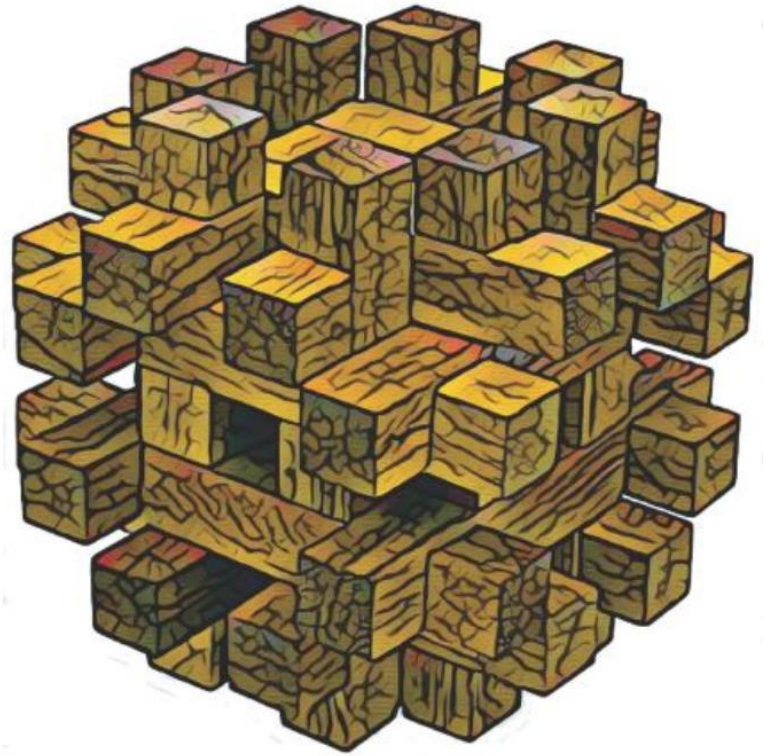
Distributed Systems

Maarten Van Steen & Andrew S. Tanenbaum



3th Edition – Version 3.03 - 2022

Capítulo 5

Nomeação

Quinta-feira, 04 de Agosto de 2022

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”

