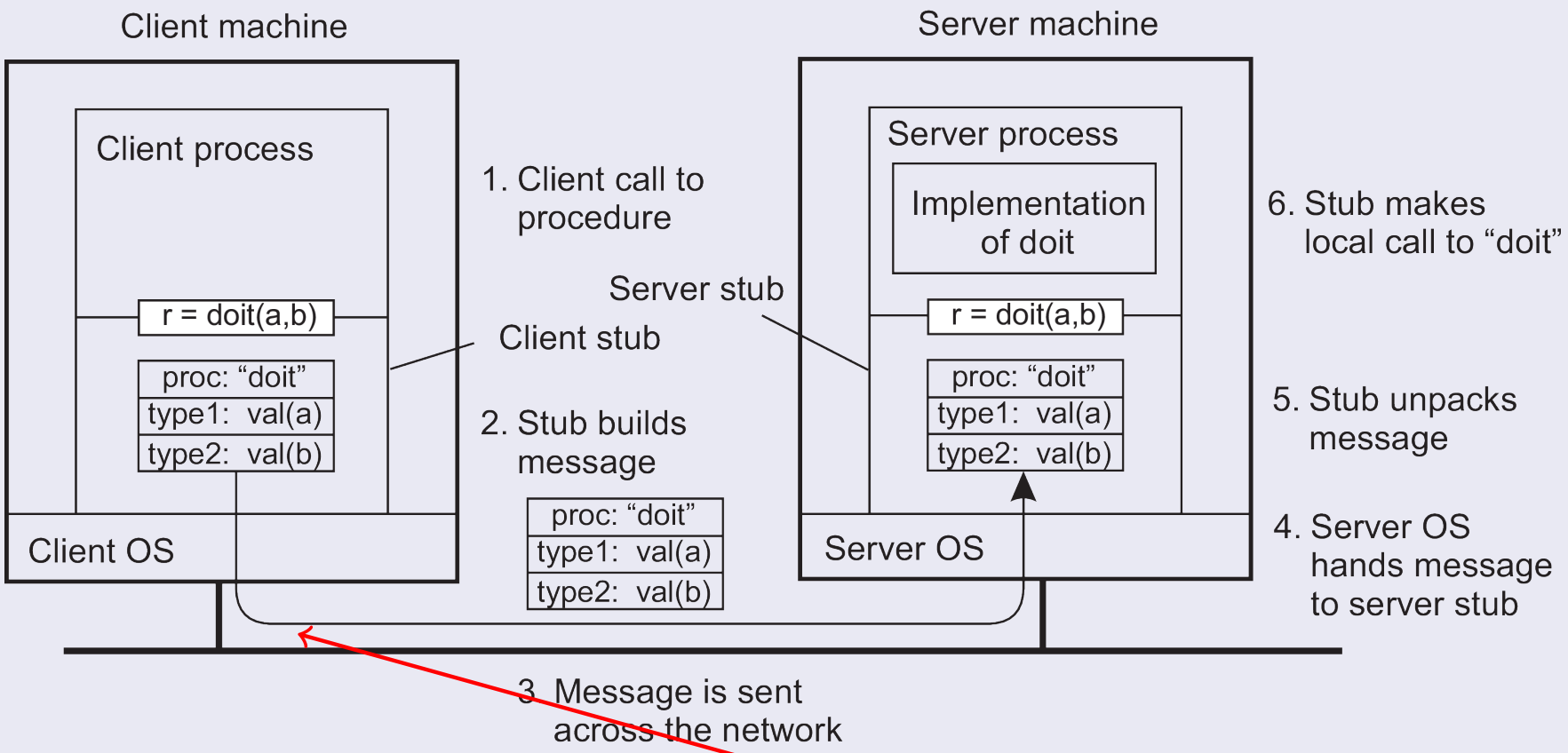
Funcionamento básico



2. O *stub* empacota os parâmetros em uma mensagem, chamando o SO para enviá-la, e bloqueando até que a resposta seja retornada

Chamada a Procedimento Remoto (RPC)

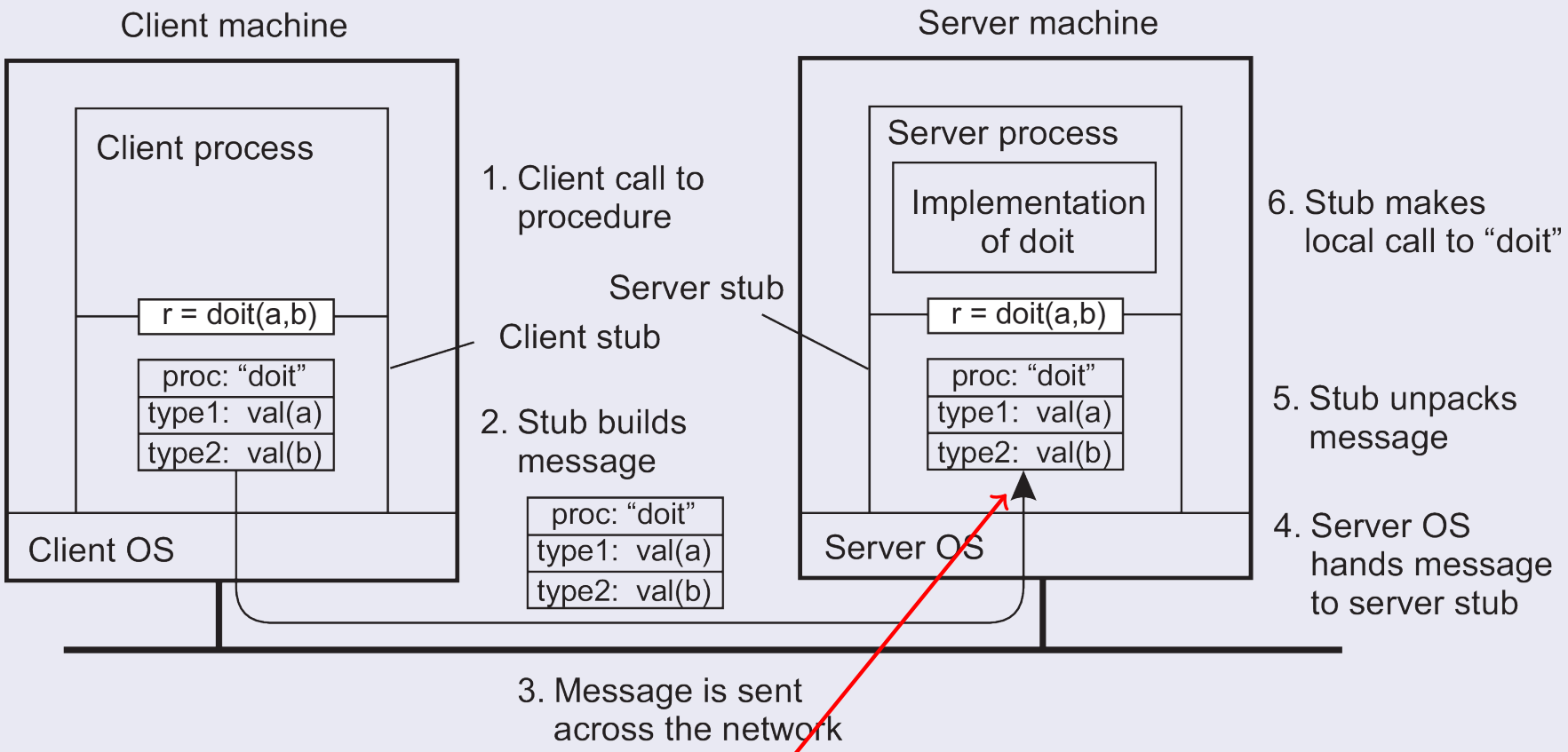
Funcionamento básico



3. O SO do cliente envia a mensagem para o SO remoto

Chamada a Procedimento Remoto (RPC)

