ACH 2147 — DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO DISTRIBUÍDOS

COMUNICAÇÃO

Daniel Cordeiro

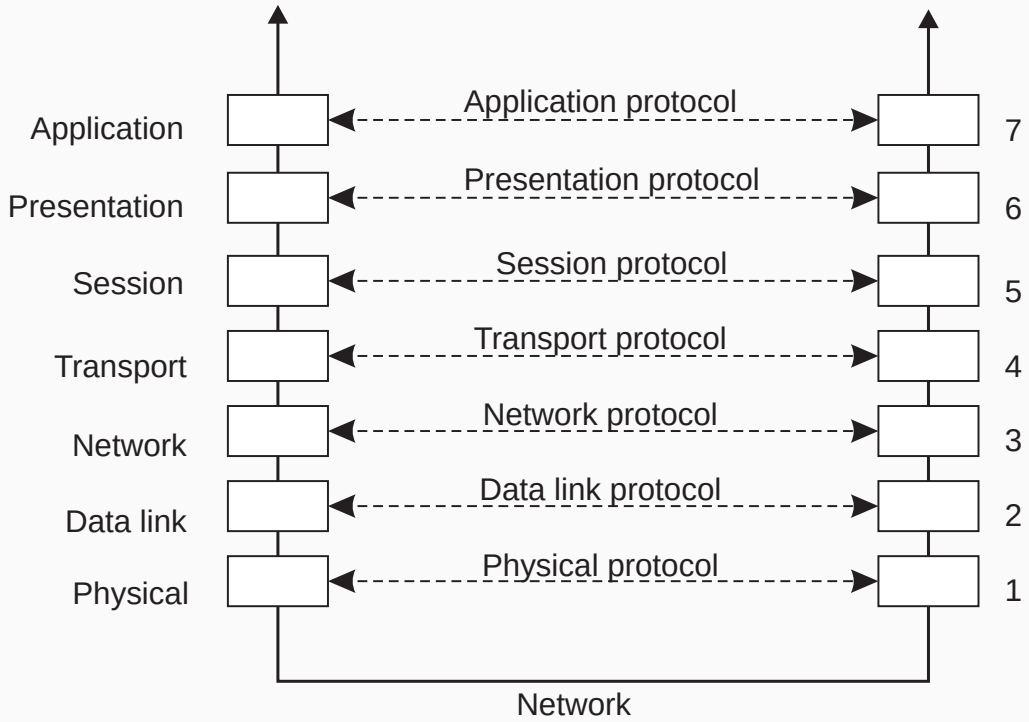
18 e 20 de abril de 2018

Escola de Artes, Ciências e Humanidades | EACH | USP

PROTOCOLOS EM CAMADAS

- Camadas de baixo nível
- Camada de transporte
- Camada de aplicação
- Camada do middleware

MODELO DE COMUNICAÇÃO BÁSICO



Desvantagens:

- Funciona apenas com passagem de mensagens
- Frequentemente possuem funcionalidades desnecessárias
- Viola a transparência de acesso

CAMADAS DE BAIXO NÍVEL

Camada física: contém a especificação e implementação dos bits em um quadro, e como são transmitidos entre o remetente e destinatário

Camada de enlace: determina o envio de séries de bits em um quadro, permite detecção de erro e controle de fluxo

Camada de rede: determina como pacotes são roteados em uma rede de computadores

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

Importante:
A camada de transporte fornece as ferramentas de comunicação efetivamente utilizadas pela maioria dos sistemas distribuídos.

Protocolos padrões da Internet

TCP: orientada a conexão, confiável, comunicação orientada a fluxo de dados

UDP: comunicação de datagramas não confiável (*best-effort*)

Nota:
IP multicasting é normalmente considerado um serviço padrão (mas essa é uma hipótese perigosa)