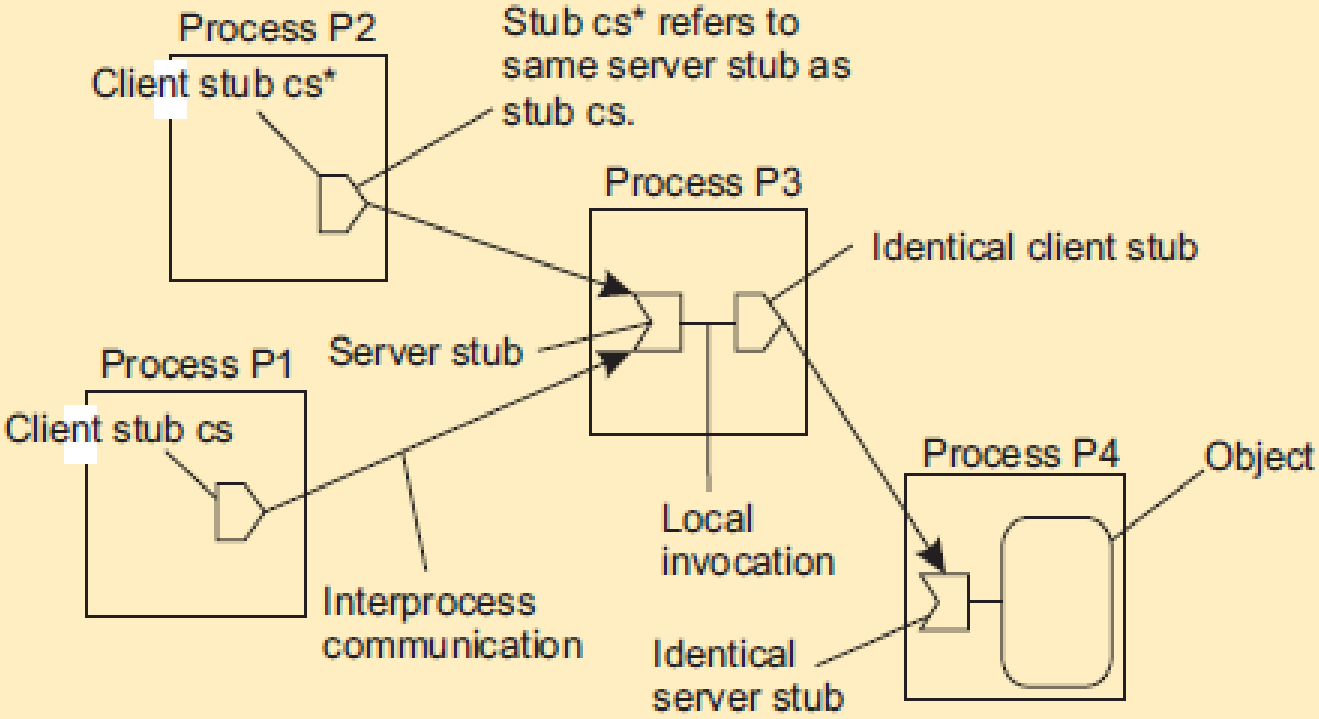
REPASSANDO PONTEIROS

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

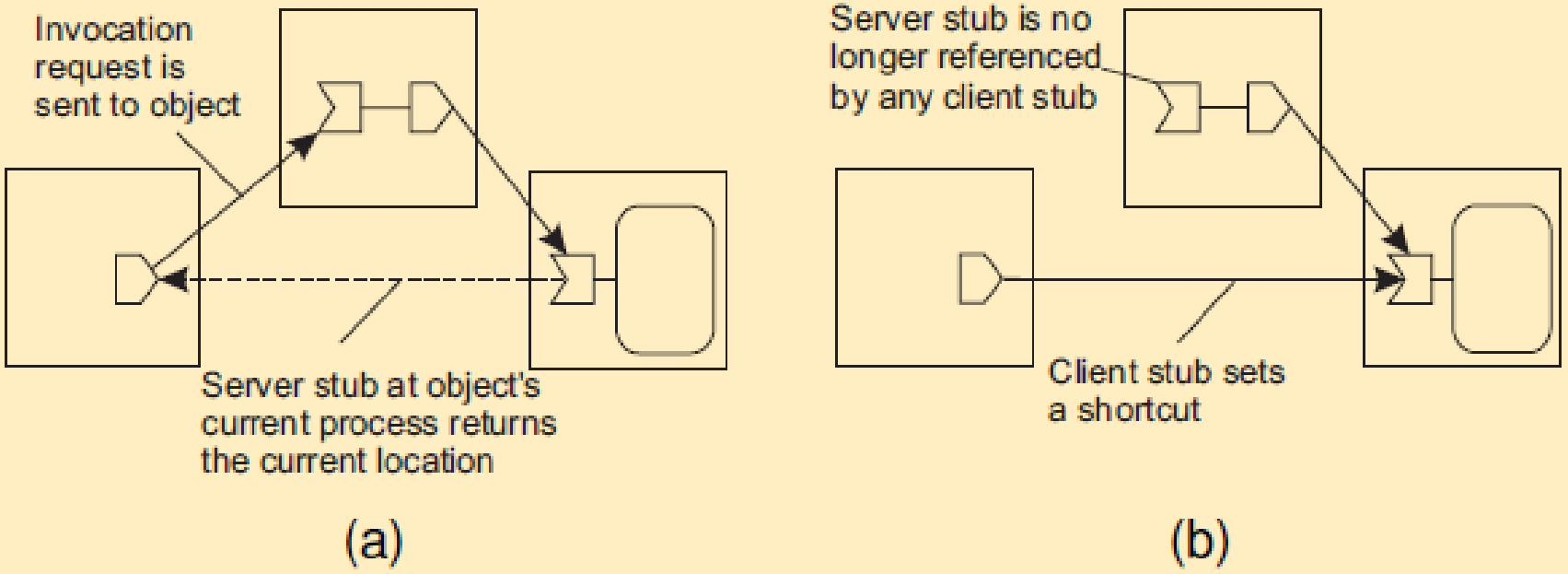
EXEMPLO CADEIAS SSP (single shared platform)