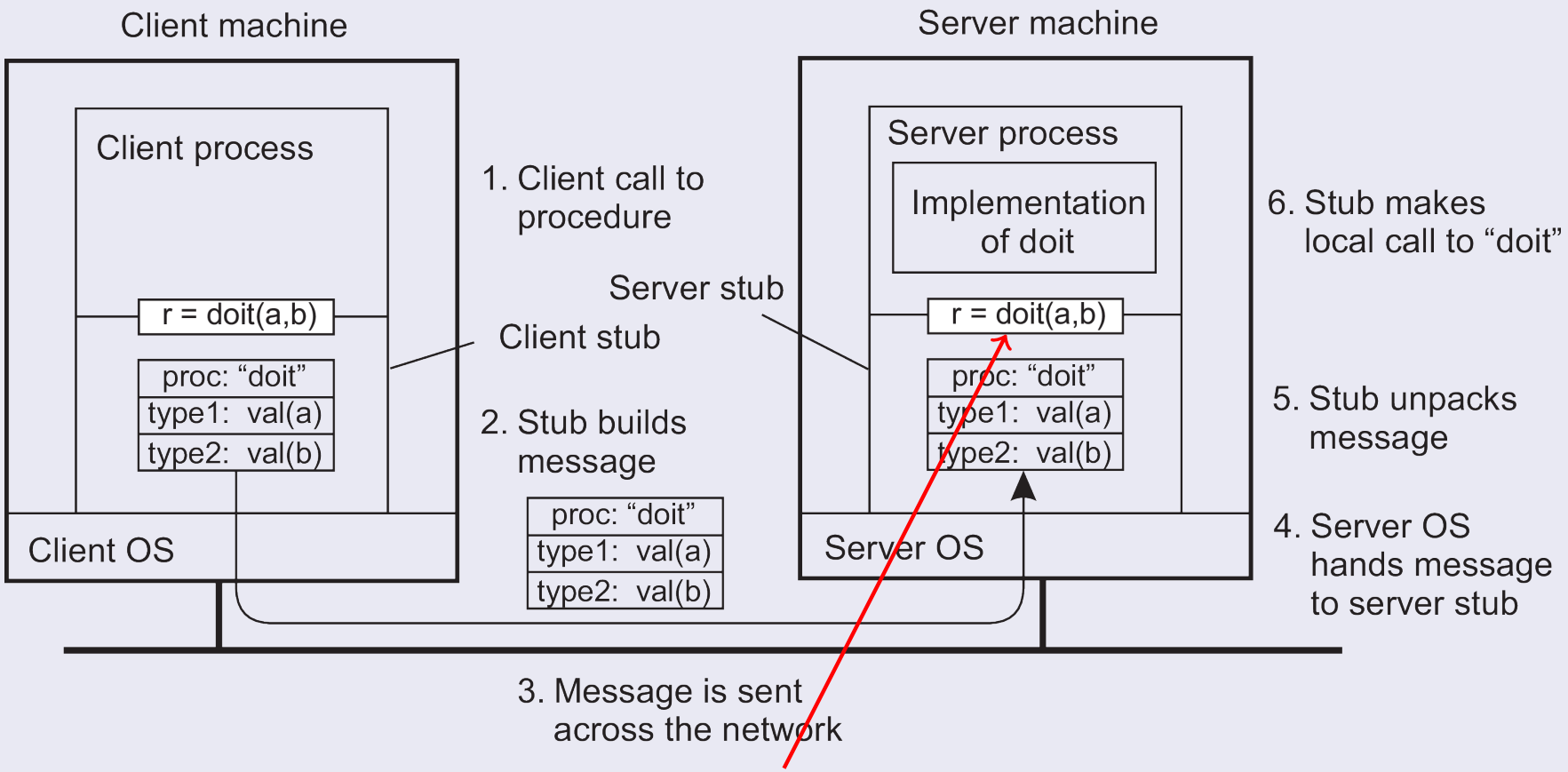
Funcionamento básico



4. Quando a mensagem chega, o SO remoto a repassa ao **stub do servidor**

Chamada a Procedimento Remoto (RPC)

