

**ESBD 3**

*Sistemas Distribuídos*



**INFORMAÇÃO,  
TECNOLOGIA  
& INOVAÇÃO**

**Sistemas Distribuídos**

**06/05/2023 - 08h30 - 12h30**

**06/05/2023 - 14h00 - 18h00**

**20/05/2023 - 08h30 - 12h30**

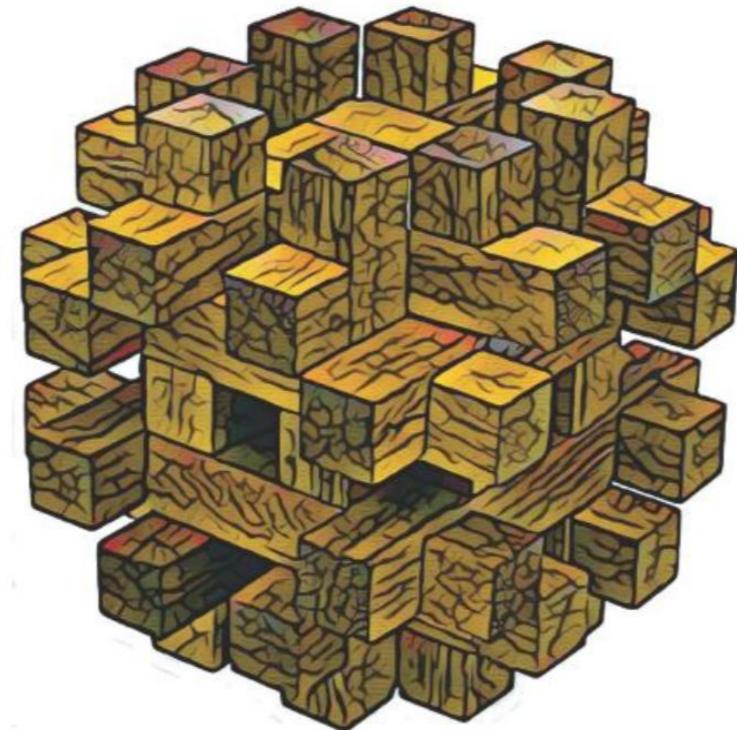
**20/05/2023 - 14h00 - 16h00**

**03/06/2023 - 08h30 - 12h30**

# LIVRO TEXTO PARA O CURSO

# Distributed Systems

Maarten Van Steen & Andrew S. Tanenbaum



3th Edition – Version 3.03 - 2020

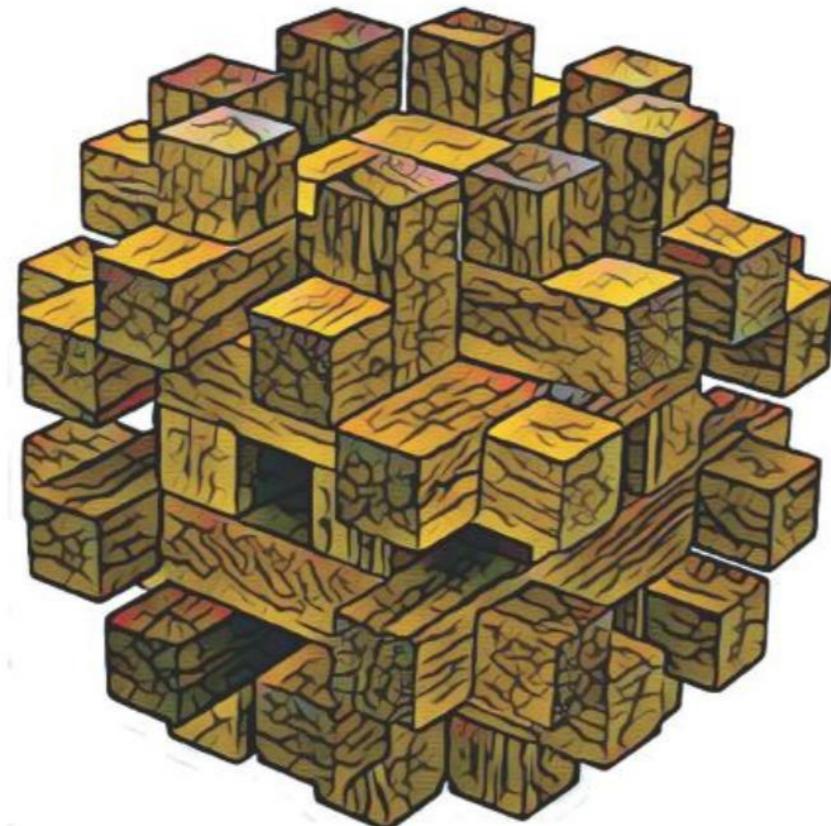
# Capítulos

1. Introdução
2. Arquiteturas
3. Processos
4. Comunicação – 20/05/2022
5. Nomes – 20/05/2021
6. Coordenação – 20/05/2022
7. Consistência e Replicação
8. Tolerância a Falhas
9. Segurança



# Distributed Systems

Maarten Van Steen & Andrew S.  
Tanenbaum



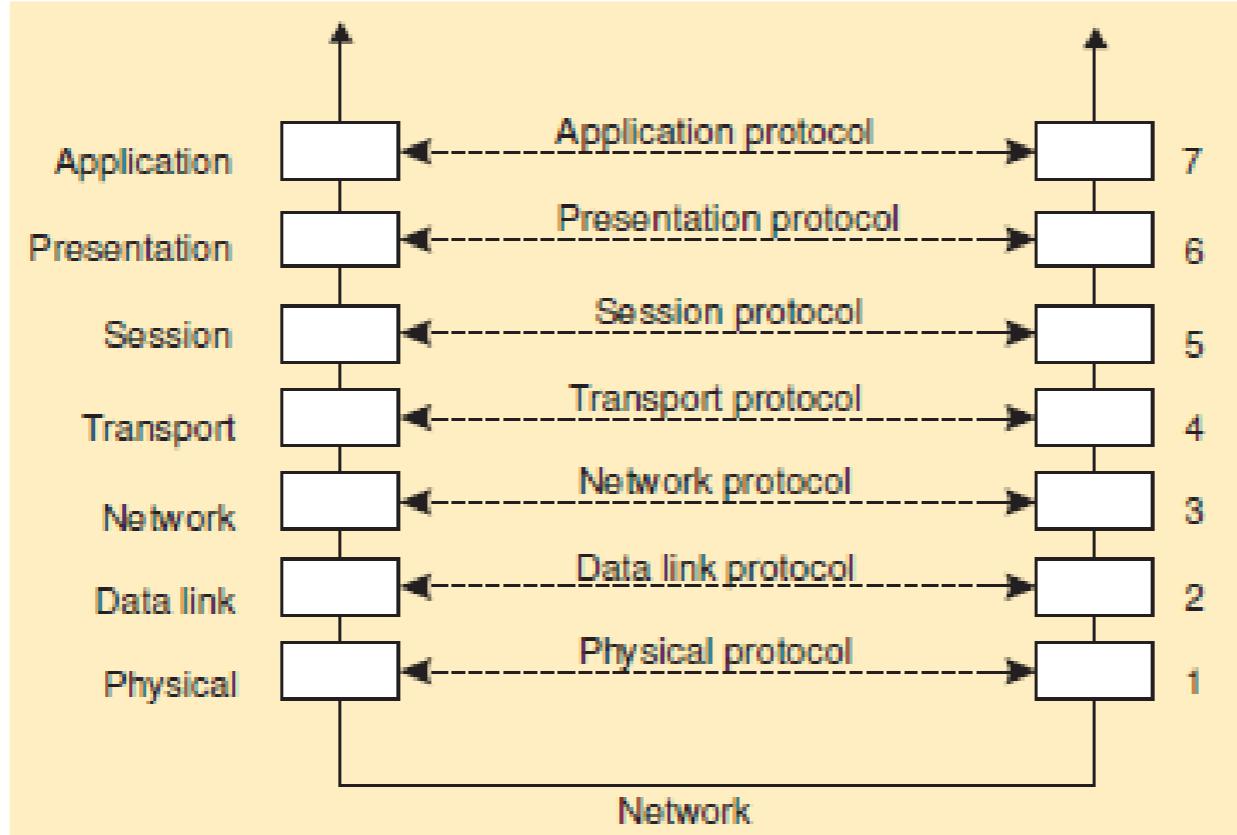
3th Edition – Version 3.01 - 2017

## Capítulo 4

## Comunicação

Sábado, 03 de Julho de 2021

# MODELO BÁSICO DE REDES



## PROBLEMAS

- Foco somente em passagem de mensagem
- Funcionalidade frequentemente indesejada e não necessária
- Viola transparência de acesso

# CAMADAS DE BAIXO NÍVEL

## RECAPITULANDO

- **Camada Física:** contém a especificação e implementação de bits, e sua transmissão entre despachante e recebedor
- **Camada Data Link:** prescreve a transmissão de uma série de bits em um Quadro (frame) para permitir controle de fluxo e de erros
- **Camada de Rede:** descreve como pacotes em uma rede de Computadores deverão ser roteados.

## OBSERVAÇÃO

- Para muitos sistemas distribuídos, a interface de mais baixo nível é a camada de rede.



# CAMADAS DE TRANSPORTE

## IMPORTANTE

- A camada de transporte provê as facilidades de comunicação para maioria dos sistemas distribuídos.

## PROTOCOLOS INTERNET PADRÃO

- TCP: orientado a conexão (connection-oriented), confiável, comunicação orientada a streams.
- UDP: não confiável (melhor esforço) comunicação por datagrama



# CAMADA MIDDLEWARE

## OBSERVAÇÃO

Middleware foi desenvolvido para prover serviços **comuns** e protocolos que podem ser usados por muitas **diferentes** aplicações

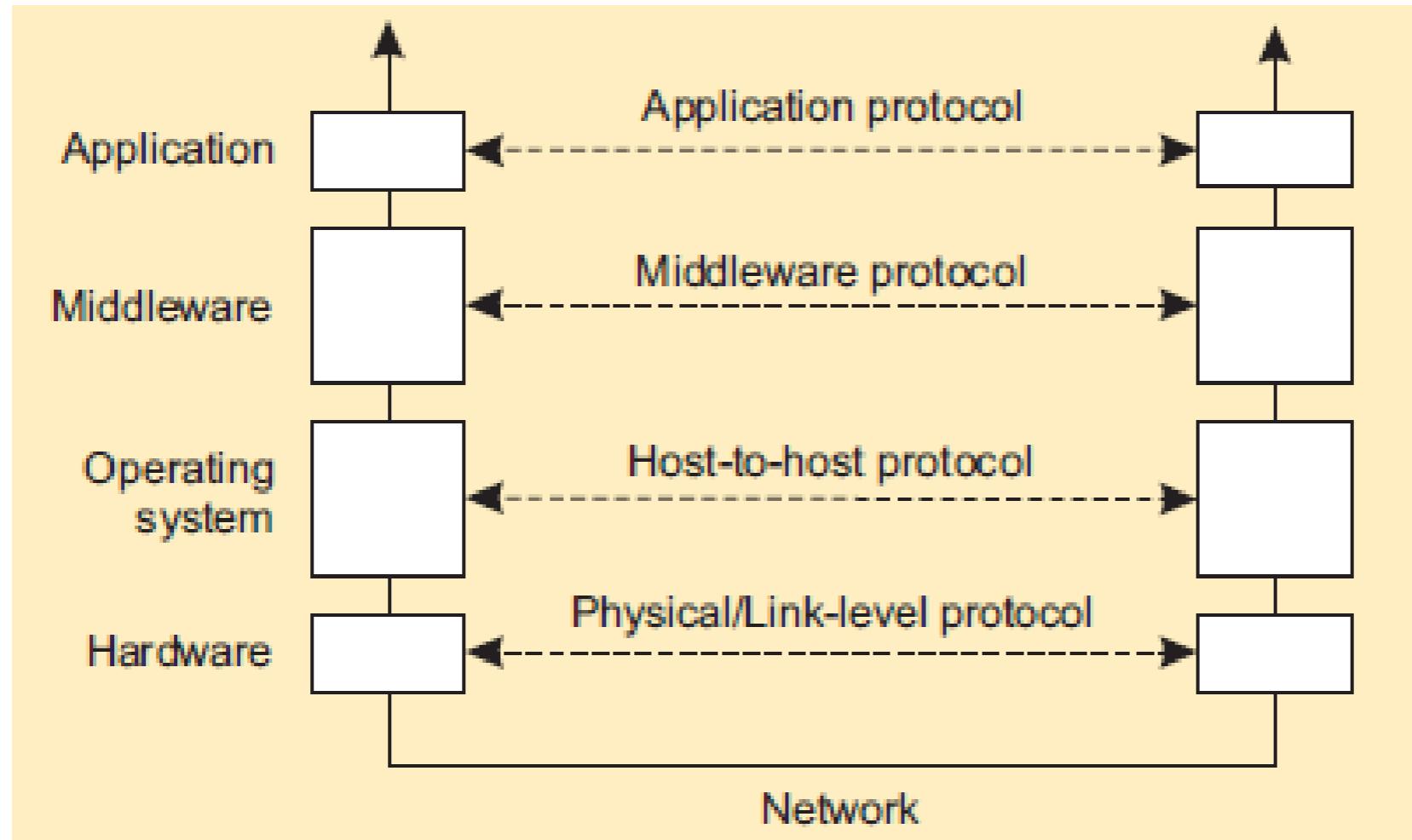
- Um rico conjunto de **protocolos de comunicação**
- **(Un)marshaling** de dados, necessário para sistemas integrados
- **Protocolos de Nomes**, para permitir o compartilhamento de recursos de forma fácil
- **Protocolos de segurança** para comunicação segura
- **Mecanismos de escalagem**, tais como replicação e caching

## NOTA

- O que sobra são protocolos **específicos de aplicação ... Tais como?**

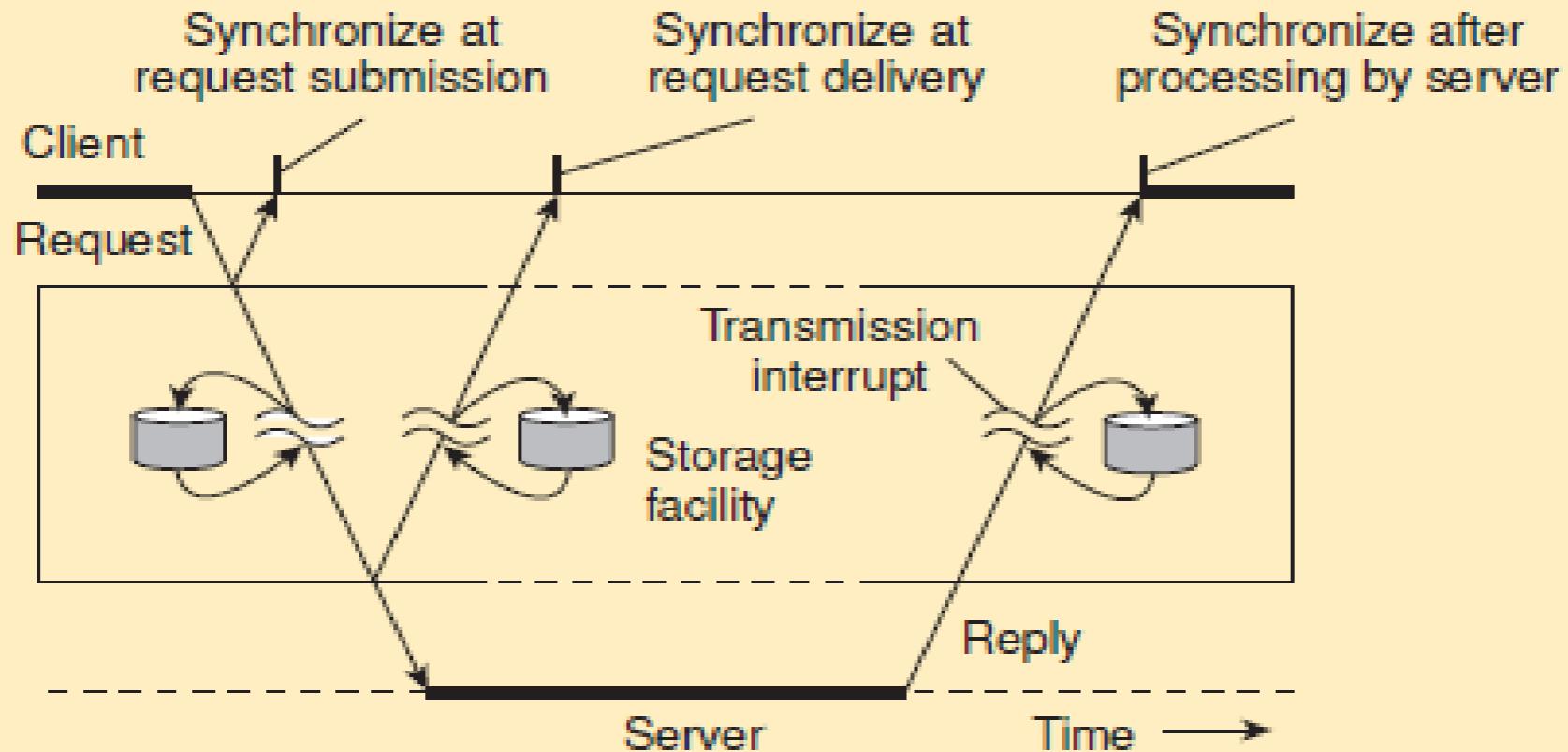


# UM ESQUEMA ADAPTADO DE CAMADAS



# TIPOS DE COMUNICAÇÃO

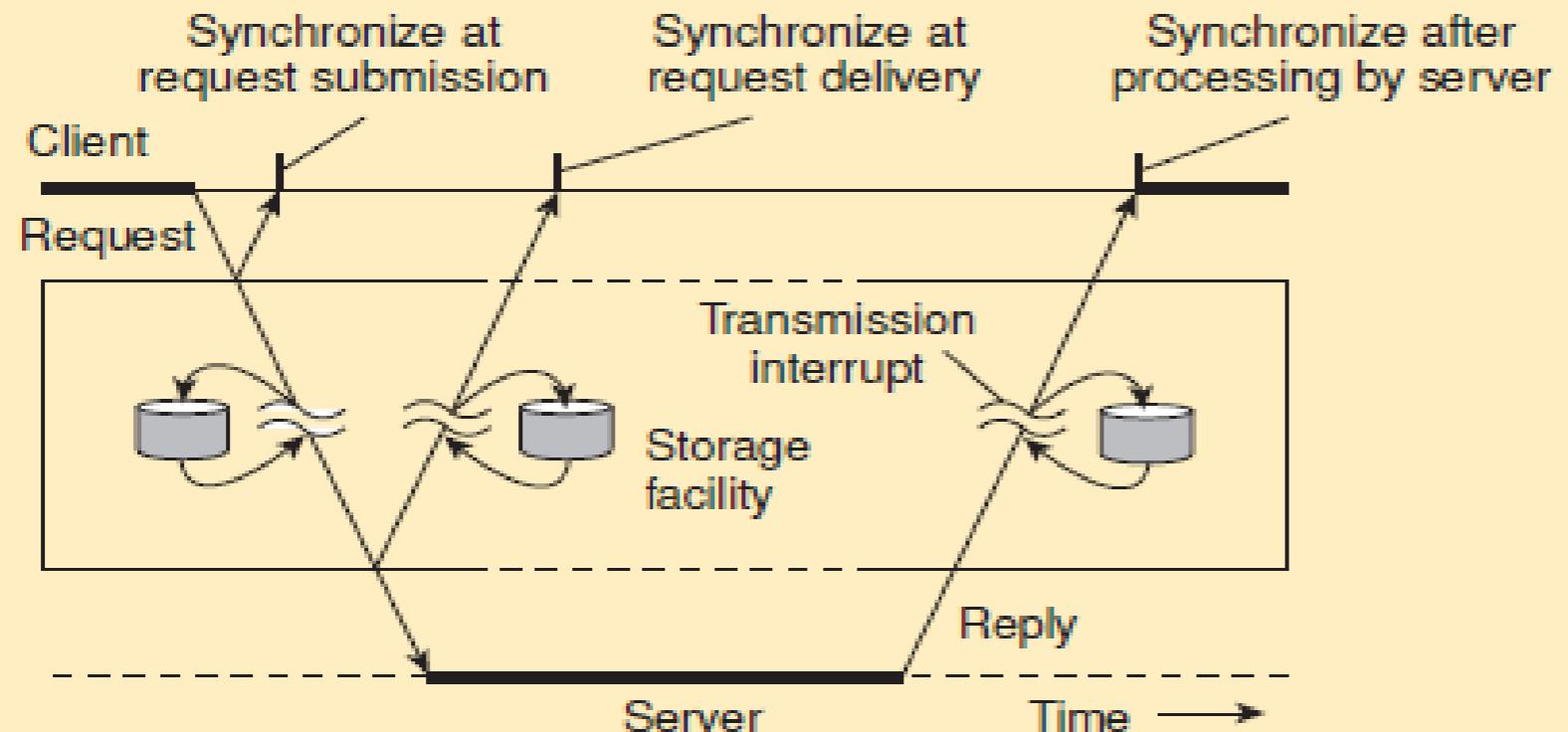
## DISTINÇÃO ..



- Comunicação Transiente X persistente
- Comunicação Assíncrona X síncrona

# TIPOS DE COMUNICAÇÃO

## TRANSIENTE X PERSISTENTE

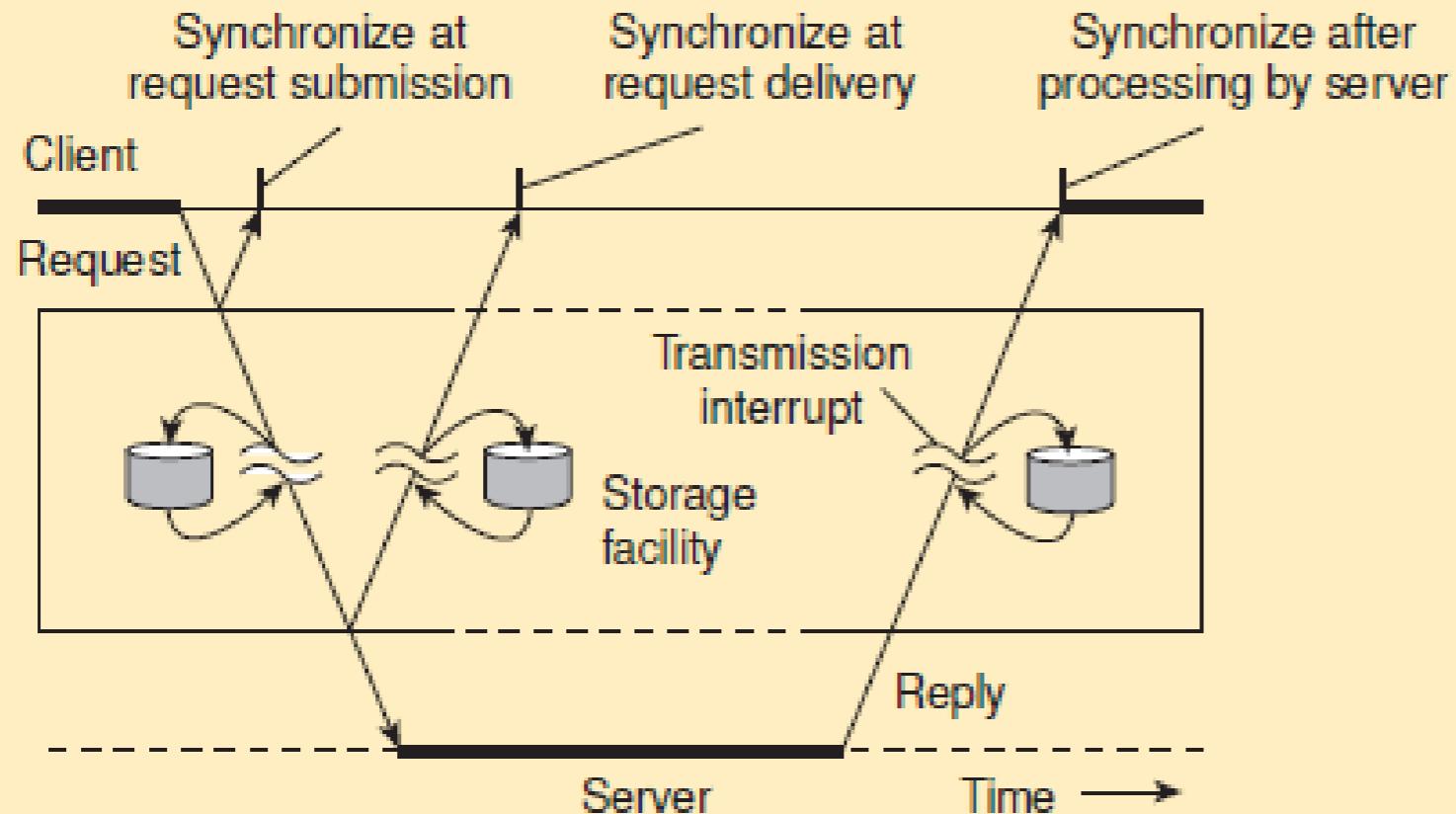


PROTOCOLOS MIDDLEWARE

- **Comunicação Transiente:** Servidor de comunicação descarta a mensagem quando ela não pode ser entregue para o próximo servidor ou destinatário.
- **Comunicação Persistente:** Uma mensagem é armazenada no servidor de comunicação pelo tempo que demorar pra ela ser entregue.

# TIPOS DE COMUNICAÇÃO

## LOCAIS PARA SÍNCRONIZAÇÃO



- Na submissão da requisição
- Na entrega da requisição
- Depois do processamento da requisição

# CLIENTE SERVIDOR

## ALGUMAS OBSERVAÇÕES ..

Computação cliente/servidor é geralmente baseada no modelo de **comunicação transiente síncrona**:

- Cliente e servidor precisam estar ativos no momento da comunicação
- Cliente emite requisições e bloqueia até receber resposta
- Servidor essencialmente espera somente por requisições entrantes, e subsequentemente as processa.

## PROBLEMAS DE COMUNICAÇÃO SÍNCRONA

- Cliente não pode fazer nenhum outro trabalho enquanto espera por resposta
- Falhas tem que ser manuseadas imediatamente: o cliente está esperando
- O modelo pode simplesmente não ser apropriado (mail, news)

# PASSANDO MENSAGENS

## MIDDLEWARE ORIENTADO A MENSAGENS

Foca na **comunicação** de alto nível **assíncrona persistente**:

- Processos enviam mensagens uns para outros, que são enfileirados (queued)
- Remetente não precisa esperar por uma resposta imediata, e pode fazer outras coisas
- Middleware normalmente garante tolerância a falhas



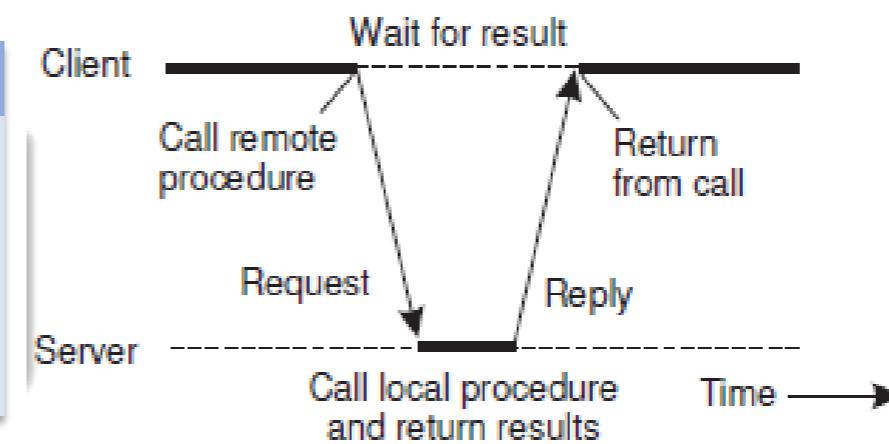
# RPC - OPERAÇÃO BÁSICA

## OBSERVAÇÕES

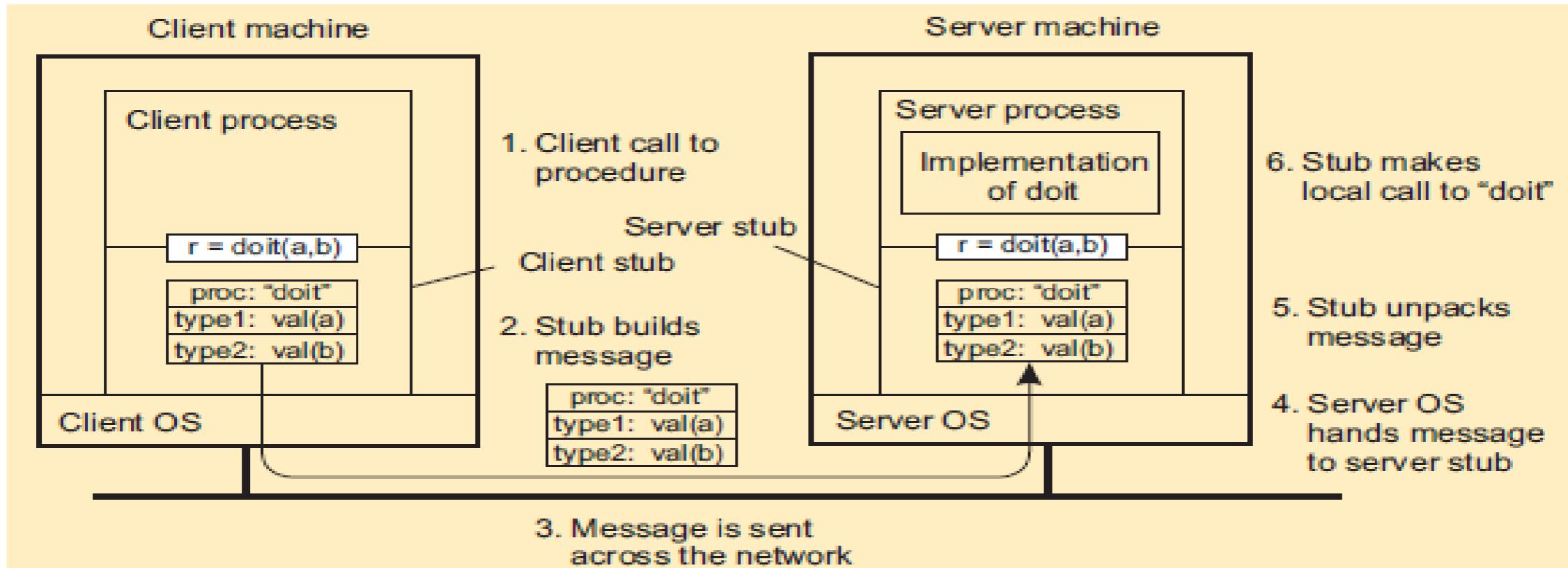
- Desenvolvedores normalmente usam o modelo procedural simples
- Procedimentos bem engenhados operam isolados (black box)
- Não existe uma razão fundamental para não executar procedimentos em máquinas separadas

## CONCLUSÃO

Comunicação entre chamador e chamado pode ser escondida usando um mecanismo de chamada de procedimento.



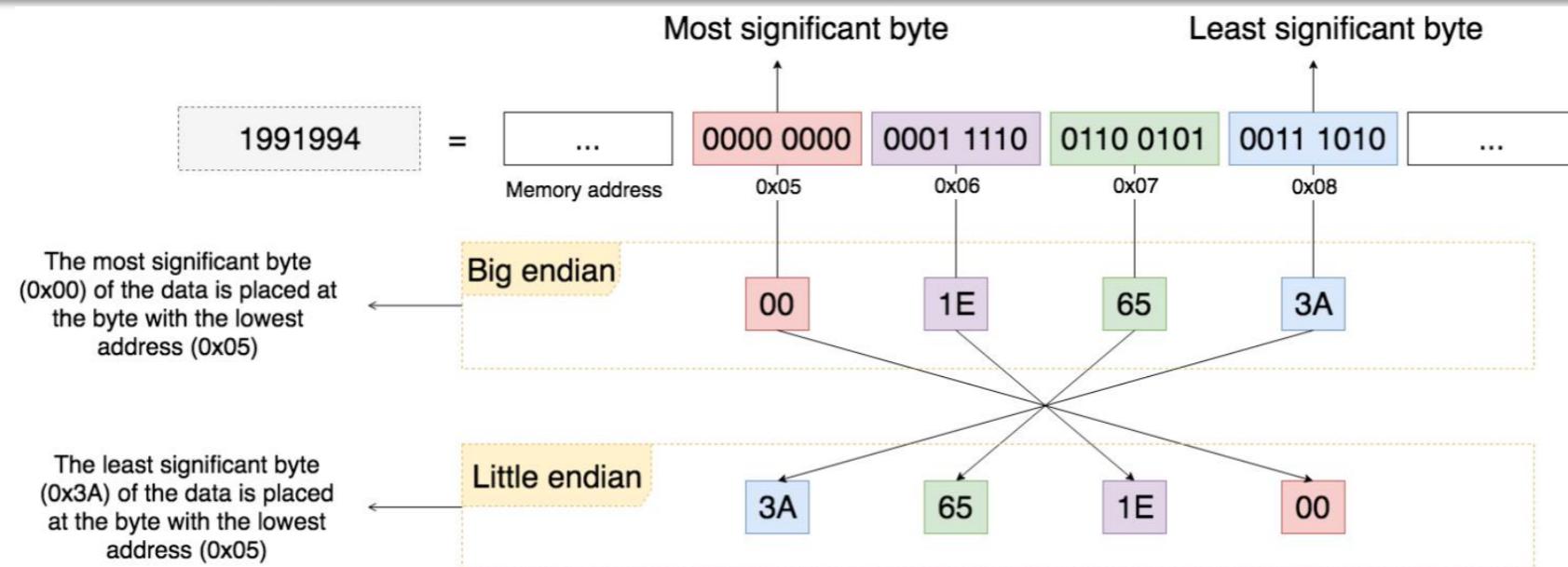
# RPC OPERAÇÃO BÁSICA



1. Procedimento Cliente chama stub cliente
2. Stub monta mensagem; chama SO local
3. SO envia mensagem para SO remoto.
4. SO remote passa a mensagem para o stub
5. Stub desempacota parâmetros e chama servidor
6. Servidor faz uma chamada local; retorna resultado para stub.
7. Stub monta mensagem; chama o SO.
8. SO envia mensagem para o SO do cliente.
9. SO do cliente passa a mensagem para o stub.
10. Stub do cliente desempacota o resultado; retorna para o cliente.