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

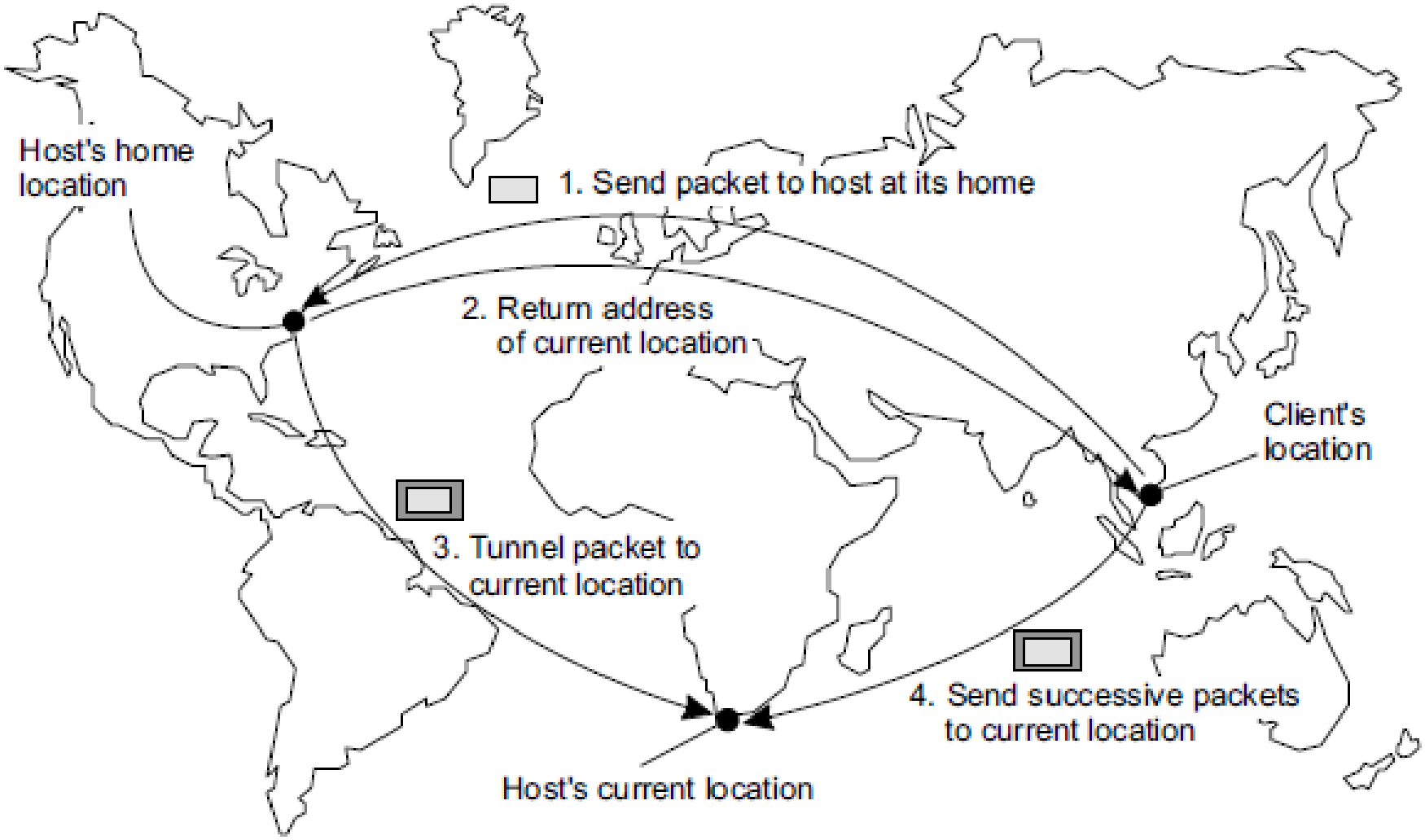


ABORDAGEM BASEADA NO ENDEREÇO LOCAL

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 suportato 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 LÓGIGO

- Cada nó é atribuído um indentificador 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* ≥ *k* (chamado de sucessor *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 possuí 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} .

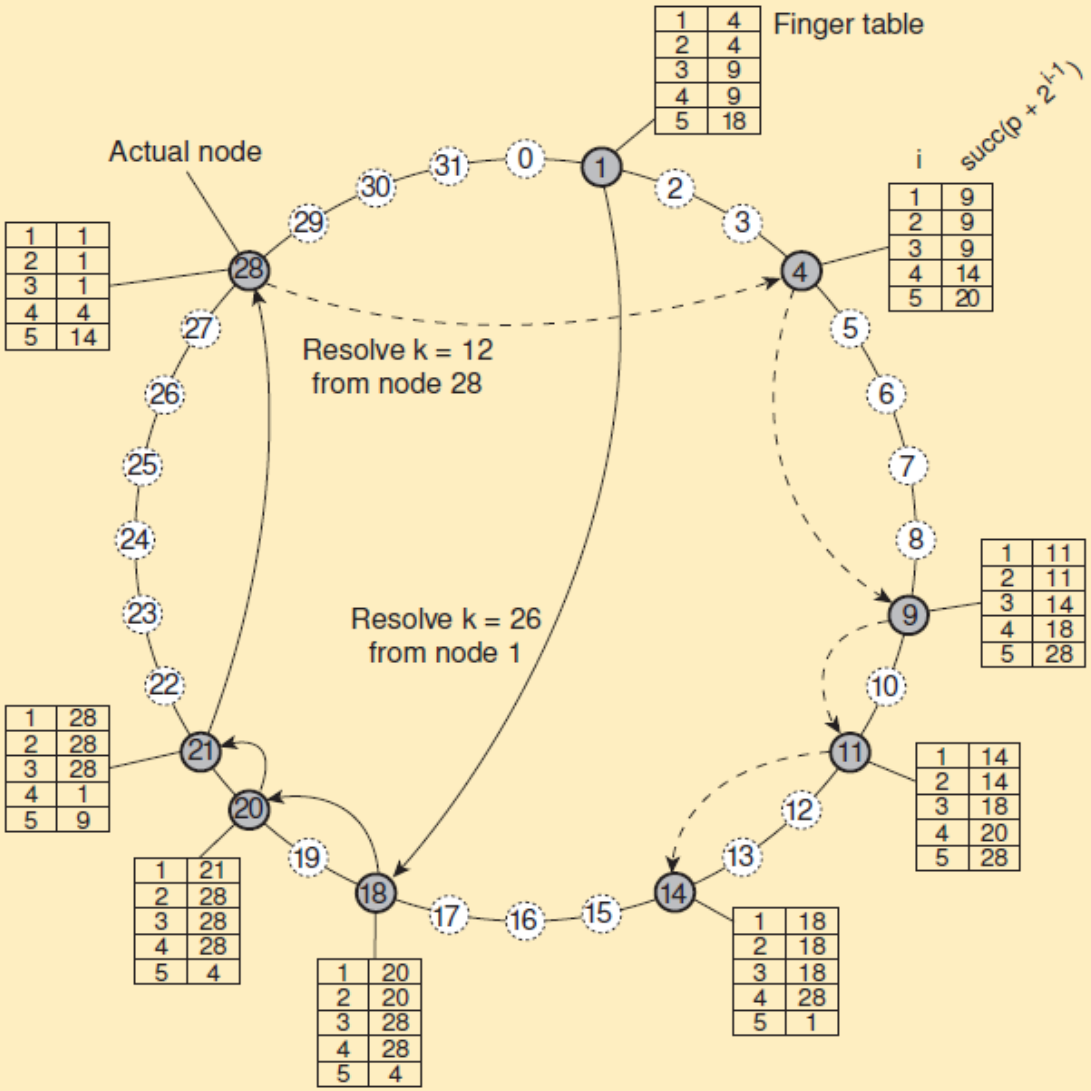
- Para procurar uma chave *k*, o nó *p* repassa a requisição para o nó com índice *j* satisfazendo

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

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

CHORD: EXEMPLO LOOKUP

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



- 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: EM PYTHON (NOMEAÇÃO PLANA / TABELA HASH DISTRIBUÍDA)

```
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]):
22         return self.FT[i]                                # in [FT[i],FT[i+1])
```

EXPLORANDO PROXIMIDADE NA REDE

PROBLEMA

- A organização lógica de nós na rede overlay pode levar a transferências erradas de mensagens na Internet abaixo: nó *p* e nó *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*.