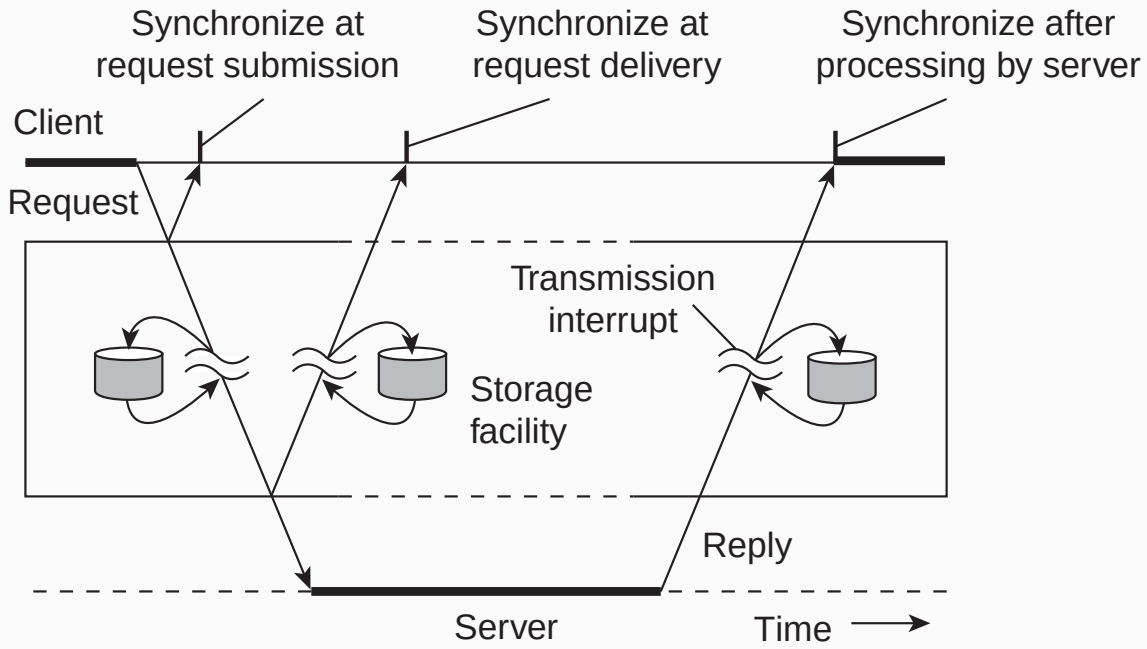
CAMADA DE MIDDLEWARE

Middleware foi inventado para prover serviços e protocolos frequentemente usados 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

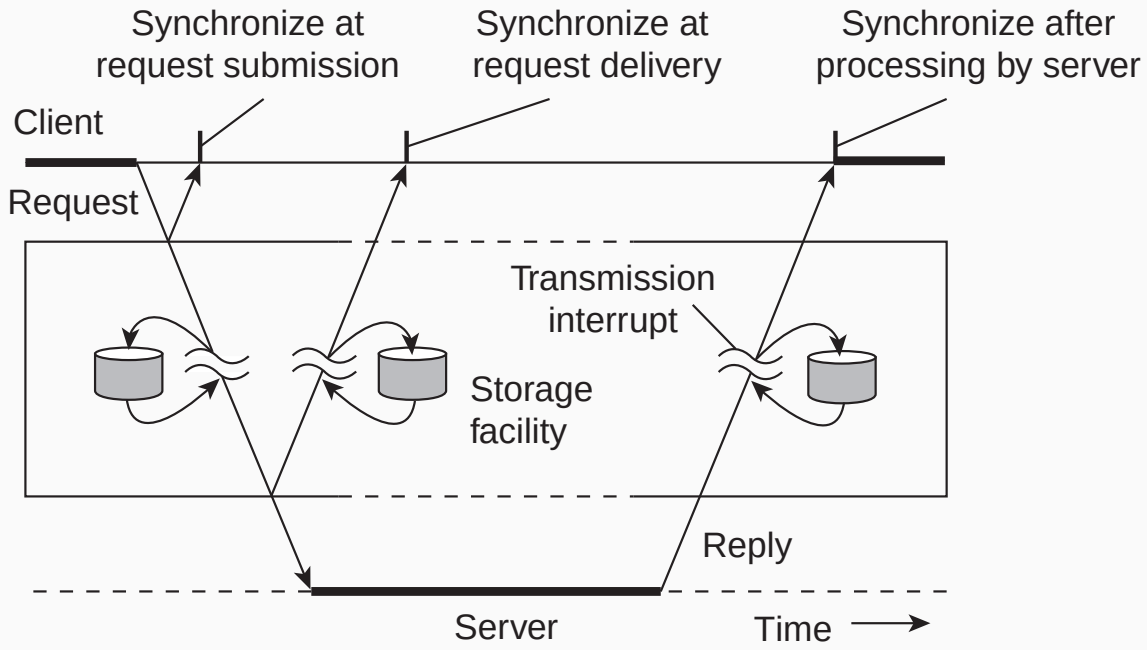
Observação:
O que realmente sobra são protocolos específicos de aplicação.

