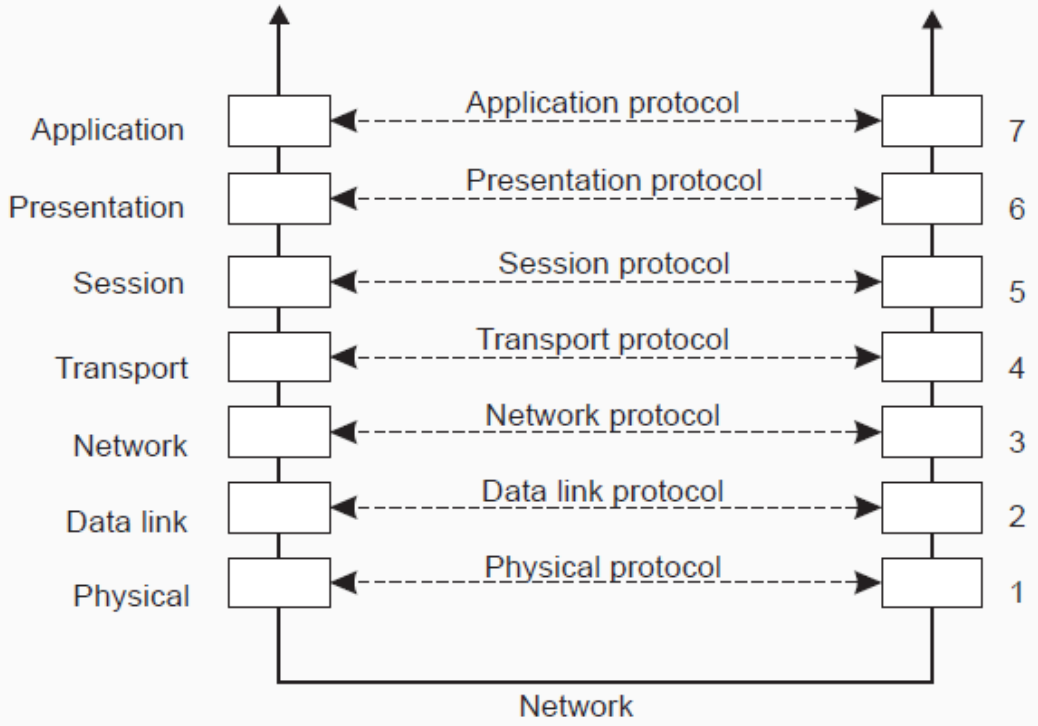
ACH 2147 — DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO DISTRIBUÍDOS

COMUNICAÇÃO

Protocolo 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.

Camada de Transporte

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)

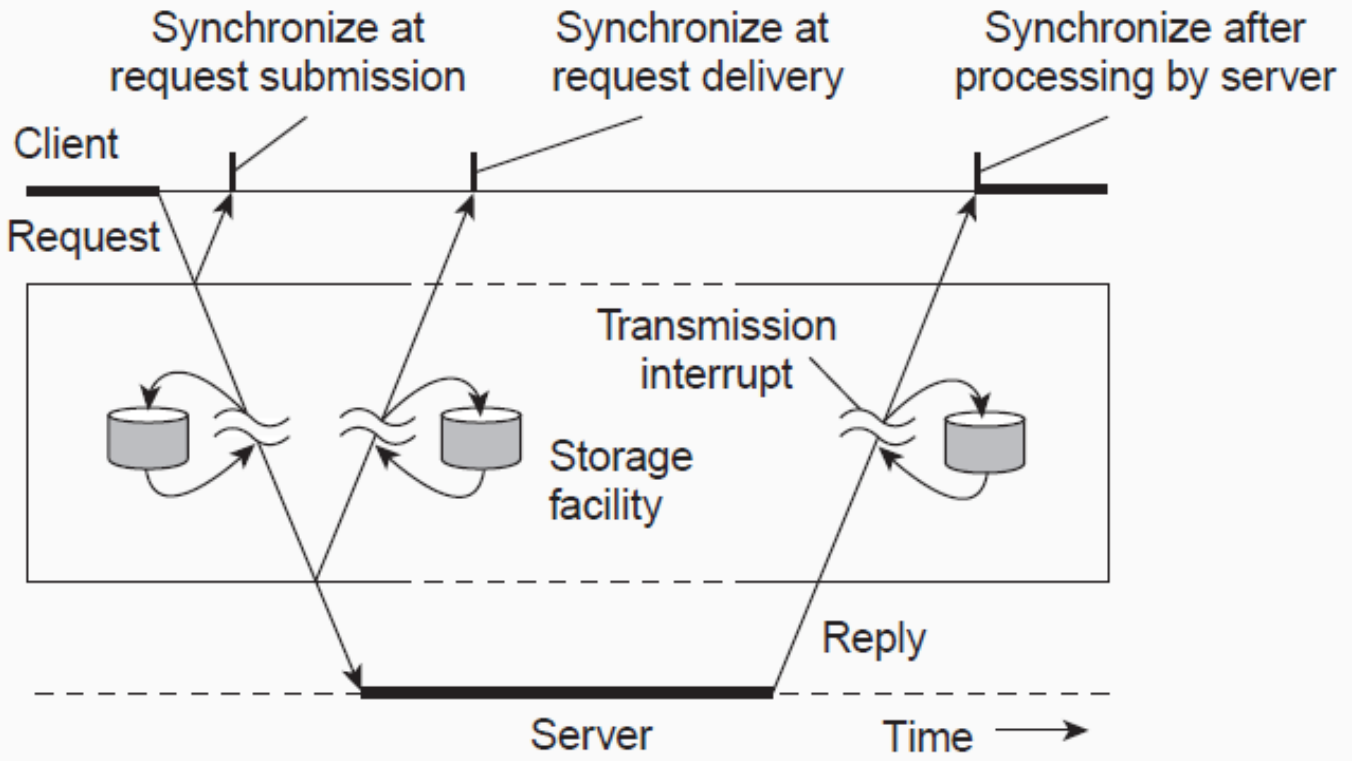
Camada de Middleware

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

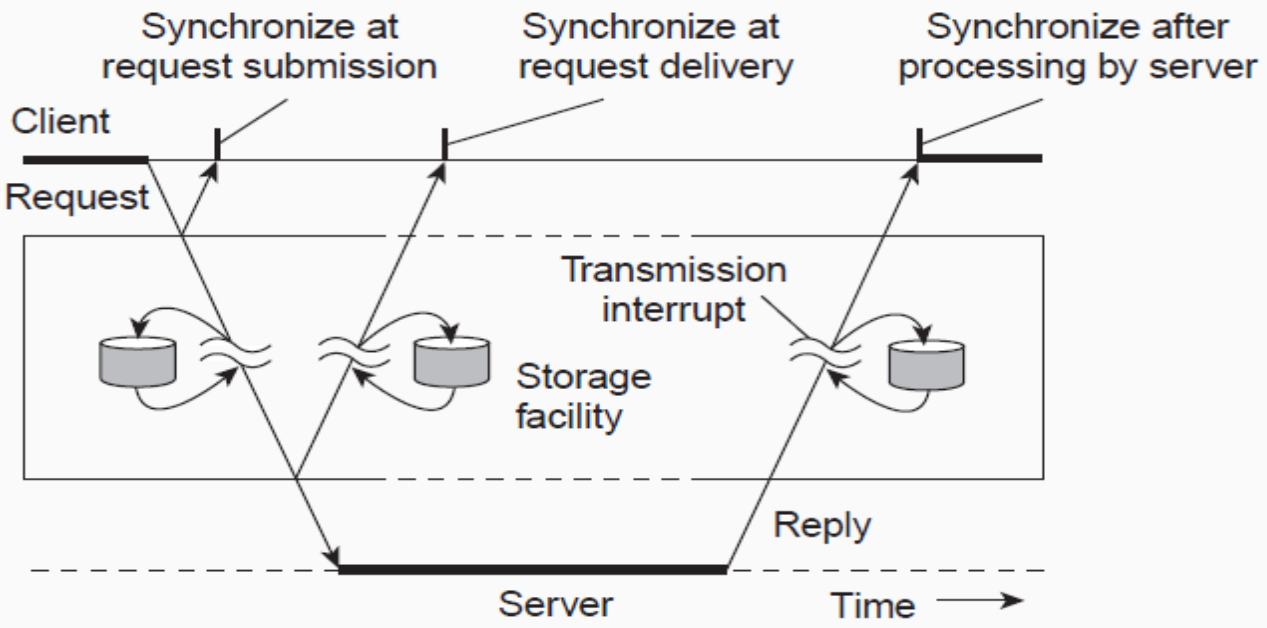
Observação:
O que realmente sobra são protocolos específicos de aplicação.