### OBSERVAÇÕES

- As máquinas cliente e servidor podem ter **diferentes representações de dados** (big/little endian)
- Wrapping de parâmetros significa **transformer um valor em uma sequência de bytes**
- Cliente e servidor tem que concordar em uma mesma codificação:
  - Como os valores de dados básicos são representados (integers, floats, characters)
  - Como os valores complexos são representados (arrays, unions)



### CONCLUSÃO

Cliente e servidor precisam interpretar corretamente mensagens, e transformá-las em representações dependentes de máquina.

# RPC PASSAGEM DE PARÂMETROS

RPC: PASSAGEM DE PARÂMETROS

## ALGUMAS SUPOSIÇÕES

- **Copy in/copy out** semântica: enquanto o procedimento é executado, nada pode ser assumido sobre os valores dos parâmetros
- **Todos** dados que precisam ser operados são passados como parâmetros. Exclui passagem de referência para **(global) dados**.

## CONCLUSÃO

Transparência de acesso total não pode ser realizada ...

## UM MECANISMO DE REFERÊNCIA REMOTA MELHORA A TRANSPARÊNCIA DE ACESSO

- Referências remotas oferecem acesso unificado a dados remotos
- Referências remotas podem ser passadas como parâmetros em RPCs
- Nota: stubs podem às vezes serem usados como tais referências

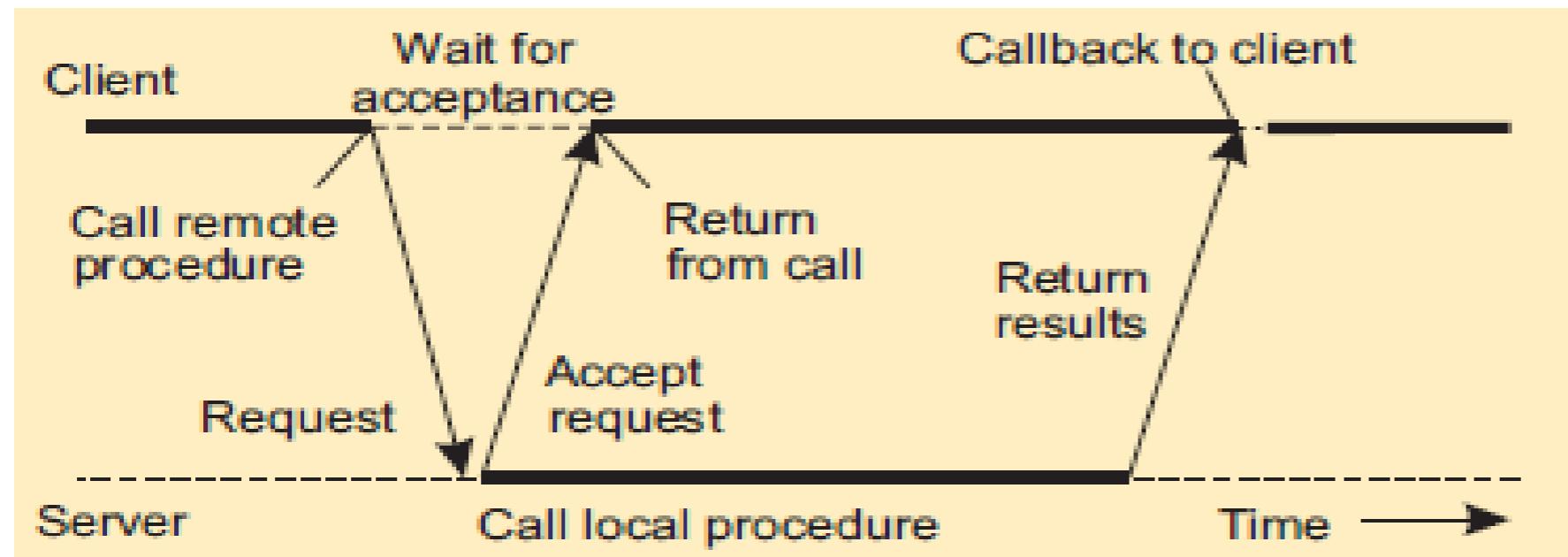


# RPC ASSÍNCRONA

## ESSÊNCIA

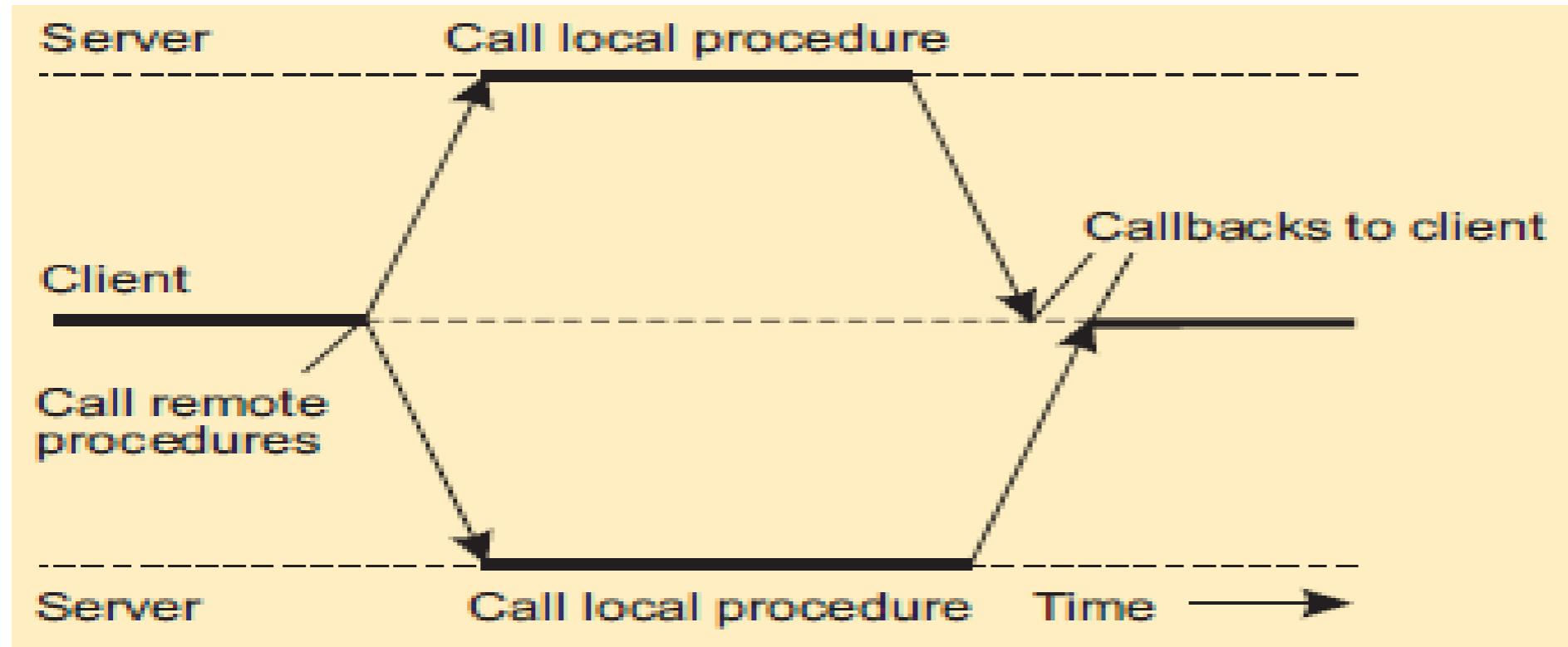
- Tente se livrar do comportamento estrito request/reply, mas deixe o cliente continuar sem esperar por uma resposta do servidor.

RPC CHAMADA ASSÍNCRONA



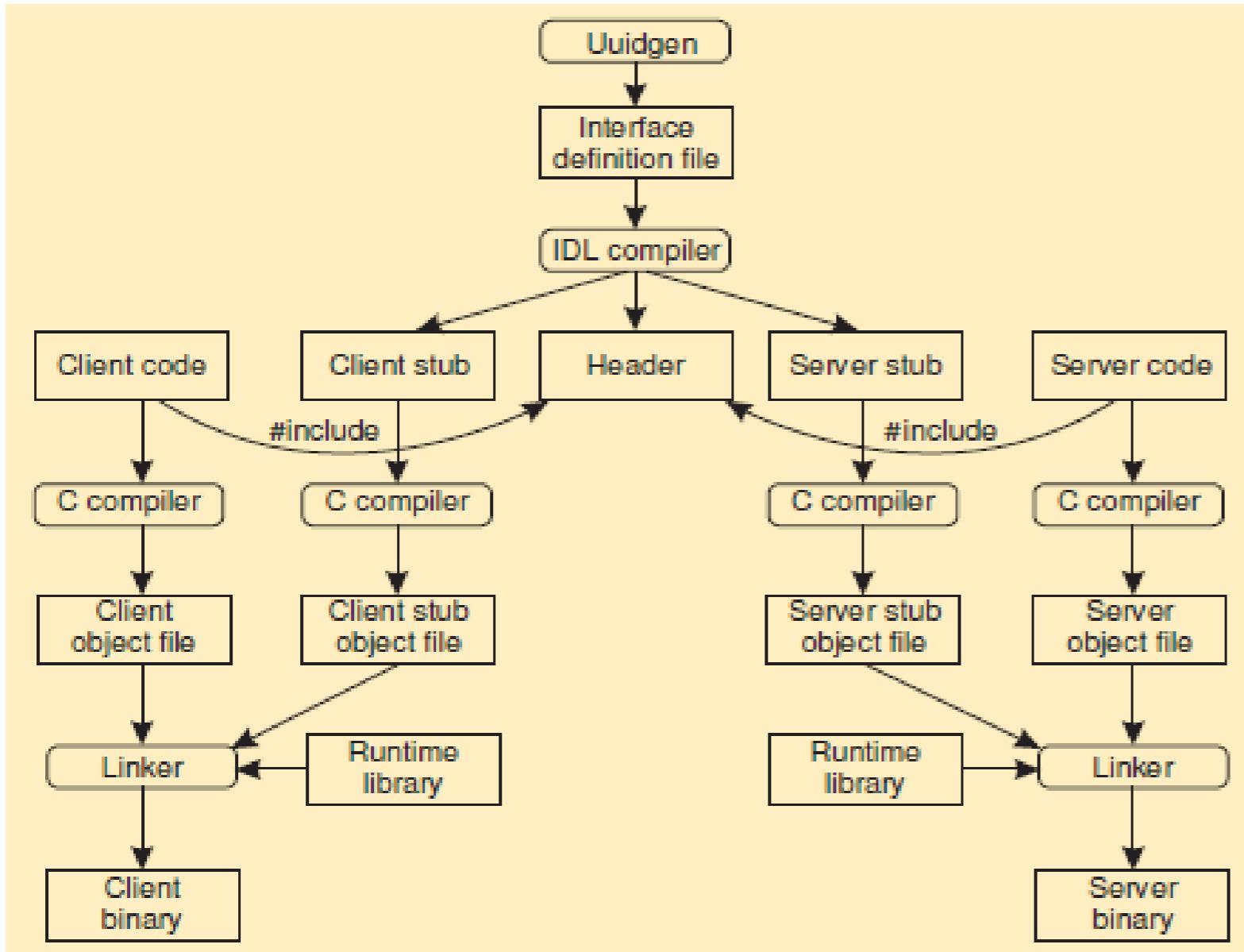
## ESSÊNCIA

- Enviando uma requisição RPC para um grupo de servidores.



# RPC NA PRÁTICA

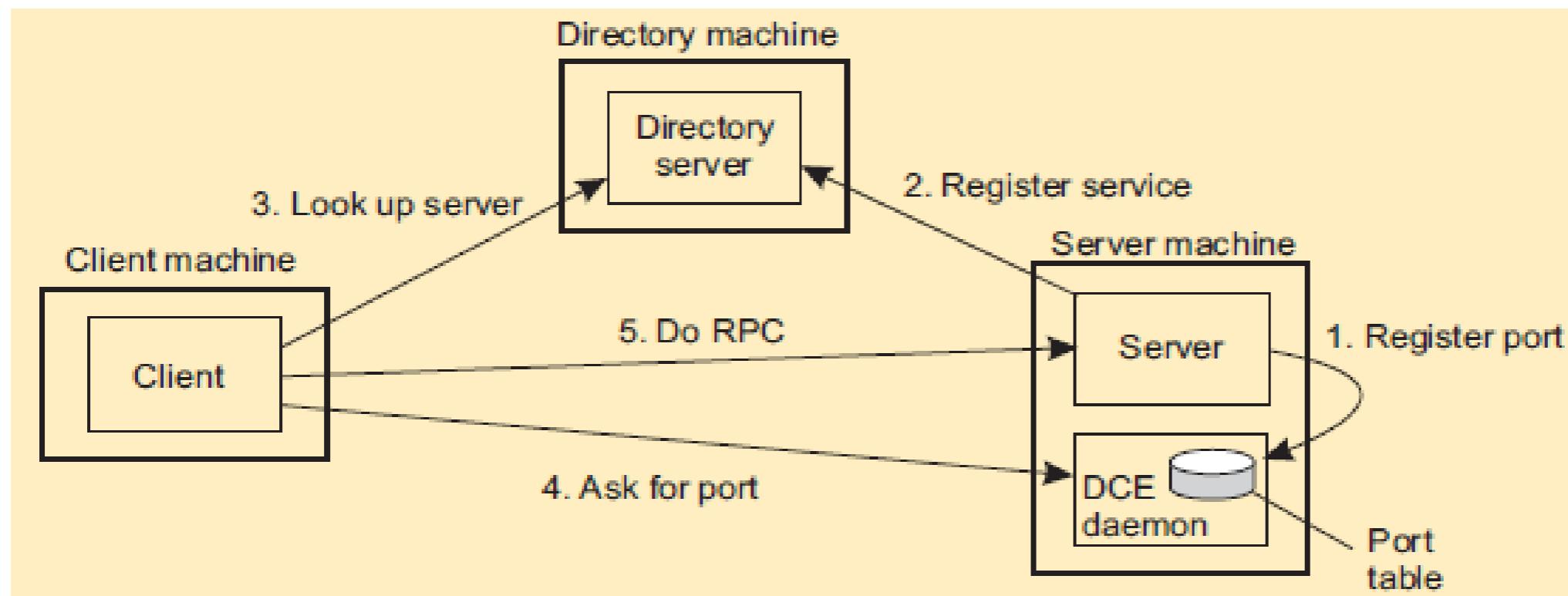
ESCREVENDO UM CLIENTE E UM SERVIDOR



# BIDING CLIENTE/SERVIDOR (DCE)

## QUESTÕES

(1) Cliente deve localizar a máquina servidor, e (2) localizar o servidor.

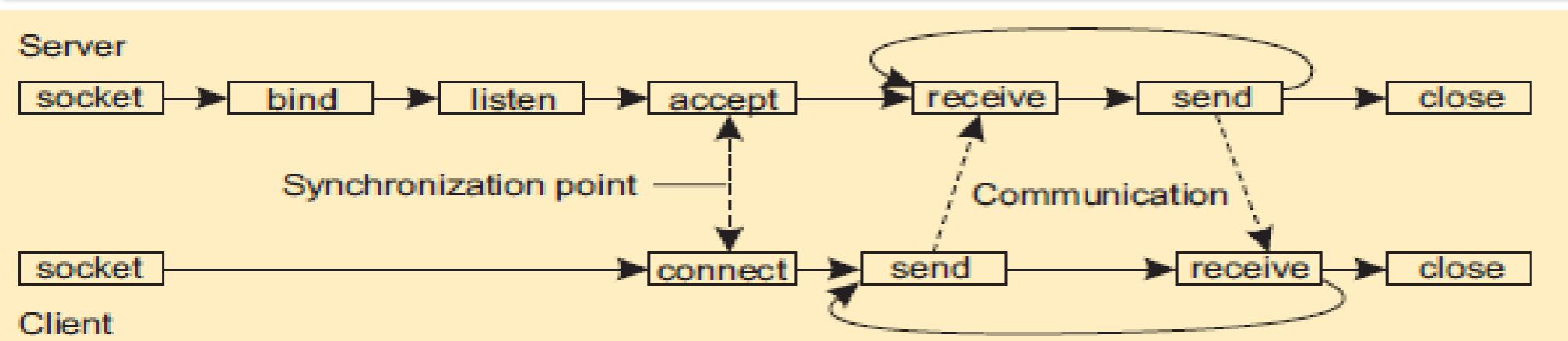


# MENSAGENS TRANSIENTES: SOCKETS

## API SOCKETS BERKELEY

Operation	Description
socket	Create a new communication end point
bind	Attach a local address to a socket
listen	Tell operating system what the maximum number of pending connection requests should be
accept	Block caller until a connection request arrives
connect	Actively attempt to establish a connection
send	Send some data over the connection
receive	Receive some data over the connection
close	Release the connection

CLIENTE/SERVIDOR BIDING



### SERVIDOR

```
1 from socket import *
2 s = socket(AF_INET, SOCK_STREAM)
3 (conn, addr) = s.accept()      # returns new socket and addr. client
4 while True:                   # forever
5     data = conn.recv(1024)      # receive data from client
6     if not data: break         # stop if client stopped
7     conn.send(str(data)+"*")   # return sent data plus an "*"
8 conn.close()                  # close the connection
```

### CLIENTE

```
1 from socket import *
2 s = socket(AF_INET, SOCK_STREAM)
3 s.connect((HOST, PORT))        # connect to server (block until accepted)
4 s.send('Hello, world')        # send some data
5 data = s.recv(1024)           # receive the response
6 print data                   # print the result
7 s.close()                     # close the connection
```

# FACILITANDO O USO DE SOCKETS

## OBSERVAÇÃO

- Sockets representa um nível bem baixo de programação e erros podem acontecer facilmente. Entretanto, a forma como eles são usados não muda (como exemplo em ajustes de cliente-servidor).

## ALTERNATIVA: ZeroMQ

- Provê um nível mais alto para expressar o **pareamento** de sockets: um para enviar mensagens do processo P e um correspondente para receber mensagens em Q. Toda comunicação é **assíncrona**.

## TRÊS PADRÕES

- Request-reply
- Publish-subscribe
- Pipeline



## SERVIDOR

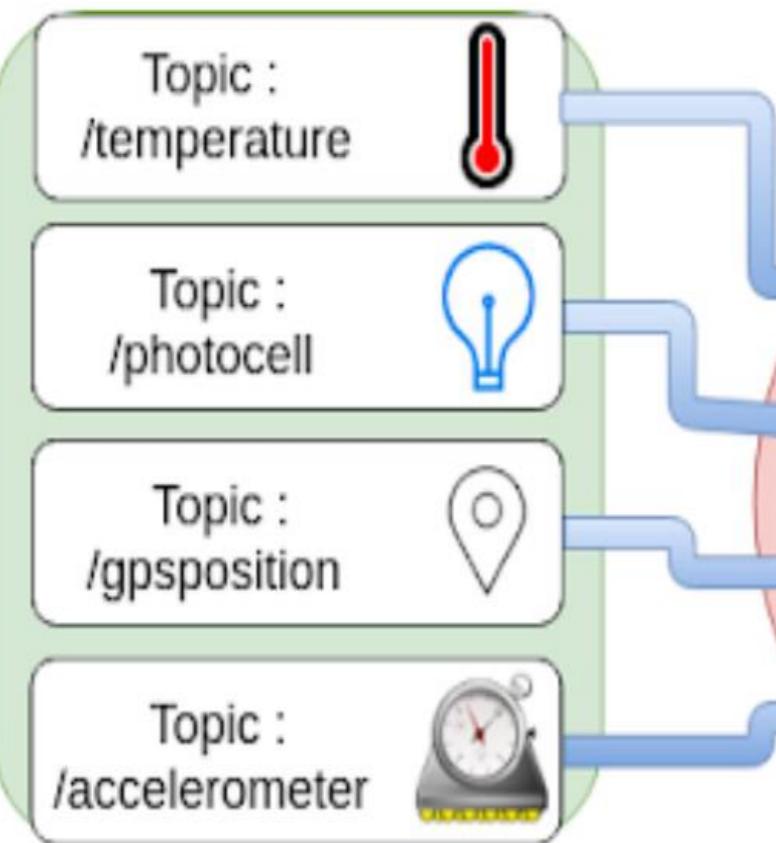
```
1 import zmq
2 context = zmq.Context()
3
4 p1 = "tcp://" + HOST + ":" + PORT1 # how and where to connect
5 p2 = "tcp://" + HOST + ":" + PORT2 # how and where to connect
6 s = context.socket(zmq.REP)       # create reply socket
7
8 s.bind(p1)                      # bind socket to address
9 s.bind(p2)                      # bind socket to address
10 while True:
11     message = s.recv()          # wait for incoming message
12     if not "STOP" in message:
13         s.send(message + "*")   # if not to stop...
14     else:
15         break                   # append "*" to message
16                             # else...
17                             # break out of loop and end
```

## CLIENTE

```
1 import zmq
2 context = zmq.Context()
3
4 php = "tcp://" + HOST + ":" + PORT # how and where to connect
5 s = context.socket(zmq.REQ)         # create socket
6
7 s.connect(php)                    # block until connected
8 s.send("Hello World")            # send message
9 message = s.recv()               # block until response
10 s.send("STOP")                  # tell server to stop
11 print message                   # print result
```

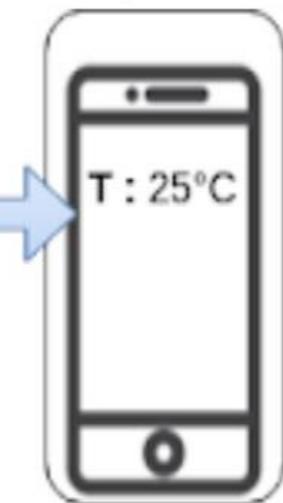
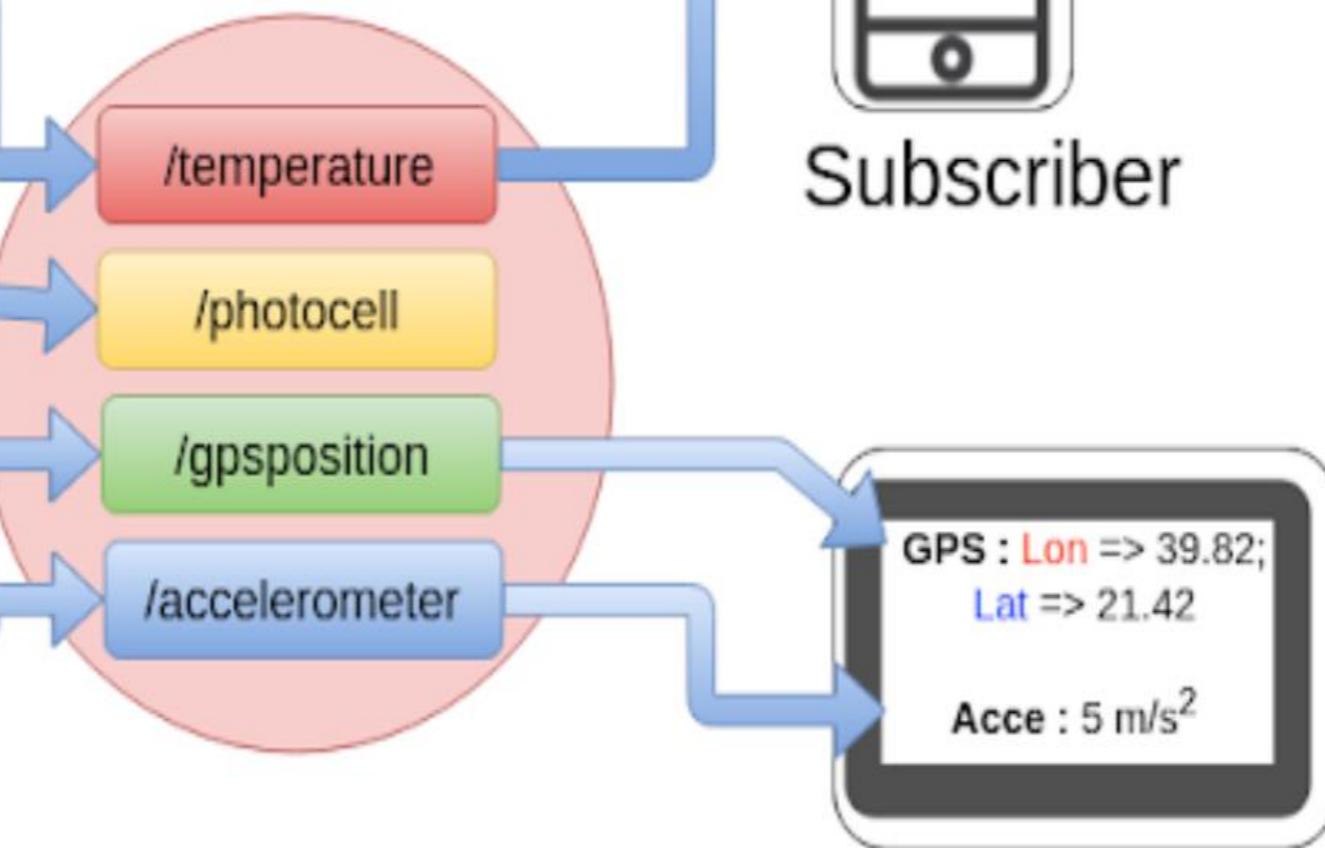


Publisher



Modelo MQTT (Pub/Sub)

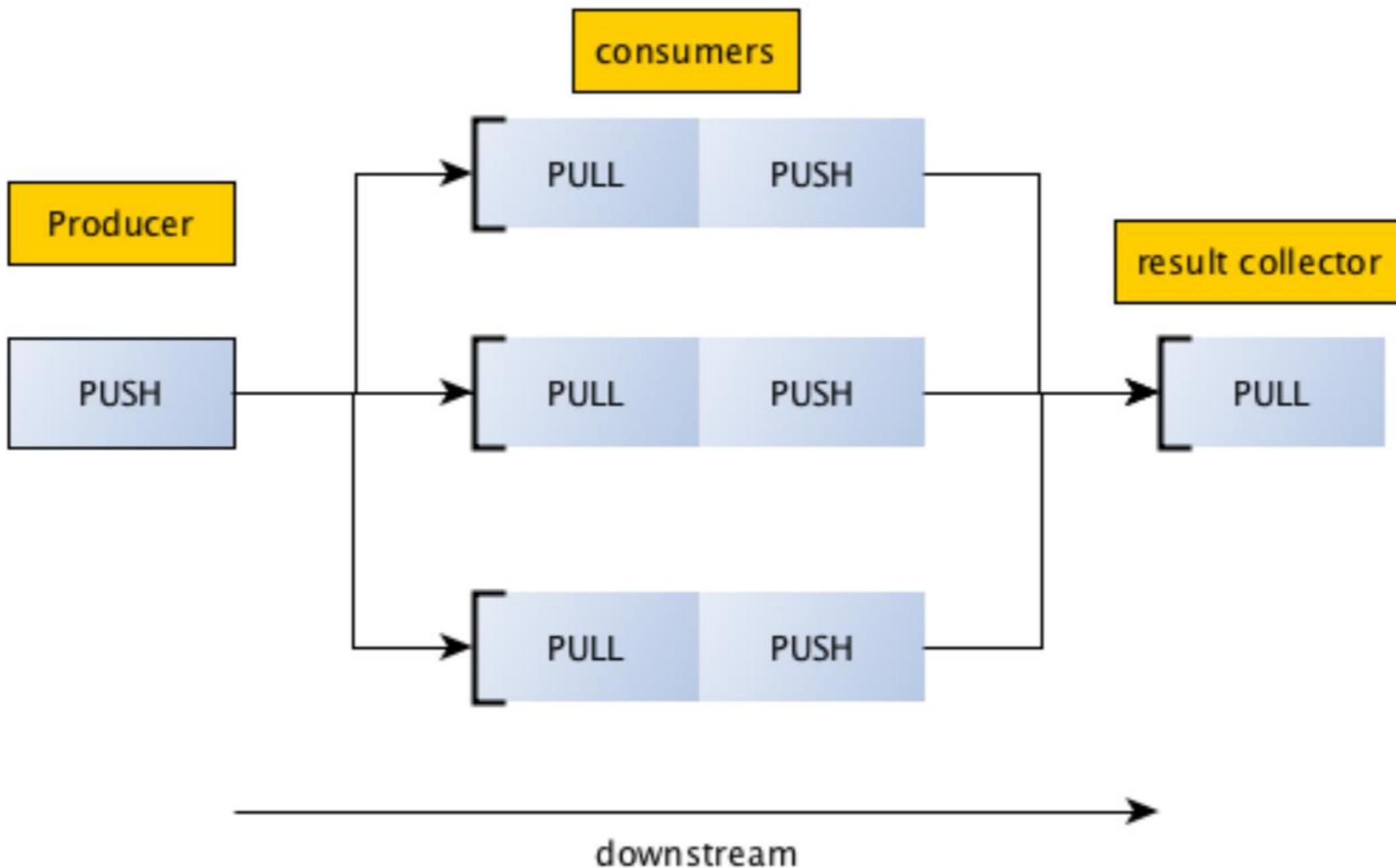
Broker



Subscriber



Subscriber



# MPI QUANDO É NECESSÁRIO FLEXIBILIDADE

## OPERAÇÕES REPRESENTATIVAS

Operation	Description
<code>MPI_bsend</code>	Append outgoing message to a local send buffer
<code>MPI_send</code>	Send a message and wait until copied to local or remote buffer
<code>MPI_ssend</code>	Send a message and wait until transmission starts
<code>MPI_sendrecv</code>	Send a message and wait for reply
<code>MPI_isend</code>	Pass reference to outgoing message, and continue
<code>MPI_issend</code>	Pass reference to outgoing message, and wait until receipt starts
<code>MPI_recv</code>	Receive a message; block if there is none
<code>MPI_irecv</code>	Check if there is an incoming message, but do not block



# MESSAGE ORIENTED MIDDLEWARE

## ESSÊNCIA

Comunicação assíncrona persistente com suporte de enfileiramento em nível de middleware. Filas correspondem a buffers em servidores de comunicação.