TIPOS DE COMUNICAÇÃO



- Comunicação transiente vs. persistente
- Comunicação assíncrona vs. síncrona

TIPOS DE COMUNICAÇÃO

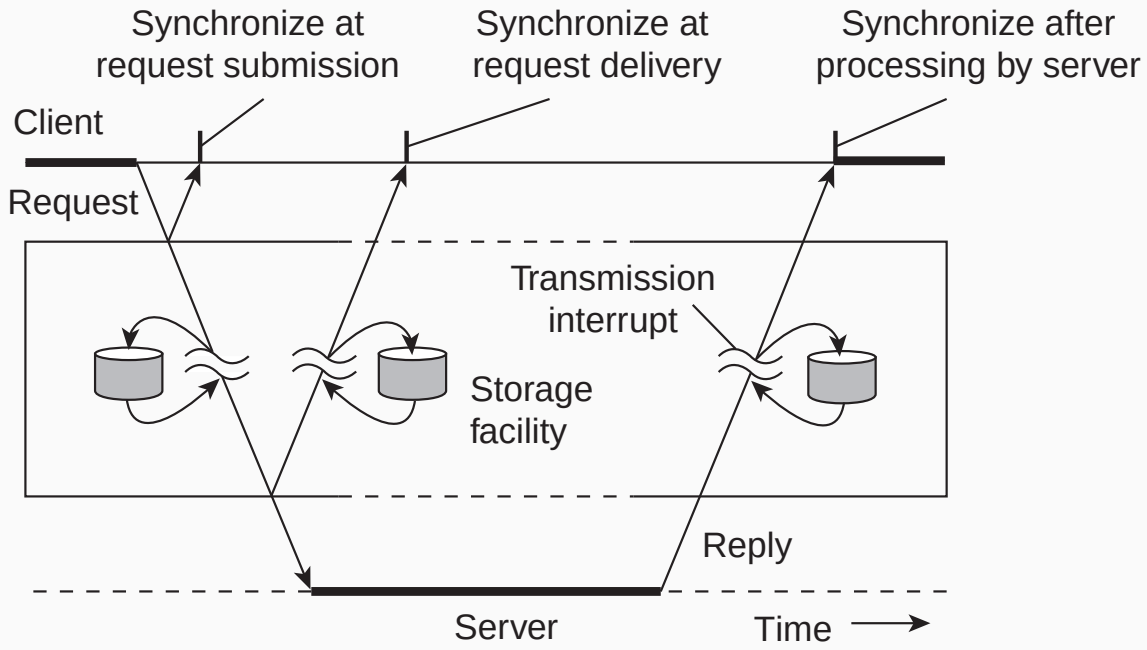


Transiente vs. persistente

Comunicação transiente: remetente descarta a mensagem se ela não puder ser encaminhada para o destinatário

Comunicação persistente: uma mensagem é guardada no remetente pelo tempo que for necessário, até ser entregue no destinatário

TIPOS DE COMUNICAÇÃO



Pontos de sincronização

- No envio da requisição
- Na entrega da requisição
- Após o processamento da requisição

CLIENTE/SERVIDOR

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

- Cliente e servidor devem estar ativos no momento da comunicação
- Cliente envia uma requisição e bloqueia até que receba sua resposta
- Servidor essencialmente espera por requisições e as processa

CLIENTE/SERVIDOR

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

- Cliente e servidor devem estar ativos no momento da comunicação
- Cliente envia uma requisição e bloqueia até que receba sua resposta
- Servidor essencialmente espera por requisições e as processa

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)

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
- Middleware normalmente assegura tolerância a falhas