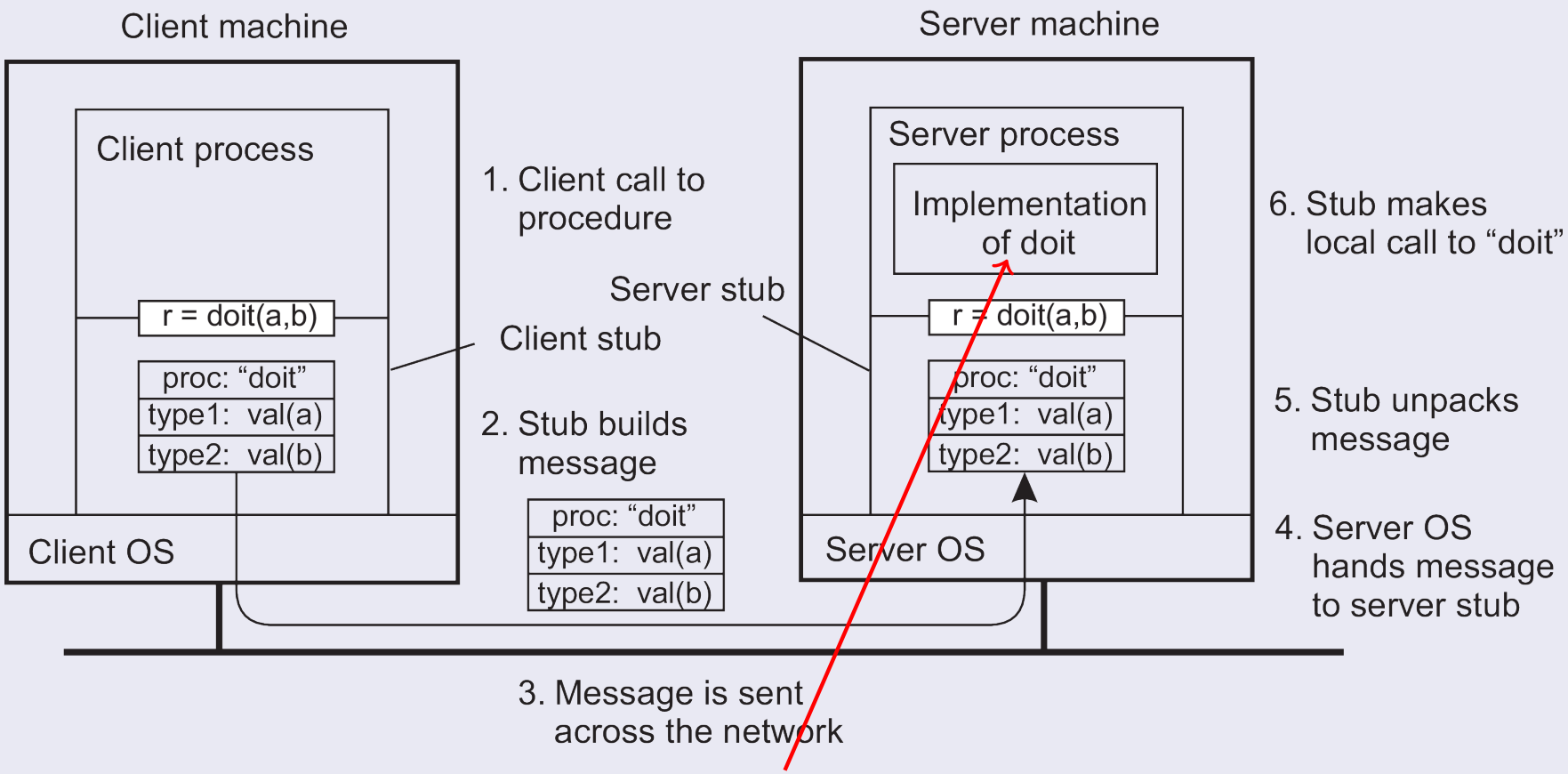
Funcionamento básico



5. O *stub* do servidor desempacota os parâmetros da mensagem e então chama o procedimento correspondente no servidor

Chamada a Procedimento Remoto (RPC)

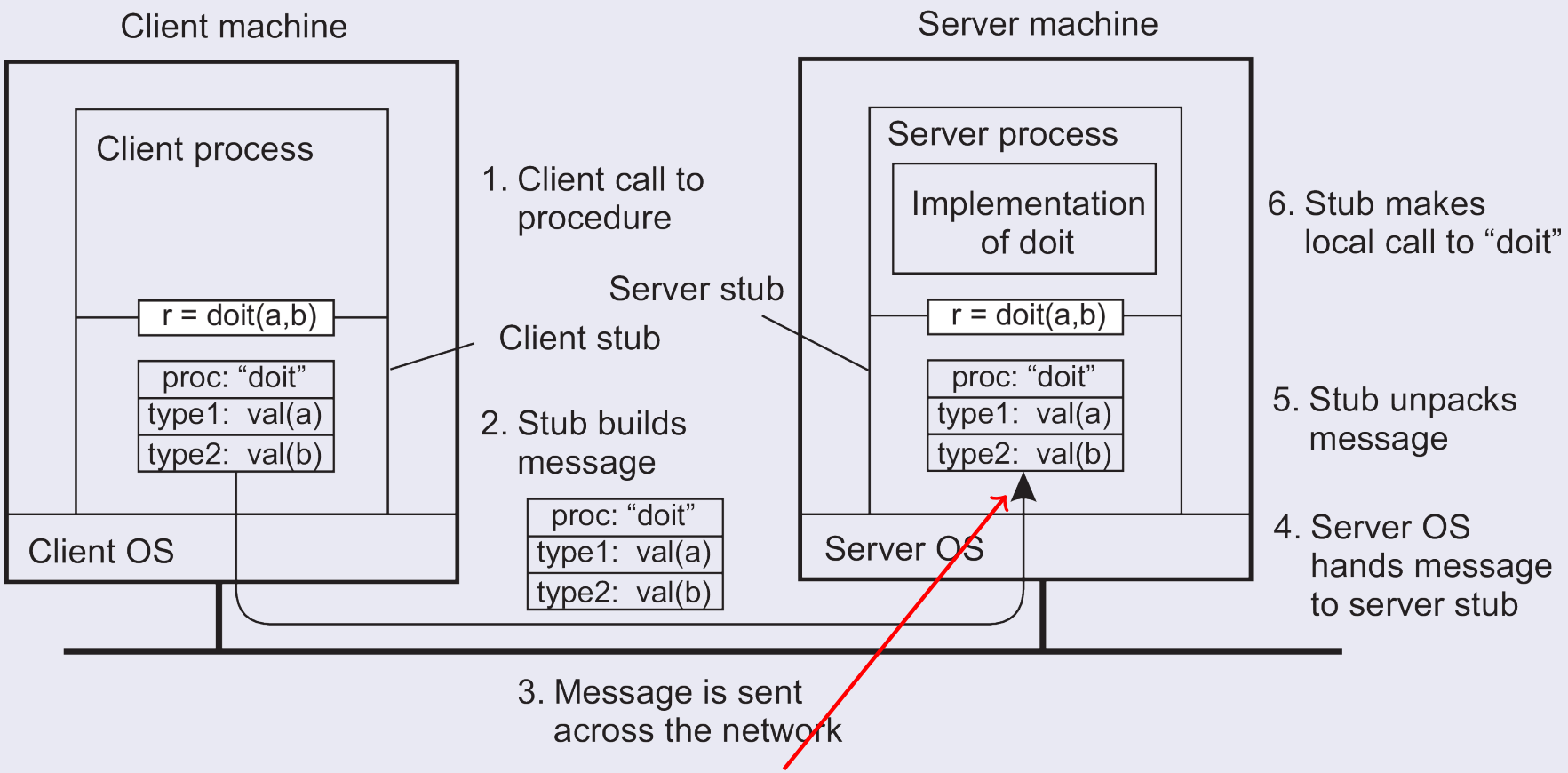
Funcionamento básico



6. O procedimento é rodado no servidor e retorna ao chamador (no caso, o *stub* do servidor)

Chamada a Procedimento Remoto (RPC)