Tipos de Comunicação



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

Tipos de Comunicação (cont.)

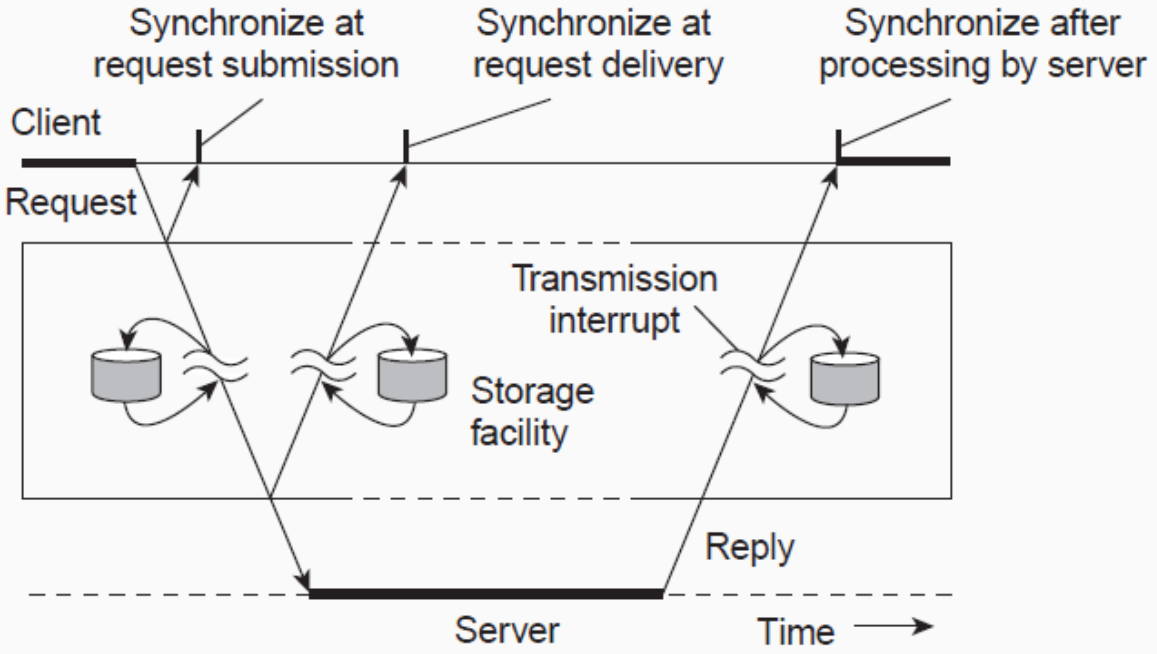


Transiente vs. persistente

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 (cont.)



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

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)

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
- Middleware normalmente assegura tolerância a falhas

Remote Procedure Call

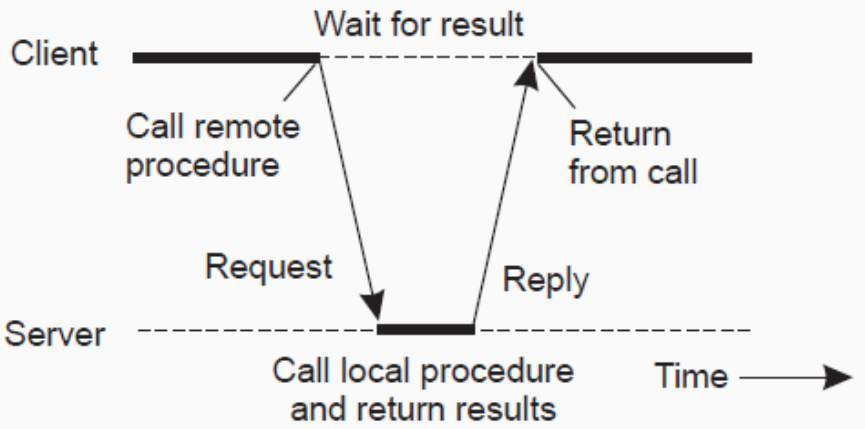
- Funcionamento básico de RPCs
- Passagem de parâmetros
- Variações

Mecanismo Básico - RPC

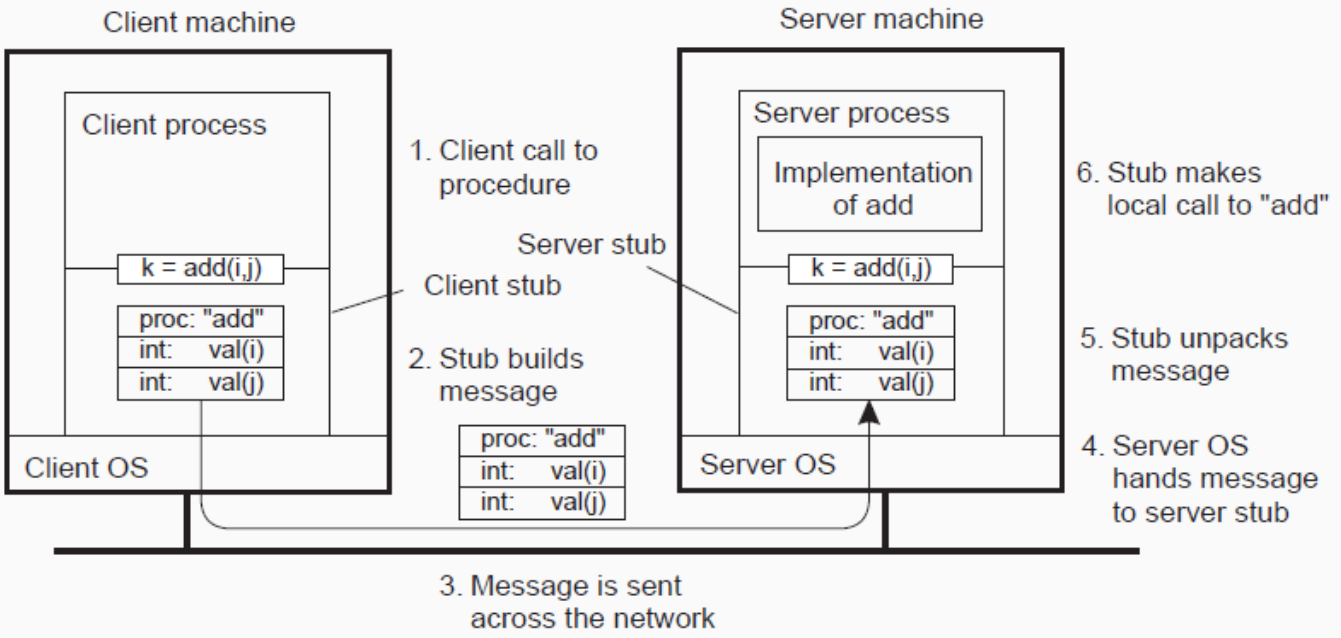
- Desenvolvedores estão familiarizados com o modelo de procedimentos
- Procedimentos bem projetados operam isoladamente (*black box*)
- Então não há razão para não executar esses procedimentos em máquinas separadas