RPC — Chamadas a procedimentos remotos

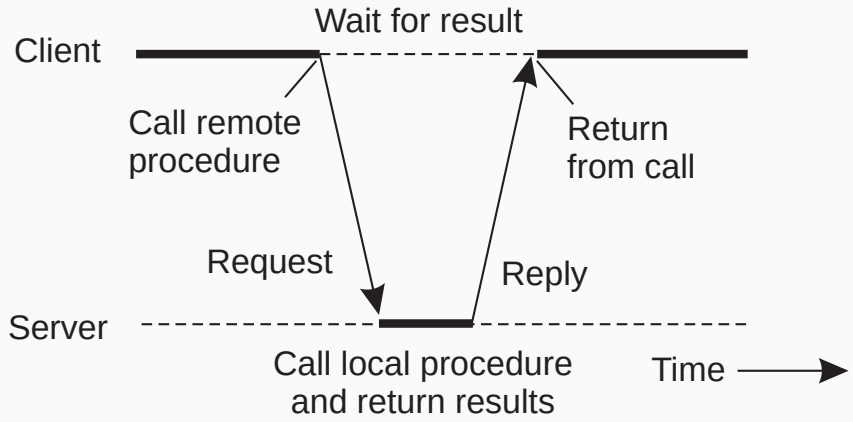
Chamadas a procedimentos remotos (RPC)

- Funcionamento básico de RPCs
- Passagem de parâmetros
- Variações

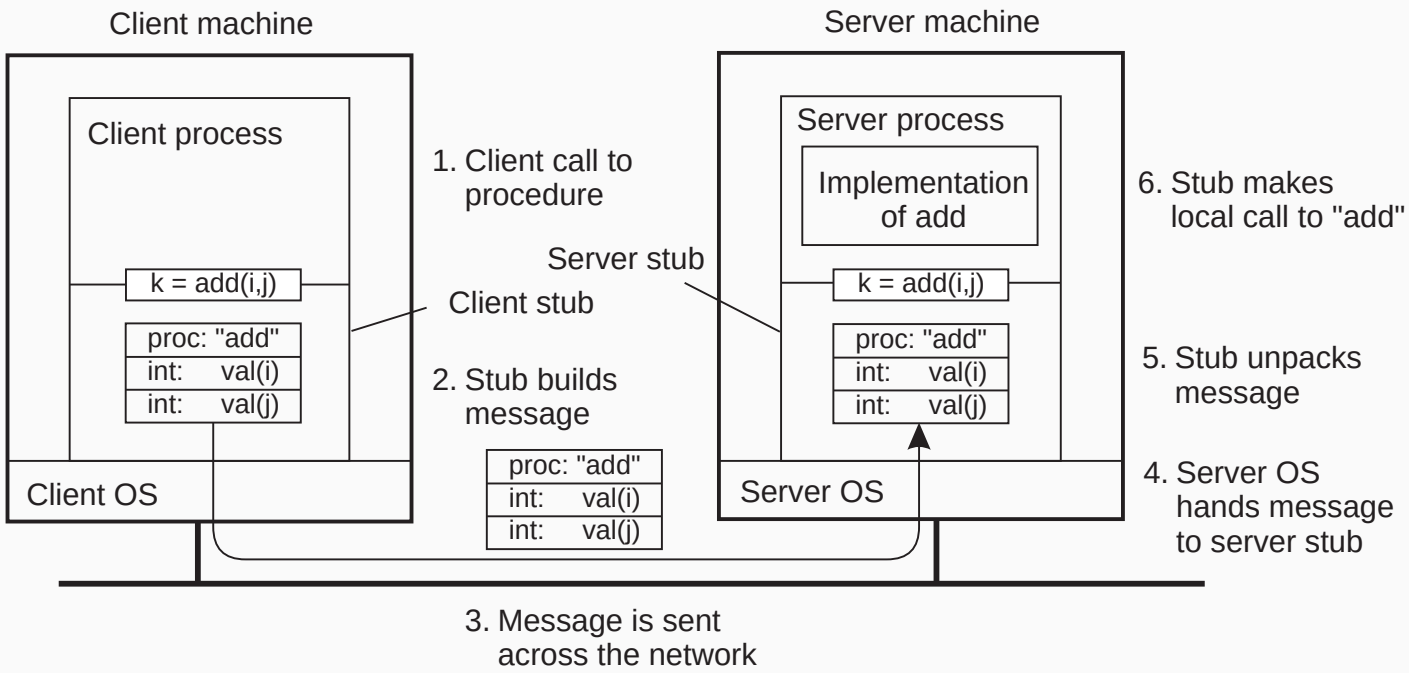
FUNCIONAMENTO BÁSICO DE RPC

- 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

Conclusão
Comunicação entre o chamador & chamado podem ser escondida com o uso de mecanismos de chamada a procedimentos.