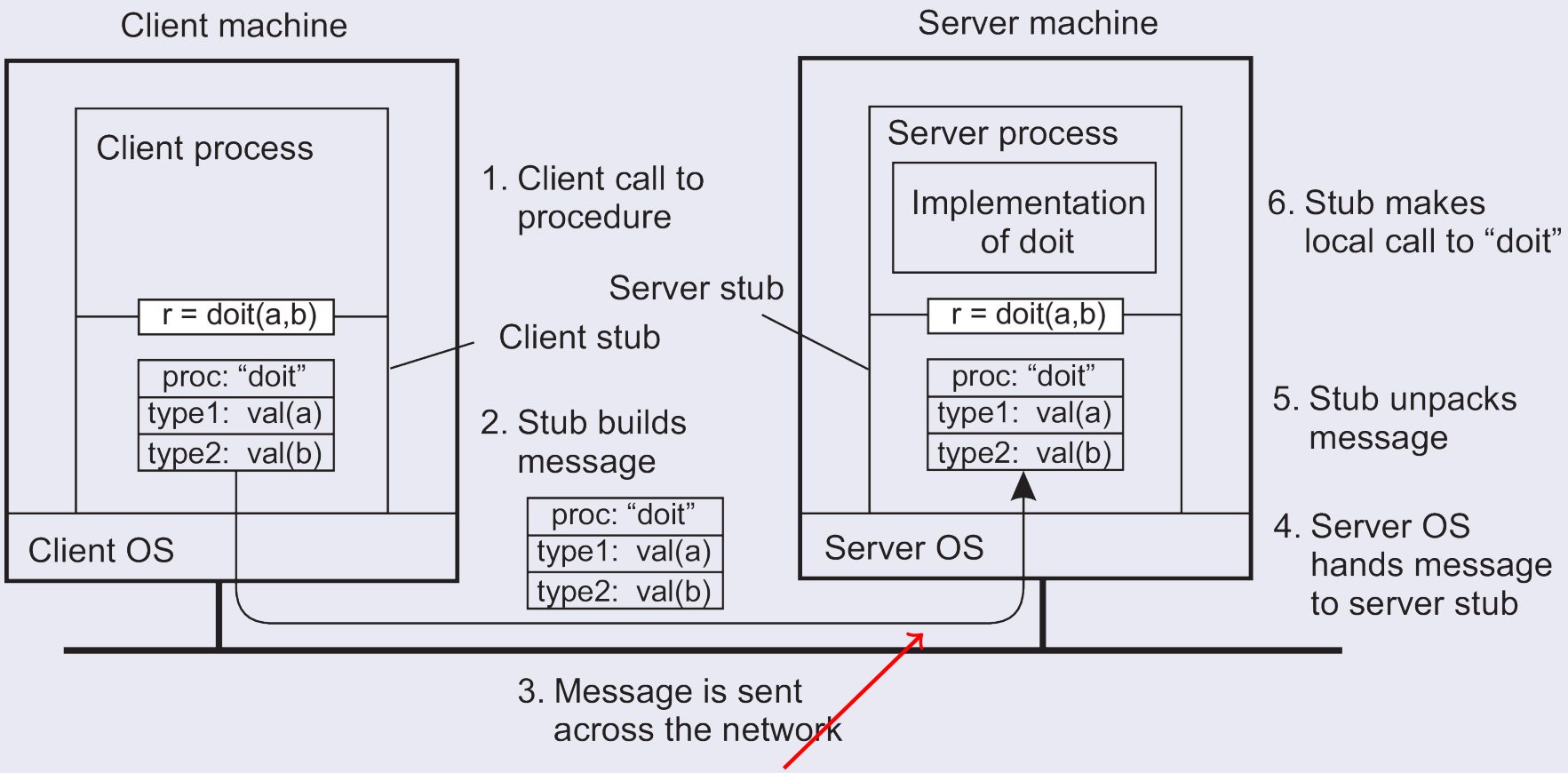
Funcionamento básico



7. O *stub* do servidor empacota o resultado em uma mensagem e chama o SO local para enviá-la

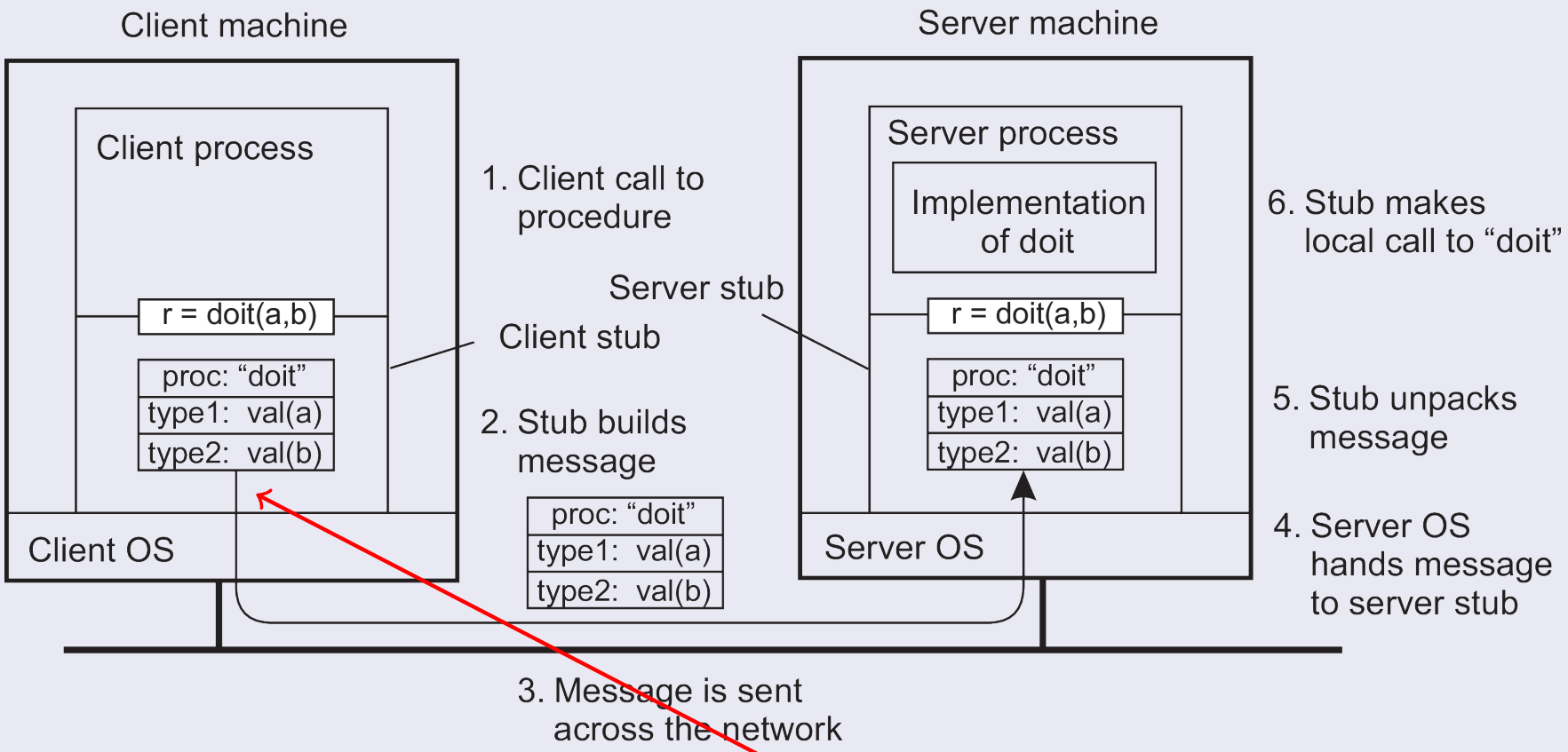
Chamada a Procedimento Remoto (RPC)