Conclusão

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



Funcionamento Básico - 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

Passagem de Parâmetros - RPC

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

Passagem de Parâmetros - RPC

Algumas suposições:

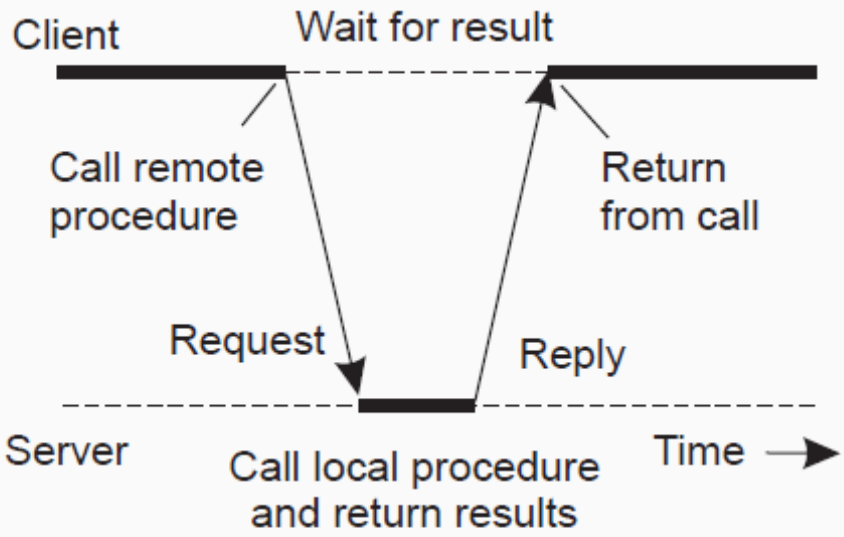
- semântica de copy in/copy out: enquanto um procedimento é executado, nada pode ser assumido sobre os valores dos parâmetros
- Todos os dados são passado apenas por parâmetro. Exclui passagem de *referências para dados (globais)*.

Conclusão
Não é possível assumir transparência total de acesso.

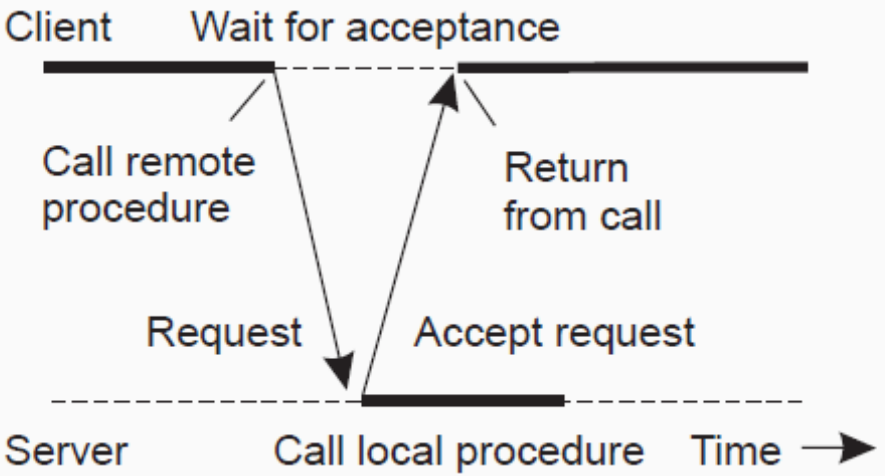
RPC Assíncrono

Ideia geral

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

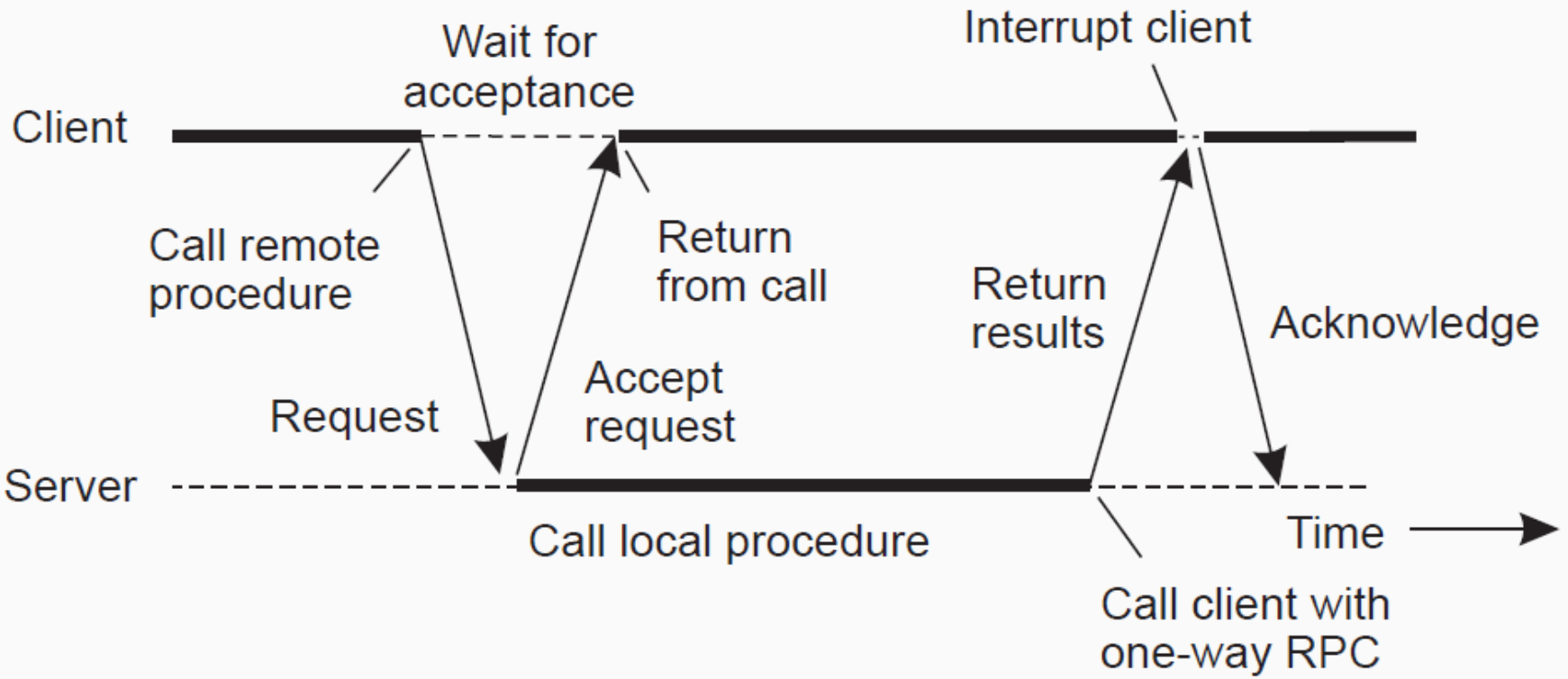


(a)