## OPERAÇÕES

Operation	Description
put	Append a message to a specified queue
get	Block until the specified queue is nonempty, and remove the first message
poll	Check a specified queue for messages, and remove the first. Never block
notify	Install a handler to be called when a message is put into the specified queue

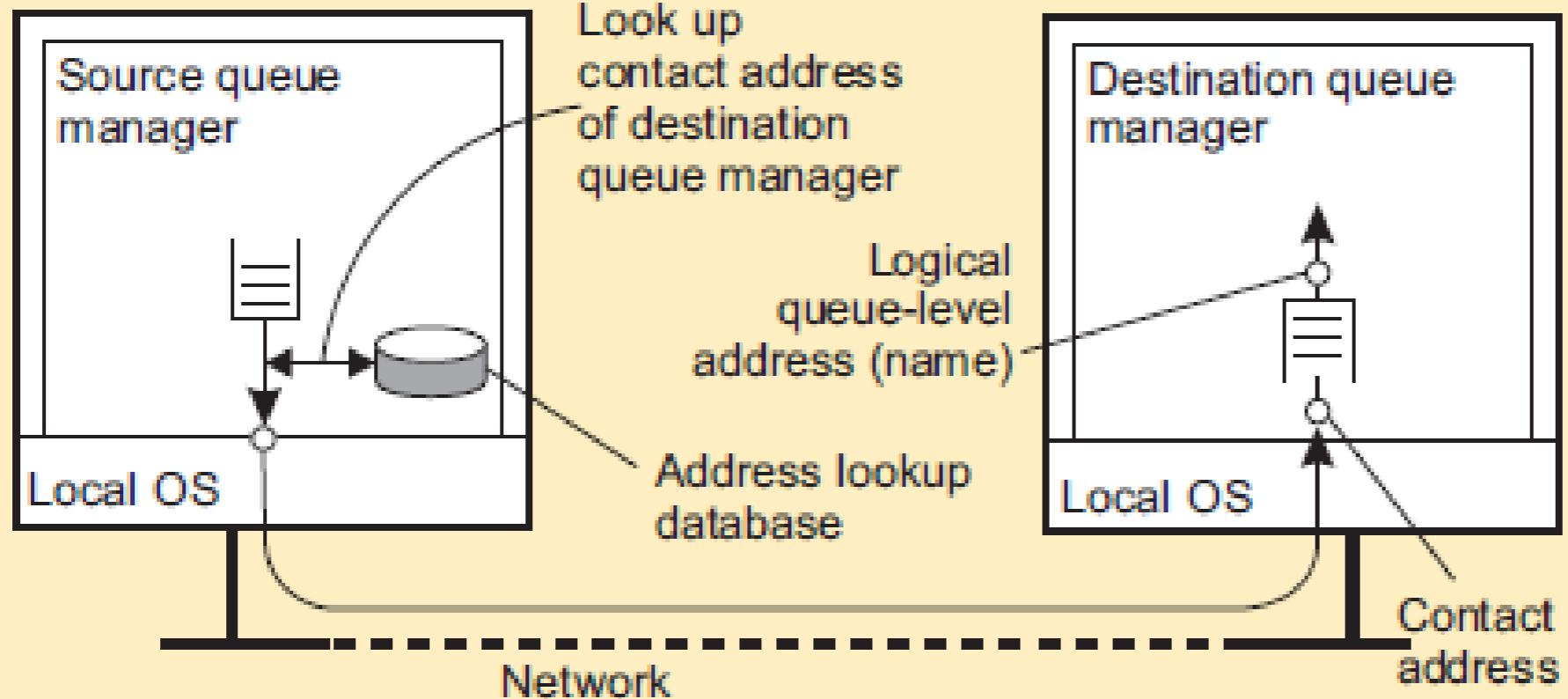


# MODELO GERAL

## GERENCIADORES DE FILA

Comunicação assíncrona persistente com suporte de enfileiramento em nível de middleware. Filas correspondem a buffers em servidores de comunicação.

## ROTEAMENTO



# MESSAGE BROKER

## OBSERVAÇÃO

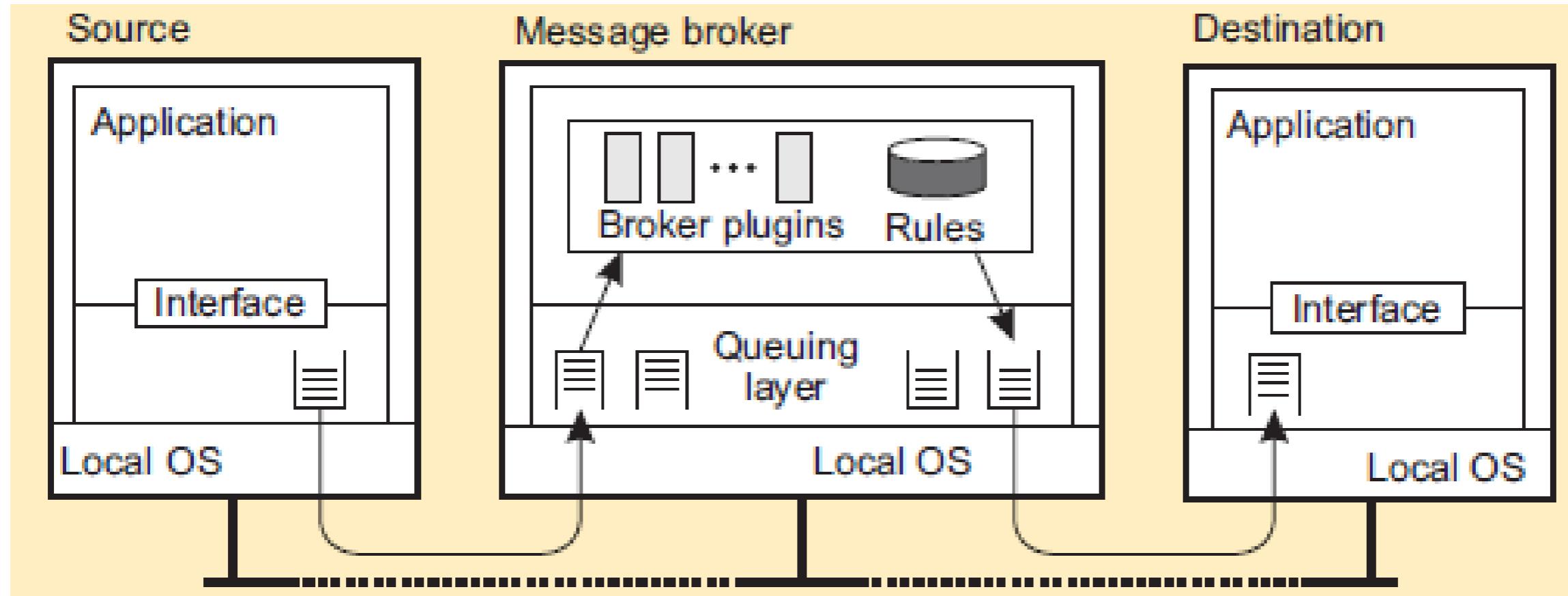
Sistemas de enfileiramento de mensagens assumem um protocolo comum de mensageria: todas as aplicações concordam com o formato da mensagem (i.e. representação e estrutura de dados)

## BROKER MANIPULA A HETEROGENEIDADE DE UM SISTEMA MQ

- Transforma mensagens entrantes em formato do destinatário alvo
- Age como um *gateway* de aplicação de forma frequente
- Pode prover capacidade de roteamento baseado no tópico (*subject-based*) (i.e., capacidades *publish-subscribe*)



# MESSAGE BROKER ARQUITETURA GERAL



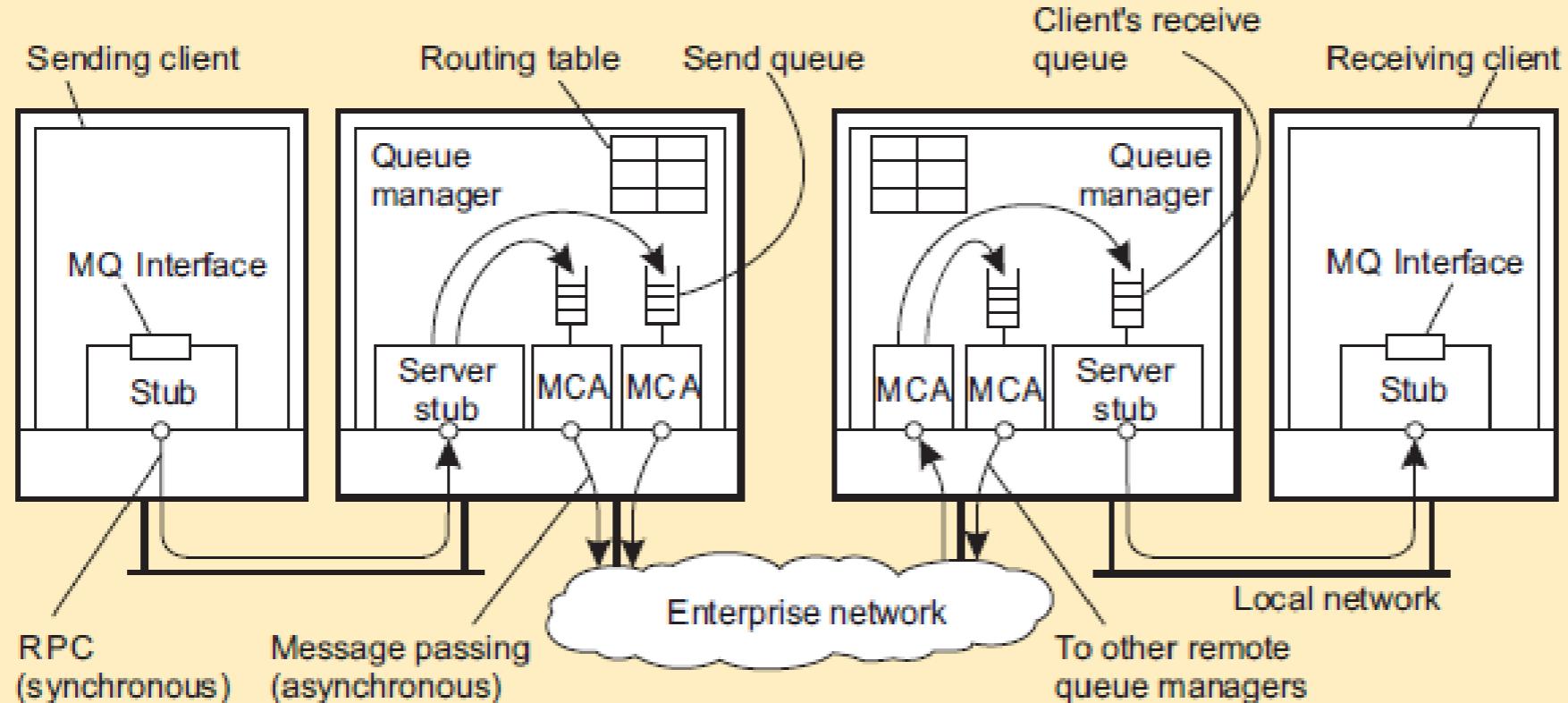
## CONCEITO BÁSICO

- Mensagens específicas da aplicação a são colocadas e removidas de filas
- Filas reside em um regime do gerenciador da fila
- Processos podem colocar mensagens somente em filas locais, ou através de um mecanismo RPC

## TRANSFERÊNCIA DE MENSAGEM

- Mensagens são transferidas entre filas
- transferência de mensagem entre filas em diferentes processos, demandam o uso de um canal (*channel*)
- Em cada *end point* de um canal existe um agente de mensagem do canal (*message channel agent*)
- *Message channel agents* são responsáveis por:
  - Criação (setup) de canais usando facilidades de comunicação de baixo nível da rede (ex. TCP/IP)
  - (*Un*)wrapping de mensagens de/em pacotes no nível do transporte
  - Envio / Recebimento de pacotes

## TRANSFERÊNCIA DE MENSAGEM



- Canais são inherentemente unidirecionais
- Automaticamente iniciam MCA's (message channel agent) quando uma mensagem chega
- Qualquer rede de gerenciadores de filas pode ser criada
- Roteadores são criados manualmente (administração do sistema)

# MCA Message Channel Agent

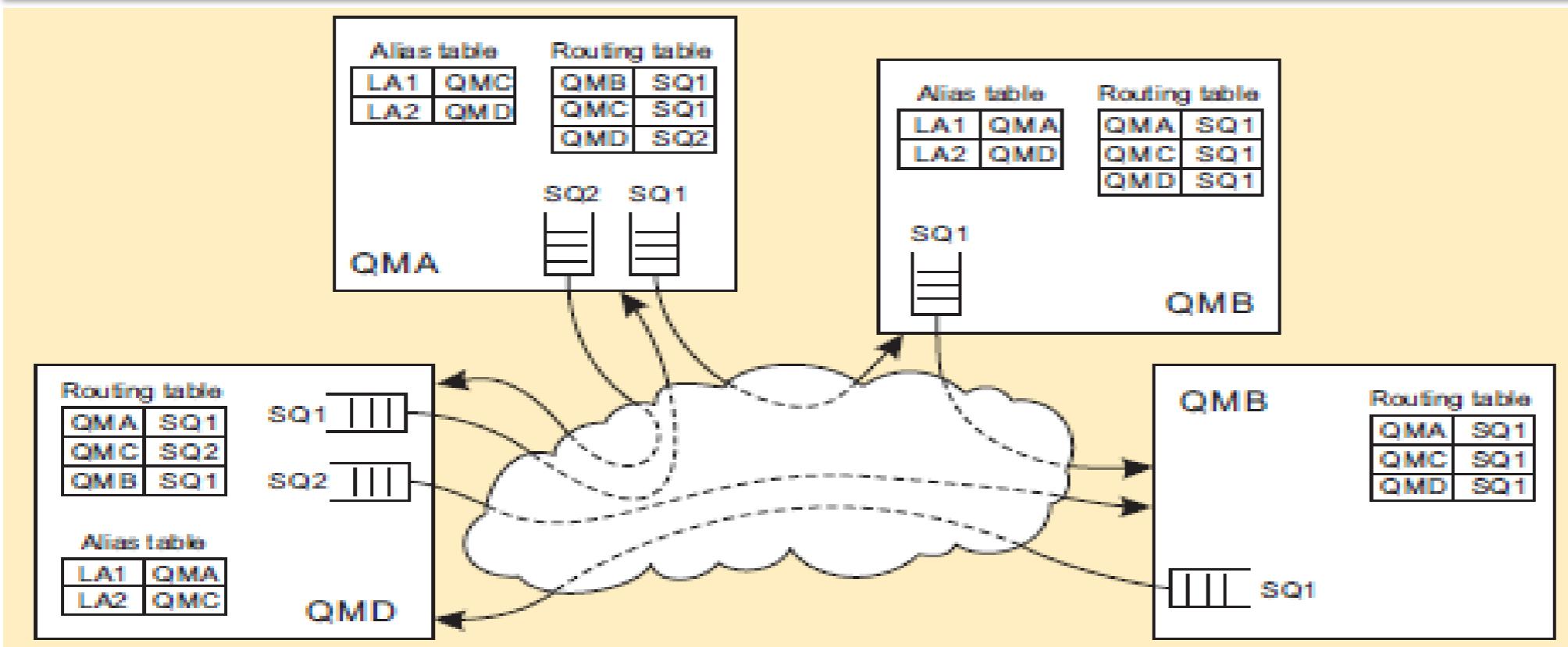
## ALGUNS ATRIBUTOS ASSOCIADOS COM MCA

Attribute	Description
Transport type	Determines the transport protocol to be used
FIFO delivery	Indicates that messages are to be delivered in the order they are sent
Message length	Maximum length of a single message
Setup retry count	Specifies maximum number of retries to start up the remote MCA
Delivery retries	Maximum times MCA will try to put received message into queue



## ROTEAMENTO

Usando **nomes lógicos**, em combinação com a resolução do nome para filas locais, é possível colocar uma mensagem em uma **fila remota**



# MULTICASTING EM NÍVEL APLICAÇÃO

## ESSÊNCIA

Organiza nós de um sistema distribuído em uma **rede overlay** e usa esta rede para disseminar dados:

- Muitas vezes uma **árvore**, levando a rotas únicas
- Alternativamente, também **redes Mesh**, que demanda uma forma de **roteamento**

# CHORD MULTICASTING EM NÍVEL APLICAÇÃO

MULTICAST NÍVEL APLICAÇÃO BASEADO EM ÁRVORE

## ABORDAGEM BÁSICA

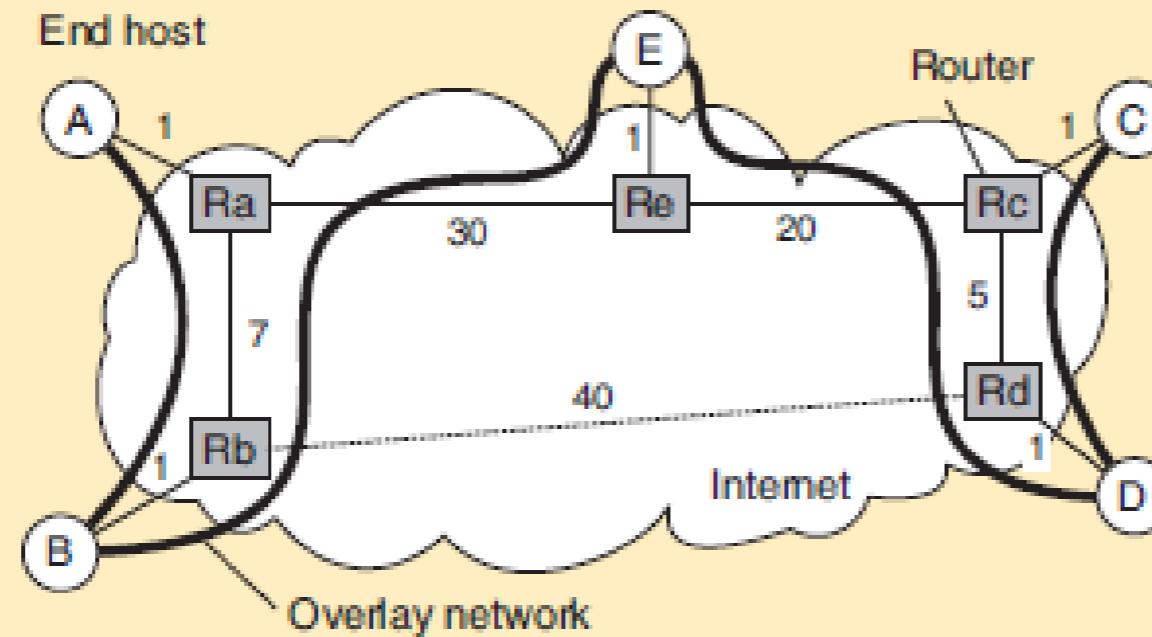
1. Iniciador gera um identificador multicast  $mid$
2. Lookup  $\text{succ}(mid)$ , o nó responsável por  $mid$
3. A requisição é roteada para  $\text{succ}(mid)$ , que vai se tornar a root
4. Se  $P$  quer se juntar, então  $P$  envia uma requisição **join** para root
5. Quando a requisição chega em  $Q$ :
  - $Q$  ainda não conhece uma requisição join  $\rightarrow$  e se torna um **forwarder**;  $P$  se torna filho de  $Q$ . **A requisição join continua a ser repassada.**
  - $Q$  sabe sobre a árvore  $\rightarrow P$  se torna filho de  $Q$ . **Não existe mais necessidade de repassar a requisição join.**



# CUSTOS DE MENSAGERIA EM NÍVEL APLICAÇÃO

## DESEMPENHO EM OVERLAYS

## DIFERENTES MÉTRICAS



**Stress no link:** com qual frequência uma ALM (application level message) atravessa o mesmo link físico ? **Exemplo:** mensagem de *A* para *D* precisa atravessar  $\langle R_a, R_b \rangle$  duas vezes.

**Stretch:** Razão em atraso entre caminho ALM e caminho de rede.

**Exemplo:** mensagens de *B* para *C* seguem um caminho de comprimento 73 em ALM, mas 47 em nível de rede → stretch = 73/47

# COMUNICAÇÃO MULTICAST **FLOODING**

## ESSÊNCIA

$P$  envia mensagem  $m$  para todos seus vizinhos. Cada vizinho repassa esta mensagem, exceto para  $P$ , e somente se não viu a mesma mensagem  $m$  antes.

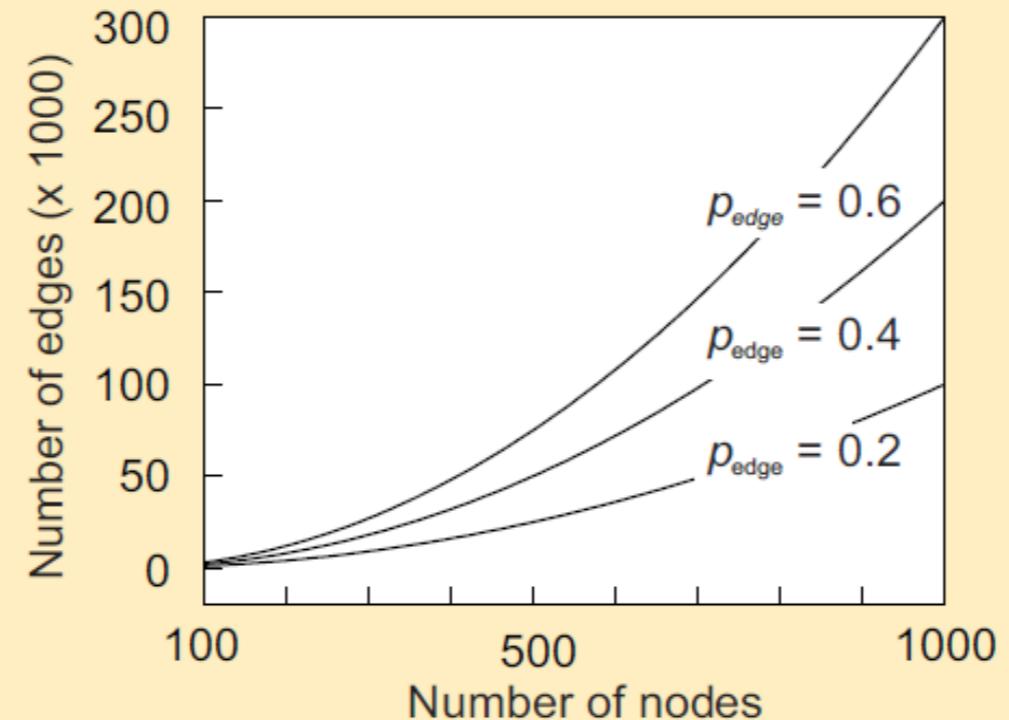
## DESEMPENHO

Quanto mais bordas, mais custoso !

## VARIAÇÃO

Se  $Q$  repassar uma mensagem com uma certa probabilidade  $P_{flood}$ , possivelmente dependente de seu próprio número de vizinhos (i.e. **node degree**) ou o grau de seus vizinhos.

## O TAMANHO DE OVERLAYS



FLOODING-BASED MULTICAST

# PROTOCOLOS EPIDÊMICOS

## ASSUMA QUE NÃO EXISTE CONFLITO ESCRITA-ESCRITA

- Operações de atualização (update) são executadas em um único servidor
- Uma replica passa o estado da atualização para somente alguns vizinhos
- A propagação da atualização é preguiçosa, i.é., não imediata
- Finalmente cada atualização chega em cada réplica

## DUAS FORMAS DE EPIDEMIA

- **Anti-entropia:** Cada replica regularmente escolhe outra replica de forma aleatória e troca as diferenças de estado, o que leva a estados idênticos em ambos depois da troca
- **Espalhamento de rumor:** Uma replica que foi recentemente atualizada (i.é., foi **contaminada**), conta para outras replicas sobre sua atualização (contaminando elas também)

# ANTI ENTROPIA

## PRINCIPAIS OPERAÇÕES

- Um nó  $P$  seleciona outro nó  $Q$  de um Sistema de forma aleatória
- **Pull**:  $P$  puxa somente dados novos de  $Q$  (de atualizações)
- **Push**:  $P$  somente empurra sua própria atualização para  $Q$
- **Push-Pull**:  $P$  e  $Q$  enviam atualizações para ambos

## OBSERVAÇÃO

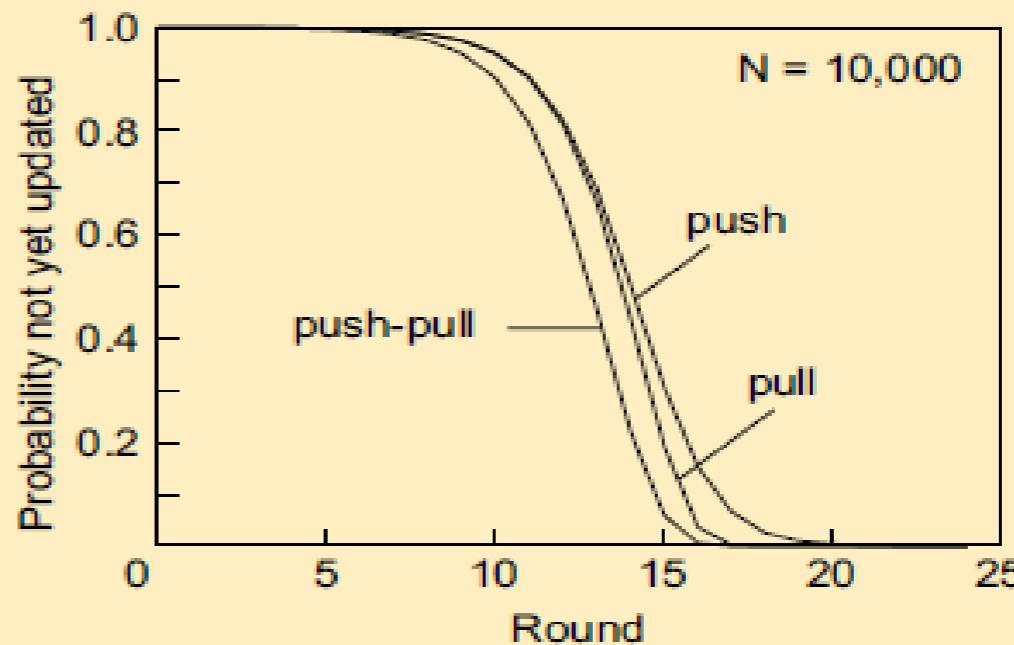
- Para push-pull, demora  $\mathcal{O}(\log(N))$  rodadas para atualizar todos os  $N$  nós (rodada = quando cada nó tomou a iniciativa de iniciar uma troca)



### BÁSICO

Considere uma única fonte propaganda suas atualizações. Faça  $P_j$  ser a probabilidade de um nó não ter recebido a atualização depois da  $i$ -enésima rodada.

- With **pull**,  $p_{i+1} = (p_i)^2$ : the node was not updated during the  $i^{th}$  round and should contact another ignorant node during the next round.
- With **push**,  
 $p_{i+1} = p_i(1 - \frac{1}{N})^{N(1-p_i)} \approx p_i e^{-1}$  (for small  $p_i$  and large  $N$ ): the node was ignorant during the  $i^{th}$  round and no updated node chooses to contact it during the next round.
- With **push-pull**:  $(p_i)^2 \cdot (p_i e^{-1})$



# ESPALHAMENTO DE RUMOR

## MODELO BÁSICO

Um servidor  $S$  possuindo uma atualização para reporter contato outros servidores. Se um servidor é contatado e o mesmo já recebeu uma atualização propaganda,  $S$  para de contatar outros servidores com probabilidade  $P_{stop}$

## OBSERVAÇÃO

Se  $S$  é uma fração de servidores ignorantes (i.é, não sabem da atualização), pode ser mostrado que para muitos servidores

$$s = e^{-(1/P_{stop} + 1)(1-s)}$$

# ANÁLISE FORMAL

## NOTAÇÕES

Faça  $S$  denotar uma fração de nós que ainda não foram atualizados (i.é. **Suscetível**),  $i$  a fração de atualizados (**infetados**) e nós ativos; e  $r$  a fração de nós atualizados que desistiram (**removidos**)

## DA TEORIA DA EPIDEMIA

$$(1) \quad ds/dt = -s \cdot i$$

$$(2) \quad di/dt = s \cdot i - p_{stop} \cdot (1 - s) \cdot i$$

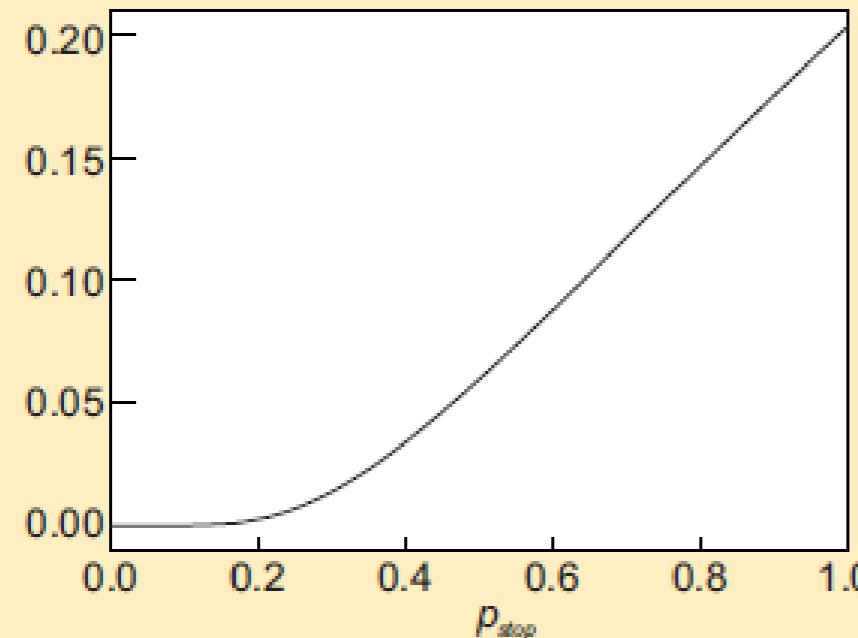
$$\Rightarrow \quad di/ds = -(1 + p_{stop}) + \frac{p_{stop}}{s}$$

$$\Rightarrow \quad i(s) = -(1 + p_{stop}) \cdot s + p_{stop} \cdot \ln(s) + C$$

## RESUMINDO

$i(1) = 0 \Rightarrow C = 1 + p_{stop} \Rightarrow i(s) = (1 + p_{stop}) \cdot (1 - s) + p_{stop} \cdot \ln(s)$ . We are looking for the case  $i(s) = 0$ , which leads to  $s = e^{-(1/p_{stop}+1)(1-s)}$

### O EFEITO DE PARAR



Consider 10,000 nodes		
$1/p_{stop}$	$s$	$N_s$
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

### OBSERVAÇÃO

Se tivermos que garantir que todos servidores serão atualizados, somente o espalhamento de rumor não é suficiente.

# DELETANDO VALORES

## PROBLEMA FUNDAMENTAL

Não podemos remover um valor antigo de um servidor e esperar a remoção se propagar. Ao invés disto, a mera remoção será desfeita no tempo devido usando algoritmos epidêmicos

## SOLUÇÃO

A remoção tem que ser registrada como uma atualização especial através da inserção de um **certificado de morte**



# DELETANDO VALORES

## QUANDO REMOVER UM CERTIFICADO DE MORTE (NÃO É PERMITIDO FICAR PRA SEMPRE)

- Rode um algoritmo global para detetar se a remoção é conhecida em todos lugares, e então cole os certificados de morte (se parece com coleta de lixo)
- Assuma que certificados de morte se propagam em tempo finito, e associe um tempo de vida máximo para um certificado (pode ser feito mas com risco de não atingir todos servidores)

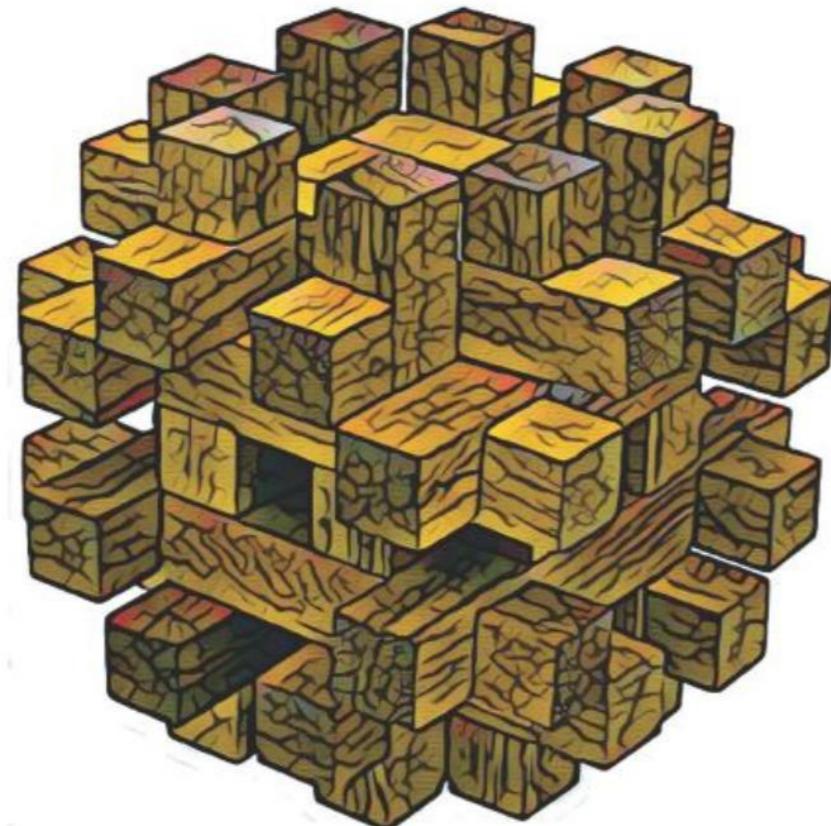
### NOTA

É necessário que a remoção de fato atinja todos servidores



# Distributed Systems

Maarten Van Steen & Andrew S.  
Tanenbaum



3th Edition – Version 3.01 - 2017

## Capítulo 5

### Nomeação

Sábado, 03 de Julho de 2021

# NOMEAÇÃO (NAMING)

## ESSÊNCIA

- Nomes são usados para denotar entidades em um sistema distribuído. Para operar uma entidade, precisamos ter acesso ao seu **ponto de acesso (access point)**. Pontos de acesso são nomeados por intermédio de **endereços**.