Funcionamento básico



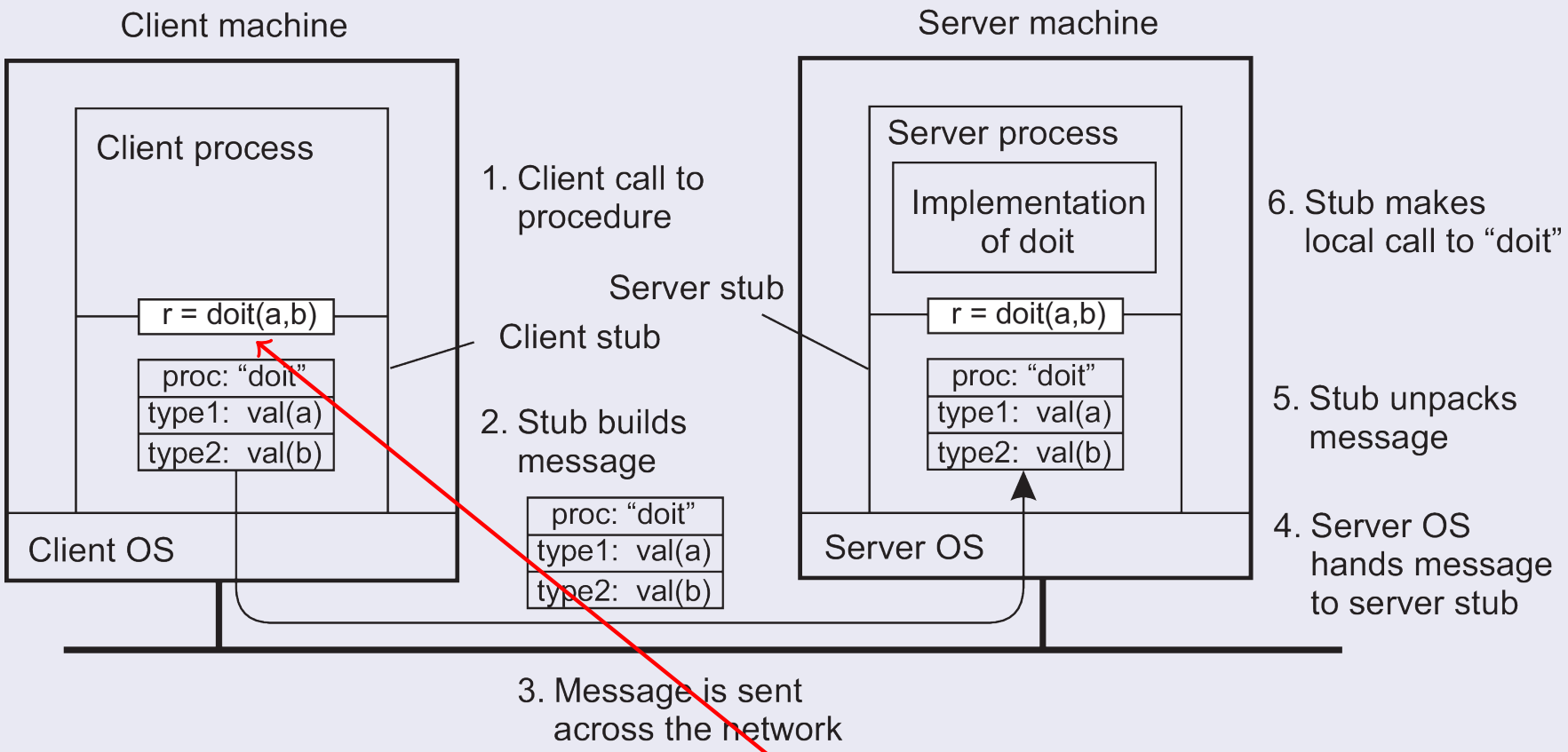
Chamada a Procedimento Remoto (RPC)

Funcionamento básico



Chamada a Procedimento Remoto (RPC)

Funcionamento básico



10. O cliente é desbloqueado. O *stub* desempacota o resultado e o retorna ao cliente, como se fosse um retorno de procedimento comum

Chamada a Procedimento Remoto (RPC)

Passagem de parâmetros

- Não basta apenas empacotar parâmetros em uma mensagem
  - As máquinas cliente e servidor podem ter representação de dados diferentes (ex: ordem dos bytes)
  - Empacotar um parâmetro significa transformar um valor em uma sequência de bytes
  - Cliente e servidor precisam concordar com a mesma regra de codificação (*encoding*):
    - Como os valores dos dados básicos (inteiros, números em ponto flutuante, caracteres) são representados?
    - Como os valores de dados complexos (arranjos, *unions*) são representados?

Chamada a Procedimento Remoto (RPC)

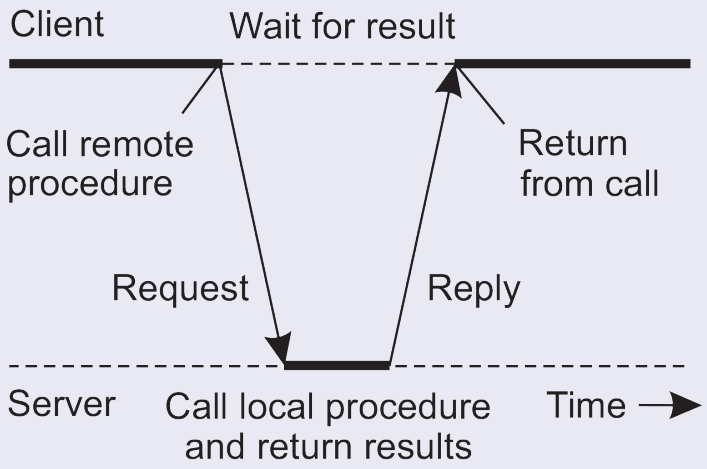
Passagem de parâmetros

- Cliente e servidor precisam interpretar corretamente as mensagens,
  - Transformando-as em representações dependentes da máquina

Chamada a Procedimento Remoto (RPC)

RPC assíncrona

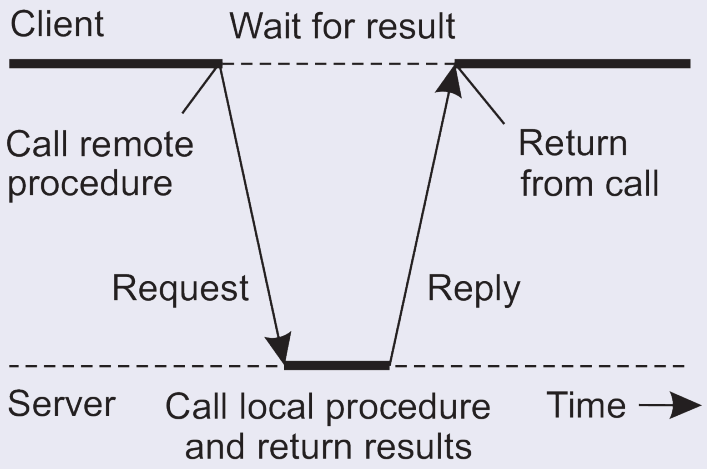
Na RPC tradicional, quando um cliente faz uma chamada ele bloqueia até obter a resposta