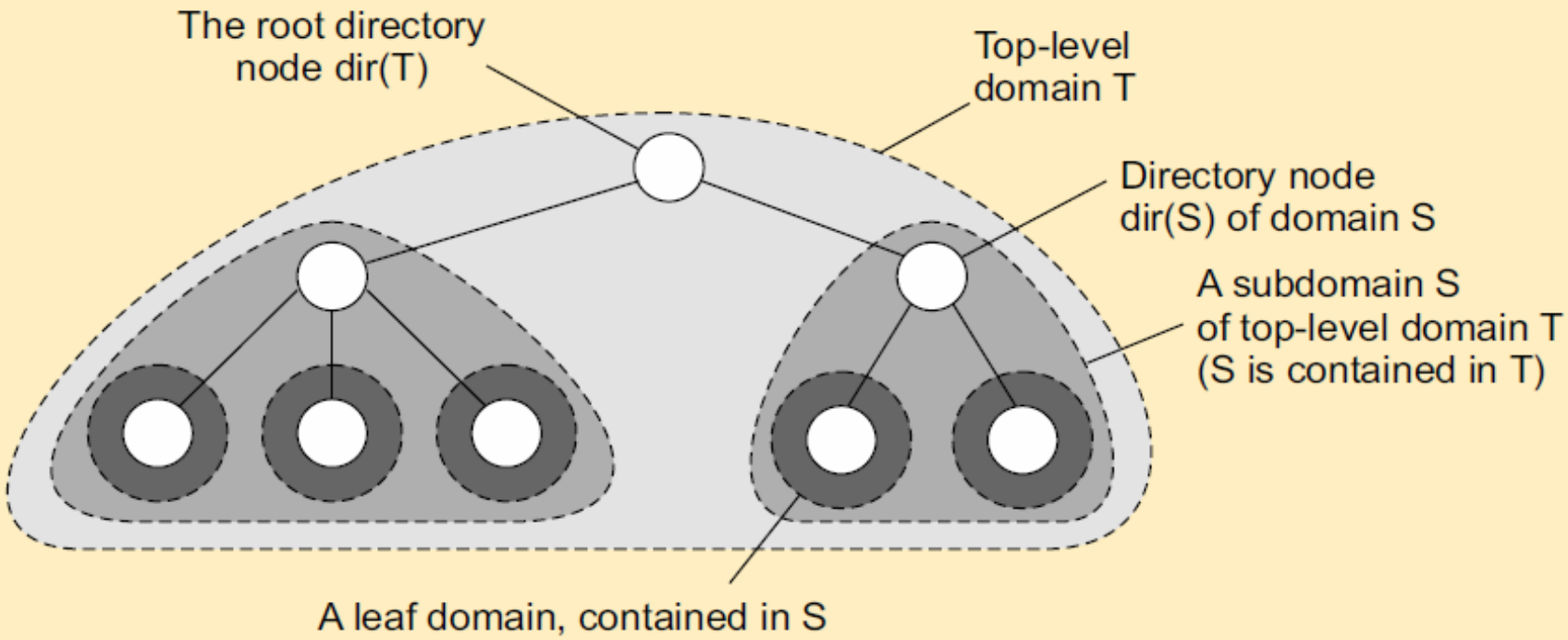
- Seleção de vizinho na proximidade: quando existe escolha para selecionar quem o vizinho será (não no Chord), pegue o mais próximo.