## NOTA

- Um nome **independente de localidade** para uma entidade  $E$ , é independente do endereço dos endereços dos pontos de acesso oferecidos por  $E$ .

# IDENTIFICADORES

## NOME PURO

Um nome que não tem significado nenhum; é somente um *string* aleatório. Nomes puros podem ser usados para comparação somente.

## IDENTIFICADOR

1. Um identificador se refere a no máximo uma entidade.
2. Cada entidade é referenciada por no máximo um identificador.
3. Um identificador sempre se refere a mesma entidade (i.é, nunca é reusado)

## OBSERVAÇÃO

Um identificador não precisa ser um nome puro – i.é, ele pode ter conteúdo



# BROADCASTING

## BROADCAST DO ID, REQUISITANDO UMA ENTIDADE A RETORNAR SEU ENDEREÇO ATUAL

- Nunca pode escalar além da rede local
- Demanda que todos processos ouçam (*listen*) requisições locais entrantes.

## ARP

- Para descobrir qual endereço MAC está associado com um endereço IP, faça *broadcast* da pergunta “quem tem o endereço IP”



# REPASSANDO PONTEIROS

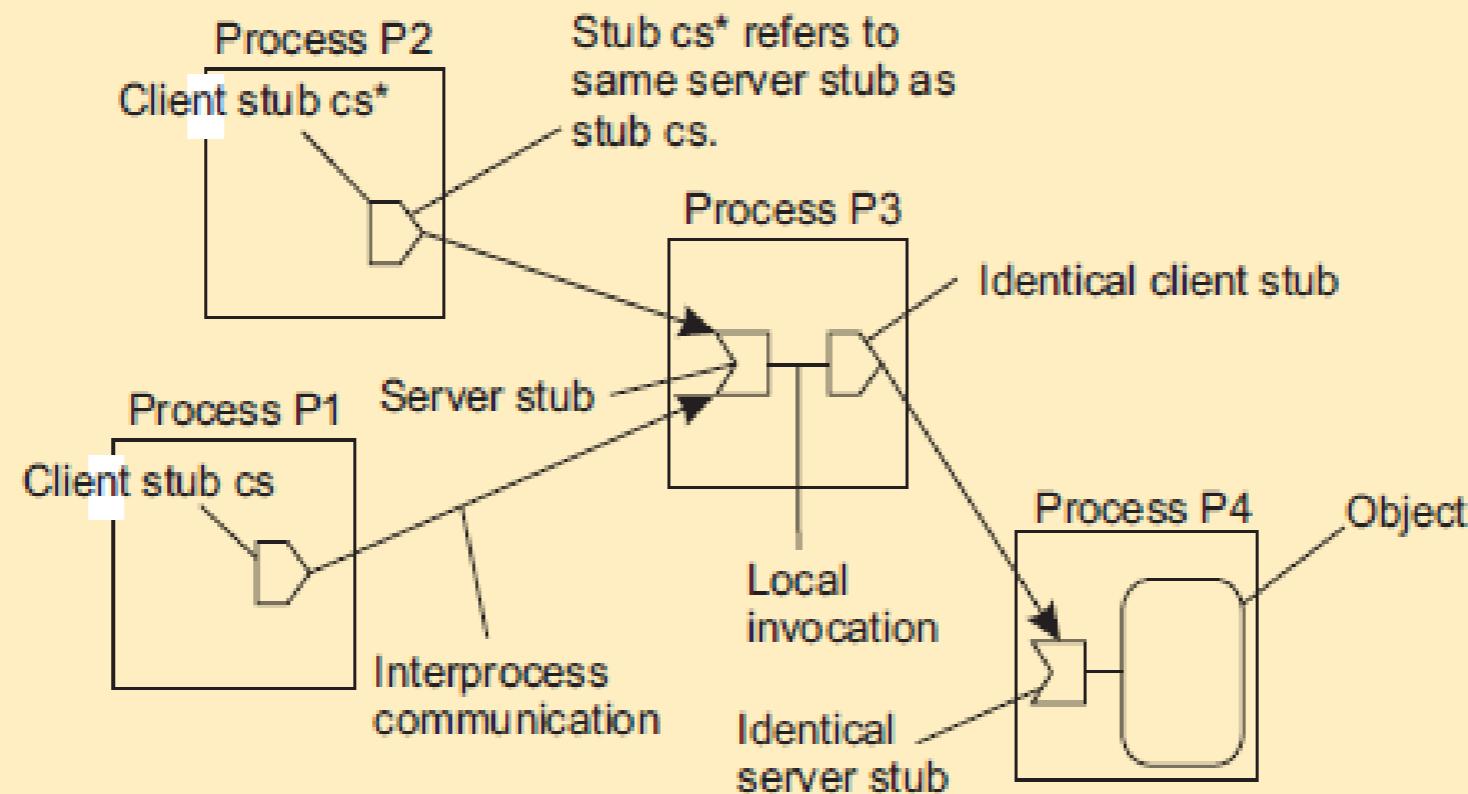
QUANDO UMA ENTIDADE SE MOVE, ELA DEIXA PRA TRÁS UM PONTEIRO COM SUA PRÓXIMA LOCALIZAÇÃO

- Desreferenciamento pode ser feito totalmente de forma transparente para clientes através do seguimento de uma cadeia de ponteiros
- Atualização da referência do cliente quando a localização local é achada
- Problemas de escalabilidade geográfica (para o qual mecanismos separados de cadeia de redução são necessárias):
  - Longas cadeias não são tolerantes a falha
  - Aumenta a latência da rede no desreferenciamento



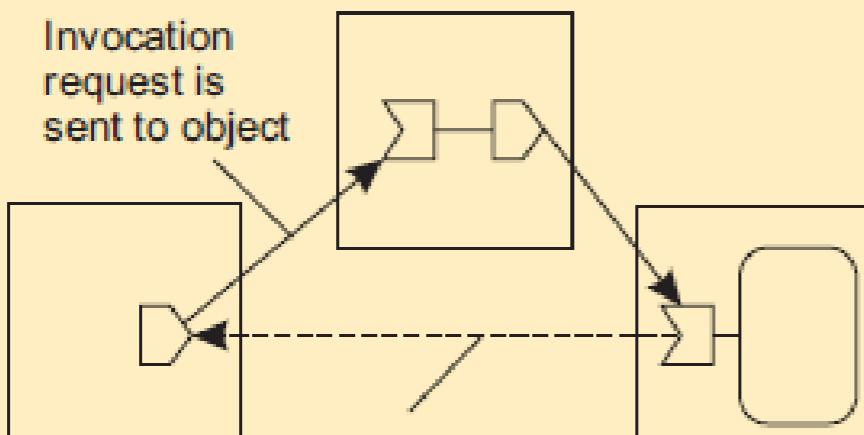
# EXEMPLO CADEIAS SSP (single shared platform)

## O PRINCÍPIO DE REPASSE DE PONTEIROS USANDO STUB CLIENTE E STUB SERVIDOR

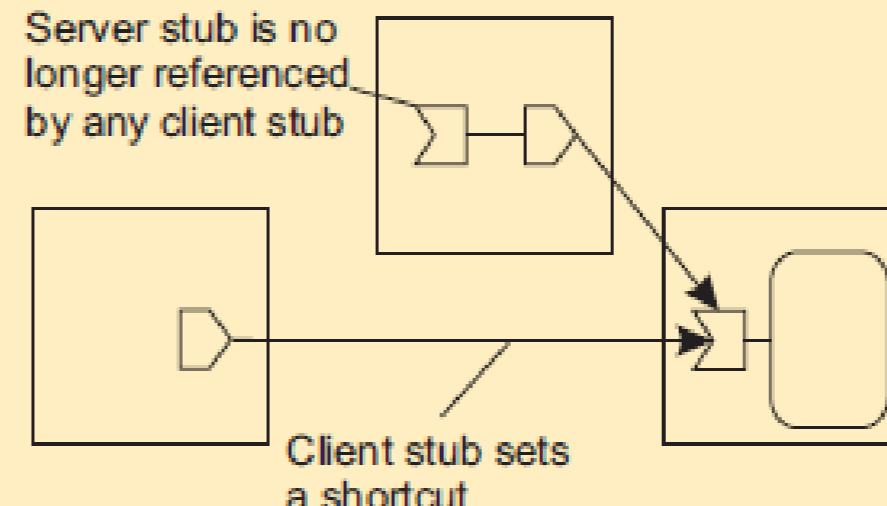


# EXEMPLO CADEIAS SSP

## REDIRECIONANDO UM PONTEIRO REPASSADO ATRAVÉS DO ARMAZENAMENTO DE UM ATALHO NO STUB CLIENTE



(a)



(b)

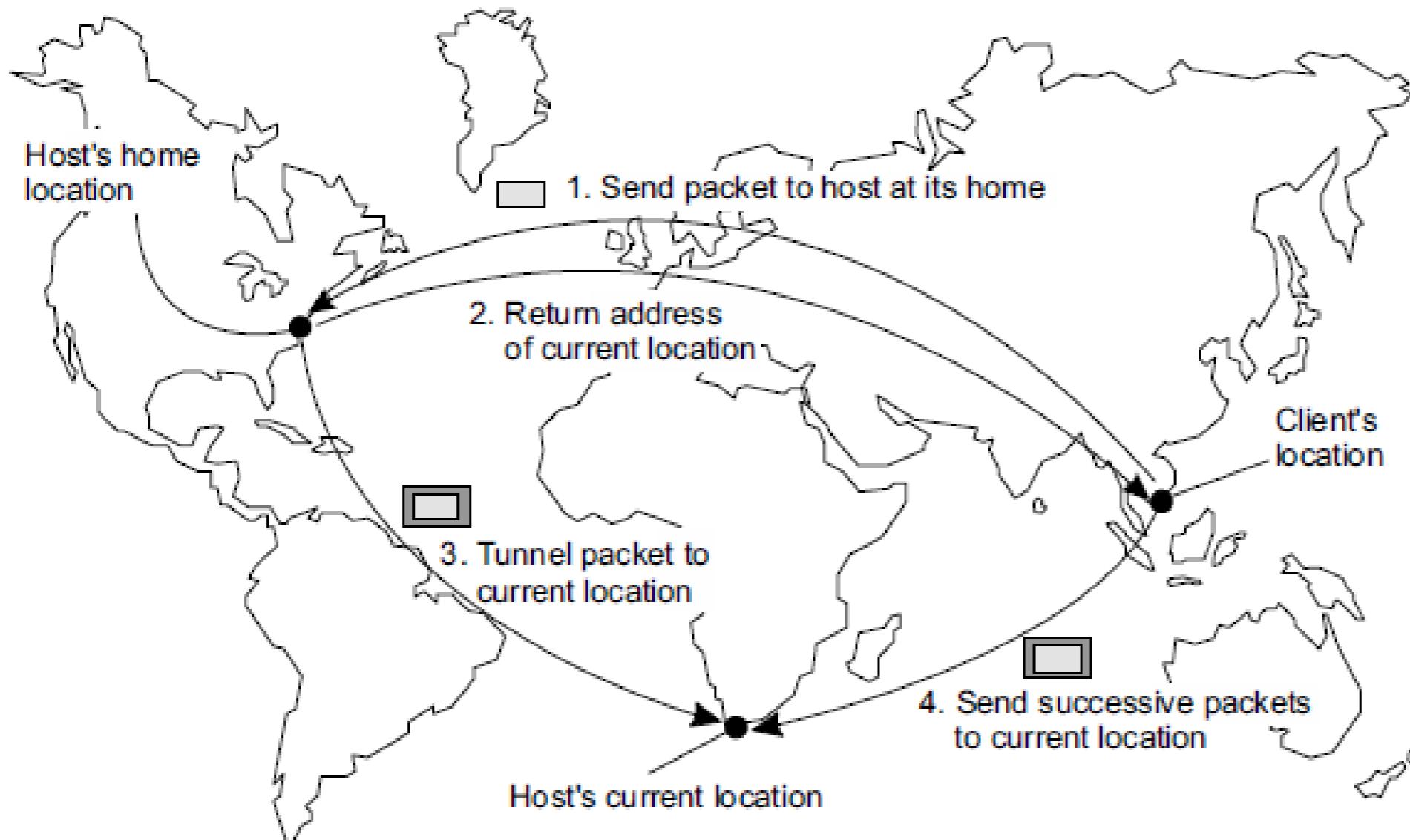
# ABORDAGEM BASEADA NO ENDEREÇO LOCAL

## ESQUEMA SINGLE-TIERED: DEIXE O LOCAL MANTER A LOCALIZAÇÃO DA ENTIDADE

- O endereço local registrado em um serviço de nomes
- Os registradores locais e endereço estrangeiro de uma entidade
- Cliente contata local primeiro e então contata registrador remoto



# O PRINCÍPIO DO IP MÓVEL



# ABORDAGEM HOME-BASED

## PROBLEMAS COM ABORDAGEM HOME BASED

- Endereço home tem que ser suportado por todo ciclo de vida da entidade
- O endereço home é fixo -> carga desnecessária quando a entidade se move de forma permanente
- Escalabilidade geográfica pobre (entidade pode estar perto do cliente)

## NOTA

- Mudança permanente pode ser atacada com outro nível de nomeação (DNS)



# EXEMPLO: CHORD

## PROBLEMAS COM ABORDAGEM HOME BASED

- Endereço home tem que ser suportado por todo ciclo de vida da entidade
- O endereço home é fixo -> carga desnecessária quando a entidade se move de forma permanente
- Escalabilidade geográfica pobre (entidade pode estar perto do cliente)

## NOTA

- Mudança permanente pode ser atacada com outro nível de nomeação (DNS)



## EXEMPLO: CHORD

### CONSIDERE A ORGANIZAÇÃO DE MUITOS NÓS EM UM ANEL LÓGICO

- Cada nó é atribuído um **identificador** randômico  $m$ -bit
- Cada entidade é atribuída uma chave única  $m$ -bit
- Entidade com chave  $k$  fica na jurisdição do nó com o menor  $id \geq k$  (chamado de **sucessor**  $\text{succ}(k)$ )

### NÃO SOLUÇÃO

- Deixe cada nó manter seus vizinhos e começar uma busca linear ao longo do anel

### NOTAÇÃO

- Quando falarmos nó  $p$  é o nó que possui identificador  $p$



# TABELAS: CHORD

## PRINCÍPIO

- Cada nó  $p$  mantém uma **tabela de apontamento**  $FT_p[]$  com no máximo  $m$  entradas:

$$FT_p[i] = succ(p + 2^{i-1})$$

**Nota:** a entrada  $i$ -ésima aponta para o primeiro nó sucessivo a  $p$  por pelo menos  $2^{i-1}$ .

- Para procurar uma chave  $k$ , o nó  $p$  repassa a requisição para o nó com índice  $j$  satisfazendo

$$q = FT_p[j] \leq k < FT_p[j + 1]$$

- Se  $p < k < FT_p[1]$ , a requisição é também repassada para  $FT_p[1]$



# TABELAS: CHORD

## PRINCÍPIO

- Cada nó  $p$  mantém uma **tabela de apontamento**  $FT_p[]$  com no máximo  $m$  entradas:

$$FT_p[i] = succ(p + 2^{i-1})$$

**Nota:** a entrada  $i$ -ésima aponta para o primeiro nó sucessivo a  $p$  por pelo menos  $2^{i-1}$ .

- Para procurar uma chave  $k$ , o nó  $p$  repassa a requisição para o nó com índice  $j$  satisfazendo

$$q = FT_p[j] \leq k < FT_p[j + 1]$$

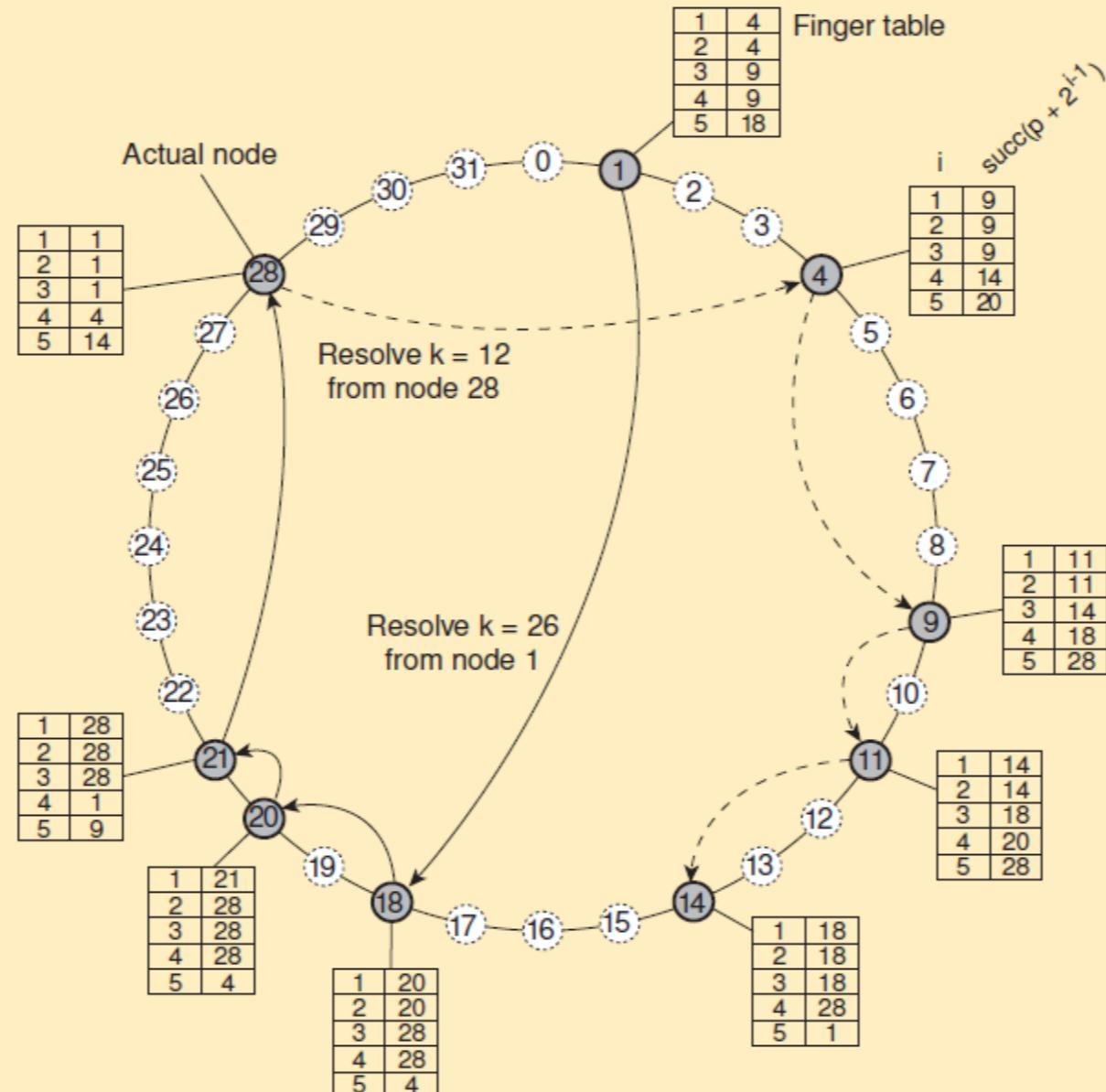
- Se  $p < k < FT_p[1]$ , a requisição é também repassada para  $FT_p[1]$



# CHORD: EXEMPLO LOOKUP

RESOLVENDO A CHAVE 26 A PARTIR DO NÓ 1 E CHAVE 12 A PARTIR DO NÓ 28

MECANISMO GERAL



# CHORD: EM PYTHON

(NOMEAÇÃO PLANA / TABELA HASH DISTRIBUÍDA)

MECANISMO GERAL

```
1  class ChordNode:
2      def finger(self, i):
3          succ = (self.nodeID + pow(2, i-1)) % self.MAXPROC      # succ( $p+2^{i-1}$ )
4          lwbi = self.nodeset.index(self.nodeID)                  # self in nodeset
5          upbi = (lwbi + 1) % len(self.nodeset)                  # next neighbor
6          for k in range(len(self.nodeset)):                      # process segments
7              if self.inbetween(succ, self.nodeset[lwbi]+1, self.nodeset[upbi]+1):
8                  return self.nodeset[upbi]                         # found successor
9          (lwbi, upbi) = (upbi, (upbi+1) % len(self.nodeset))    # next segment
10
11     def recomputeFingerTable(self):
12         self.FT[0] = self.nodeset[self.nodeset.index(self.nodeID)-1] # Pred.
13         self.FT[1:] = [self.finger(i) for i in range(1, self.nBits+1)] # Succ.
14
15     def localSuccNode(self, key):
16         if self.inbetween(key, self.FT[0]+1, self.nodeID+1): # in (FT[0], self]
17             return self.nodeID                                # responsible node
18         elif self.inbetween(key, self.nodeID+1, self.FT[1]): # in (self, FT[1])
19             return self.FT[1]                                 # succ. responsible
20         for i in range(1, self.nBits+1):                   # rest of FT
21             if self.inbetween(key, self.FT[i], self.FT[(i+1) % self.nBits]): # in [FT[i], FT[i+1]]
22                 return self.FT[i]
```



### PROBLEMA

- A organização lógica de nós na rede overlay pode levar a **transferências erradas de mensagens** na Internet abaixo: nó  $p$  e nó  $\text{succ}(p + 1)$  podem estar muito separados.

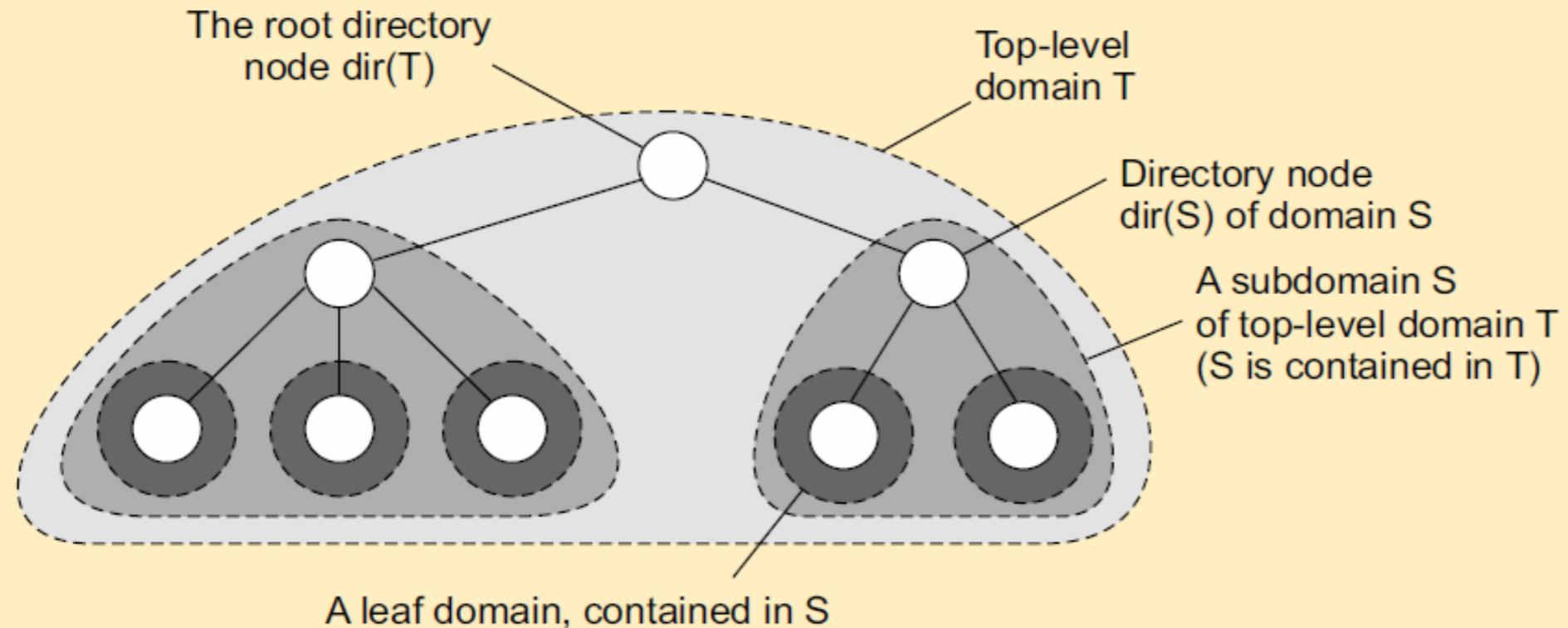
### SOLUÇÕES

- **Atribuição de nó ciente da topologia:** quando for fazer uma atribuição de um ID para um nó, tenha certeza que nós perto do espaço de ID estão também perto da rede. **Pode ser muito difícil.**
- **Roteamento na proximidade:** Mantém mais que um possível sucessor e repassa para o mais perto. **Exemplo:** no Chord  $FT_p[i]$  aponta para o primeiro nó em  $INT = [p + 2^{i-1}, p + 2^i - 1]$ . Nó  $p$  pode também armazenar ponteiros para outros nós em  $INT$ .
- **Seleção de vizinho na proximidade:** quando existe escolha para selecionar quem o vizinho será (não no Chord), pegue o mais próximo.

## IDÉIA BÁSICA

Construa uma árvore de busca em larga escala para a qual a rede inferior é dividida em domínios hierárquicos. Cada domínio é representado por um nó diretório separado.

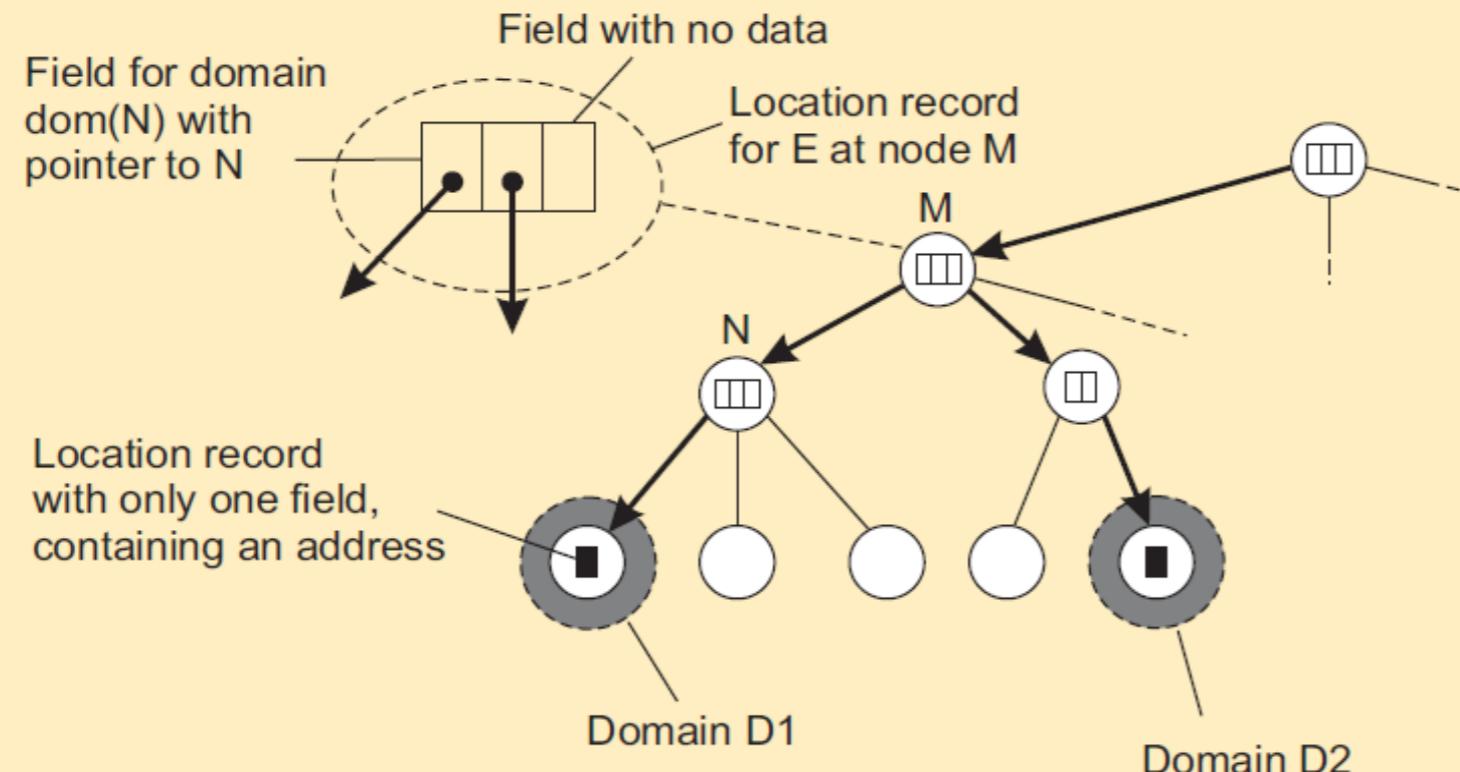
## PRINCÍPIO



### INVARIANTES

- Endereço de uma entidade  $E$  é armazenado em uma folha ou nó intermediário
- Nós intermediários contém um ponteiro para um nó filho se e somente se a sub-árvore roteada no nó filho armazena um endereço de uma entidade
- A raiz conhece todas entidades

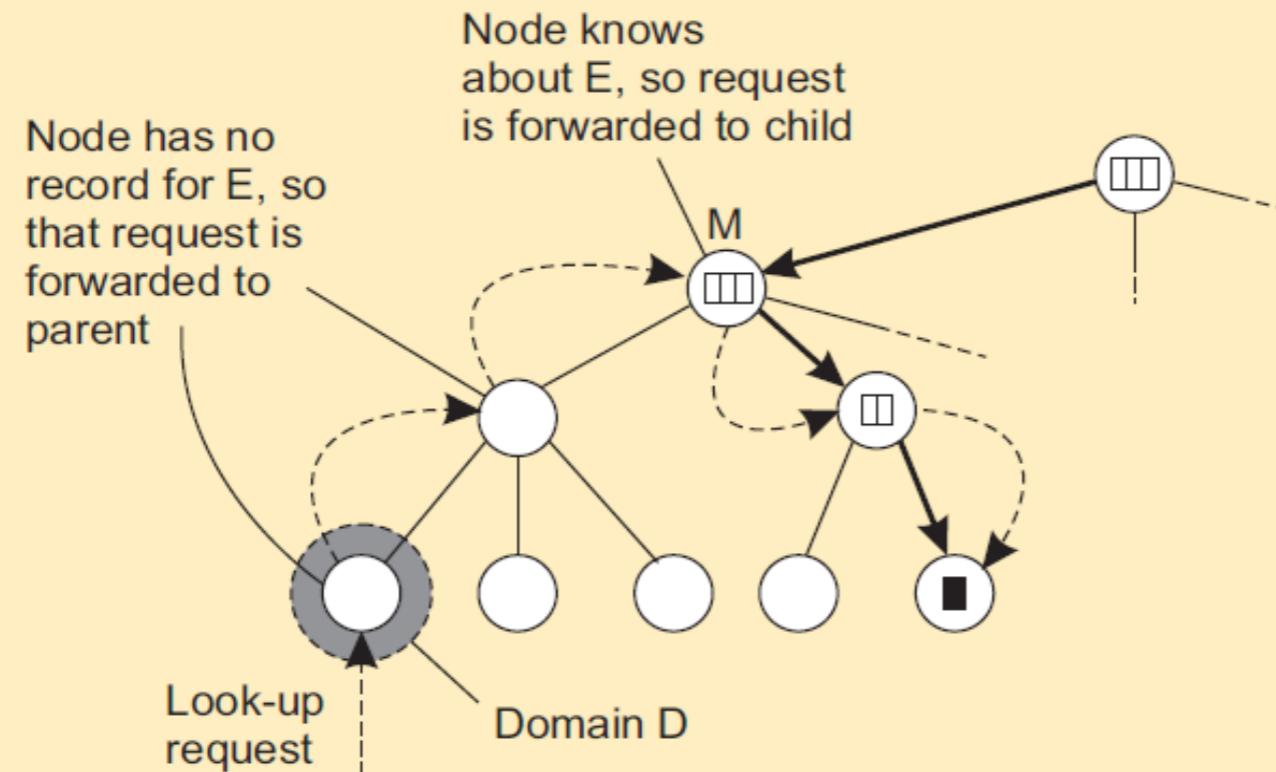
### ARMAZENANDO INFORMAÇÃO DE UMA ENTIDADE TENDO DOIS ENDEREÇOS EM DIFERENTES DOMÍNIOS FOLHA