Chamada a Procedimento Remoto (RPC)

RPC assíncrona

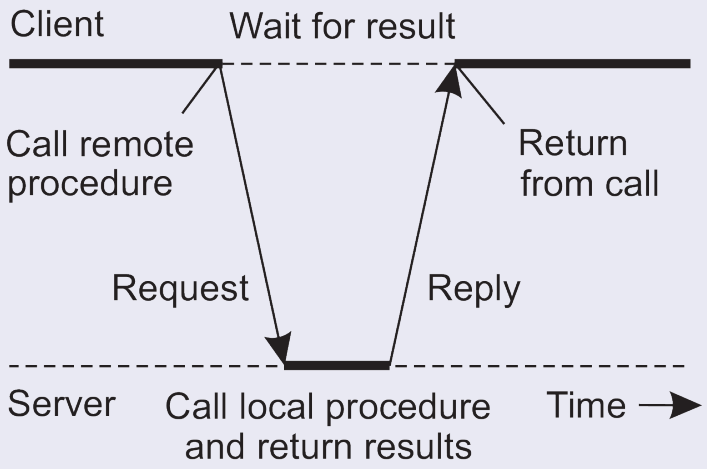
Mas e se não houver resultado a ser retornado, o que fazer?



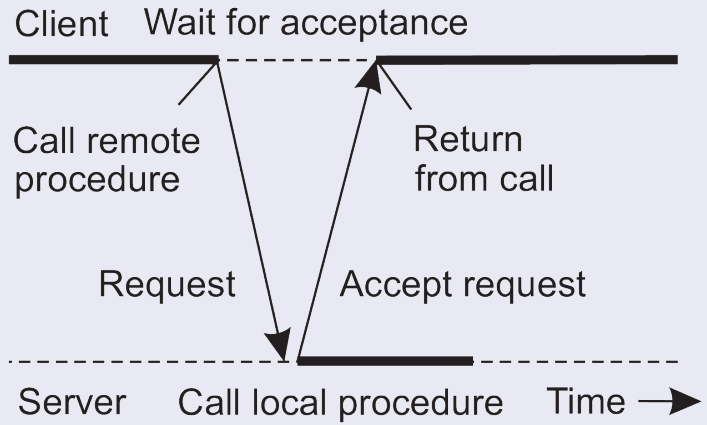
Chamada a Procedimento Remoto (RPC)

RPC assíncrona

Mas e se não houver resultado a ser retornado, o que fazer?



RPC assíncrona: o servidor envia uma resposta no instante do recebimento da mensagem

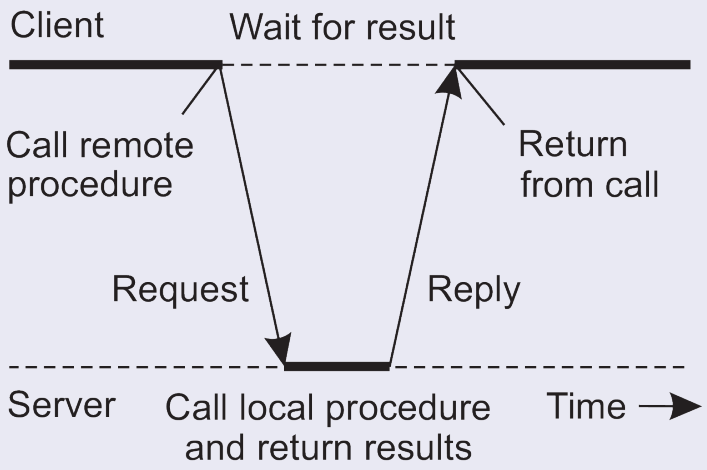


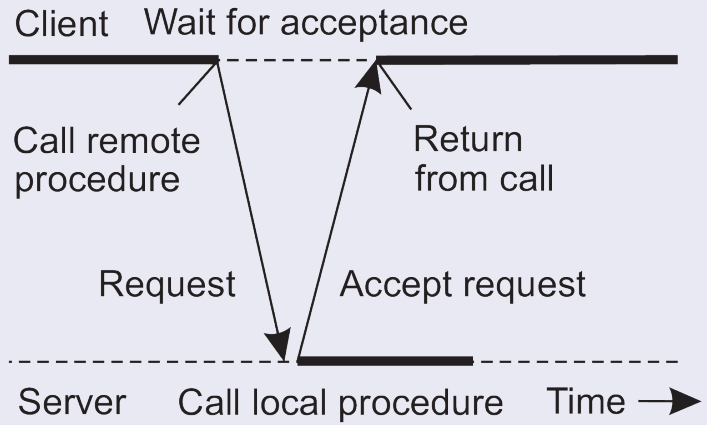
Chamada a Procedimento Remoto (RPC)

RPC assíncrona

Mas e se não houver resultado a ser retornado, o que fazer?

Só então chama o procedimento requisitado (a resposta serve como um *acknowledge*)



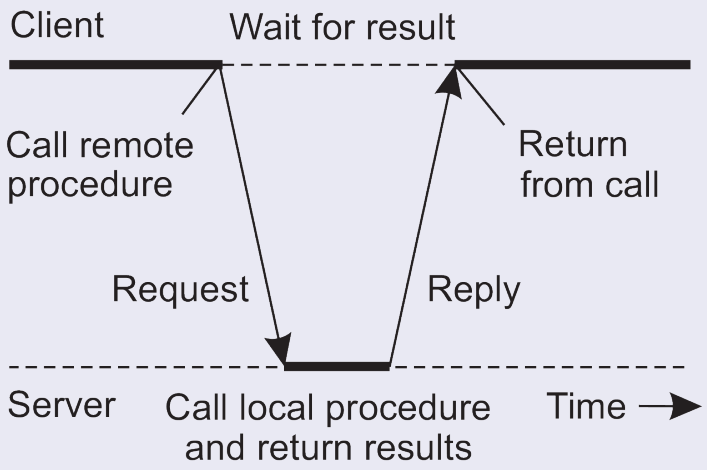


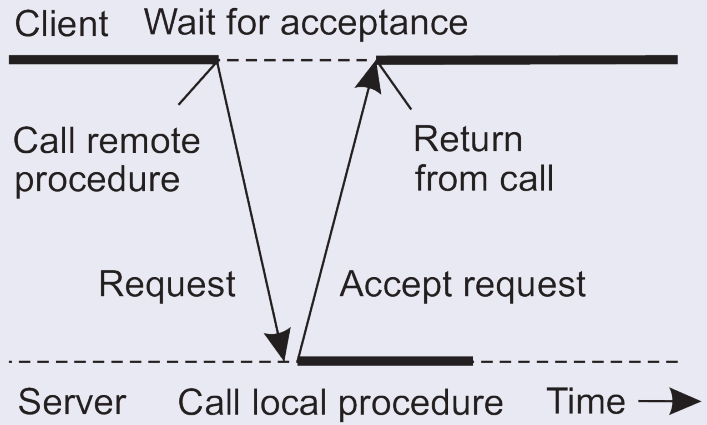
Chamada a Procedimento Remoto (RPC)

RPC assíncrona

Mas e se não houver resultado a ser retornado, o que fazer?