(b)

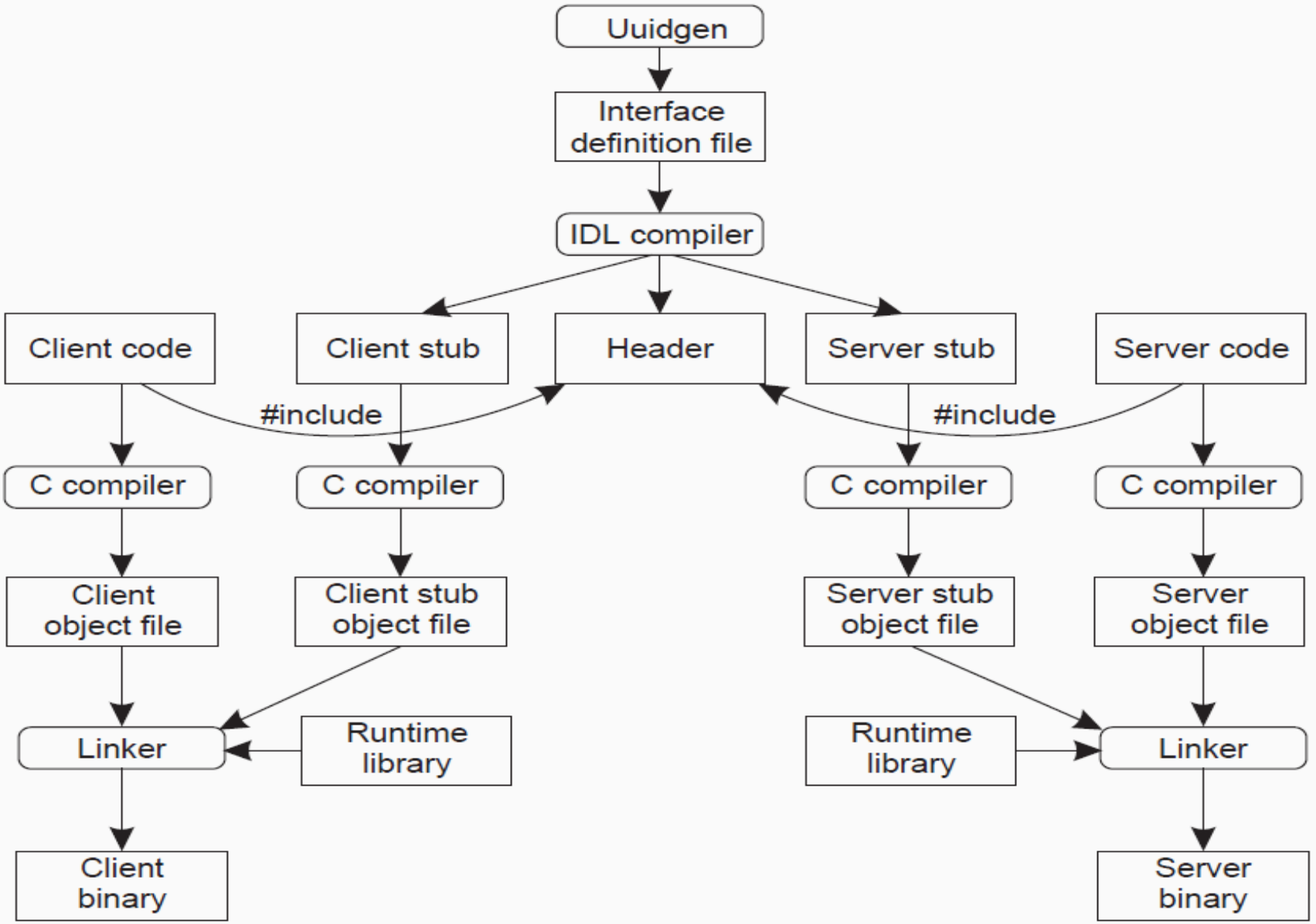
RPC Síncrono Diferido



Variação

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

RPC na prática

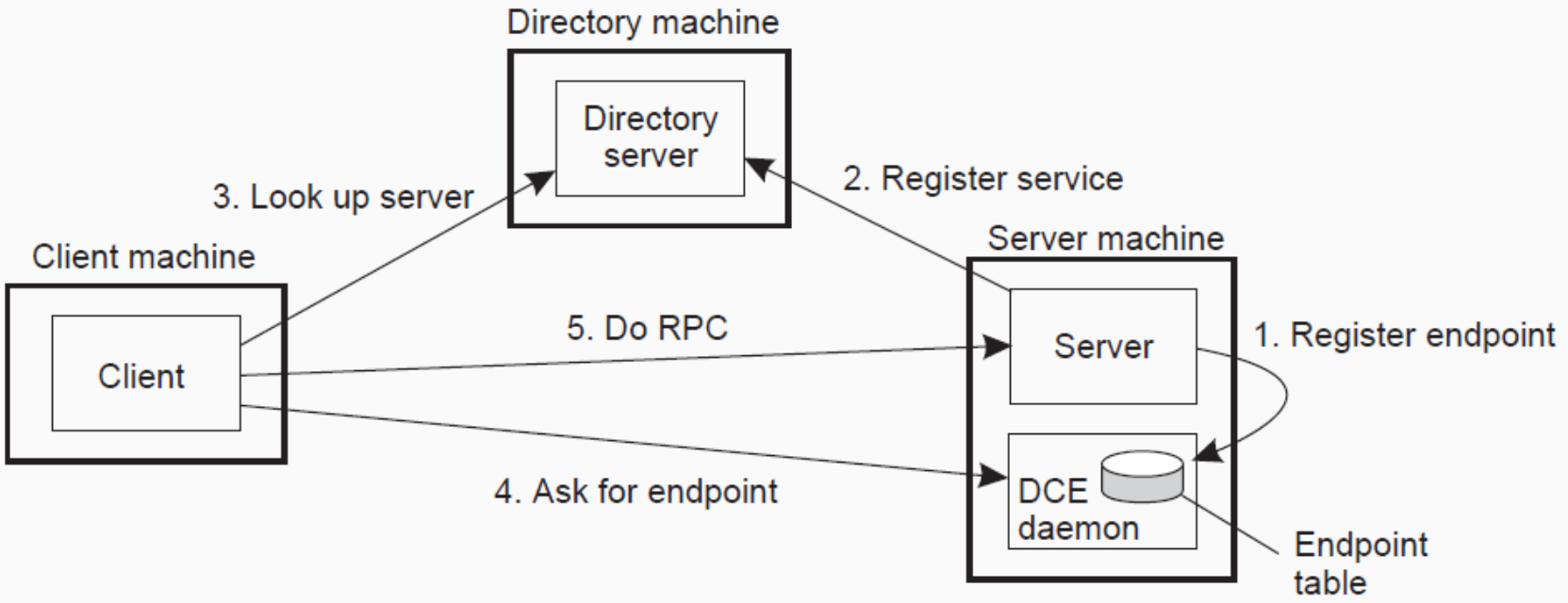


Vinculação Cliente Servidor - DCE

Problemas

(1) Cliente precisa localizar a máquina com o servidor e,

(2) precisa localizar o servidor.