FUNCIONAMENTO BÁSICO DE RPC



1. Procedimento no cliente chama o *stub* do cliente
2. *Stub* constrói mensagem; chama o SO local
3. SO envia msg. para o SO remoto
4. SO remoto repassa mensagem para o *stub*
5. *Stub* desempacota parâmetros e chama o servidor
6. Servidor realiza chamada local e devolve resultado para o *stub*
7. *Stub* constrói mensagem; chama SO
8. SO envia mensagem para o SO do cliente
9. SO do cliente repassa msg. para o *stub*
10. *Stub* do cliente desempacota resultado e devolve para o cliente

Empacotamento de parâmetros
há mais do que apenas colocá-los nas mensagens:

- 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 (vetores, *unions*) são representados?
- Cliente e servidor precisam interpretar corretamente as mensagens, transformando seus valores usando representações dependentes da máquina

RPC: PASSAGEM DE PARÂMETROS

Algumas suposições:

- semântica de copy in/copy out: enquanto um procedimento é executado, nada pode ser assumido sobre os valores dos parâmetros
- Todos os dados são passado apenas por parâmetro. Exclui passagem de *referências para dados (globais)*.

RPC: PASSAGEM DE PARÂMETROS

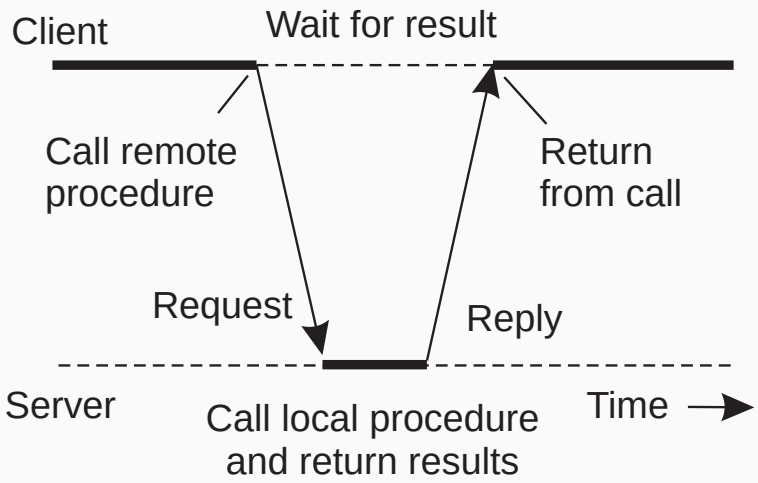
Algumas suposições:

- semântica de copy in/copy out: enquanto um procedimento é executado, nada pode ser assumido sobre os valores dos parâmetros
- Todos os dados são passado apenas por parâmetro. Exclui passagem de *referências para dados (globais)*.

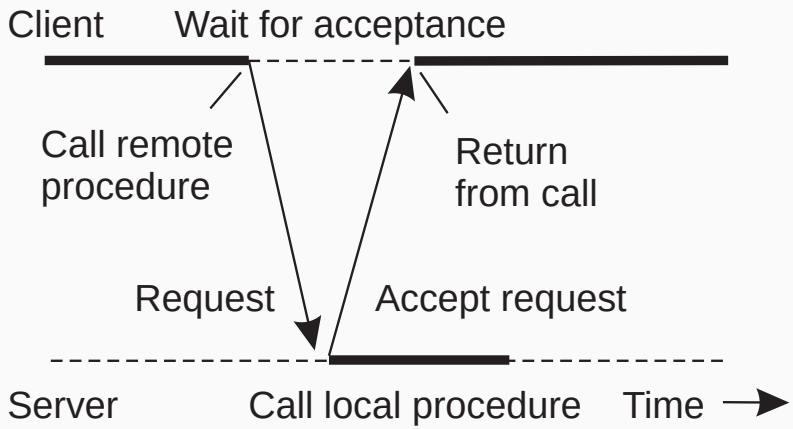
Conclusão
Não é possível assumir transparência total de acesso.