## PRINCÍPIOS BÁSICOS

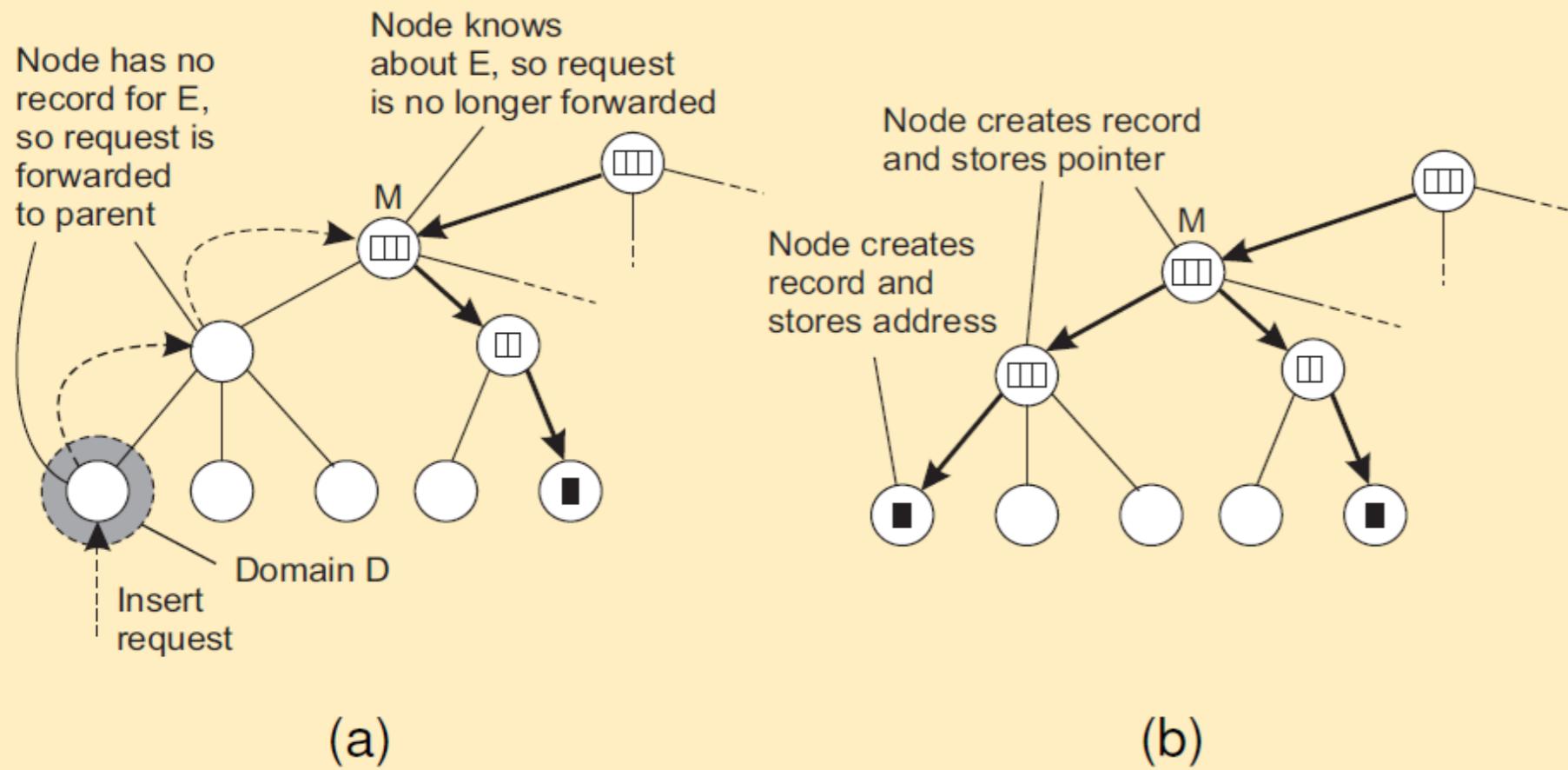
- Inicia *lookup* em um nó folha local
- Nó sabe sobre E → segue ponteiro pra baixo, ou vai pra cima
- *Lookup* pra cima sempre vai parar na raiz

## BUSCANDO (LOOKING UP) UMA LOCALIZAÇÃO



# HLS - OPERAÇÃO INSERÇÃO

- (a) Uma requisição inserção (*insert*) é repassada para o primeiro nó que conhece entidade E
- (b) Uma cadeia de ponteiros para repasse (*chain of forwarding pointers*) para o nó folha é criada

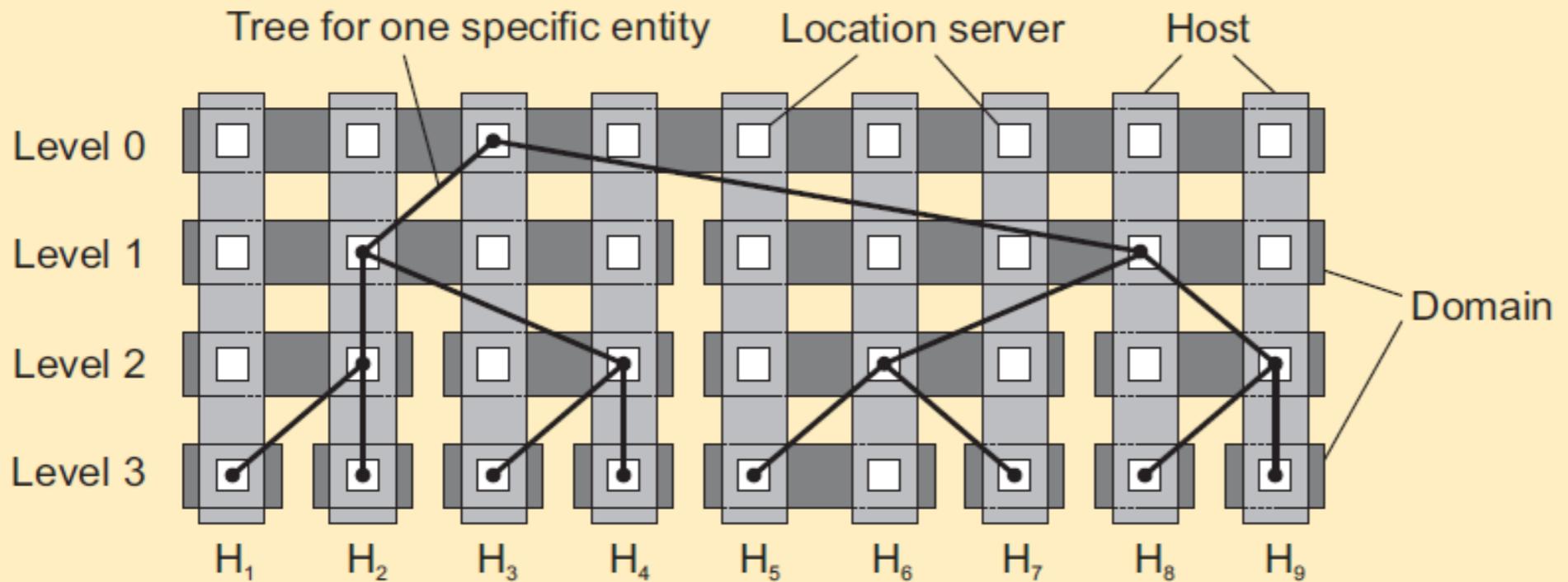


# HLS - PODE ESCALAR ?

## SOLUÇÃO

- $D_k = \{D_{k,1}, D_{k,2}, \dots, D_{k,N_k}\}$  denota os  $N_k$  domínios no nível k
- Nota:  $N_0 = |D_0| = 1$
- Para cada nível k, o conjunto de hospedeiros é partitionado em  $N_k$  subconjuntos com cada hospedeiro um servidor de localização representando exatamente um dos domínios  $D_{k,i}$  de  $D_k$ .

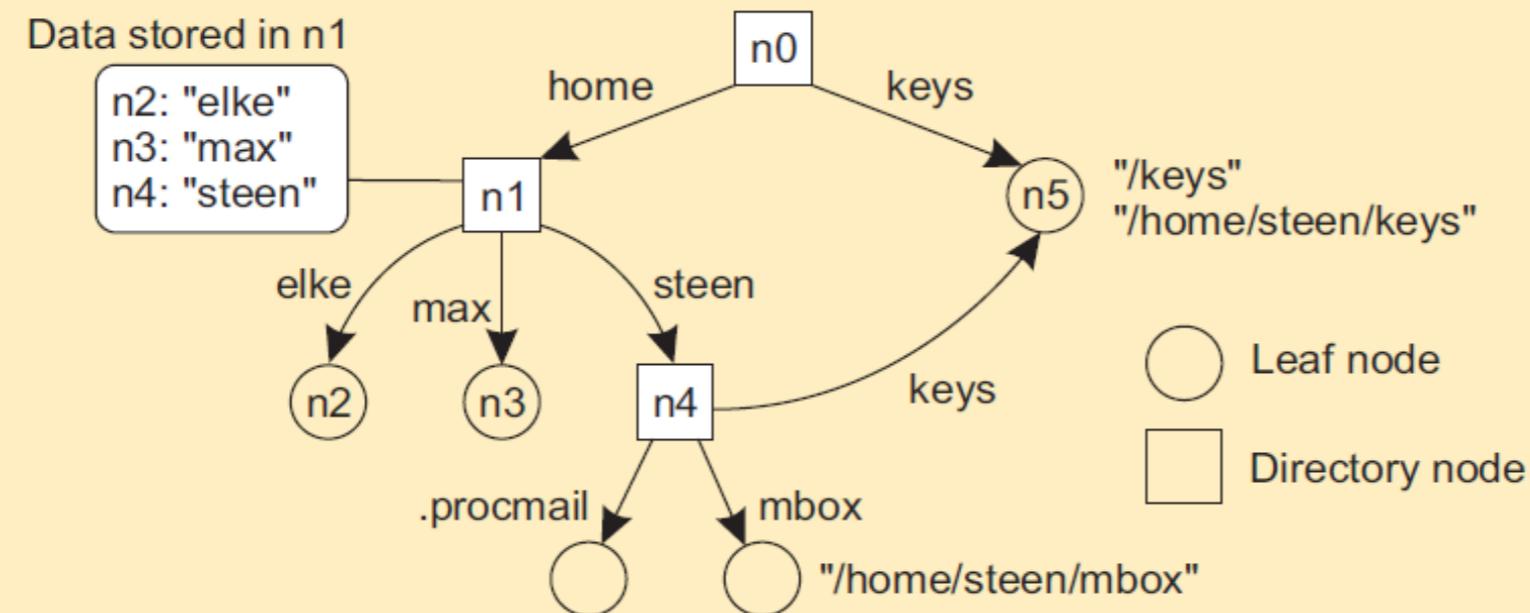
## PRINCÍPIO DE DISTRIBUIÇÃO LÓGICA DE SERVIDORES DE LOCALIZAÇÃO



### GRAFO DE NOMES

Um nome que não tem significado nenhum; é somente um *string* aleatório. Nomes puros podem ser usados para comparação somente.

### UM GRAFO DE NOMEAÇÃO GERAL COM UM NÓ RAIZ ÚNICO



### NOTA

Um nó diretório contém uma tabela de (nó identificador, rótulo de borda) pares

# ESPAÇO DE NOMES

**PODEMOS ARMAZENAR TODO TIPO DE ATRIBUTOS EM UM NÓ**

- Tipo de entidade
- Um identificador para a entidade
- Endereço da localidade da entidade
- Apelidos
- ...

## NOTA

Um nó diretório contém uma tabela de (nó identificador, rótulo de borda) pares

# RESOLUÇÃO DE NOMES

ESPAÇO DE NOMES: NOMES ESTRUTURADOS

## PROBLEMA

- Para resolver um nome precisamos de um nó diretório. Como de fato encontramos este nó (inicial) ?

## MECANISMO DE FECHAMENTO: O MECANISMO PARA SELECIONAR O CONTEXTO IMPLÍCITO A PARTIR DO QUAL IREMOS INICIAR A RESOLUÇÃO DE NOME

- `www.distributed-systems.net`: inicia em um servidor de nomes DNS
- `/home/maarten/mbox`: inicia no local do servidor de arquivos NFS (possível busca recursiva)
- `0031 20 598 7784`: disca um número de telefone
- `77.167.55.6`: roteia mensagens para um endereço IP específico

## NOTA

Se não tiver um mecanismo de fechamento explícito – como começar ?

## LIGAÇÃO DURA (HARD LINK)

O que descrevemos até agora como um caminho de nome (*path name*): um nome que é resolvido seguindo um caminho específico em um grafo de nomes de um nó para outro.

## LIGAÇÃO MACIA (SOFT LINK): PERMITE A UM NÓ N CONTER UM NOME DE OUTRO NÓ

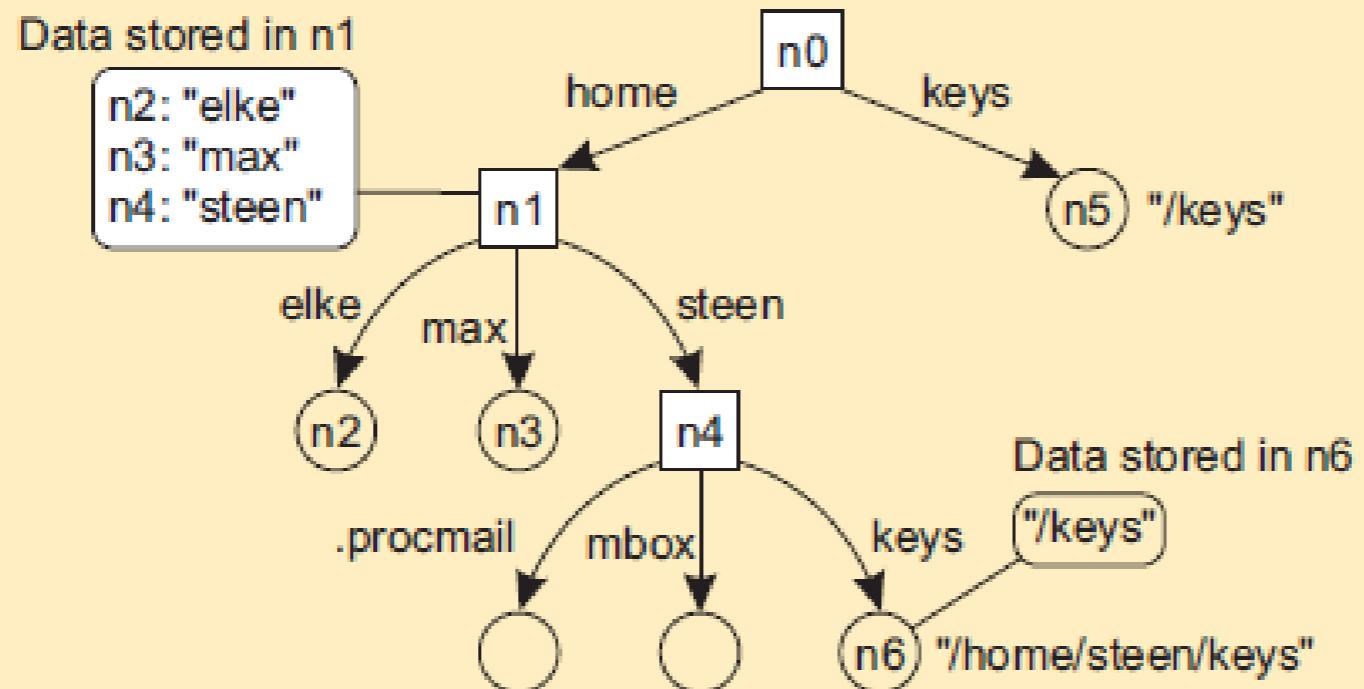
- Primeiro resolve o nome de  $N$  (que leva a  $N$ )
- Lê o conteúdo de  $N$ , resultando em  $N$
- A resolução de nome continua com *nome*

## OBSERVAÇÕES

- O processo de resolução de nome determina a leitura de conteúdo de um nó, em particular, o nome em outro nó pra onde precisamos ir;
- No caminho para outro nó, sabemos onde e como começar a resolução de nome, dado o nome.

# LIGANDO NOMES

## O CONCEITO DE UM LINK SIMBÓLICO EXPLICADO EM UM GRAFO DE NOMES



### OBSERVAÇÃO

Nó 5 tem somente um nome

## QUESTÃO

Resolução de nomes pode também ser usada para juntar (*merge*) diferentes espaços de nomes de forma transparente através de montagem (*mounting*): associando um identificador de nó de um outro espaço de nomes com um nó em um espaço de nós atual.

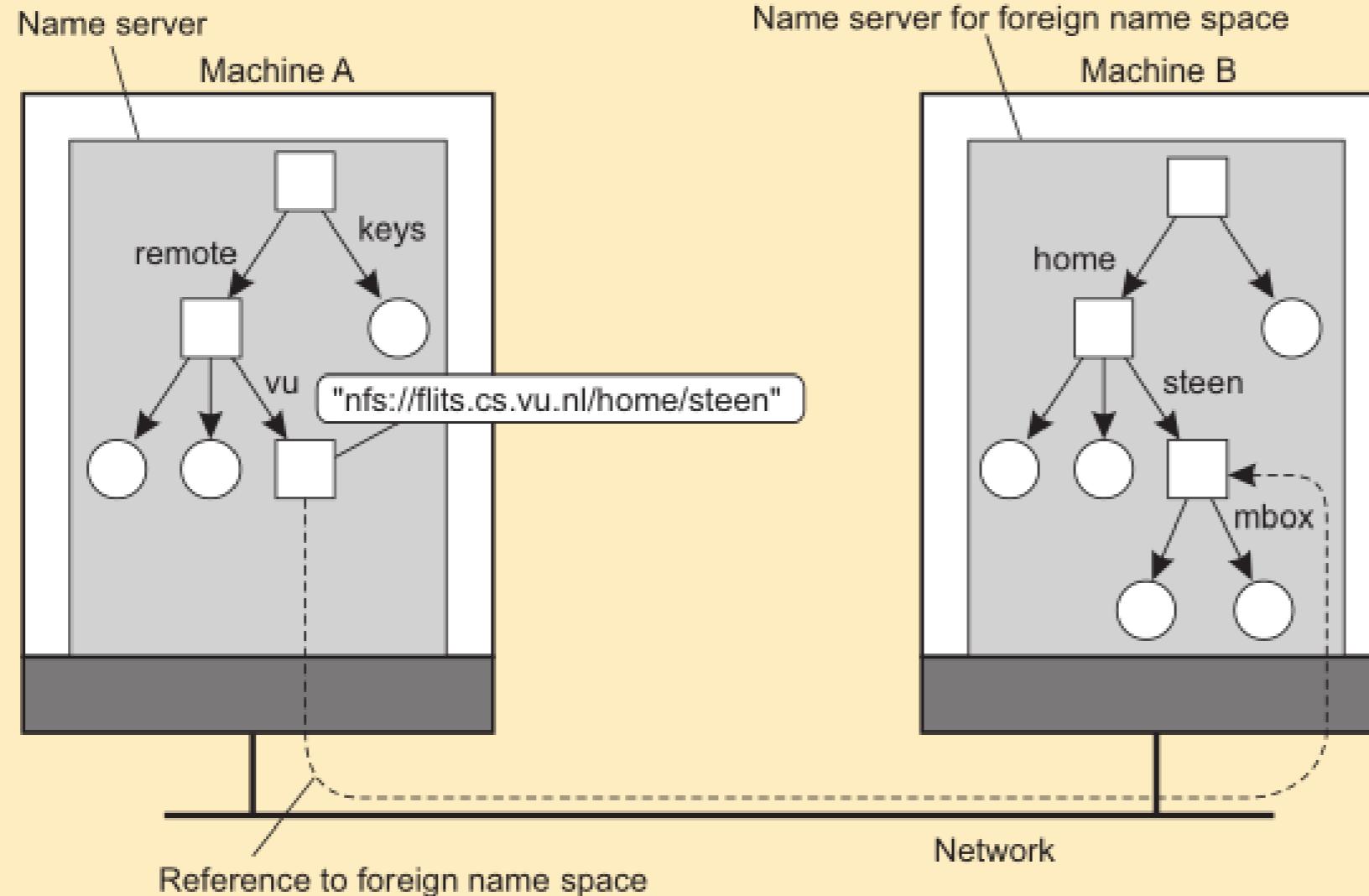
## TERMINOLOGIA

- **Espaço de nomes estrangeiro**: o espaço de nomes que queremos acessar;
- **Ponto montado (mount point)**: o nó do espaço de nomes atual contendo o identificador do espaço de nomes estrangeiro;
- **Ponto de montagem (mounting point)**: o nó de um espaço de nomes estrangeiro onde iremos continuar a resolução de nomes.

## MONTANDO ATRAVÉS DA REDE

- O nome de um protocolo de acesso
- O nome de um servidor
- O nome de um ponto de montagem de um espaço de nomes estrangeiro

## MONTANDO ESPAÇO DE NOMES REMOTOS ATRAVÉS DE UM PROTOCOLO DE ACESSO ESPECÍFICO



# IMPLEMENTAÇÃO DE ESPAÇO DE NOMES

## QUESTÃO BÁSICA

O processo de resolução de nomes distribuído bem como o gerenciamento do espaço de nomes através de máquinas múltiplas, através da distribuição de nós no grafo de nomes

DISTRIBUIÇÃO DE ESPAÇO DE NOMES

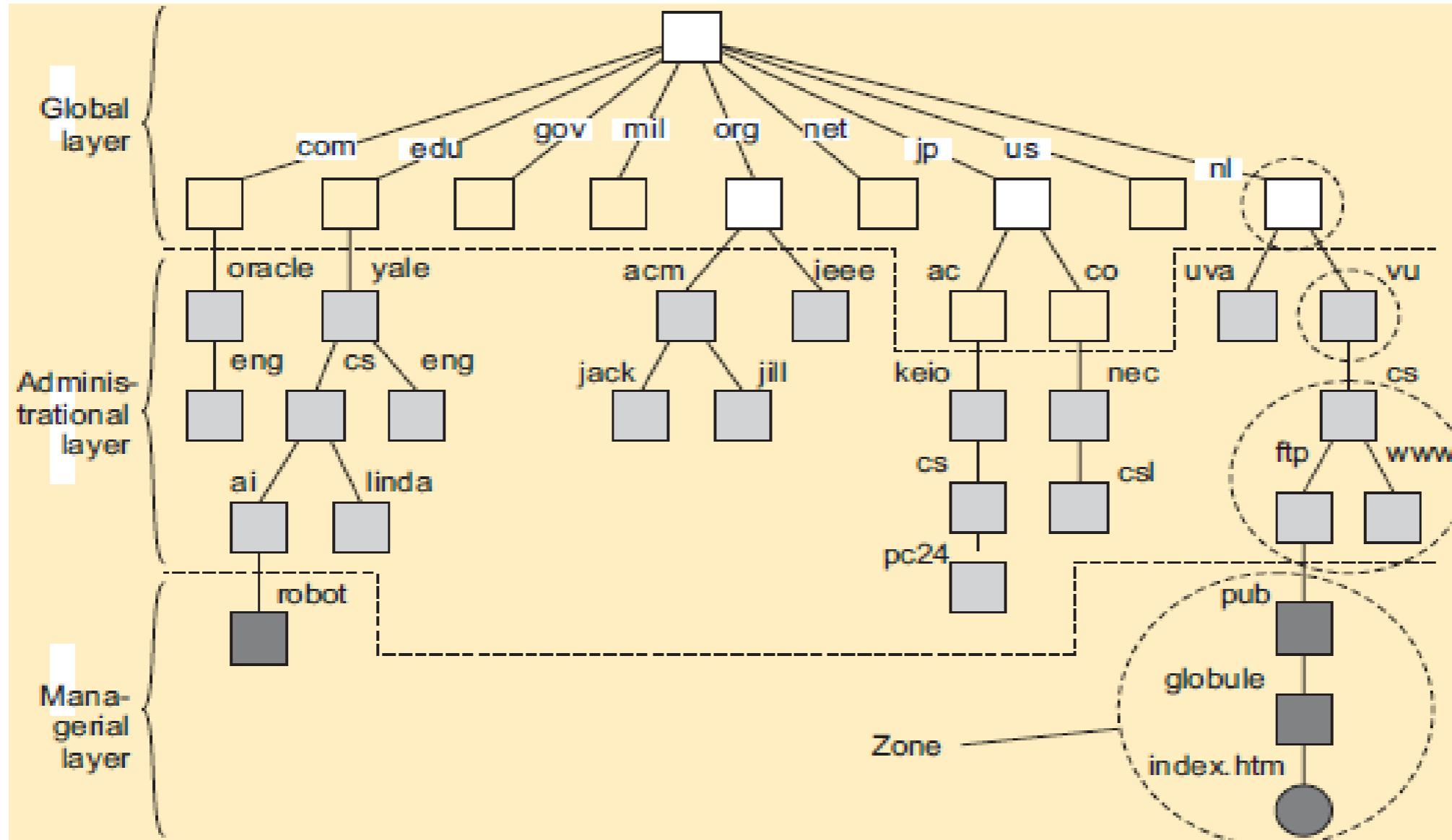
## DISTINÇÃO DE TRÊS NÍVEIS

- **Nível global:** consiste de nós de diretório de alto nível. O aspecto principal é que este diretório tem que ser gerenciado de forma conjunta por diferentes administrações;
- **Nível administrativo:** Contém os nós de diretório de nível médio que pode ser agrupado de tal forma que cada grupo pode ser atribuído a uma administração separada;
- **Nível gerencial:** Consiste de nós de diretório de baixo nível de uma única administração. A questão principal é o mapeamento efetivo de nós de diretório a servidores de nomes locais.

# IMPLEMENTAÇÃO DE ESPAÇO DE NOMES

DISTRIBUIÇÃO DE ESPAÇO DE NOMES

UM EXEMPLO DO PARTICIONAMENTO DO ESPAÇO DE NOMES DO DNS, INCLUÍNDΟ ARQUIVOS DE REDE



# IMPLEMENTAÇÃO DE ESPAÇO DE NOMES

DISTRIBUIÇÃO DE ESPAÇO DE NOMES

UMA COMPARAÇÃO ENTRE SERVIDORES DE NOME PARA IMPLEMENTAÇÃO DE NÓS EM UM ESPAÇO DE NOMES

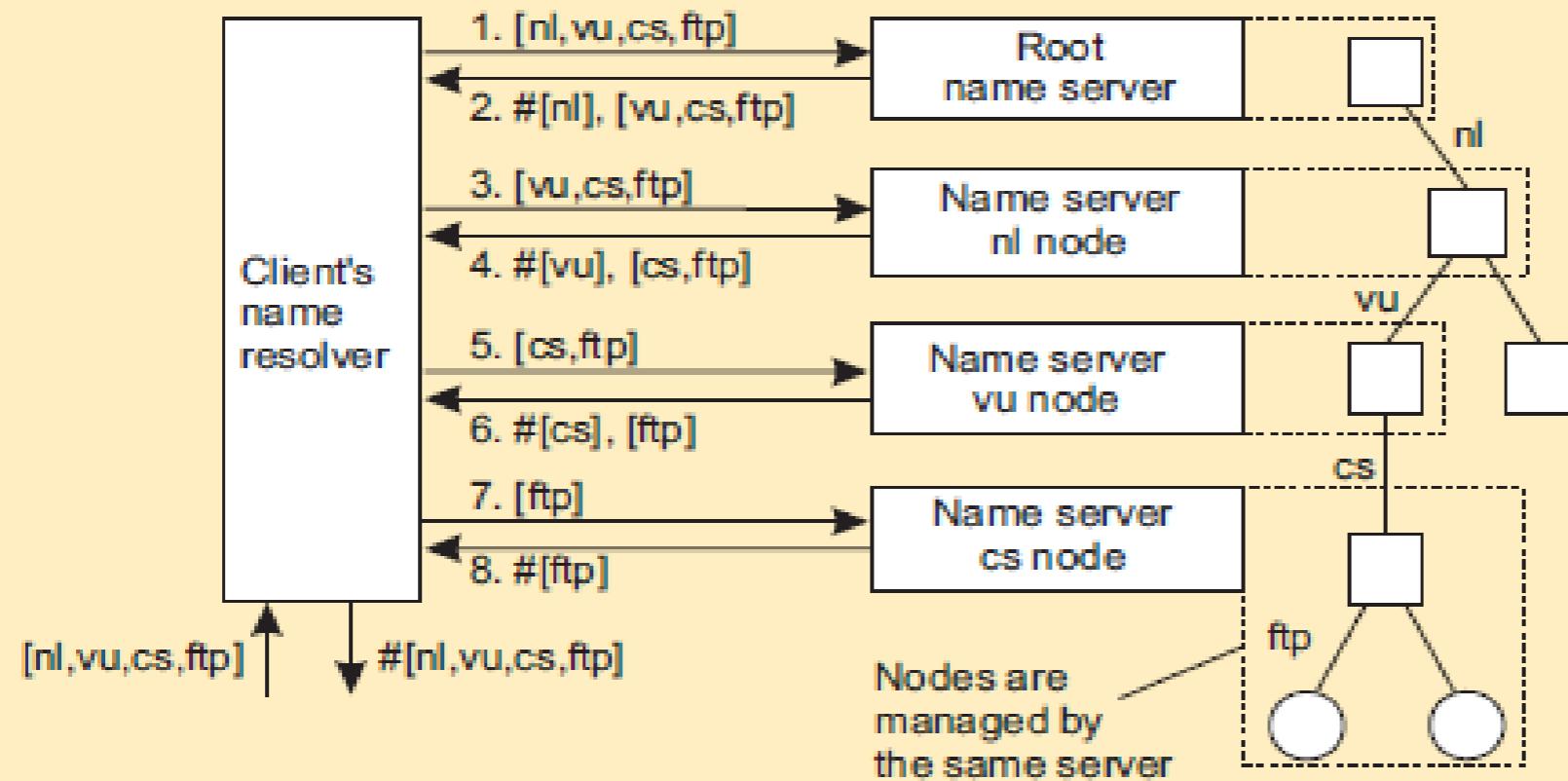
Item	Global	Administrational	Managerial
1	Worldwide	Organization	Department
2	Few	Many	Vast numbers
3	Seconds	Milliseconds	Immediate
4	Lazy	Immediate	Immediate
5	Many	None or few	None
6	Yes	Yes	Sometimes

1: Geographical scale  
2: # Nodes  
3: Responsiveness

4: Update propagation  
5: # Replicas  
6: Client-side caching?