Comunicação Orientada a Mensagens

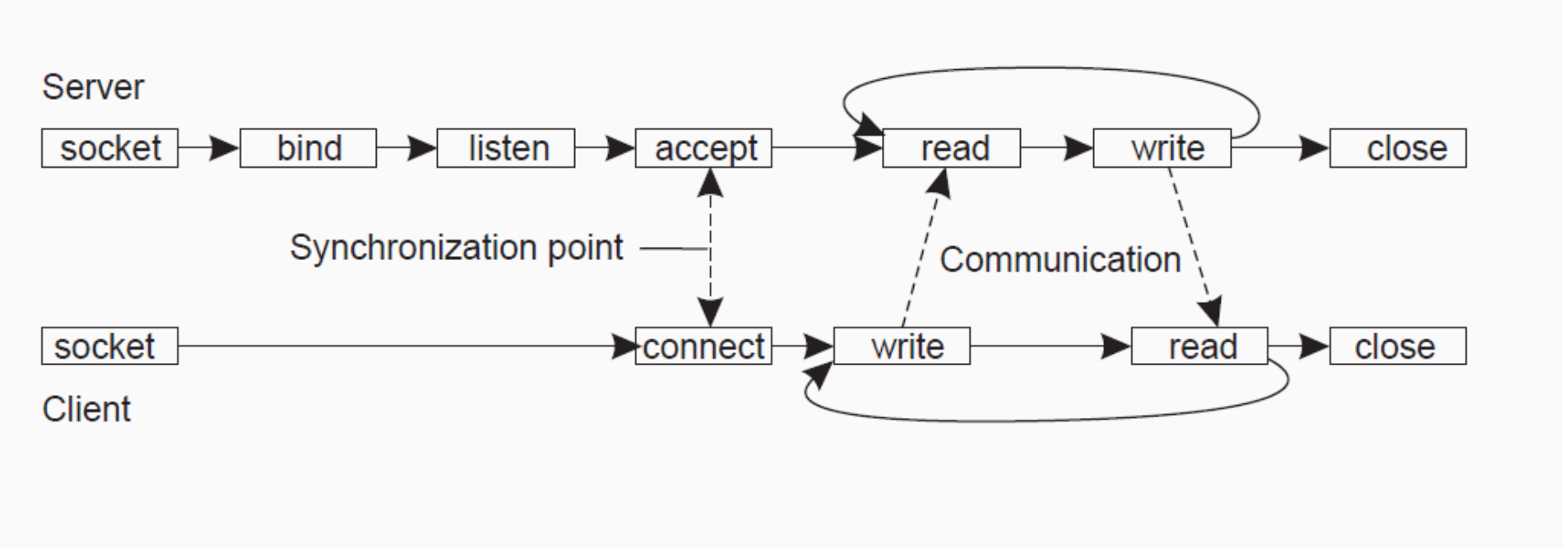
- Mensagens transientes
- Sistema de enfileiramento de mensagens
- *Message brokers*
- Exemplo: IBM Websphere

Mensagens Transientes - Sockets

Berkeley socket interface

SOCKET	Cria um novo ponto de comunicação
BIND	Especifica um endereço local ao socket
LISTEN	Anuncia a vontade de receber *N* conexões
ACCEPT	Bloqueia até receber um pedido de estabelecimento de conexão
CONNECT	Tenta estabelecer uma conexão
SEND	Envia dados por uma conexão
RECEIVE	Recebe dados por uma conexão
CLOSE	Libera a conexão

Mensagens Transientes - Sockets



Sockets – um exemplo em Python

```
import socket
HOST = socket.gethostname()                    # e.g. 'localhost'
PORT = SERVERPORT                              # e.g.  80
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(N)              # listen to max N queued connection
conn, addr = s.accept()        # new socket + addr client
while 1: # forever
  data = conn.recv(1024)
  if not data: break
  conn.send(data)
conn.close()
```

Middleware orientado a mensagens

Ideia geral

Comunicação assíncrona e persistente graças ao uso de filas pelo middleware. Filas correspondem a buffers em servidores de comunicação.

PUT	Adiciona uma mensagem à fila especificada
GET	Bloqueia até que a fila especificada tenha alguma mensagem e remove a primeira mensagem
POLL	Verifica se a fila especificada tem alguma mensagem e remove a primeira. Nunca bloqueia
NOTIFY	Instala um tratador para ser chamado sempre que uma mensagem for inserida em uma dada fila

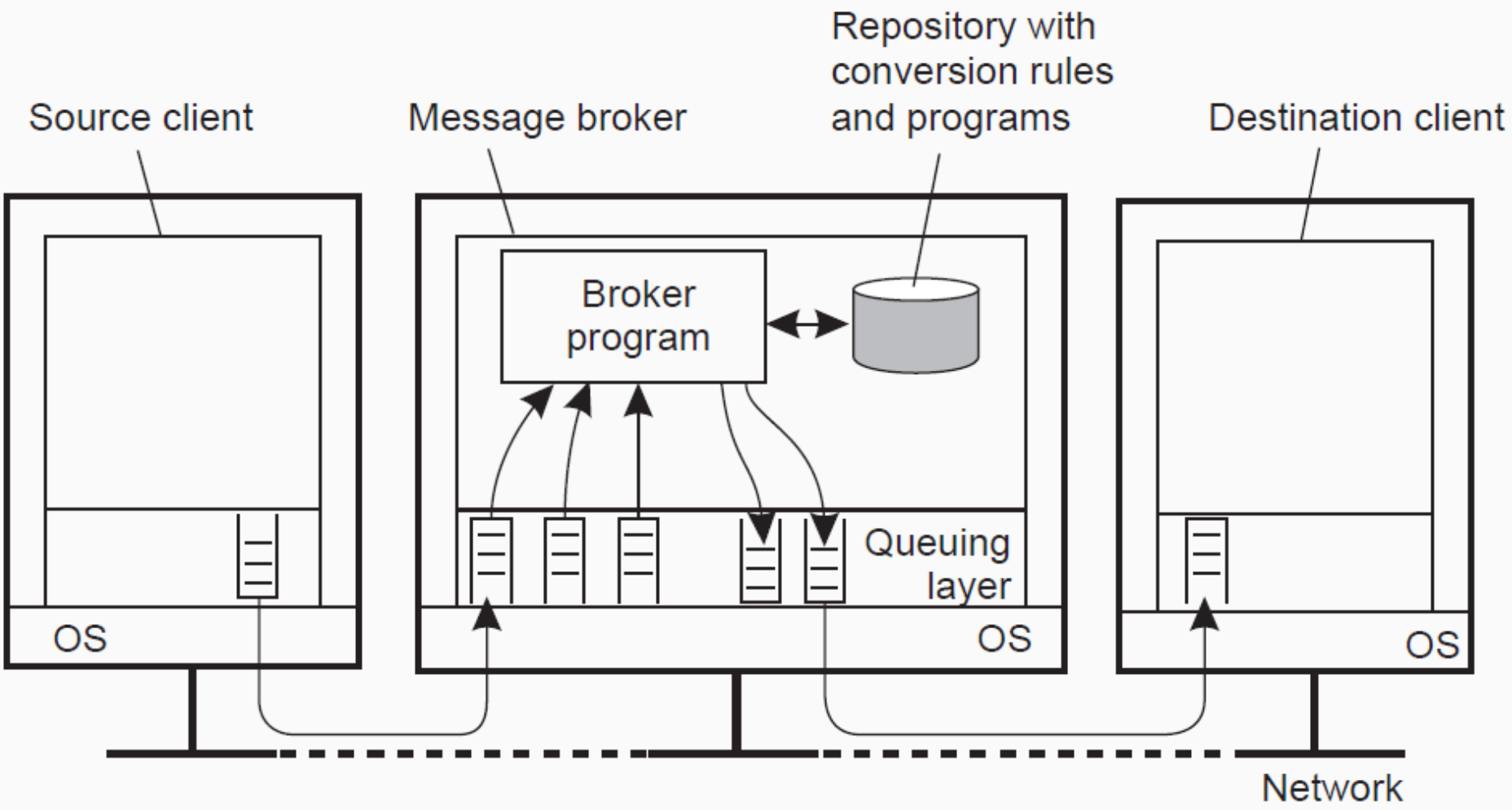
Message Broker

Observação:
Sistemas de filas de mensagens assumem um protocolo comum de troca de mensagens: todas as aplicações usam o mesmo formato de mensagem (i.e., estrutura e representação de dados)