Ideia geral

Tentar se livrar do comportamento estrito de requisição–resposta, mas permitir que o cliente continue sem esperar por uma resposta do servidor.

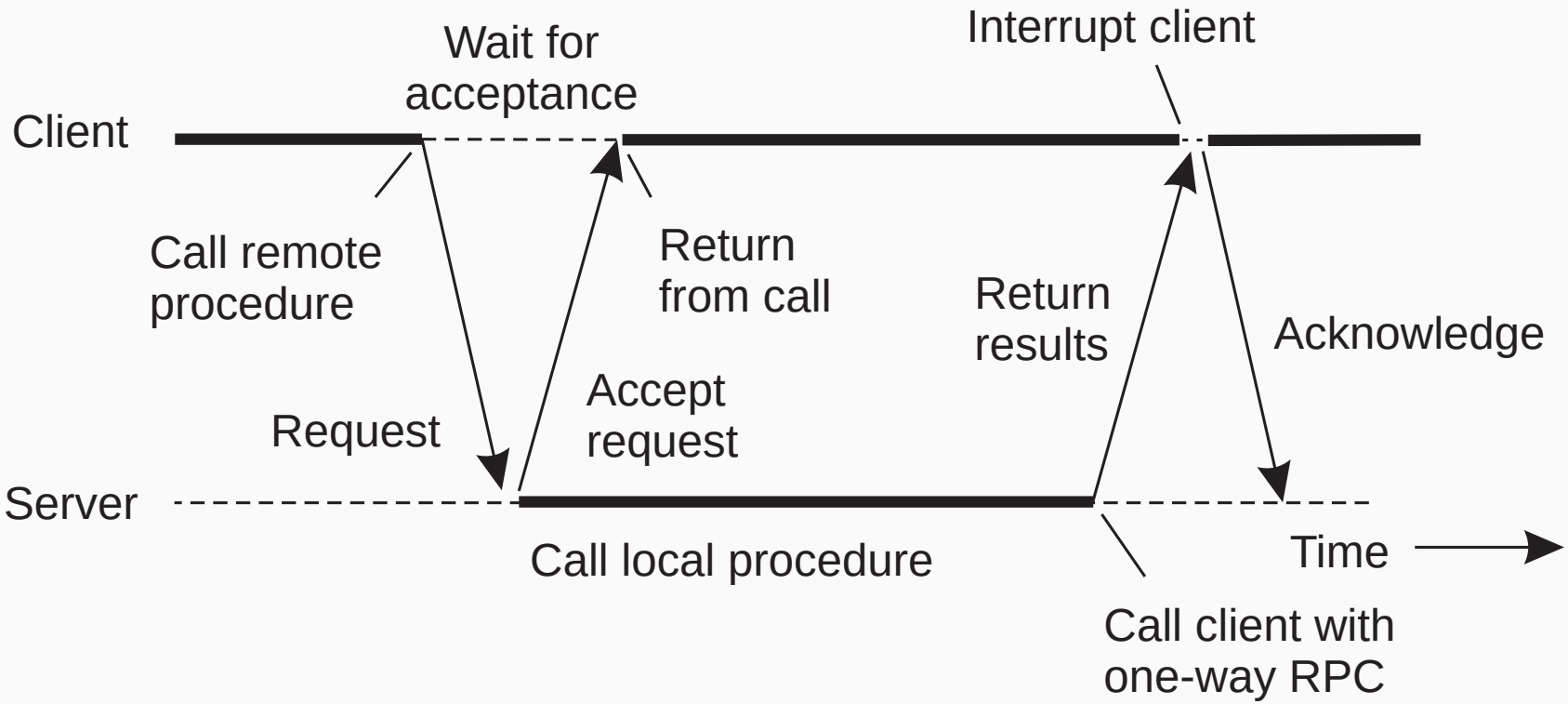


(a)



(b)

RPC SÍNCRONO DIFERIDO



Variação

Cliente pode também realizar uma consulta (*poll*) (bloqueante ou não) para verificar se os resultados estão prontos.

RPC NA PRÁTICA