O cliente, ao receber a resposta, pode seguir seu processamento

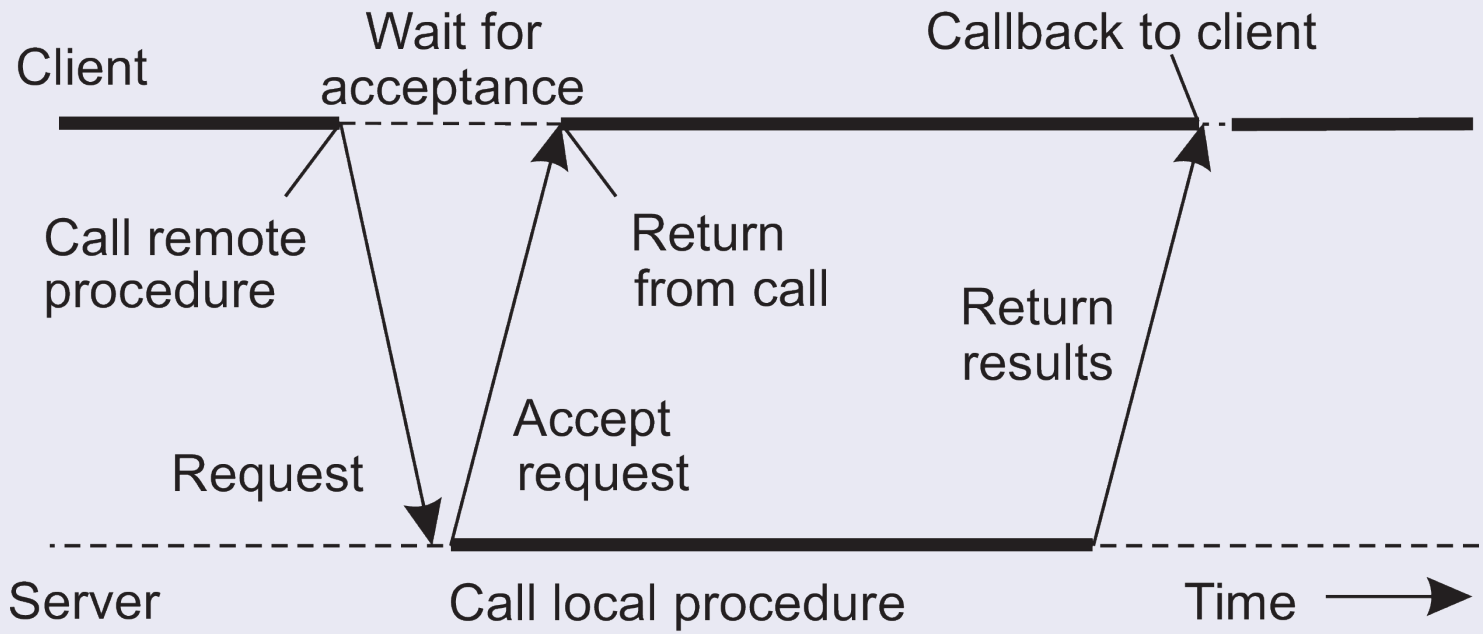




Chamada a Procedimento Remoto (RPC)

RPC síncrona diferida

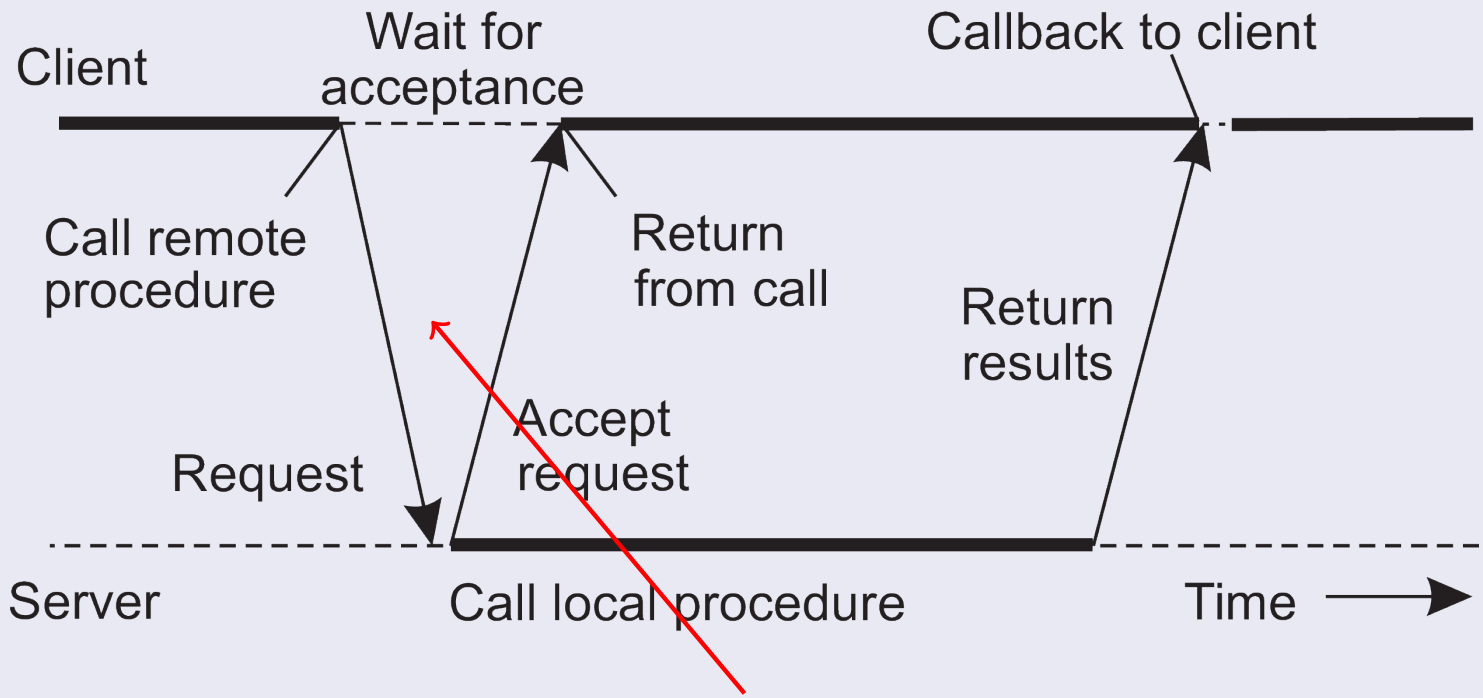
- Podemos combinar RPC assíncrona com *callback*:



Chamada a Procedimento Remoto (RPC)