Message broker
Componente centralizado que lida com a heterogeneidade das aplicações:

- transforma as mensagens recebidas para o formato apropriado
- frequentemente funciona como um application gateway
- podem rotear com base no conteúdo ⇒ Enterprise Application Integration

Message Broker



IBM Websphere Message Queue

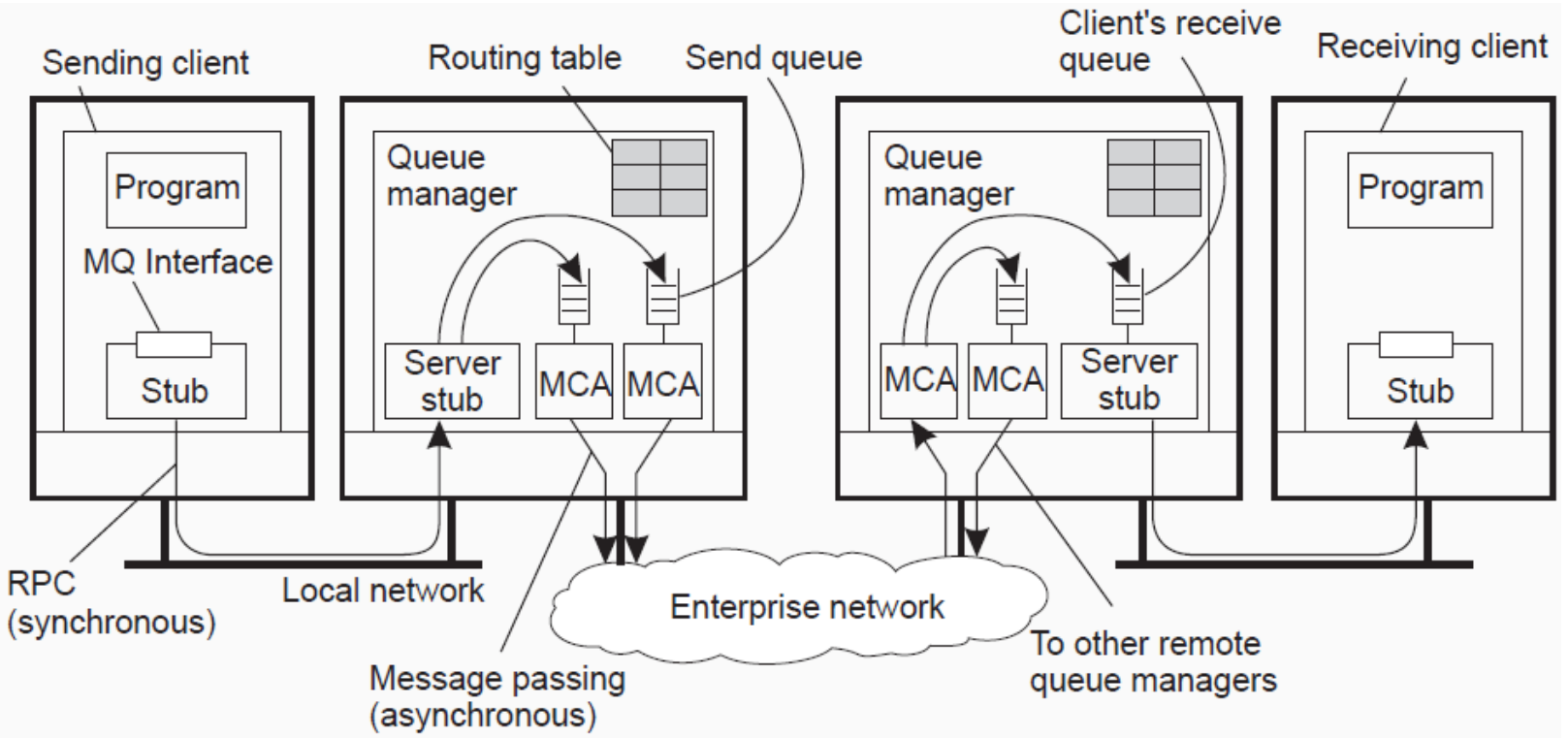
- Mensagens específicas da aplicação são colocadas e removidas de filas
- As filas são controladas por um gerenciador de filas
- Processos podem colocar mensagens apenas em filas locais, ou usando um mecanismo de RPC

IBM Websphere MQ

Transferência de mensagens

- Mensagens são transferidas entre filas
- Mensagens transferidas entre filas em diferentes processos requerem um canal
- Em cada ponta do canal existe um agente de canal, responsável por:
  - configurar canais usando ferramentas de rede de baixo nível (ex: TCP/IP)
  - (Des)empacotar mensagens de/para pacotes da camada de transporte
  - Enviar/receber pacotes