### PRINCÍPIO

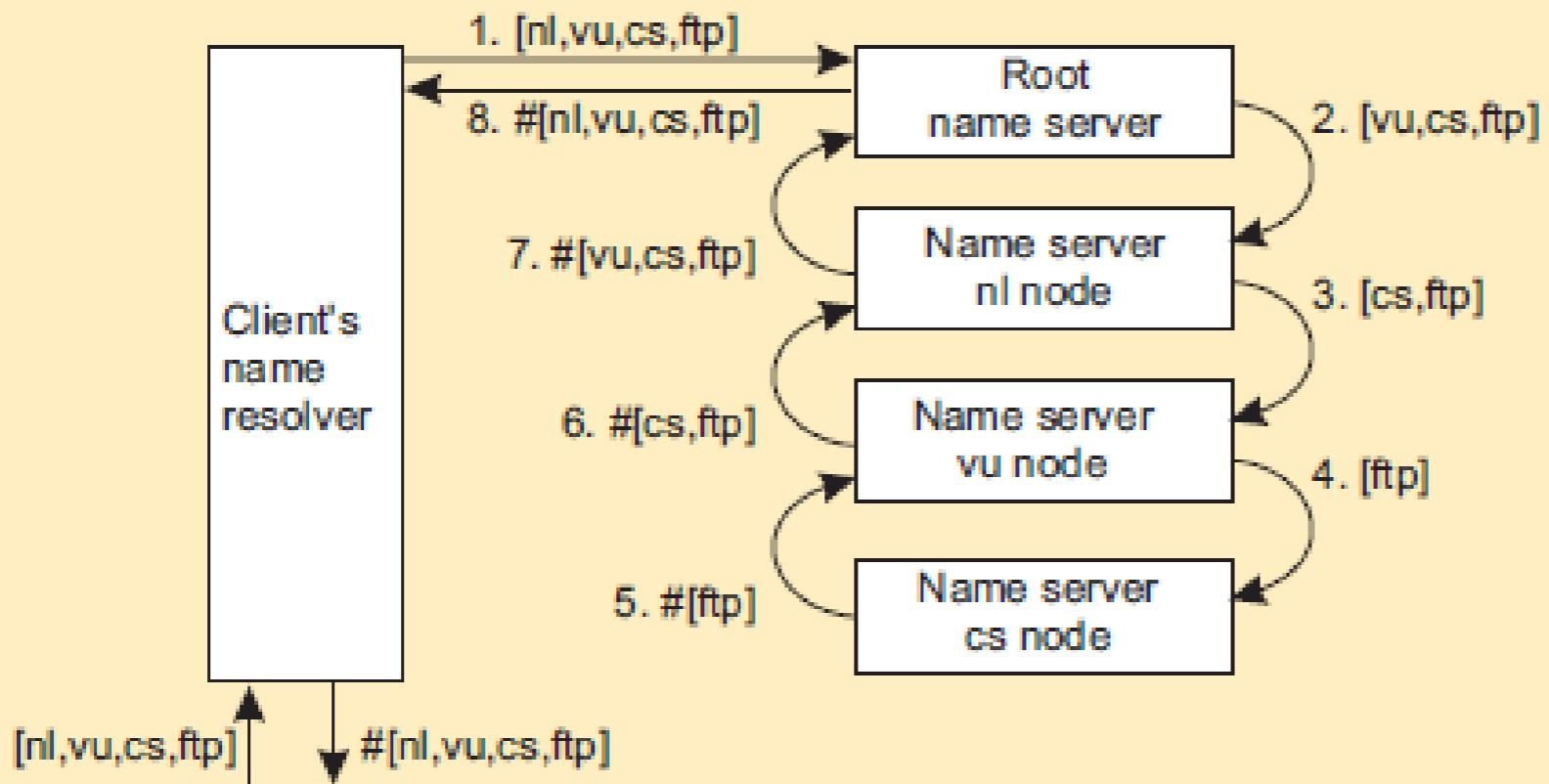
1.  $\text{resolve}(\text{dir}, [\text{nome}_1, \dots, \text{nome}_k])$  enviado para  $\text{servidor}_0$  responsável por  $\text{dir}$
2.  $\text{Servidor}_0$  resolve  $\text{resolve}(\text{dir}, \text{nome}) \rightarrow \text{dir}_1$ , retornando o identificador (endereço) do  $\text{servidor}_1$ , que armazena  $\text{dir}_1$ .
3. Cliente envia  $\text{resolve}(\text{dir}_1, [\text{nome}_2, \dots, \text{nome}_k])$  para  $\text{servidor}_1$ , etc



# RESOLUÇÃO RECURSIVA DE NOMES

## PRINCÍPIO

1.  $\text{resolve}(\text{dir}, [\text{nome}_1, \dots, \text{nome}_k])$  enviado para  $\text{servidor}_0$  responsável por  $\text{dir}$
2.  $\text{Servidor}_0$  resolve  $\text{resolve}(\text{dir}, \text{nome}) \rightarrow \text{dir}_1$ , e envia  $\text{resolve}(\text{dir}, [\text{nome}_1, \dots, \text{nome}_k])$  para  $\text{servidor}_1$ , que armazena  $\text{dir}_1$ .
3.  $\text{Servidor}_0$  espera pelo resultado do  $\text{servidor}_1$ , e retorna para o cliente



DISTRIBUIÇÃO DE ESPAÇO DE NOMES

# CACHING DE RESOLUÇÃO RECURSIVA DE NOMES

## RESOLUÇÃO RECURSIVA DE NOMES DE $[nl, vu, cs, ftp]$

Server for node	Should resolve	Looks up	Passes to child	Receives and caches	Returns to requester
cs	[ftp]	# [ftp]	—	—	# [ftp]
vu	[cs, ftp]	# [cs]	[ftp]	# [ftp]	# [cs] # [cs, ftp]
nl	[vu, cs, ftp]	# [vu]	[cs, ftp]	# [cs] # [cs, ftp]	# [vu] # [vu, cs] # [vu, cs, ftp]
root	[nl, vu, cs, ftp]	# [nl]	[vu, cs, ftp]	# [vu] # [vu, cs] # [vu, cs, ftp]	# [nl] # [nl, vu] # [nl, vu, cs] # [nl, vu, cs, ftp]

## ESCALABILIDADE DE TAMANHO

Precisamos garantir que servidores podem manusear um grande número de requisições por unidade de tempo => servidores de alto nível estão com grande problema

## SOLUÇÃO

Assuma (pelo mesmo nos níveis global e administrativo) que o conteúdo de nós dificilmente muda. Podemos então aplicar replicação extensiva mapeando nós a múltiplos servidores e iniciar a resolução de nomes no servidor mais próximo.

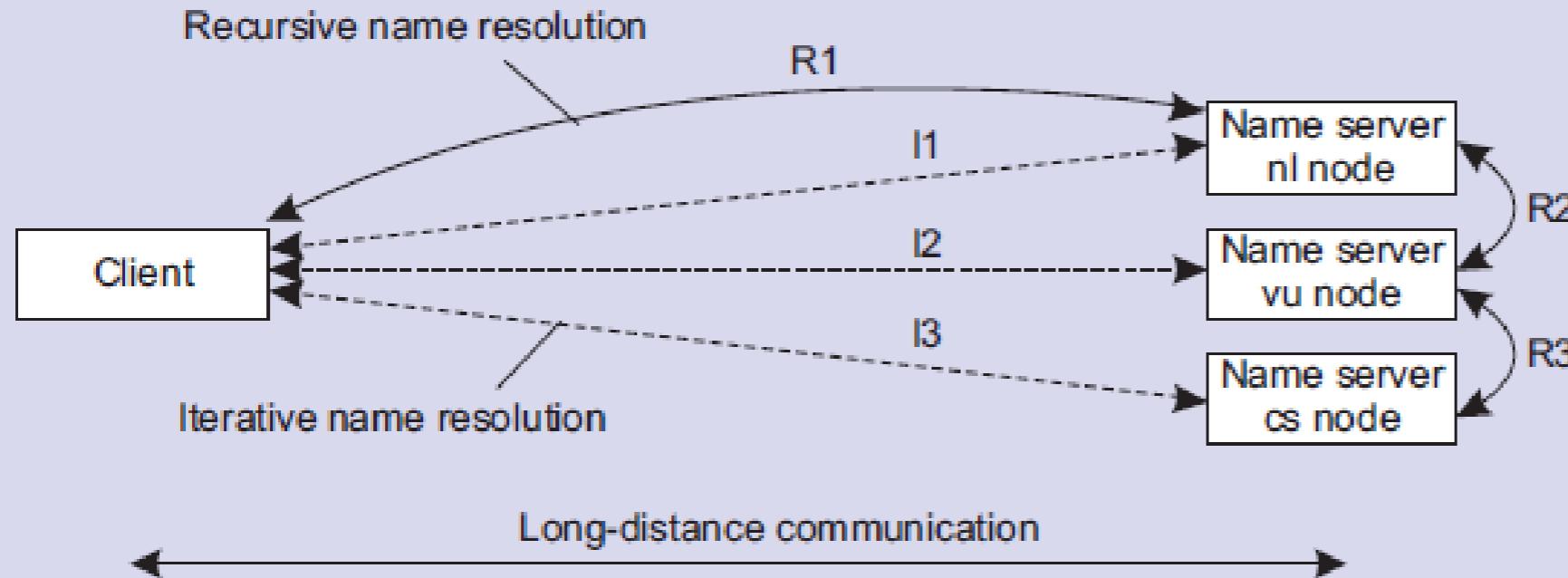
## OBSERVAÇÃO

Um atributo importante de muitos nós é o endereço onde a representação da entidade pode ser contatada. A replicação de nós torna servidores de larga escala tradicionais não adequados para localização de entidades móveis.

# QUESTÕES DE ESCALABILIDADE

ESPAÇO DE NOMES: RESOLUÇÃO

PRECISAMOS GARANTIR QUE O PROCESSO DE RESOLUÇÃO DE NOMES ESCALA ATRAVÉS DE GRANDES DISTÂNCIAS GEOGRÁFICAS



## PROBLEMA

A dependência de localização implícita é introduzida através do mapeamento de nós a servidores que podem ser localizados em qualquer lugar.

# DNS

## ESSÊNCIA

- Espaço de nomes organizado hierarquicamente com cada nó tendo exatamente uma borda de entrada => rótulo da borda = rótulo do nó
- **Domínio**: uma sub-árvore
- **Domínio de nomes**: um caminho de nomes para o nó raiz do domínio

## INFORMAÇÃO EM UM NÓ

Type	Refers to	Description
SOA	Zone	Holds info on the represented zone
A	Host	IP addr. of host this node represents
MX	Domain	Mail server to handle mail for this node
SRV	Domain	Server handling a specific service
NS	Zone	Name server for the represented zone
CNAME	Node	Symbolic link
PTR	Host	Canonical name of a host
HINFO	Host	Info on this host
TXT	Any kind	Any info considered useful

# NOMEAÇÃO BASEADA EM ATRIBUTOS

## OBSERVAÇÃO

Em muitos casos é muito mais conveniente nomear e buscar entidades através de seus atributos => **serviços de diretório tradicionais (aka yellow pages)**

## PROBLEMA

Operações lookup podem ser muito custosas, porque demandam o casamento de **valores de atributos requisitados**, contra **valores atuais de atributos** => inspeção de **todas entidades** (em princípio)



# IMPLEMENTANDO SERVIÇOS DE DIRETÓRIO

DISTRIBUIÇÃO DE ESPAÇO DE NOMES

## SOLUÇÃO ESCALÁVEL PARA BUSCA

Implementa serviços básicos de diretório como uma base de dados e combina com estrutura tradicional de sistema de nomes

## INFORMAÇÃO EM UM NÓ

Cada entrada no diretório consiste de pares (atributo, valor), e é **nomeado** **unicamente** para facilitar lookups.

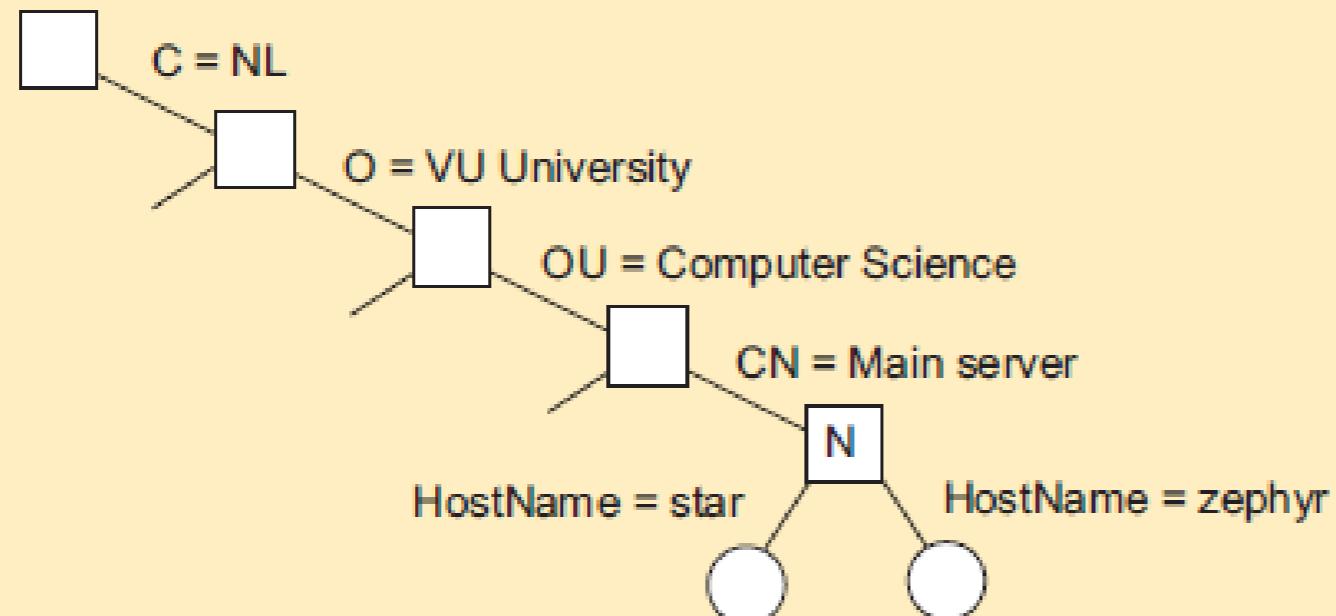
Attribute	Abbr.	Value
Country	C	NL
Locality	L	Amsterdam
Organization	O	VU University
OrganizationalUnit	OU	Computer Science
CommonName	CN	Main server
Mail_Servers	–	137.37.20.3, 130.37.24.6, 137.37.20.10
FTP_Server	–	130.37.20.20
WWW_Server	–	130.37.20.20



## ESSÊNCIA

- **Base de informação de diretório:** coleção de todas entradas em diretório em um serviço LDAP
- Cada registro é unicamente nomeado como uma sequência de atributos de nomes (chamados **Nomes Distintos Relativos**), de forma que podem ser procurados (looked up).
- **Árvore de Informações do Diretório:** o grafo de nomes de um serviço LDAP; cada nó representa uma entrada no diretório.

## PARTE DE UMA ÁRVORE DE INFORMAÇÃO DE DIRETÓRIO



# LDAP

DUAS ENTRADAS EM DIRETÓRIO TENDO *HostName* COMO RDN

Attribute	Value	Attribute	Value
Locality	Amsterdam	Locality	Amsterdam
Organization	VU University	Organization	VU University
OrganizationalUnit	Computer Science	OrganizationalUnit	Computer Science
CommonName	Main server	CommonName	Main server
HostName	star	HostName	zephyr
HostAddress	192.31.231.42	HostAddress	137.37.20.10

Result of search(``(C=NL) (O=VU University) (OU=\*) (CN>Main server)'' )



### IDÉIA BÁSICA

- Assuma um conjunto de atributos  $\{a^1, \dots, a^N\}$
- Cada atributo  $a^k$  pega valores de um conjunto  $R^k \{s_1^k, \dots, s_{n_k}^k\}$
- Para cada atributo  $a^k$  associe um conjunto  $S^k =$  de  $n_k$  servidores
- Mapeamento global:  $F: F(a^k, v) = S_j^k$  com  $S_j^k \in S^k$  e  $v \in R^k$

### OBSERVAÇÃO

Se  $L(a^k, v)$  é um conjunto de chaves retornado por  $F(a^k, v)$ , então uma *query* bem formulada como uma expressão lógica, p.ex.,

$$(F(a^1, v^1) \wedge F(a^2, v^2)) \vee F(a^3, v^3)$$

Que pode ser processado por um cliente através da construção do conjunto

$$(L(a^1, v^1) \cap L(a^2, v^2)) \cup L(a^3, v^3)$$

# PROBLEMAS COM ÍNDICE DISTRIBUÍDO

## ALGUNS PROBLEMAS

- Uma *query* envolvendo  $k$  atributos requer o contato em  $k$  servidores
- Imagine *looking up* “*lastName = Smith ^ firstName = Pheriby*”: o cliente pode precisar processar muitos arquivos porque existe muitas pessoas com nome “*Smith*”
- Não facilmente suportado para consulta de intervalos (*range queries*), tal como “*preço = [1000 – 2500]*”

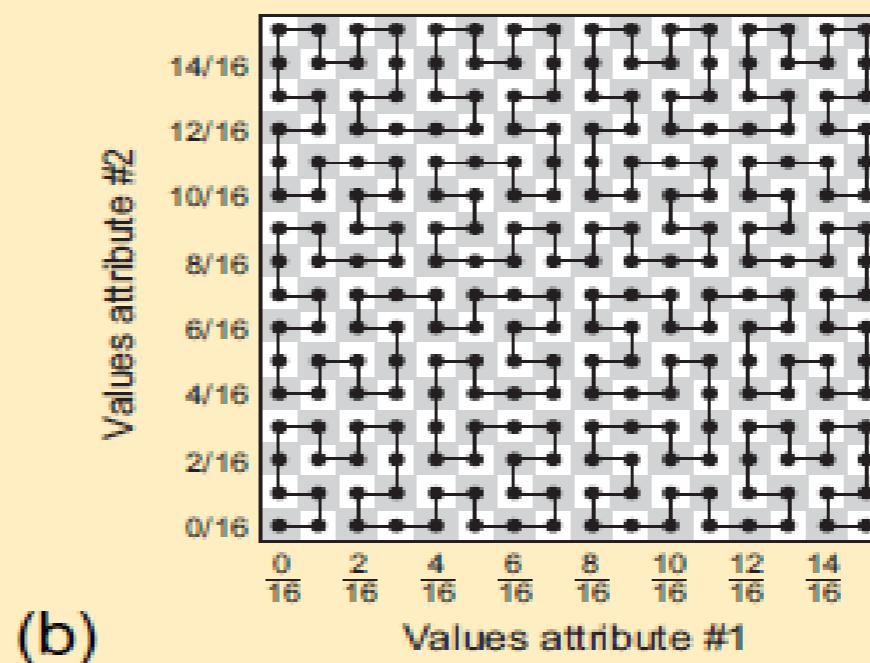
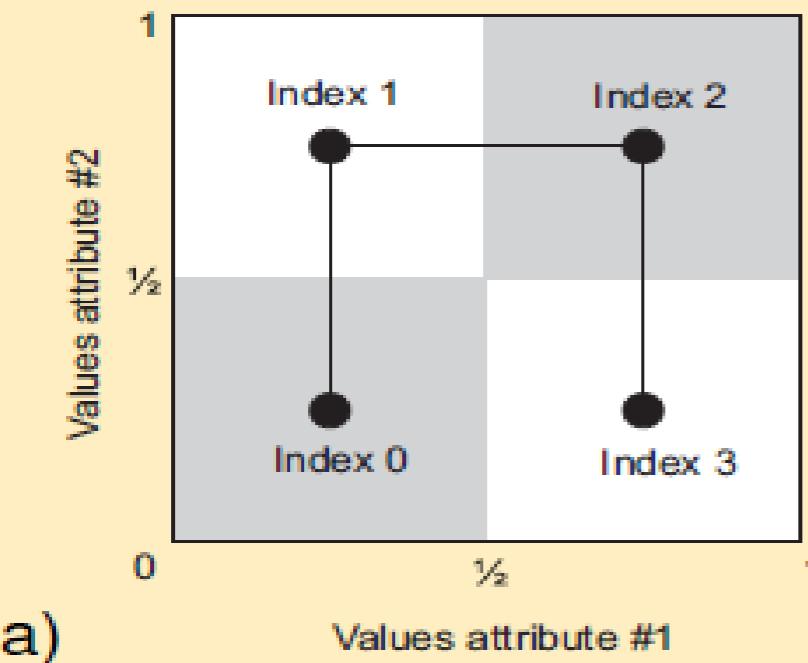


## ALTERNATIVA: MAPEAR TODOS ATRIBUTOS EM 1 DIMENSÃO E ENTÃO INDEXAR

### CURVAS DE PREENCHIMENTO DE ESPAÇO: PRINCÍPIO

1. Mapeie o espaço N dimensional coberto por N atributos  $\{a^1, \dots, a^N\}$  em uma única dimensão
2. Fazer hash de valores de forma a distribuir o espaço 1 -dimensional entre servidores de índice

### CURVA DE PREENCHIMENTO DE ESPAÇO DE HILBERT PARA (a) ORDEM 1 E (b) ORDEM 4



# CURVA PREENCHIMENTO ESPAÇO

## UMA VEZ QUE A CURVA TENHA SIDO DESENHADA

Considere o caso de duas dimensões:

- uma curva de Hilbert de ordem  $k$  conecta  $2^{2k}$  subquadrados => tem  $2^{2k}$  índices.
- Uma faixa de query corresponde a um **retângulo  $R$**  em um caso de 2 dimensões
- $R$  cruza com um número de subquadrados, cada um correspondendo a um índice => e temos agora uma **série de índices** associados com  $R$

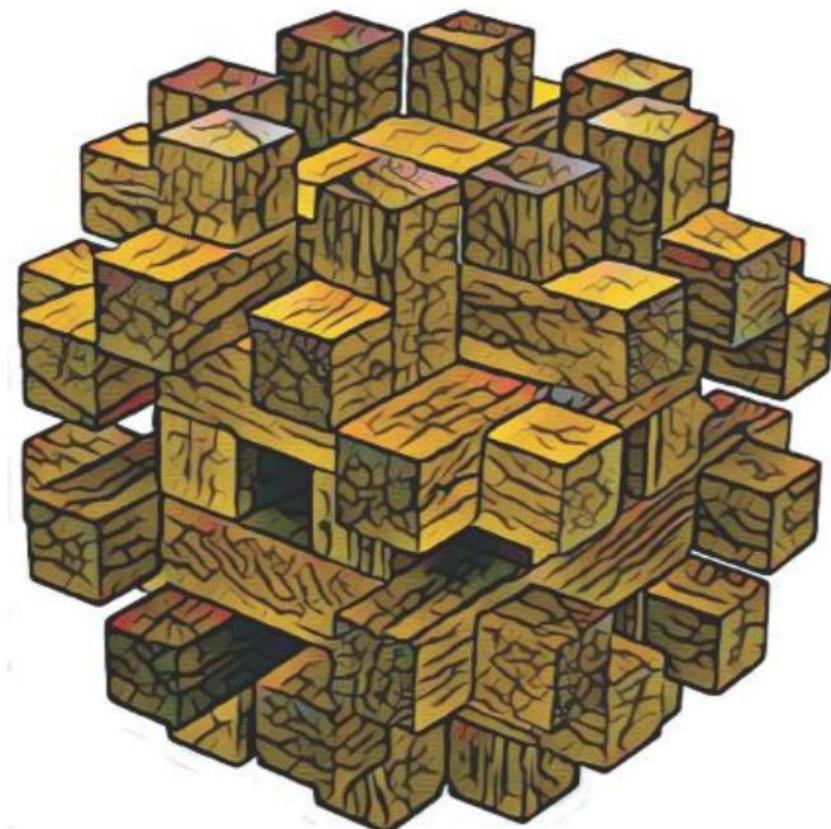
## INDO PARA ENTIDADES

Cada índice deve ser mapeado a um servidor, que mantém a referência para a entidade associada. Uma possível solução: usar uma **DHT (distributed hash table)**



# Distributed Systems

Maarten Van Steen & Andrew S.  
Tanenbaum



3th Edition – Version 3.01 - 2017

## Capítulo 6

### Coordenação

Sábado, 03 de Julho de 2021

## PROBLEMA

As vezes precisamos da hora exata e não somente de um enfileiramento

## SOLUÇÃO: UTC – UNIVERSAL COORDINATED TIME

- Baseado em um número de transições por segundo do átomo de césio 133
- Até o momento, o tempo real é calculado como a média de 50 relógios atômicos ao redor do mundo
- Introduz um segundo de “leap” de tempos em tempos para compensar o fato que os dias estão de alongando.

## NOTA

- UTC é transmitido através de ondas curtas de rádio e satélite. Satélites tem precisão de  $\pm 0,5\text{ms}$

## PRECISÃO

O objetivo é manter um desvio **entre dois relógios ou duas máquinas quaisquer** dentro de um limite específico, conhecido como **precisão  $\pi$** :

$$\forall t, \forall p, q : |C_p(t) - C_q(t)| \leq \pi$$

Com  $C_p(t)$  o tempo de relógio computado na máquina  $p$  no tempo  $t$  hora UTC

## ACURÁCIA

- No caso da acurácia, queremos manter o relógio vinculado a um valor  $\alpha$ :

$$\forall t, \forall p : |C_p(t) - t| \leq \alpha$$

## SINCRONIZAÇÃO

- **Sincronização interna:** mantém o clock preciso
- **Sincronização externa:** mantém o relógio acurado

### ESPECIFICAÇÕES DE RELÓGIO

- Um relógio vem com uma especificação de uma taxa máxima  $\rho$  de *drift* de clock
- $F(t)$  denota a frequência do oscilador do hardware do relógio no tempo  $t$
- $F$  é a frequência ideal (constante) -> atendendo as especificações

$$\forall t : (1 - \rho) \leq \frac{F(t)}{F} \leq (1 + \rho)$$

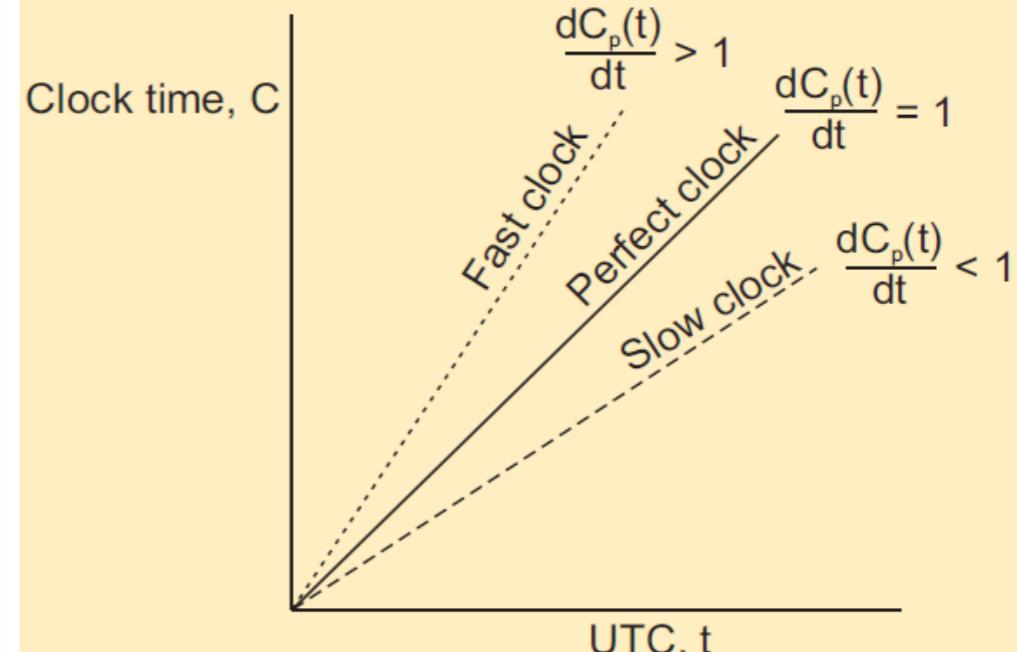
### OBSERVAÇÃO

Usando interrupções de hardware, fazemos o casamento de um relógio de software com o relógio de hardware e assim a taxa de drift de relógio fica:

$$C_p(t) = \frac{1}{F} \int_0^t F(t) dt \Rightarrow \frac{dC_p(t)}{dt} = \frac{F(t)}{F}$$

$$\Rightarrow \forall t : 1 - \rho \leq \frac{dC_p(t)}{dt} \leq 1 + \rho$$

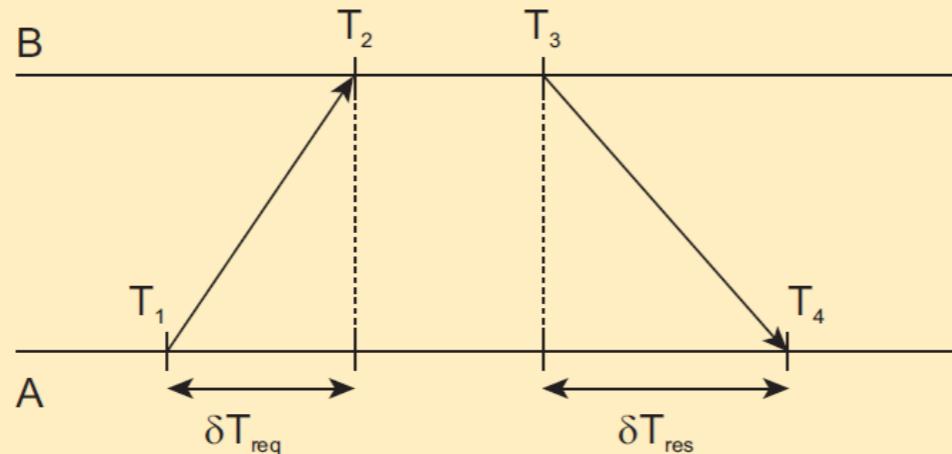
### RELÓGIOS: RÁPIDO, PERFEITO, DEVAGAR



# DETECTANDO E AJUSTANDO HORAS INCORRETAS

NTP: NETORK TIME PROTOCOL

## PEGANDO HORÁRIO ATUAL EM SERVIDOR DE HORAS



## OBSERVAÇÃO

supondo:  $\delta T_{req} = T_2 - T_1 \approx T_4 - T_3 = \delta T_{res}$

$$\theta = T_3 + ((T_2 - T_1) + (T_4 - T_3)) / 2 - T_4 = ((T_2 - T_1) + (T_3 - T_4)) / 2$$

$$\delta = ((T_4 - T_1) - (T_3 - T_2)) / 2$$

## NTP – Network Time Protocol

Colete 8 pares  $(\theta, \delta)$  e escolha  $\theta$  para o qual o atraso associado foi mínimo.

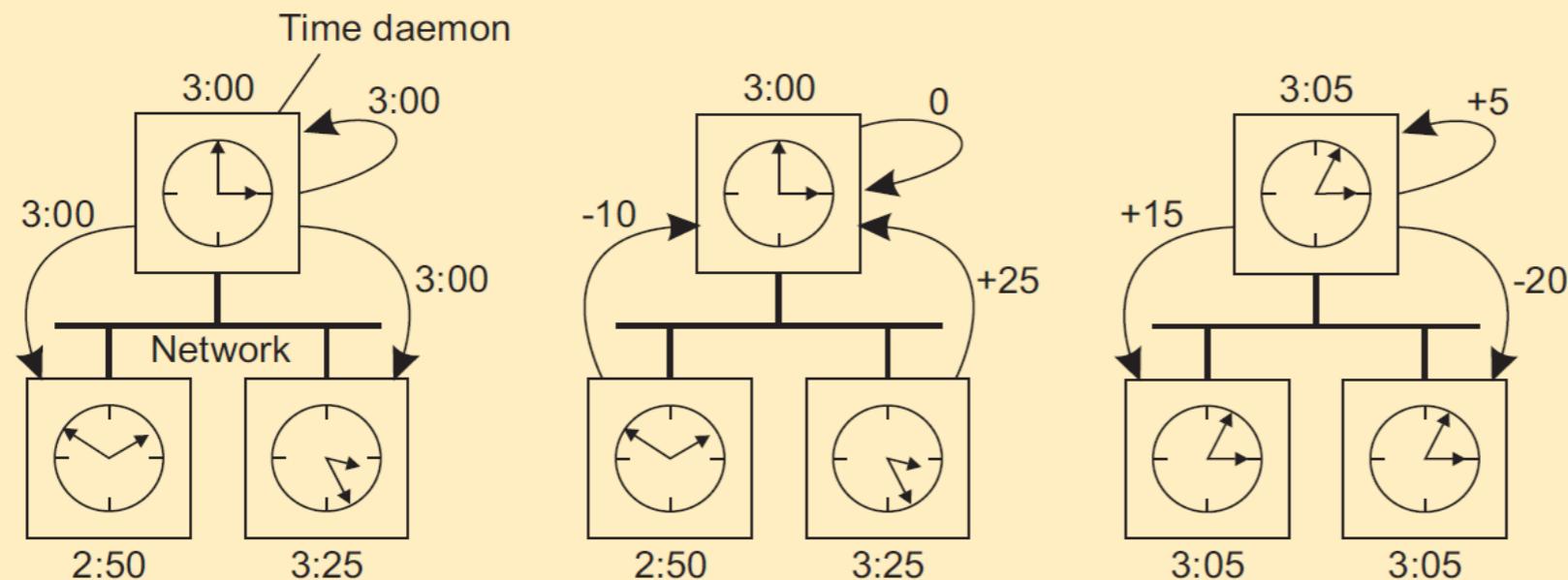
# MANTENDO A HORA SEM UTC

NTP: NETORK TIME PROTOCOL

## PRINCÍPIO

Deixe o servidor de hora escanear todas as máquinas periodicamente, calcule a media, e informe cada máquina como ela deve ajustar seu horário **relative ao seu horário atual**

## USANDO UM SERVIDOR DE HORA



## FUNDAMENTAL

É necessário considerer que o ajuste de horário para trás não é permitido => ajustes suaves (i.e , rodar mais rápido ou mais devagar).