HLS - HIERARCHICAL LOCATION SERVICES

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

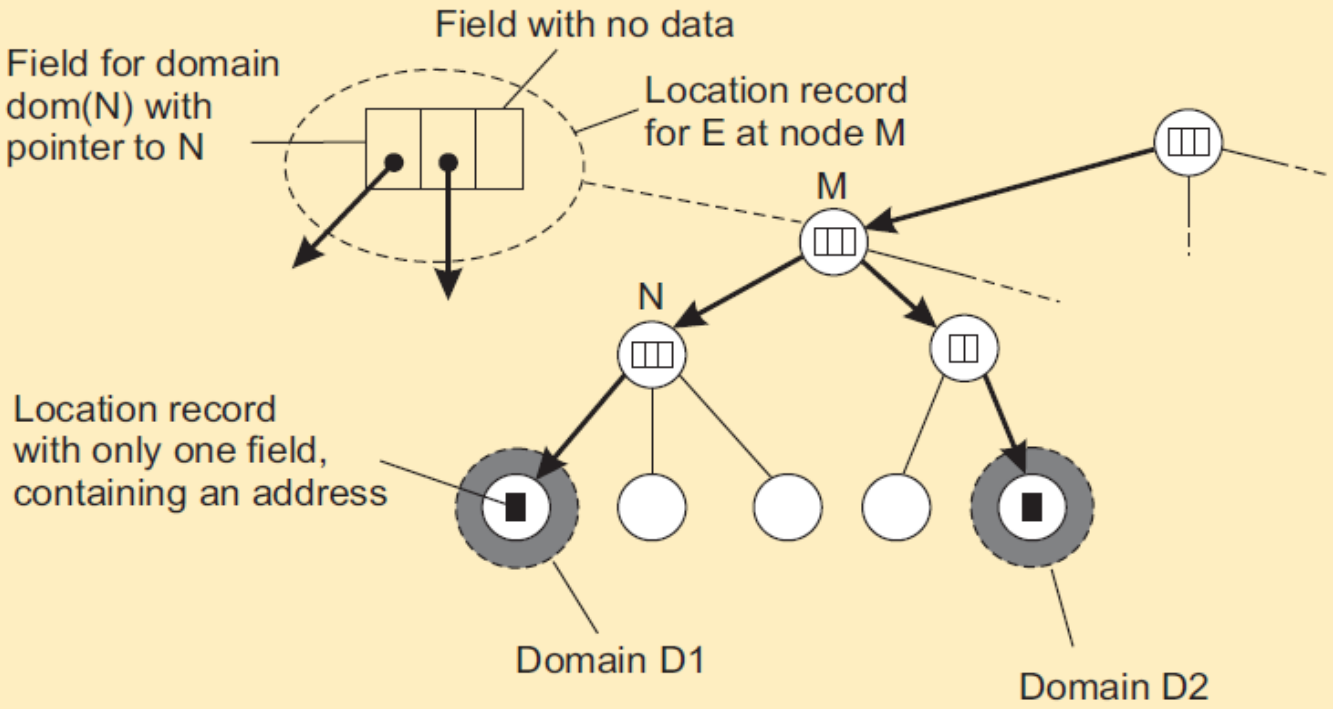


HLS - ORGANIZAÇÃO EM ÁRVORE

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

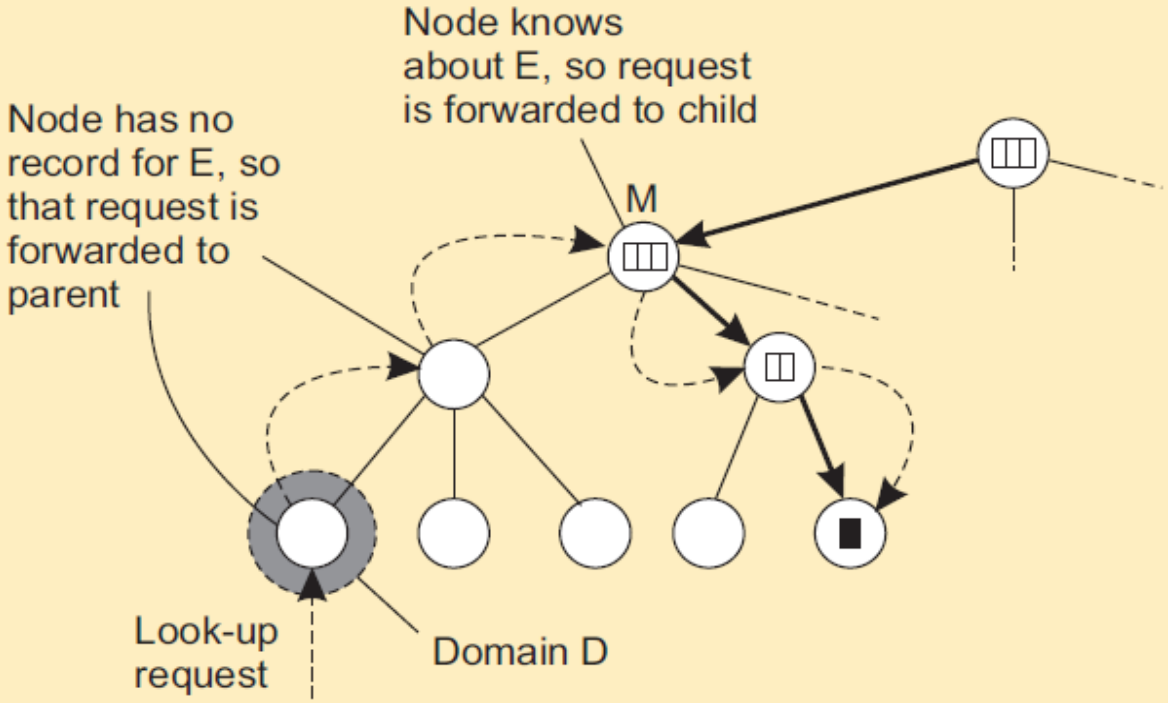


HLS - OPERAÇÃO LOOKUP

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