IBM Websphere MQ

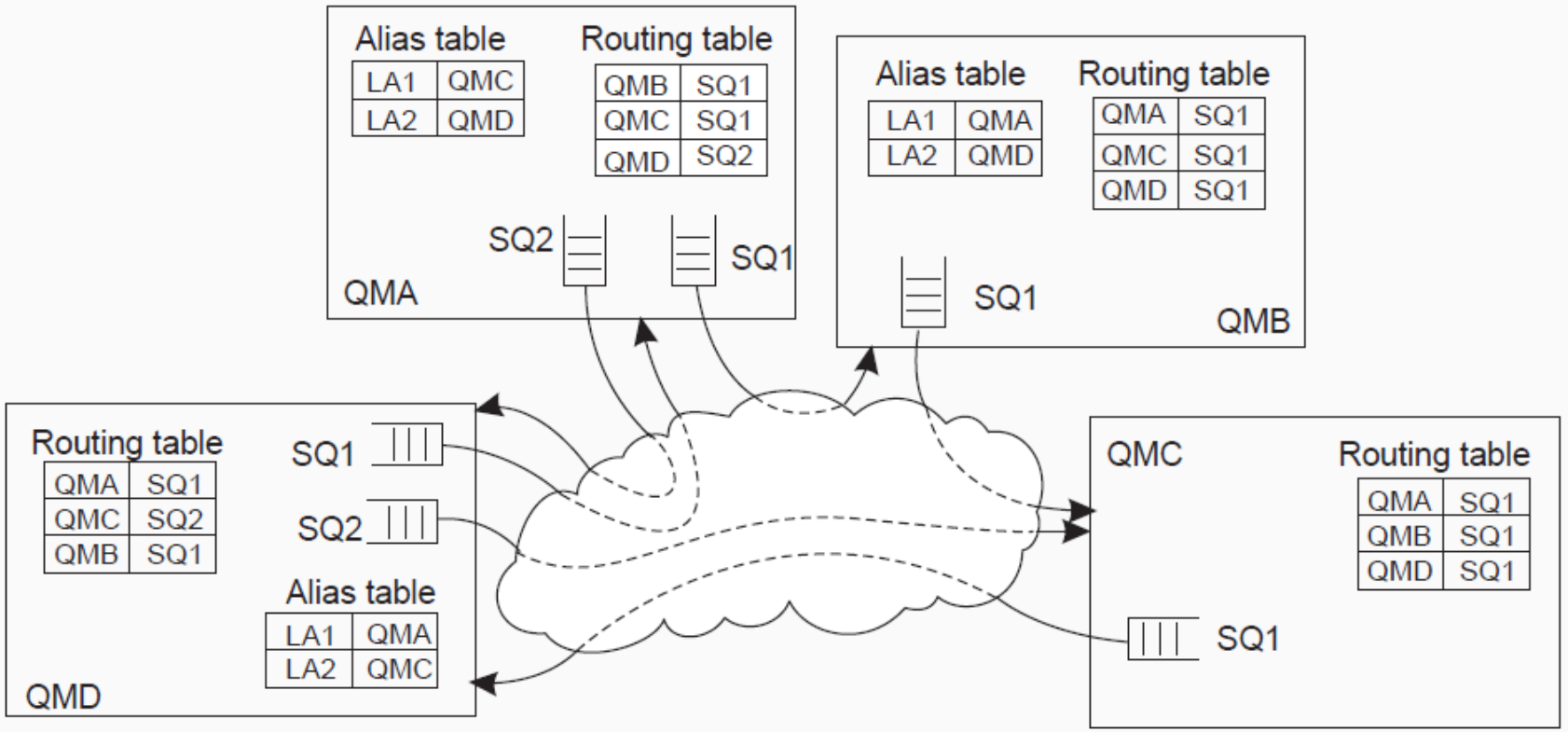


- Canais são unidirecionais
- Agentes de canais são automaticamente iniciados quando uma mensagem chega
- Pode-se criar redes de gerenciadores de filas
- Rotas são configuradas manualmente (pelo admin do sistema)

IBM Websphere MQ

Roteamento

O uso de nomes lógicos, combinados com resolução de nomes para filas locais, permitem que uma mensagem seja colocada em uma fila remota.



Comunicação Multicast

- Transmissão de mensagens multicast no nível da aplicação
- Disseminação de dados com métodos de *gossiping*

Multicast no nível de aplicação

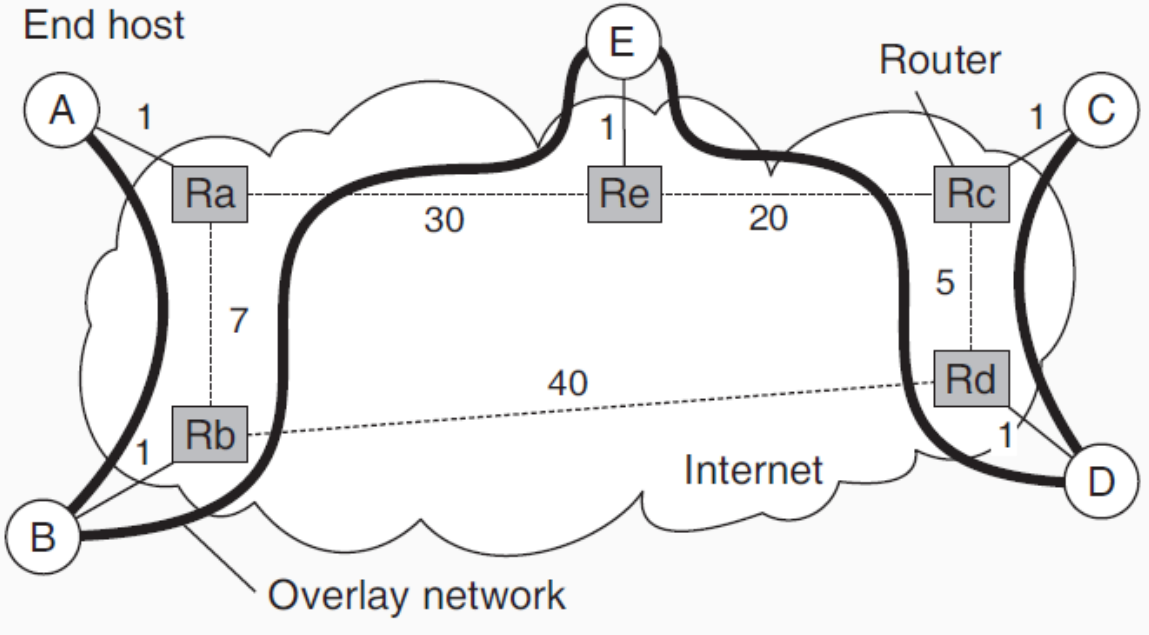
Ideia geral

Organizar os nós de um sistema distribuído em uma rede overlay e usar a rede para disseminar os dados.

Construção de árvores pelo protocolo Chord

1. O iniciador gera um identificador multicast `mid`
2. Lookup por `succ(mid)`, o nó responsável por `mid`
3. Requisição é roteada para `succ(mid)`, que será designado root
4. Se *P* quiser se juntar, ele envia uma requisição do tipo `join` ao root.
5. Quando uma requisição chegar em *Q*:
   - se *Q* não viu nenhuma requisição `join` antes, ele se torna um forwarder; *P* se torna filho de *Q* e a requisição de `join` continua a ser repassada
   - se *Q* sabe sobre a existência da árvore, *P* se torna filho de *Q* (como antes), mas não é mais necessário repassar a requisição de `join`

Multicast no nível de aplicação



- Stress nos links: com que frequência uma mensagem de multicast será enviada pelo mesmo enlace físico? Exemplo: uma mensagem de A para D precisa atravessar $\langle Ra, Rb \rangle$ duas vezes
- Stretch: razão entre o atraso da comunicação usando o caminho multicast e usando a rede. Exemplo: mensagens de B para C seguem o caminho de tamanho 73 no multicast, mas um de 47 existe no nível da rede. stretch = 73/47

Protocolos Epidêmicos

- Contexto
- Modelos de atualização
- Remoção de objetos

Princípios

Ideia básica: assuma que não existem conflitos de write–write (má ideia)

- atualizações são realizadas em um único servidor
- uma réplica passa o estado atualizado para alguns vizinhos
- propagação da atualização é *lazy*, i.e., não é imediata
- eventualmente, todo update deveria alcançar todas as réplicas

Duas formas de epidemias:

Anti-entropy: cada réplica regularmente escolhe outra réplica ao acaso e uniformiza seus estados