# SINCRONIZAÇÃO COM BROADCAST DE REFERÊNCIA

## ESSÊNCIA

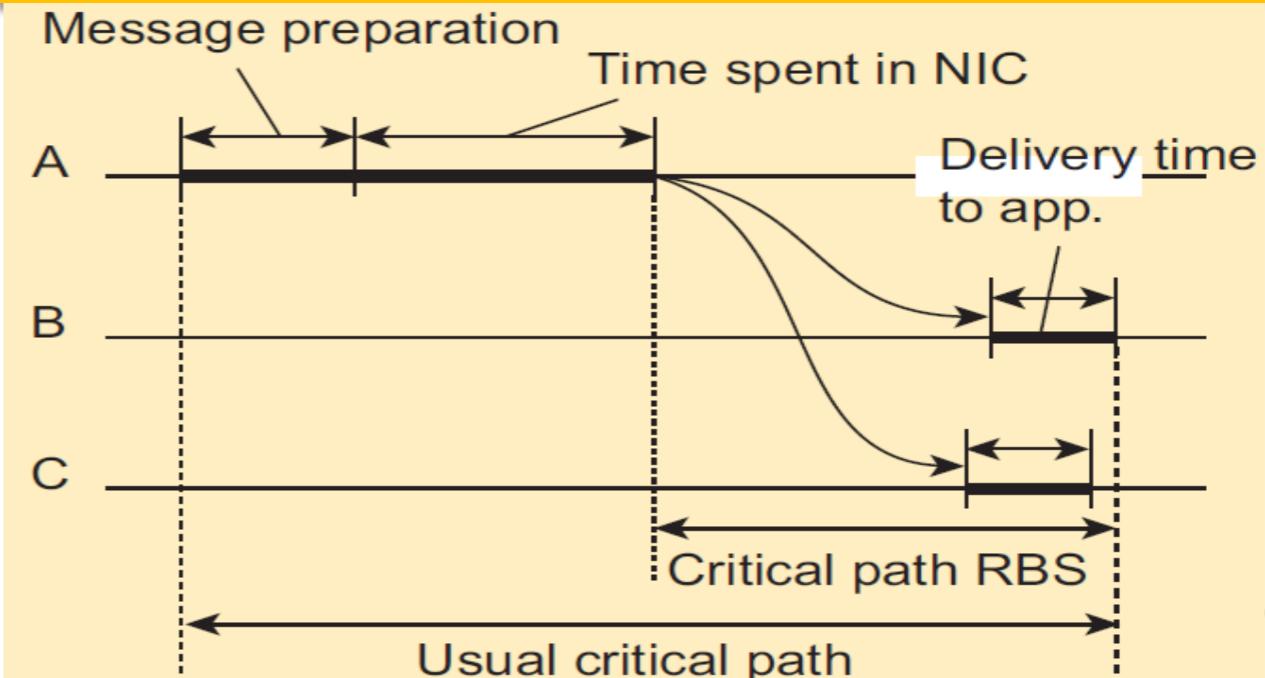
- Um nó transmite (*broadcast*) uma mensagem de referência  $m \Rightarrow$  cada nó recebedor  $p$  registra o horário  $T_{p,m}$  que recebeu  $m$
- Nota:  $T_{p,m}$  é lido do relógio local de  $p$

**PROBLEMA: A CAPTURA DA MÉDIA NÃO CAPTURA DRIFT  $\Rightarrow$  USE REGRESSÃO LINEAR**

NO:  $\text{Offset}[p, q](t) = \frac{\sum_{k=1}^M (T_{p,k} - T_{q,k})}{M}$

YES:  $\text{Offset}[p, q](t) = \alpha t + \beta$

## RBS MINIMIZA O CAMINHO CRÍTICO



# A RELAÇÃO ACONTEceu ANTES

RELÓGIOS LÓGICOS DE LAMPORT

## QUESTÃO

O que normalmente importa não é que todos processos concordam exatamente em que hora é, mas que eles concordam em **que ordem os eventos aconteceram**. Demanda uma noção de ordenamento.

## A RELAÇÃO ACONTEceu ANTES (**HAPPENED-BEFORE**)

- Se  $a$  e  $b$  são eventos no mesmo processo e  $a$  vem antes de  $b$ , então  $a \rightarrow b$ .
- Se  $a$  é o envio de uma mensagem e  $b$  é a recepção desta mensagem, então  $a \rightarrow b$ .
- Se  $a \rightarrow b$  e  $b \rightarrow c$ , então  $a \rightarrow c$ .

## NOTA

- Isto introduz um **ordenamento parcial de eventos** em um sistema com processos operacionais concorrentes.

### PROBLEMA

Como mantemos uma visão global sobre o comportamento do sistema com a relação aconteceu-antes?

**ANEXE UM TIMESTAMP  $C(e)$  EM CADA EVENTO  $e$ , SATISFAZENDO AS SEGUINTE PROPRIEDADES**

**P1:** Se  $a$  e  $b$  são eventos no mesmo processo, e  $a \rightarrow b$ , então requisitamos que  $C(a) < C(b)$  (*timestamp de  $a$  < timestamp de  $b$* )

**P2:** Se  $a$  corresponde ao envio de uma mensagem  $m$ , e  $b$  ao recebimento desta mensagem, então  $C(a) < C(b)$  (*recebimento só pode ocorrer após envio*)

### PROBLEMA

Como anexar um *timestamp* a um evento quando **não existe um relógio global**  
=> manter um conjunto consistente de relógios lógico, um por processo

CADA PROCESSO  $P_i$  MANTÉM UM CONTADOR LOCAL  $C_i$  E AJUSTA ESTE CONTADOR

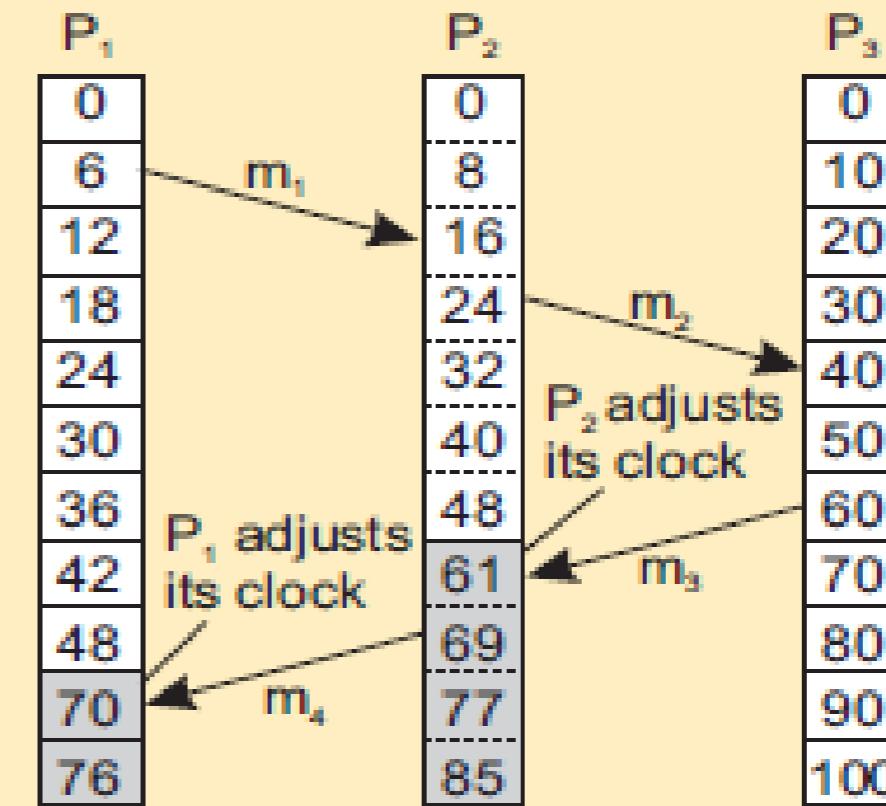
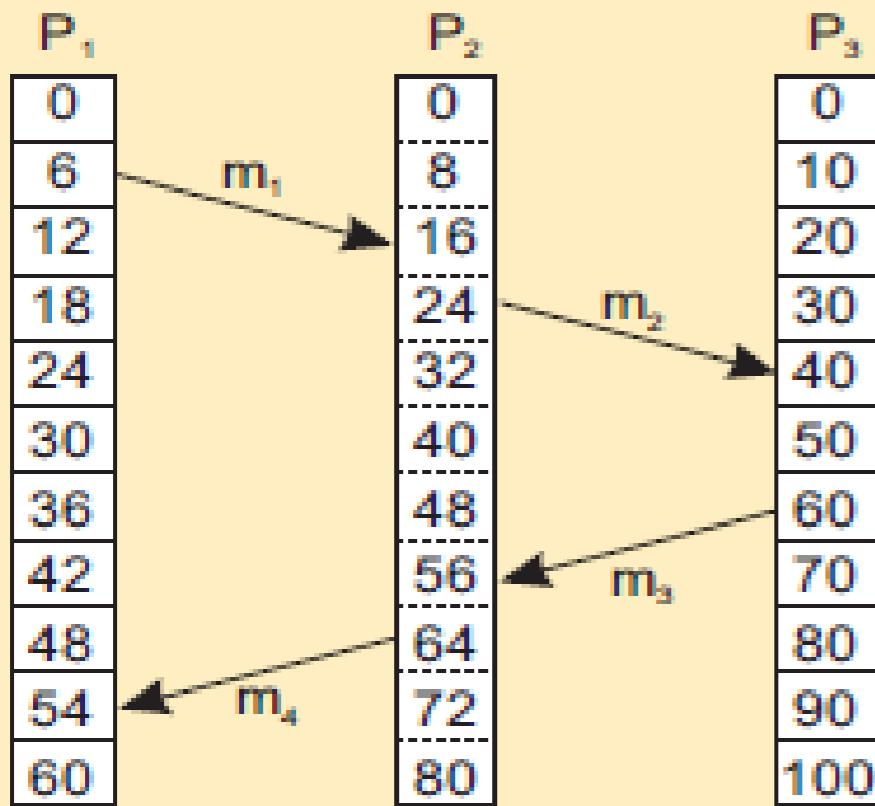
1. Para cada novo evento que acontece dentro de  $P_i$ ,  $C_i$  é incrementado de 1
2. Cada vez que uma mensagem  $m$  é enviada por um processo  $P_i$ , a mensagem recebe um timestamp  $ts(m) = C_i$ .
3. Toda vez que um mensagem  $m$  é recebida por um processo  $P_j$ ,  $P_j$  ajusta seu contador local  $C_j$  para  $\max\{C_j, ts(m)\}$ ; então executa passo 1 (*incremento de 1*) antes de passar  $m$  para a aplicação.

## NOTAS

- Propriedade P1 é satisfeita por (1); Propriedade P2 por (2) e (3).
- Ainda pode ocorrer que dois eventos acontecem ao mesmo tempo. Evite isto quebrando as amarras através de ID's de processos.

# EXEMPLO RELÓGIOS LÓGICOS

CONSIDERE TRÊS PROCESSOS COM CONTADORES DE EVENTOS OPERANDO EM TAXAS DIFERENTES



# RELÓGIOS LÓGICOS ONDE SÃO IMPLEMENTADOS

## AJUSTES IMPLEMENTADOS EM MIDDLEWARE

### Application layer

*Application sends message*

*Message is delivered to application*

*Adjust local clock and timestamp message*

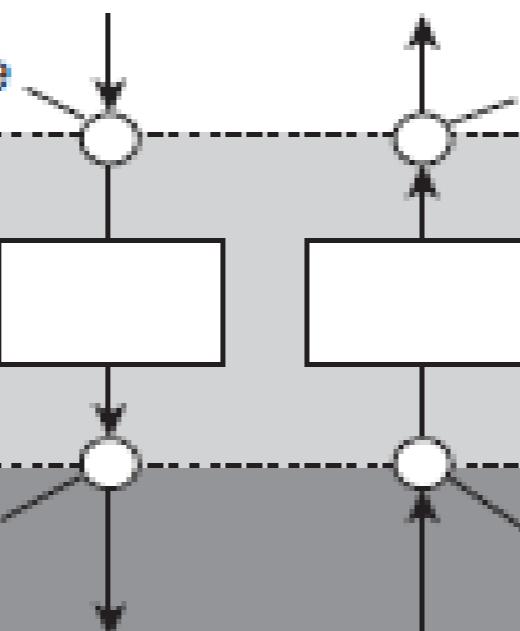
### Middleware layer

*Middleware sends message*

*Adjust local clock*

### Network layer

*Message is received*

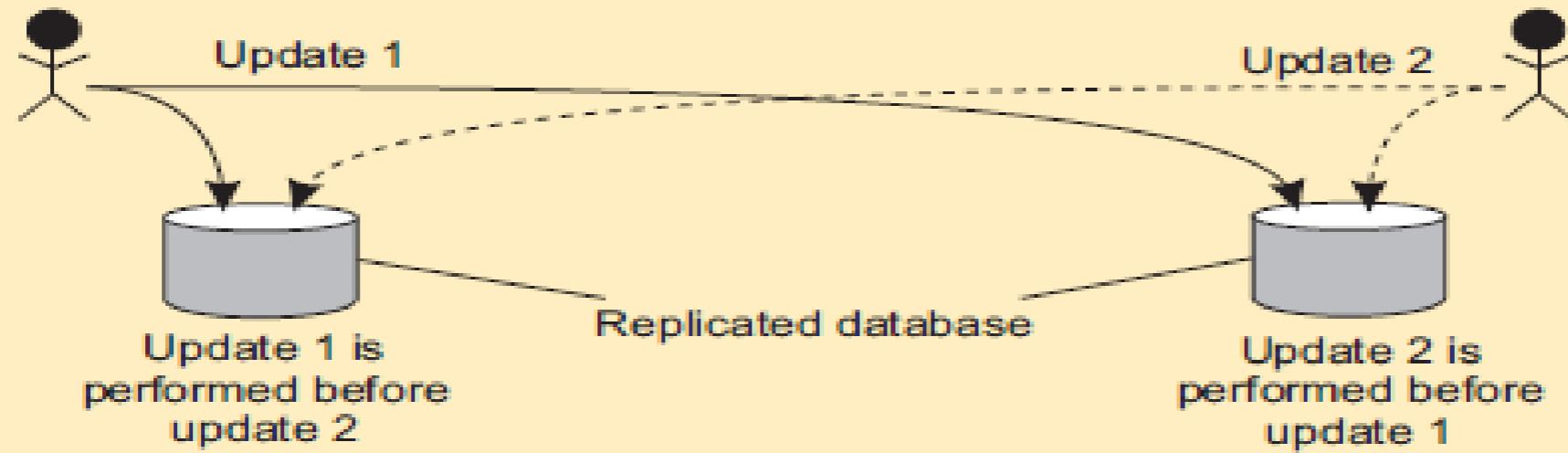


# EXEMPLO MULTICAST TOTALMENTE ORDENADO

RELÓGIOS LÓGICOS DE LAMPORT

ATUALIZAÇÕES CONCORRENTES EM BASES DE DADOS REPLICADAS SÃO VISTAS NA MESMA ORDEM EM TODOS LUGARES

- P1 adiciona \$100 a uma conta (valor inicial: \$1000)
- P2 incrementa a conta em 1%
- Existe duas replicas



## RESULTADO

- Na falta de sincronização adequada:
- Réplica #1  $\leftarrow \$1111$ , enquanto réplica #2  $\leftarrow \$1110$

## SOLUÇÃO

- Processo  $P_i$  envia mensagem com *timestamp*  $m_i$ , para todos outros processos. A mensagem é colocada em uma fila local  $queue_i$ ,
- Qualquer mensagem entrante em  $P_j$  é enfileirada em  $queue_j$ , de acordo com seu *timestamp*, e reconhecida por todos outros processos

$P_j$  PASSA A MENSAGEM  $m_i$  PARA SUA APLICAÇÃO SE:

- (1)  $m_i$  esta na cabeça da fila  $queue_j$
- (2) Para cada processo  $P_k$ , existe uma mensagem  $m_k$  na  $queue_j$  com um *timestamp* maior

## NOTA

- Estamos assumindo que a comunicação é **confiável** e ordenada por **FIFO**

```

1  class Process:
2      def __init__(self, chan):
3          self.queue      = []
4          self.clock      = 0
5
6      def requestToEnter(self):
7          self.clock = self.clock + 1
8          self.queue.append( (self.clock, self.procID, ENTER) ) # Increment clock value
9          self.cleanupQ() # Append request to q
10         self.chan.sendTo(self.otherProcs, (self.clock, self.procID, ENTER)) # Sort the queue
11
12     def allowToEnter(self, requester):
13         self.clock = self.clock + 1
14         self.chan.sendTo([requester], (self.clock, self.procID, ALLOW)) # Increment clock value
15
16     def release(self):
17         tmp = [r for r in self.queue[1:] if r[2] == ENTER] # Remove all ALLOWS
18         self.queue = tmp
19         self.clock = self.clock + 1
20         self.chan.sendTo(self.otherProcs, (self.clock, self.procID, RELEASE)) # and copy to new queue
21
22     def allowedToEnter(self):
23         commProcs = set([req[1] for req in self.queue[1:]]) # See who has sent a message
24         return (self.queue[0][1]==self.procID and len(self.otherProcs)==len(commProcs))

```

# RELÓGIOS DE LAMPORT PARA EXCLUSÃO MÚTUA

```
1  def receive(self):
2      msg = self.chan.recvFrom(self.otherProcs) [1]
3      self.clock = max(self.clock, msg[0])
4      self.clock = self.clock + 1
5      if msg[2] == ENTER:
6          self.queue.append(msg)
7          self.allowToEnter(msg[1])
8      elif msg[2] == ALLOW:
9          self.queue.append(msg)
10     elif msg[2] == RELEASE:
11         del(self.queue[0])
12         self.cleanupQ()
```

# Pick up any message  
# Adjust clock value...  
# ...and increment  
  
# Append an ENTER request  
# and unconditionally allow  
  
# Append an ALLOW  
  
# Just remove first message  
# And sort and cleanup



# RELÓGIOS DE LAMPORT PARA EXCLUSÃO MÚTUA

## ANALOGIA COM MULTICAST TOTALMENTE ORDENADO

- Com *multicast* totalmente ordenado, todos processos constroem filas idênticas, e entregam mensagens na mesma ordem
- Exclusão mútua é sobre um acordo sobre a ordem que processos poderão entrar em seções críticas

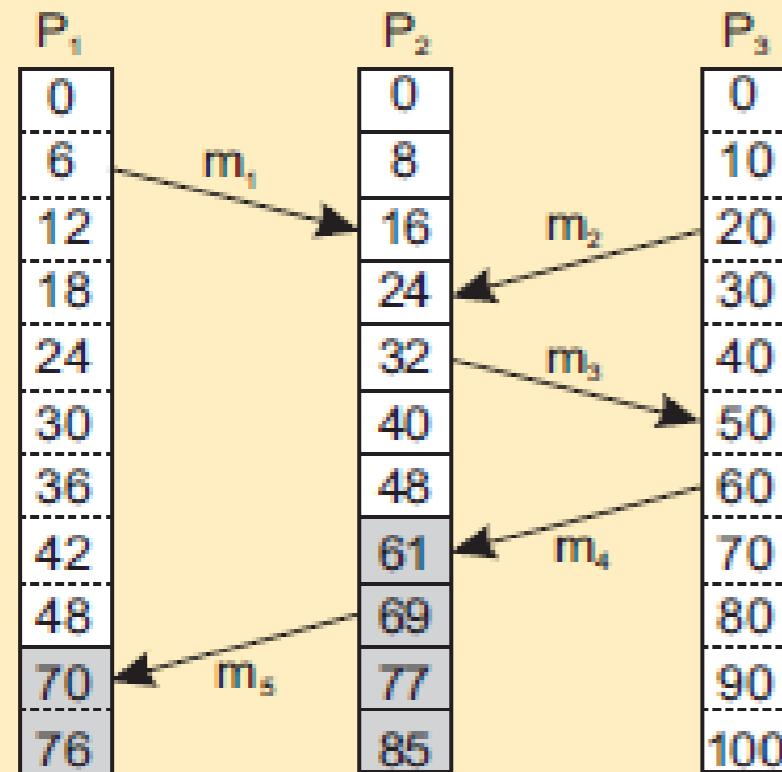


# RELÓGIOS VETORES

## OBSERVAÇÃO

Relógios de Lamport não garantem que se  $C(a) < C(b)$  então  $b$  é precedido por  $a$ .

## TRANSMISSÃO CONCORRENTE DE MENSAGENS USANDO RELÓGIOS LÓGICOS



## OBSERVAÇÃO

Evento a:  $m_1$  é recebido em  $T = 16$

Evento b:  $m_2$  é enviado em  $T = 20$

## NOTA

Não podemos concluir que causalmente  $a$  precede  $b$

# DEPENDÊNCIA CAUSAL

## DEFINIÇÃO

Dizemos que  $b$  pode causalmente depender em  $a$  se  $ts(a) < ts(b)$ , com:

- Para todo  $k$ ,  $ts(a)[k] \leq ts(b)[k]$  e
- Existe pelo menos um índice  $k'$  cujo  $ts(a)[k'] < ts(b)[k']$

## PRECEDÊNCIA VS DEPENDÊNCIA

- Dizemos que  $a$  causalmente precede  $b$
- $b$  **pode** causalmente depender em  $a$ , pois pode ter informação de  $a$  que é propagada pra  $b$

# CAPTURANDO CAUSALIDADE

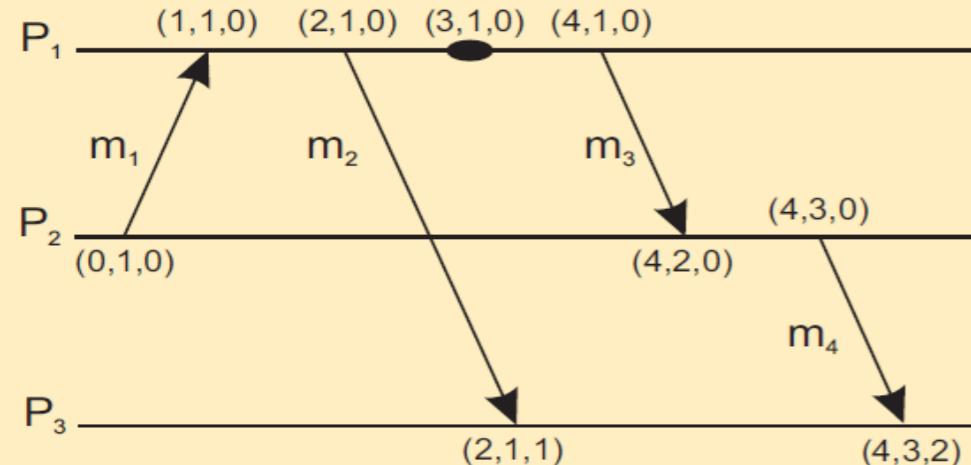
## SOLUÇÃO: CADA $P_i$ MANTÉM UM VETOR $VC_i$

- $VC_i[i]$  é o relógio local lógico no processo  $P_i$
- Se  $VC_i[j] = k$  então  $P_i$  sabe que  $k$  eventos ocorreram em  $P_j$

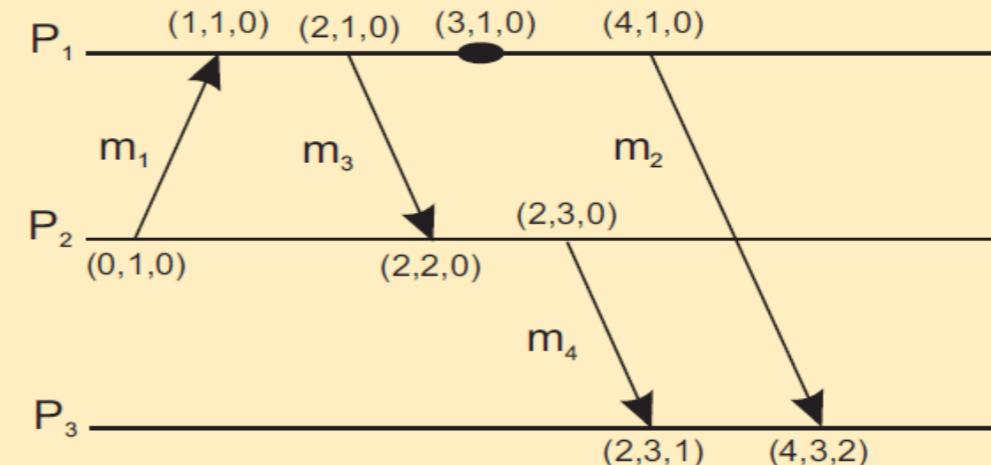
## MANTENDO RELÓGIOS VETORES (VECTOR CLOCKS)

1. Antes de executar um evento  $P_i$  execute  $VC[i] \leftarrow VC[i] + 1$ .
2. Quando processo  $P_i$  envia uma mensagem  $m$  para  $P_j$ , ele seta o vetor *timestamp*  $ts(m)$  de  $m$  igual a  $VC_i$  depois de ter executado passo 1.
3. No recebimento da mensagem  $m$ , o processo  $P_j$  seta  $VC_j[k] \leftarrow \max\{VC_j[k], ts(m)[k]\}$  para cada  $k$ , depois de executar passo 1 e então entregar a mensagem para a aplicação

## CAPTURANDO CAUSALIDADE POTENCIAL NA TROCA DE MENSAGENS



(a)



(b)

## ANÁLISE

Situation	$ts(m_2)$	$ts(m_4)$	$ts(m_2) < ts(m_4)$	$ts(m_2) > ts(m_4)$	Conclusion
(a)	(2, 1, 0)	(4, 3, 0)	Yes	No	$m_2$ may causally precede $m_4$
(b)	(4, 1, 0)	(2, 3, 0)	No	No	$m_2$ and $m_4$ may conflict

# MULTICAST ORDENADO CAUSALMENTE

## OBSERVAÇÃO

Podemos garantir que uma mensagem é entregue somente se todas mensagens causais precedentes foram entregues

## AJUSTE

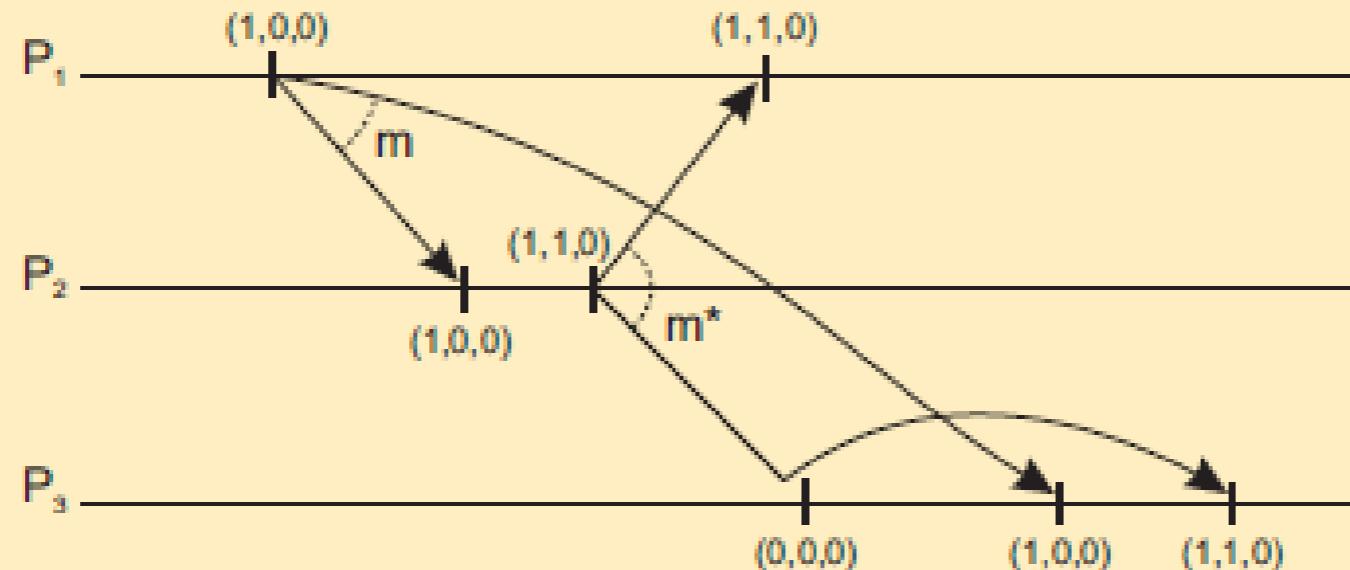
$P_i$  incrementa  $VC[i]$  somente quando estiver enviando uma mensagem e  $P_j$  ajusta  $VC_j$  quando estiver recebendo uma mensagem (i.e., efetivamente não muda  $VC_j[j]$ ).

## $P_j$ POSTERGA ENTREGA DE $m$ ATÉ:

- (1)  $ts(m)[i] = VC_j[i] + 1$
- (2)  $ts(m)[k] \leq VC_j[k]$  para todo  $k \neq i$

# MULTICAST ORDENADO CAUSALMENTE

## AJUSTE



## EXEMPLO

Pegue  $VC_3 = [0,2,2]$ ,  $ts(m) = [1,3,0]$  de  $P_1$ . Qual informação  $P_3$  tem e qual terá quando estiver recebendo  $m$  (de  $P_1$ )?

# EXCLUSÃO MÚTUA

## PROBLEMA

Um número de processos em um sistema distribuído quer acesso exclusivo a um recurso

## SOLUÇÕES BÁSICAS

Baseado em permissão: Um processo esperando para entrar em uma seção crítica ou acessar um recurso precisa permissão de outros processos

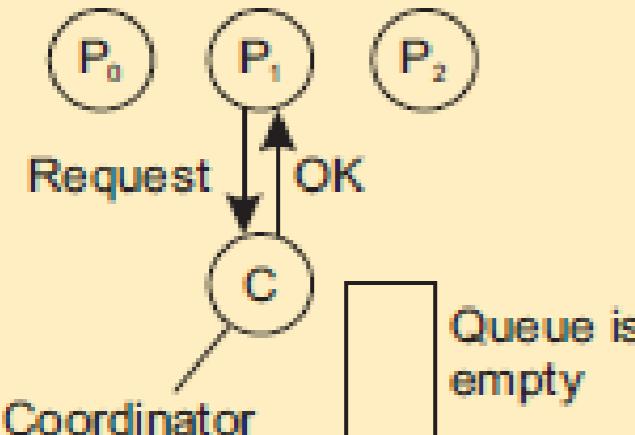
Baseado em token: Um token é passado entre processos. Aquele que tem o token pode entrar na seção crítica ou repassa o token caso não vá entrar.



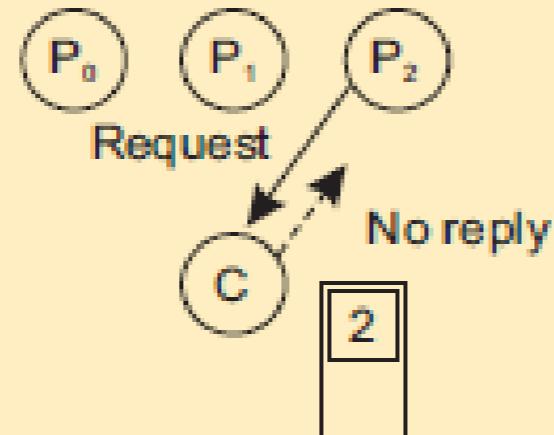
# BASEADO EM PERMISSÃO CENTRALIZADO

## SOLUÇÕES BÁSICAS

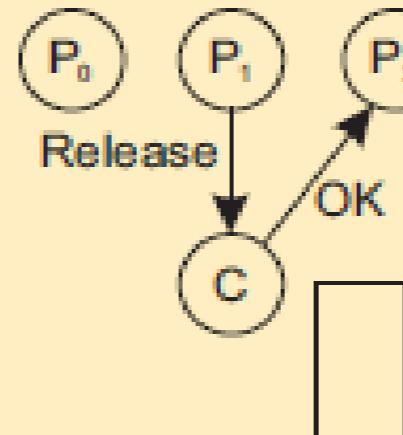
## EXCLUSÃO MÚTUA



(a)



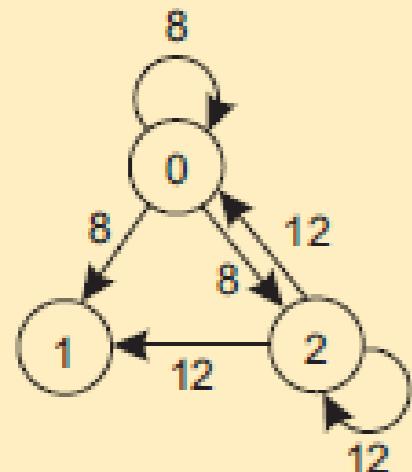
(b)



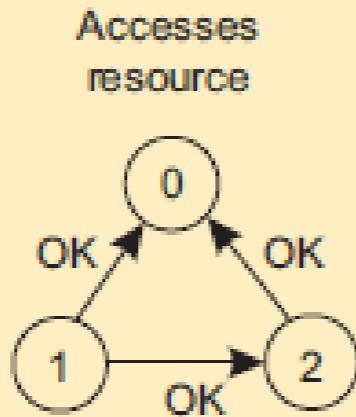
(c)

- a) Processo  $P_1$  pede ao coordenador permissão para acessar um recurso compartilhado. Permissão é dada.
- b) Processo  $P_2$  então pede permissão para acessar o mesmo recurso. O coordenador não responde.
- c) Quando  $P_1$  libera o recurso, ele avisa o coordenador que então envia resposta a  $P_2$

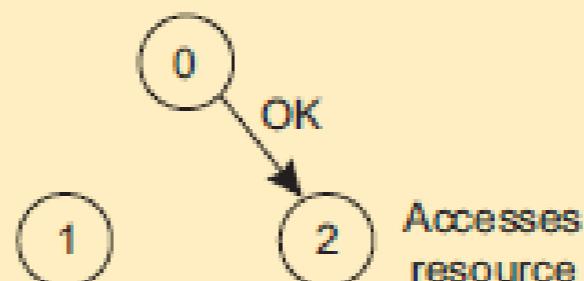
### EXEMPLO COM TRÊS PROCESSOS



(a)



(b)



(c)

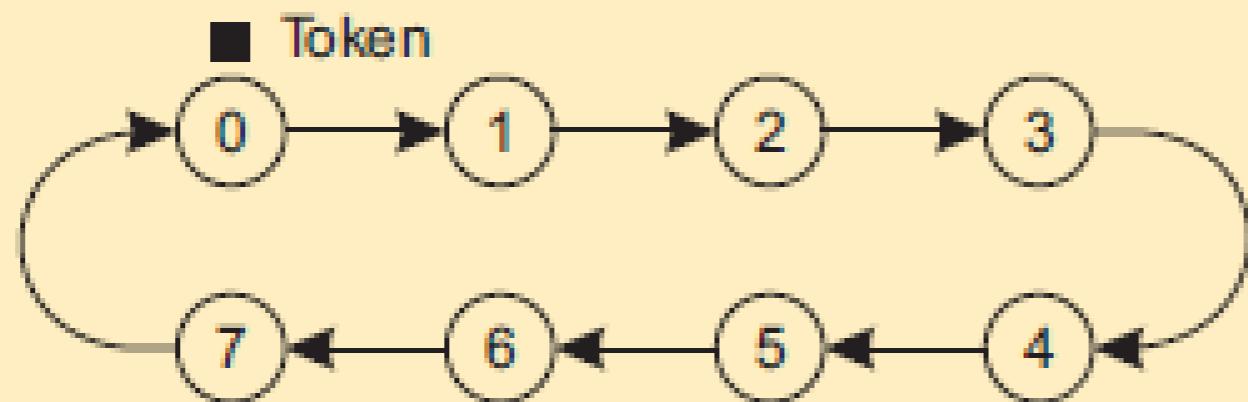
- a) Dois processos querem acessar um recurso compartilhado no mesmo momento
- b)  $P_0$  tem o menor *timestamp*, e então vence
- c) Quando processo  $P_0$  terminou, ele envia um OK também, assim  $P_2$  pode agora ir em frente

# EXCLUSÃO MÚTUA ALGORITMO TOKEN RING

## ESSÊNCIA

Organize um processo em um anel lógico e deixe um token se passado por ele. O que tiver o token terá permissão de entrar na região crítica (se quiser)

## UMA REDE OVERLAY CONSTRUÍDA COMO ANEL LÓGICO COM TOKEN CIRCULANDO



# EXCLUSÃO MÚTUA DECENTRALIZADA

## PRINCÍPIO

Assuma que cada recurso é replicado N vezes, com cada replica tendo seu próprio coordenador => acesso requer uma **maioria de votos** de  $m > N/2$  coordenadores. Um coordenador sempre responde imediatamente a uma requisição.