RPC síncrona diferida

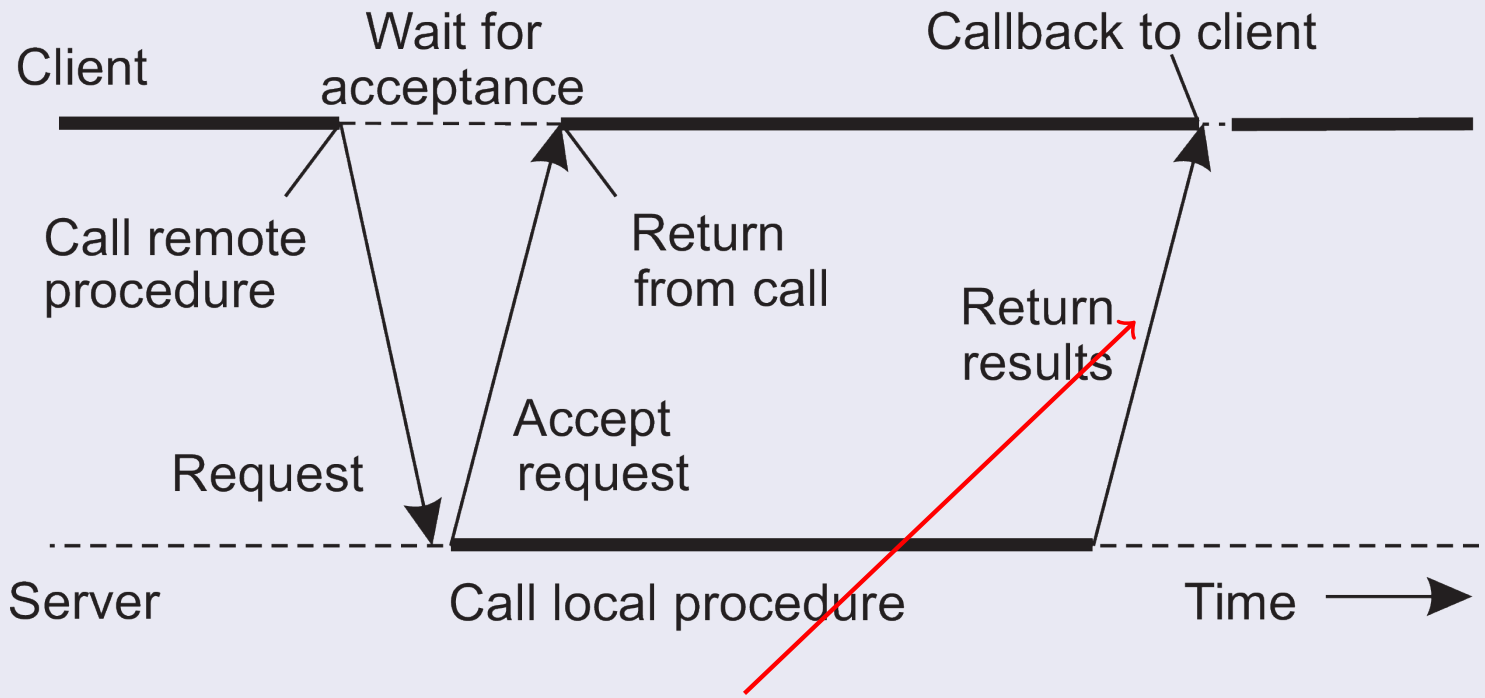
- Podemos combinar RPC assíncrona com *callback*:



Chamada a Procedimento Remoto (RPC)

RPC síncrona diferida

- Podemos combinar RPC assíncrona com *callback*:

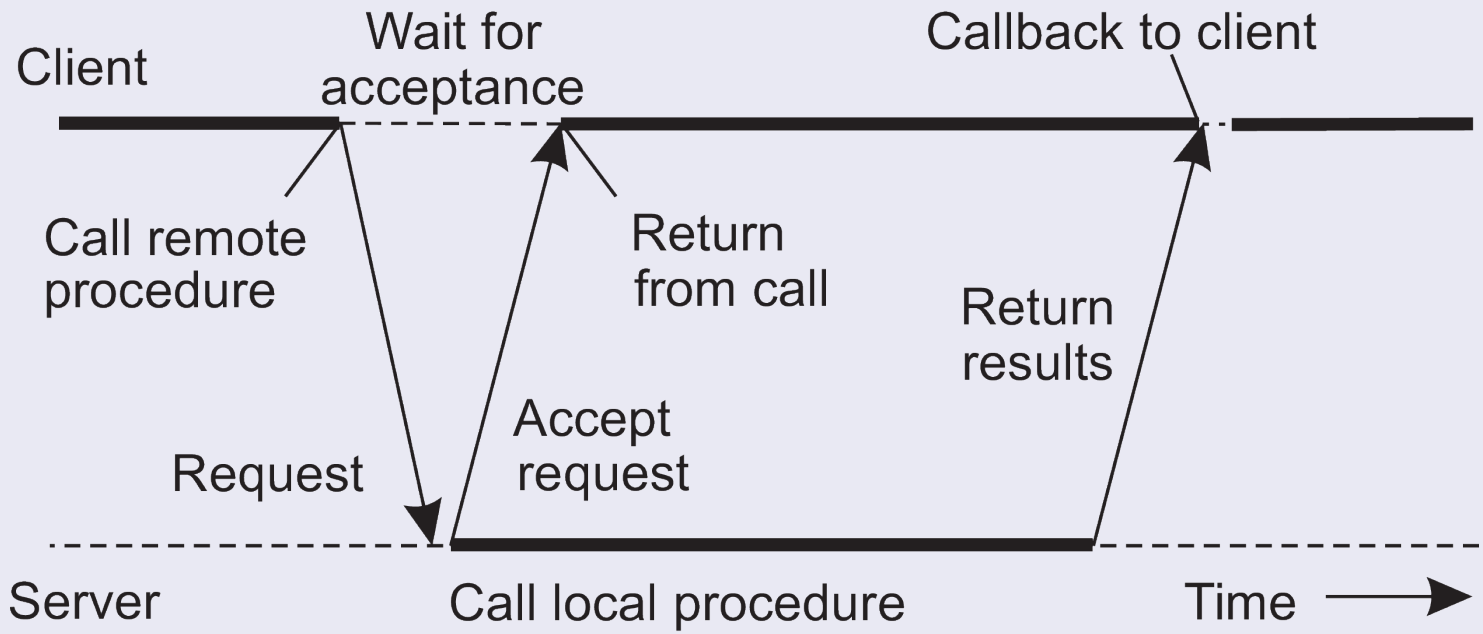


Quando os resultados estão prontos, o servidor envia uma mensagem de resposta, levando a um *callback* no cliente

Chamada a Procedimento Remoto (RPC)

RPC síncrona diferida

- Podemos combinar RPC assíncrona com *callback*:



Esse *callback* é uma função definida pelo usuário, que é invocada quando um evento especial ocorre, como a chegada de uma mensagem