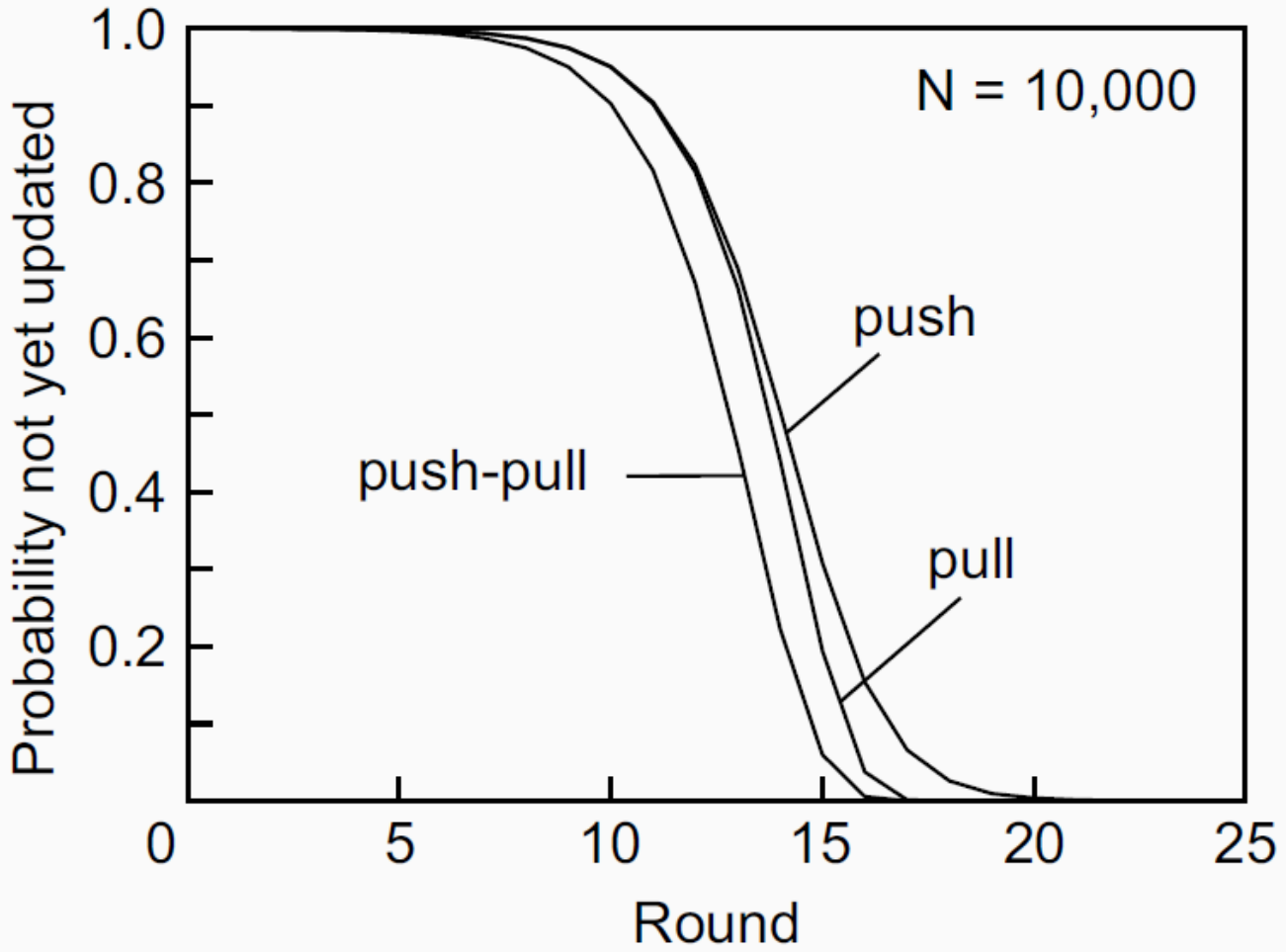
Gossiping: uma réplica que acaba de ser atualizada (*contaminada*) repassa a atualização a alguns vizinhos (contaminando elas)

Anti-entropy

- um nó P seleciona aleatoriamente outro nó Q do sistema
- Push: P só envia suas atualizações para Q
- Pull: P só recebe informações de Q
- Push–Pull: P e Q trocam atualizações entre si (e terminam com as mesmas informações)

Observação:
Cada push–pull leva $\mathcal{O}(log(N))$ rodadas de comunicação para disseminar as atualizações para todos os N nós

Desempenho Anti-entropy



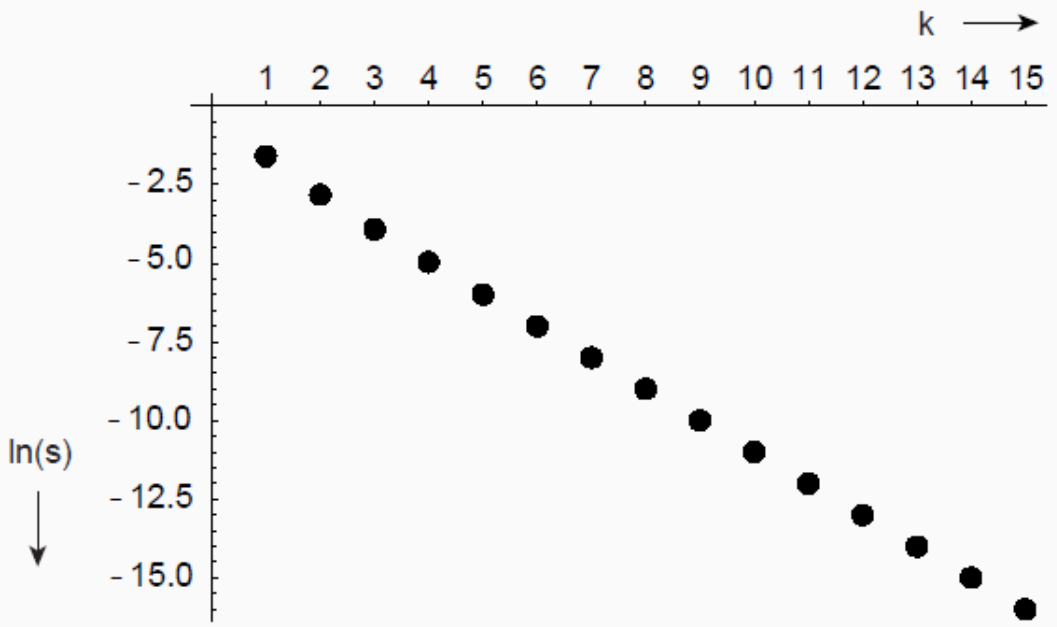
Gossiping

Um servidor S com uma atualização a reportar contacta outros servidores. Se o servidor contactado já compartilhou essa atualização, S para de conectar outros servidores com probabilidade $1/k$

Observação:
Se s for a fração de servidores que desconhecem a atualização, pode-se mostrar que, com muitos servidores,

$$s = e^{-(k+1)(1-s)}$$

Gossiping



Considere 10.000 nós		
k	s	N_s
1	0.203188	2032
2	0.059520	595
3	0.019827	198
4	0.006977	70
5	0.002516	25
6	0.000918	9
7	0.000336	3

Note que:
se você realmente quiser se assegurar de que todos os servidores serão atualizados eventualmente, gossiping sozinho não é suficiente

Remoção de Valores

Problema intrínseco:
Nós não podemos remover um valor antigo do servidor e esperar que a remoção se propague. Pior, uma remoção simples pode ser desfeita rapidamente se um protocolo epidêmico estiver sendo utilizado.

Solução
A remoção precisa ser registrada com um tipo especial de atualização: um atestado de óbito.

Remoção de Valores

Problema seguinte: como remover um atestado de óbito? (ele não pode ficar lá pra sempre)

- execute um algoritmo global para detectar se a remoção foi percebida por todos os nós e então remova os atestados
- assuma que os atestados não serão propagados para sempre e associe um tempo de vida máximo para o atestado

Observação:
É preciso que a remoção realmente alcance todos os servidores.

Problema de escalabilidade:
Quanto mais servidores, maior o tempo de propagação.