## SUPOSIÇÃO

Quando um coordenador cai, ele se recupera rapidamente mas terá esquecido as permissões que já deu



## QUÃO ROBUSTO É O SISTEMA

- Faça  $p = \Delta t / T$  ser a probabilidade de um coordenador resetar durante o intervalo  $\Delta t$ , durante o tempo de vida  $T$
- A probabilidade  $\mathbb{P}[k]$  de  $k$  de  $m$  coordenadores resetar durante o mesmo intervalo é:

$$\mathbb{P}[k] = \binom{m}{k} p^k (1-p)^{m-k}$$

- $f$  coordenadores resetam  $\Rightarrow$  exatidão é violada quando existe somente uma minoria de coordenadores sem problemas:  
quando  $m - f \leq N/2$ , ou  $f \geq m - N/2$
- A probabilidade de violação é:  $\sum_{k=m-N/2}^N \mathbb{P}[k]$ .

# EXCLUSÃO MÚTUA DECENTRALIZADA

## VIOLAÇÃO: PROBABILIDADES PARA VÁRIOS VALORES DE PARÂMETROS

N	m	p	Violation
8	5	3 sec/hour	$< 10^{-15}$
8	6	3 sec/hour	$< 10^{-18}$
16	9	3 sec/hour	$< 10^{-27}$
16	12	3 sec/hour	$< 10^{-36}$
32	17	3 sec/hour	$< 10^{-52}$
32	24	3 sec/hour	$< 10^{-73}$

N	m	p	Violation
8	5	30 sec/hour	$< 10^{-10}$
8	6	30 sec/hour	$< 10^{-11}$
16	9	30 sec/hour	$< 10^{-18}$
16	12	30 sec/hour	$< 10^{-24}$
32	17	30 sec/hour	$< 10^{-35}$
32	24	30 sec/hour	$< 10^{-49}$

ASSIM ..

O que podemos concluir ?

# EXCLUSÃO MÚTUA COMPARAÇÃO

Algorithm	Messages per entry/exit	Delay before entry (in message times)
Centralized	3	2
Distributed	$2 \cdot (N - 1)$	$2 \cdot (N - 1)$
Token ring	$1, \dots, \infty$	$0, \dots, N - 1$
Decentralized	$2 \cdot m \cdot k + m, k = 1, 2, \dots$	$2 \cdot m \cdot k$

# ALGORITMOS DE ELEIÇÃO

## PRINCÍPIO

Um algoritmo requer que alguns processos ajam como coordenadores. A questão é como selecionar este processo especial dinamicamente.

## NOTA

Em muitos sistemas o coordenador é escolhido manualmente (p.ex. servidores de arquivo). Isto leva a soluções centralizadas => ponto único de falha

## QUESTÕES

1. Se um coordenador é escolhido dinamicamente, até onde podemos falar sobre uma solução centralizada ou distribuída ?
2. Uma solução totalmente distribuída, isto é, sem um coordenador é sempre mais robusta que uma solução centralizada/coordenada ?



# SUPOSIÇÕES BÁSICAS

- Todos processos possuem ID's únicos
- Todos processos conhecem os ID's de todos processos no sistema (mas não se eles estão ativos ou não ativos)
- Eleição significa a identificação de processos com o maior ID que está ativo

# ELEIÇÃO POR BULLYING

## . PRINCÍPIO

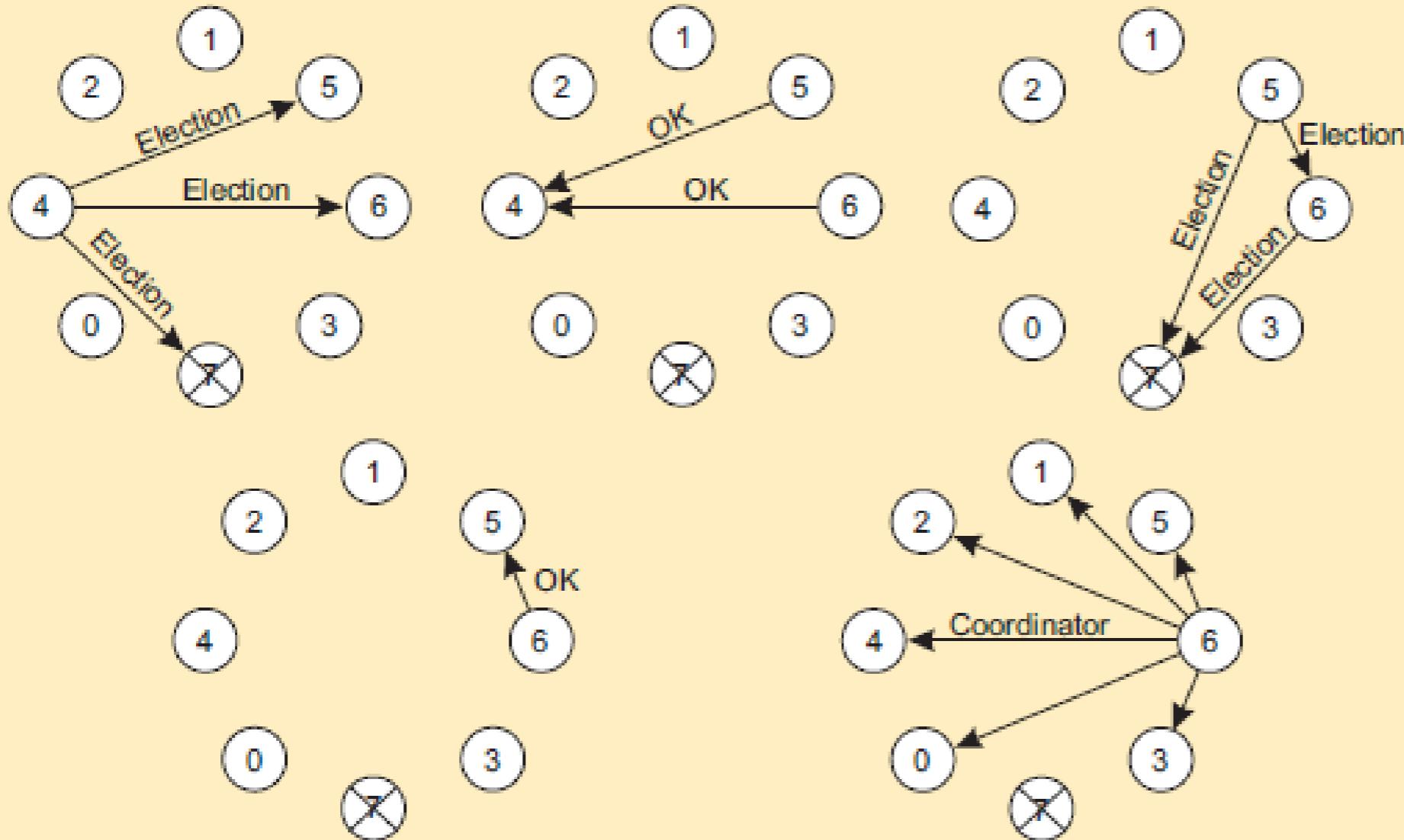
Considere  $N$  processos  $\{P_0, \dots, P_{N-1}\}$  e  $id(P_k) = k$ . Quando um processo  $P_k$  avisa que o coordenador não está mais respondendo a requisições, ele inicia uma eleição:

1.  $P_k$  envia uma mensagem ELEIÇÃO para todos processos com identificadores altos:  $P_{k+1}, P_{k+2}, \dots, P_{N-1}$ .
2. Se não há resposta,  $P_k$  vence a eleição e se torna coordenador.
3. Se um dos mais altos responde, ele assume e o trabalho de  $P_k$  estará feito.



# ELEIÇÃO POR BULLYING

## O ALGORITMO DE ELEIÇÃO POR BULLYING



ALGORITMO DE BULLYING

## PRINCÍPIO

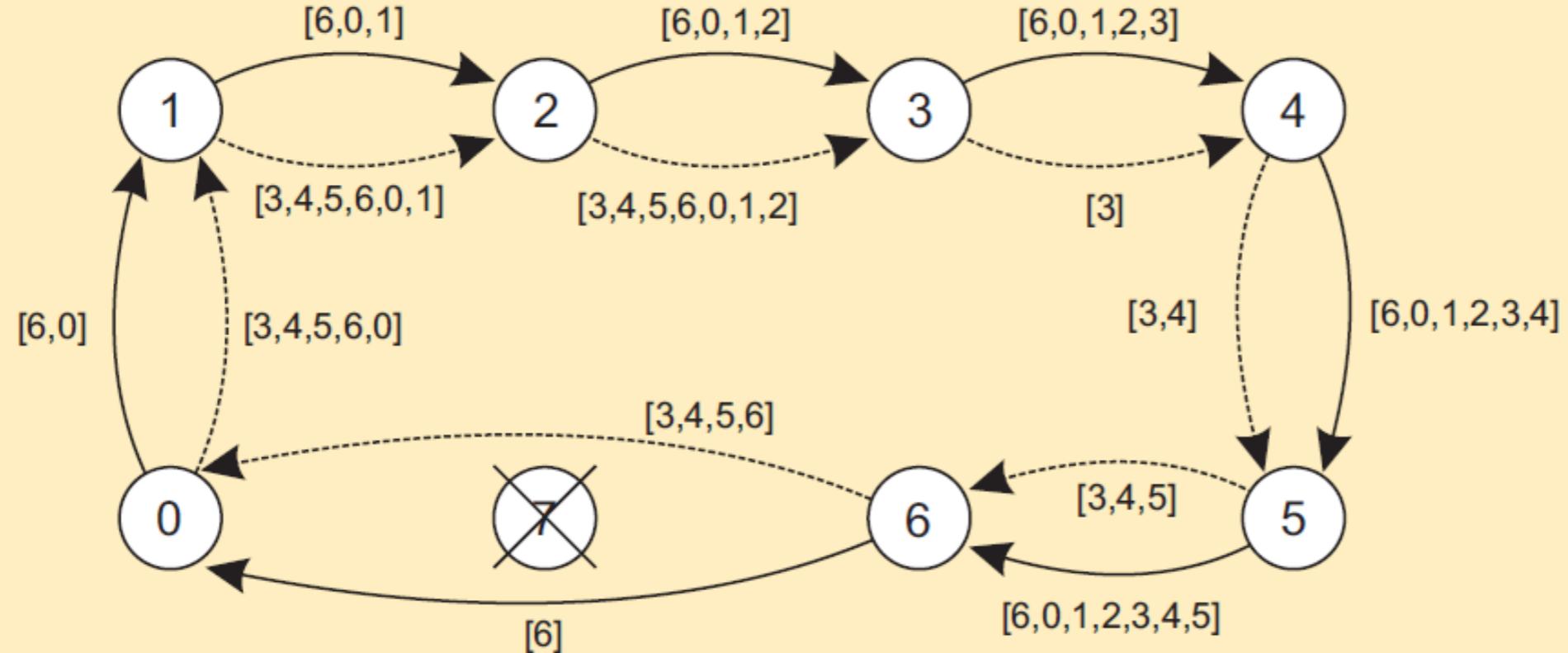
Prioridade do processo é obtida através da organização de processos em um anel lógico. Processos com a prioridade mais alta deve ser eleito como coordenador.

- Qualquer processo pode iniciar uma eleição enviando uma mensagem de eleição para seu sucessor. Se o sucessor não está disponível, a mensagem é passada para o próximo sucessor.
- Se uma mensagem é passada a frente, o despachante adiciona ele mesmo a lista. Quando a mensagem volta para o iniciador, todos tiveram a chance de tornarem sua presença conhecida.
- O iniciador envia uma mensagem de coordenação ao redor do anel contendo uma lista de processos vivos. O processo com a prioridade mais alta é eleito como coordenador

# ELEIÇÃO EM UM ANEL

## ELEIÇÃO USANDO ALGORITMO DE ANEL

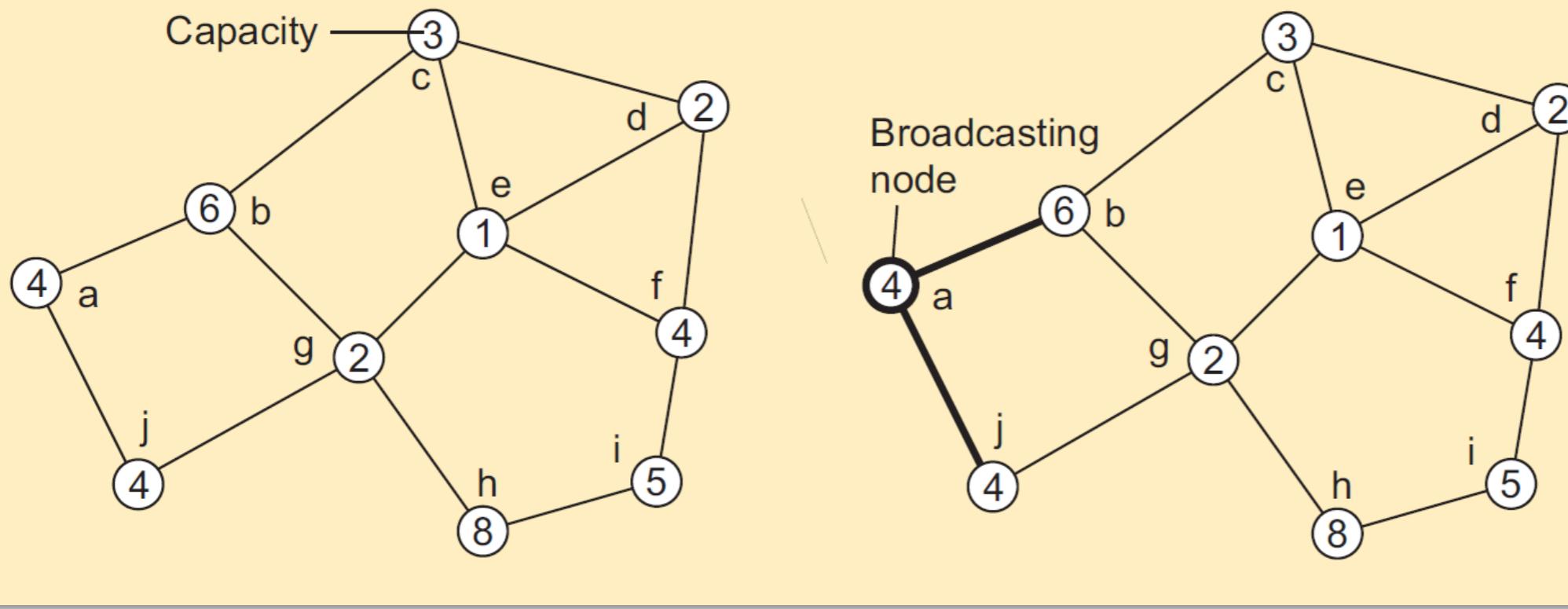
ALGORITMO ANEL



- Linha sólida mostra as mensagens de eleição iniciadas por P6
- A linha pontilhada as mensagens iniciadas por P3

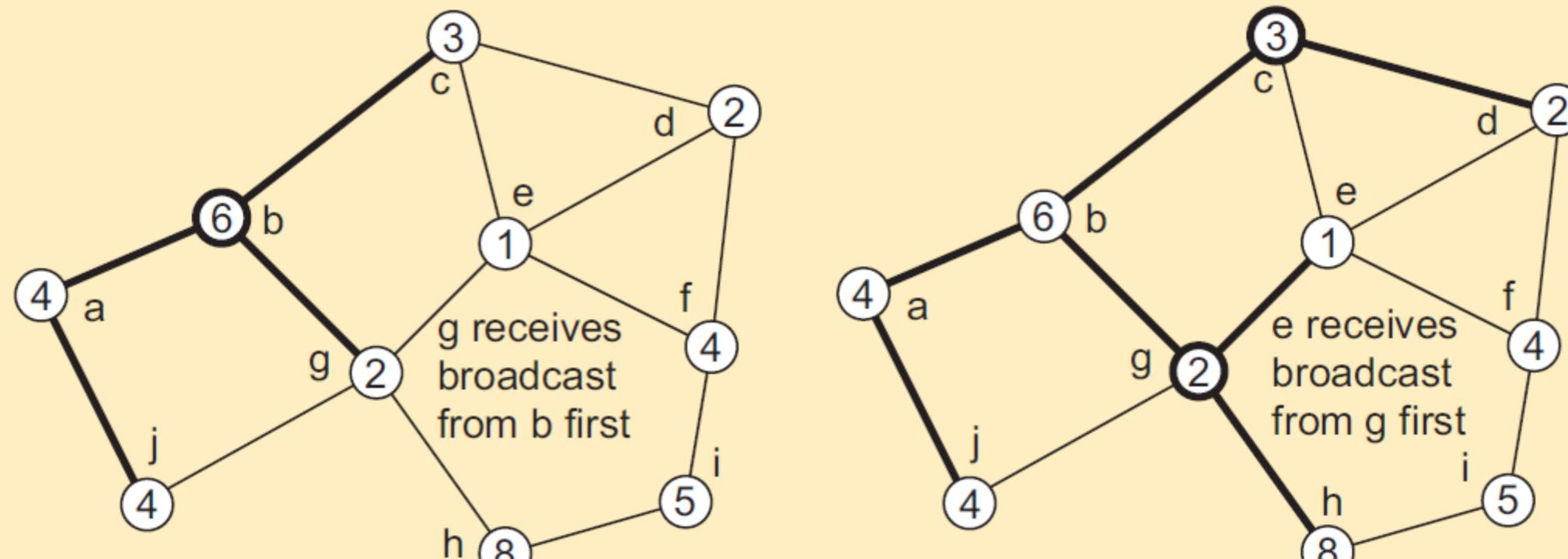
# SOLUÇÃO EM REDES SEM FIO

## EXEMPLO DE REDE



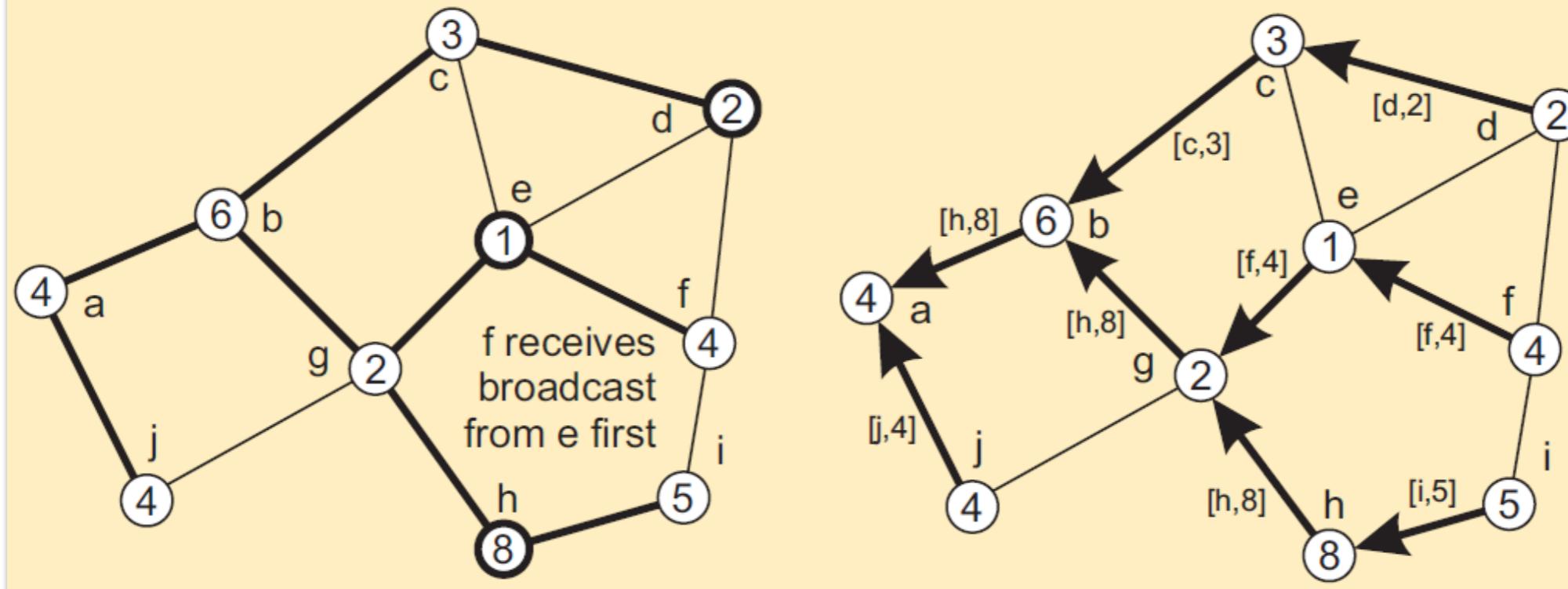
# SOLUÇÃO EM REDES SEM FIO

## EXEMPLO DE REDE



# SOLUÇÃO EM REDES SEM FIO

## EXEMPLO DE REDE



# POSICIONANDO NÓS

## QUESTÃO

- Em sistemas distribuídos de larga escala no qual nós estão dispersos através de uma rede de larga escala, frequentemente precisamos de uma noção e contabilizar a **proximidade** ou **distância** => começa com a determinação (relativa) da **localização** do nó.

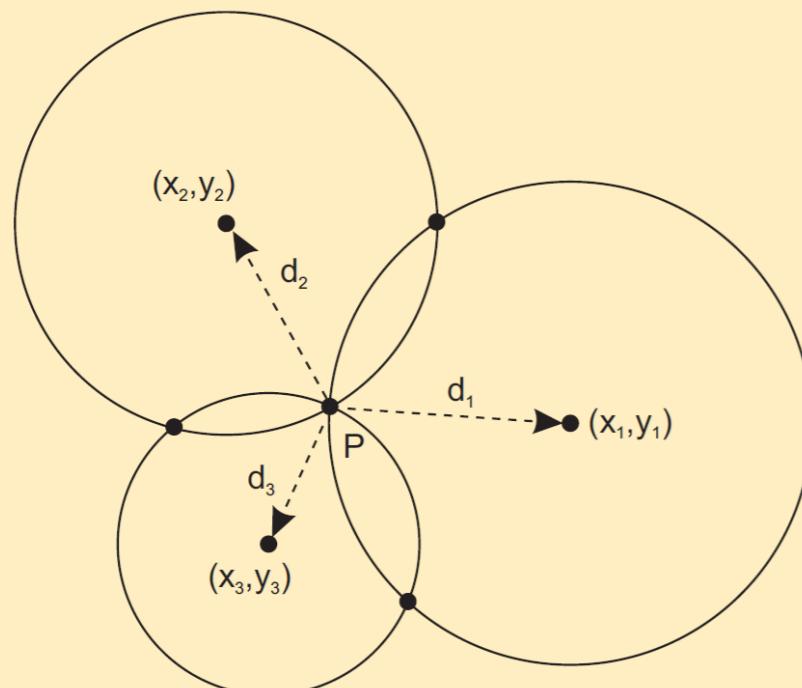


# COMPUTANDO POSIÇÃO

## OBSERVAÇÃO

- Um nó P precisa de  $d + 1$  posições (*landmarks*) para computar sua própria posição em um espaço  $d$ -dimensional. Considere o caso de 2 dimensões

## COMPUTANDO POSIÇÃO 2D



## SOLUÇÃO

$P$  precisa resolver 3 equações para dois desconhecidos  $(x_p, y_p)$

$$d_i = \sqrt{(x_i - x_P)^2 + (y_i - y_P)^2}$$

## ASSUMINDO QUE OS RELÓGIOS DE SATÉLITES SÃO ACURADOS E SINCRONIZADOS

- Leva um tempo até o sinal alcançar o receptor
- O relógio do receptor está definitivamente synchronization com o satélite

## BÁSICO

- $\Delta r$  : desvio desconhecido do relógio do receptor
- $X_r, Y_r, Z_r$  : coordenadas desconhecidas de um receptor
- $T_i$  : timestamp em uma mensagem vinda de um satélite i
- $\Delta i = (T_{\text{now}} - T_i) + \Delta r$  : atraso medido de uma mensagem enviada por um satélite i
- Distância medida até o satélite i:  $c \times \Delta i$  ( c é a velocidade da luz)
- Distância real:  $d_i = c\Delta i - c\Delta r = \sqrt{(x_i - x_r)^2 + (y_i - y_r)^2 + (z_i - z_r)^2}$

## OBSERVAÇÃO

- 4 satélites => 4 equações e 4 desconhecidos (com  $\Delta r$  como um deles)

# SERVIÇOS DE LOCALIZAÇÃO **BASEADO EM WIFI**

COORDENAÇÃO: SISTEMAS DE LOCALIZAÇÃO

## IDEIA BÁSICA

- Assuma que temos uma base de dados de pontos de acesso (APs) conhecidos e suas coordenadas
- Assuma que podemos estimar as distâncias para um AP
- Então: com 3 pontos de acesso detetados, podemos computar a posição

## COMO DETECTAR PONTOS DE ACESSO ?

- Use um dispositivo com WIFI ativado junto com GPS e ande através de uma área enquanto registra pontos de acesso observados
- Compute a posição CentroID: assuma que um ponto de acesso AP foi detetado em N diferentes localidades  $\{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_N\}$ , com localização GPS conhecida.
- Compute a localização de AP como sendo  $\vec{x}_{AP} = \frac{\sum_{i=1}^N \vec{x}_i}{N}$ .

## PROBLEMAS

- Acuracidade limitada de cada ponto GPS detectado  $\vec{x}_i$
- Um ponto de acesso tem um campo de transmissão não uniforme
- Número de pontos de amostragem detetados N pode ser muito baixo.

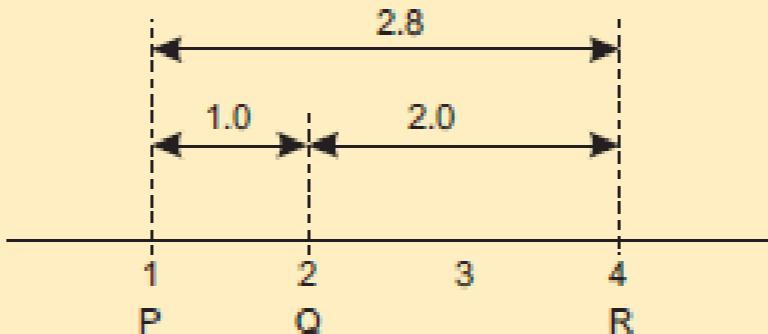
# COMPUTANDO POSIÇÃO

## POSICIONAMENTO CENTRALIZADO

### PROBLEMAS

- Latências medidas para *landmarks* flutuam
- Distâncias computadas não são nada consistentes

### DISTÂNCIAS INCOSISTENTES NO ESPAÇO 1D



### SOLUÇÃO: MINIMIZAR ERROS

- Use N *landmark* nodes especiais  $L_1, \dots, L_N$
- Landmarks medem suas latências em pares  $\tilde{d}(L_i, L_j)$
- Um nó central computa as coordenadas de cada landmark minimizando:

$$\sum_{i=1}^N \sum_{j=i+1}^N \left( \frac{\tilde{d}(L_i, L_j) - \hat{d}(L_i, L_j)}{\tilde{d}(L_i, L_j)} \right)^2$$

Onde  $\hat{d}(L_i, L_j)$  é a distância depois que os nós  $L_i$  e  $L_j$  foram posicionados

# COMPUTANDO POSIÇÃO

## ESCOLHENDO A DIMENSÃO $m$

- O parâmetro escondido na dimensão  $m$  com  $N > m$ . Um nó  $P$  mede sua distância para cada um dos  $N$  *landmarks* e computa sua coordenada minimizando

$$\sum_{i=1}^N \left( \frac{\tilde{d}(L_i, P) - \hat{d}(L_i, P)}{\tilde{d}(L_i, P)} \right)^2$$

## OBSERVAÇÃO

- A prática mostra que  $m$  pode ser tão pequeno quanto 6 ou 7 para estimativas de latências dentro de um fator 2 do valor atual

## PRINCÍPIO: REDE DE MOLAS IMPONDO FORÇAS

- Considere uma coleção de  $N$  nós  $P_1, \dots, P_N$  com  $P_i$  tendo coordenadas  $\vec{x}_i$ . Dois nós exercem força mútua:

$$\vec{F}_{ij} = (\tilde{d}(P_i, P_j) - \hat{d}(P_i, P_j)) \times u(\vec{x}_i - \vec{x}_j)$$

- Com  $u(\vec{x}_i - \vec{x}_j)$  é o vetor unidade na direção de  $\vec{x}_i - \vec{x}_j$

NÓ  $P_i$  REPETIDAMENTE EXECUTA OS PASSOS

- ➊ Measure the latency  $d_{ij}$  to node  $P_j$ , and also receive  $P_j$ 's coordinates  $\vec{x}_j$ .
- ➋ Compute the error  $e = \tilde{d}(P_i, P_j) - \hat{d}(P_i, P_j)$
- ➌ Compute the direction  $\vec{u} = u(\vec{x}_i - \vec{x}_j)$ .
- ➍ Compute the force vector  $F_{ij} = e \cdot \vec{u}$
- ➎ Adjust own position by moving along the force vector:  $\vec{x}_i \leftarrow \vec{x}_i + \delta \cdot \vec{u}$ .

# APLICAÇÕES EXEMPLO

## APLICAÇÕES TÍPICAS

- **Disseminação de dados:** talvez a aplicação mais importante. Note que existe muitas variantes de disseminação.
- **Agregação:** se cada nó  $P_i$  mantém uma variável  $v_i$ . Quando dois nós criam rumor, cada um reseta a variável para:

$$v_i, v_j \leftarrow (v_i + v_j)/2$$

- Resultado: no final, cada nó terá computado a média  $\bar{v} = \sum_i v_i / N$ .
- O que acontece no caso de inicialmente  $v_i = 1$  e  $v_j = 0$ ,  $j \neq i$  ?

# PROFESSOR DO CURSO



**FREDY JOÃO VALENTE, PhD**

**Engenheiro Elétrico pela EESC-USP / 1987, Mestre em Física Aplicada pelo IFSC-USP / 1991 e  
Ph.D. Electronics & Computer Science - University of Southampton UK / 1995**

Prof. Dr. CNPq: ICMC USP 1996-97,  
Sócio Diretor da S&V Consultoria 1997-2004: Soluções corporativas NFC/Mifare – Bilhetagem / Controle Acesso Inteligente.  
Pós Doc 2005 na Universidade de Southampton: Computação pervasiva e Computação em GRID para Supply-Chain RFID/EPC Gen2.  
Sócio Diretor da COSS Consulting 2005-2015: Cadeias de abastecimento inteligentes baseadas em RFID/EPC e Soluções IoT.  
Coordenador do projeto [Mions: Mobile Interaction on Spot, Internet of Things \(IoT\) and Context Analysis](#):  
FINEP Subvenção Econômica 2009-2012;  
Coordenador de vários projetos FAPESP PIPE II e III  
Atualmente é Prof. Dr. em Engenharia de Computação no Departamento de Computação - [UFSCar](#) – desde out / 2015.