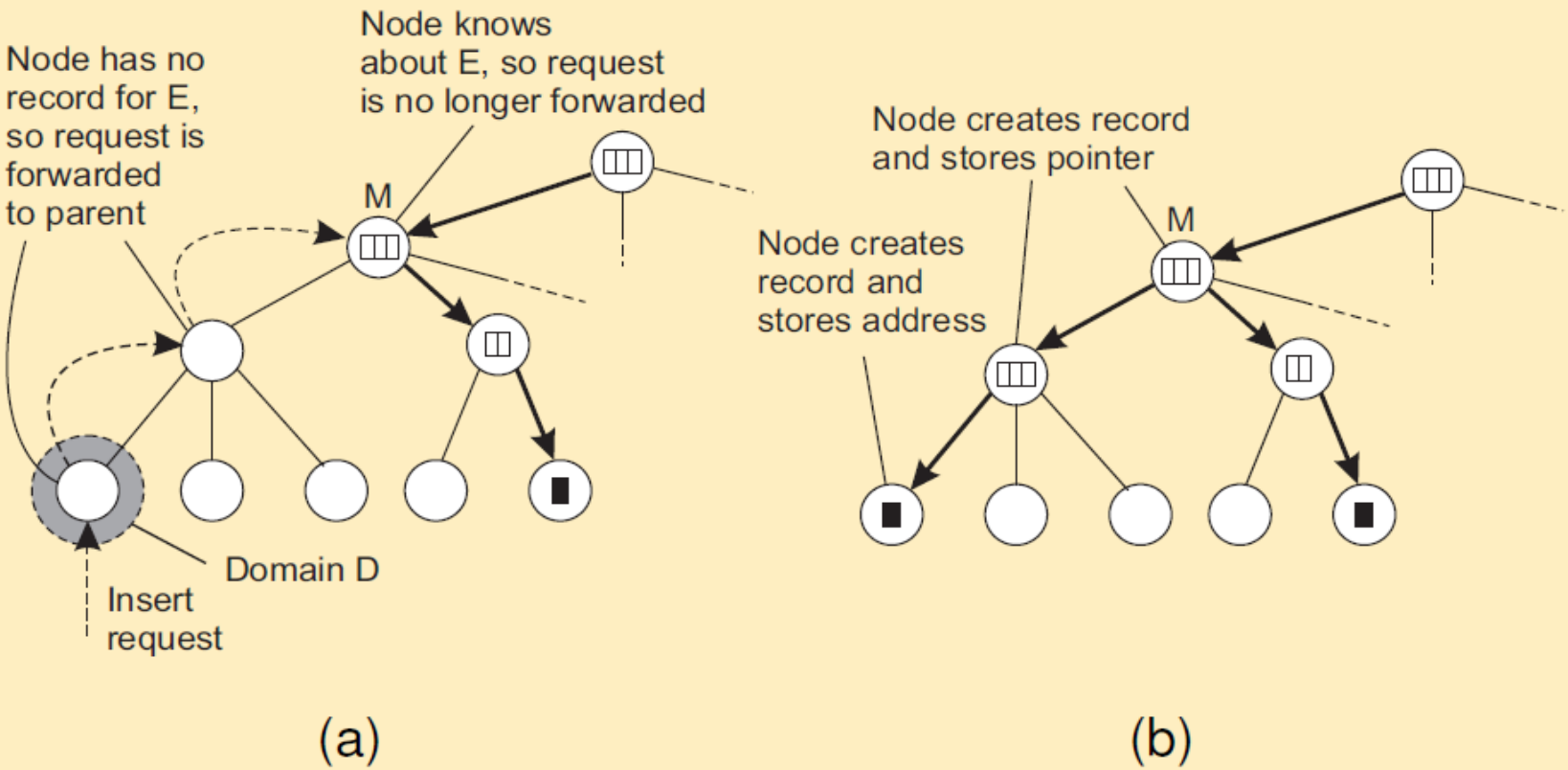
BUSCANDO (LOOKING UP) UMA LOCALIZAÇÃO



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

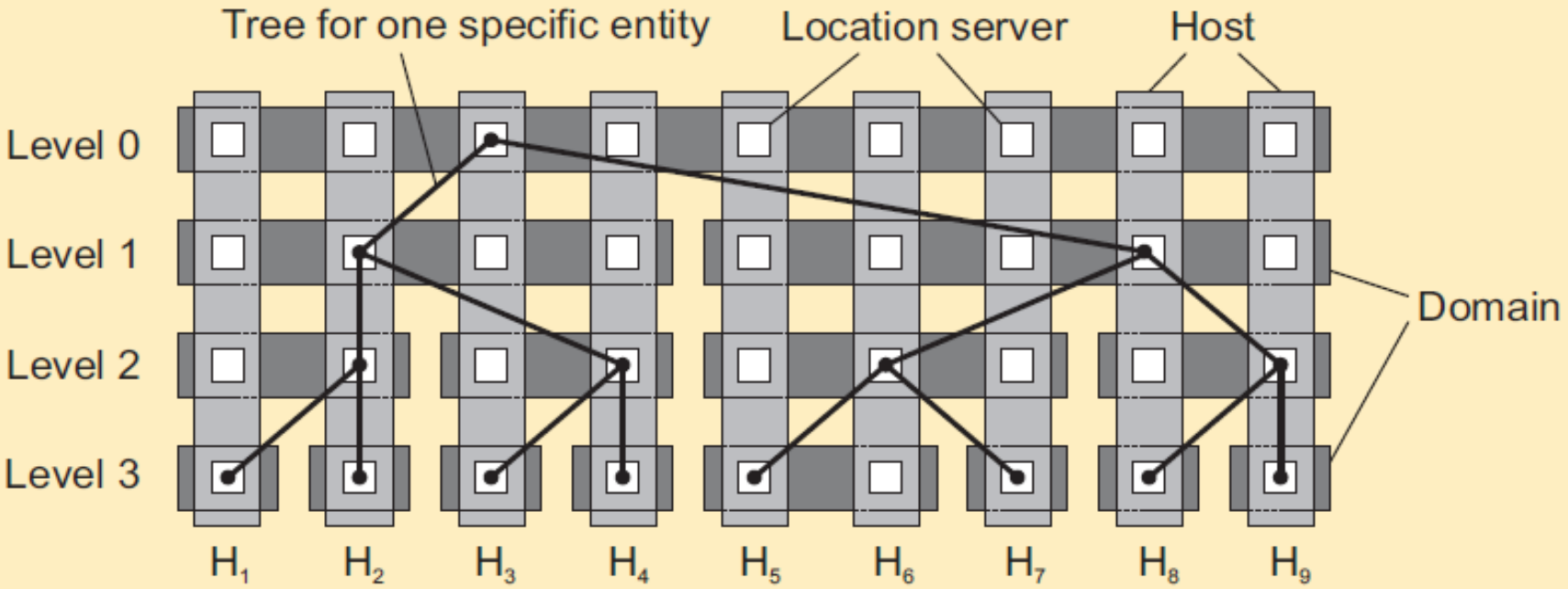


HLS - PODE ESCALAR ?

SOLUÇÃO

- $D_k = \{D_{k,1}, D_{k,2}, \ldots, D_{k,Nk}\}$ denota os N_k domínios no nível k
- Nota: $N0 = |D0| = 1$
- Para cada nível *k*, o conjunto de hospedeiros é particionado 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

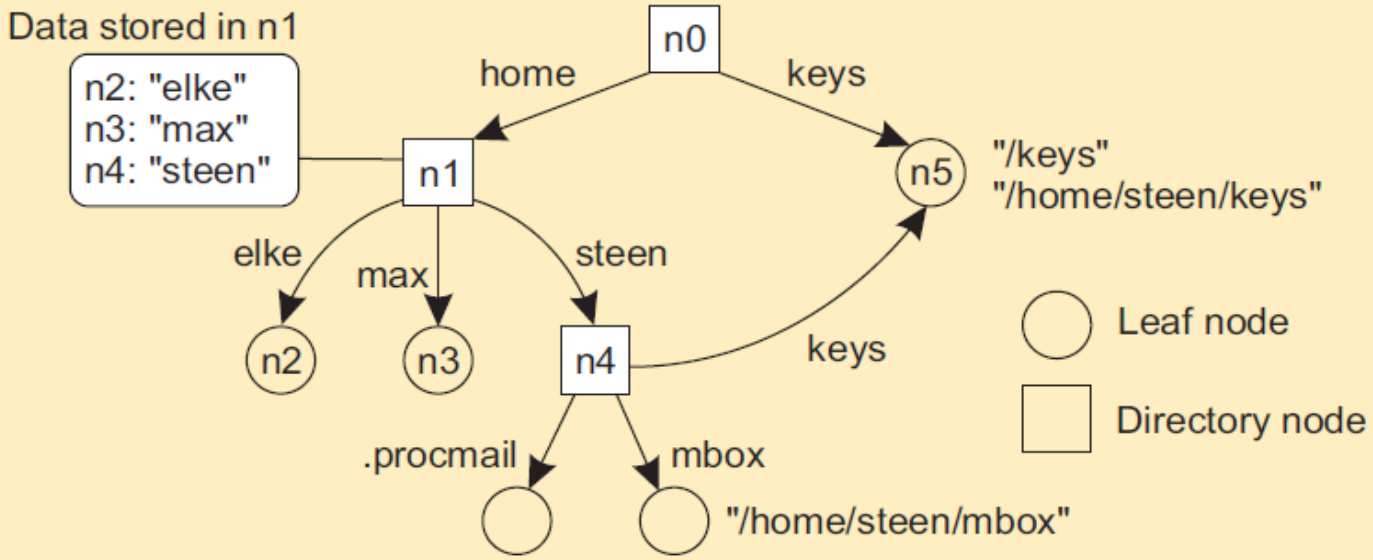


ESPAÇO DE NOMES

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

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 explicito – como começar ?

LIGAÇÃO DE NOMES

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

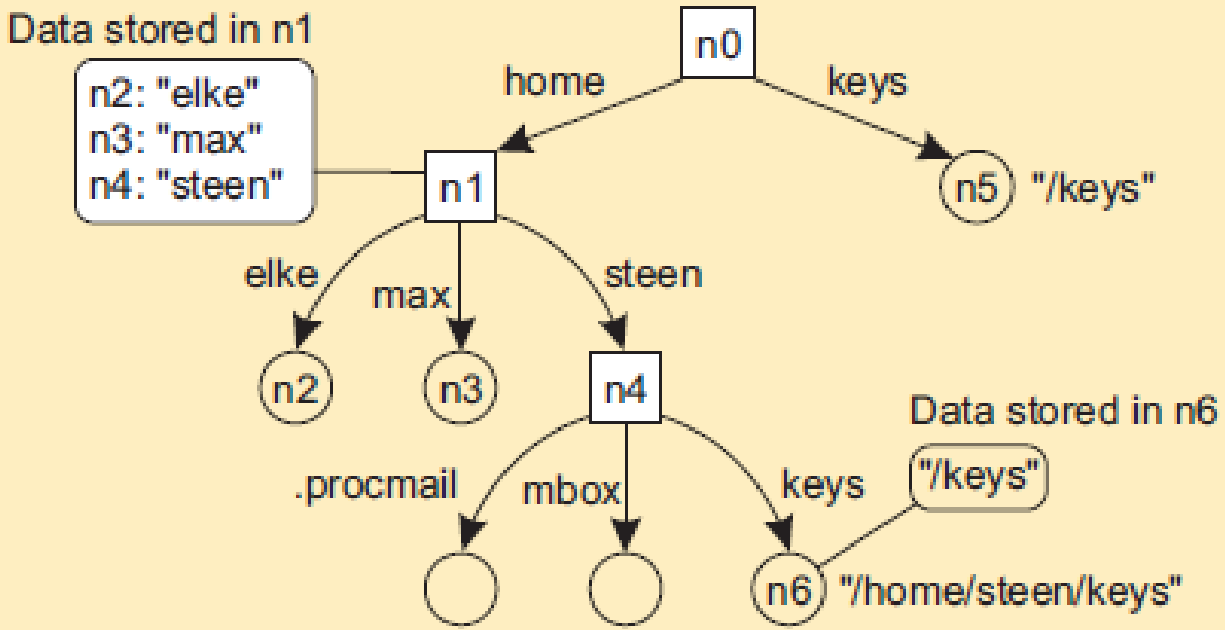
- Primeiro resolve o nome de *N* (que leva a *N*)
- Lê o conteúdo de *N*, resutando em *N*
- A resolução de nome continua com *nome*

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

MONTANDO NOMES

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

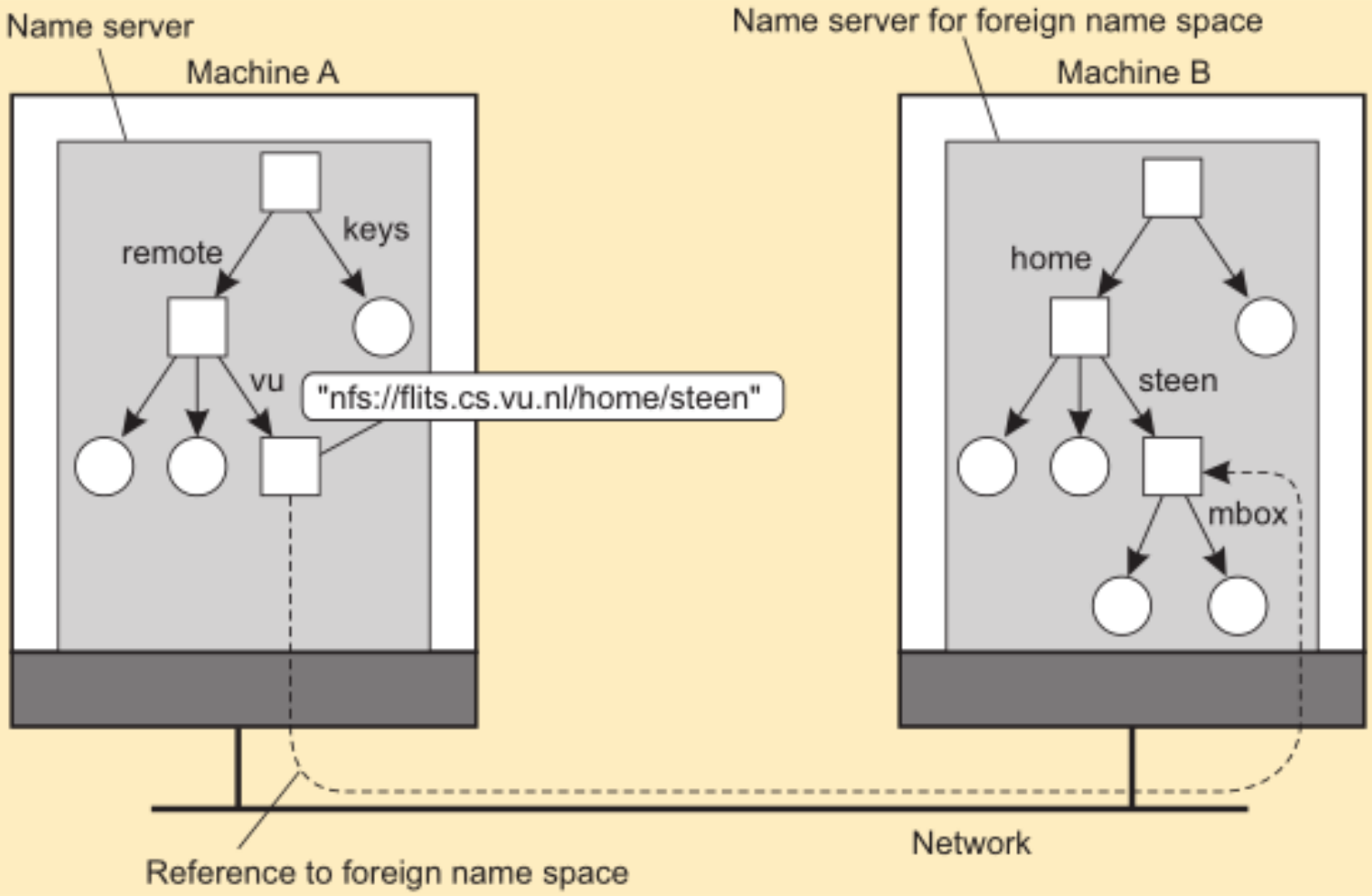
- Espaço de nomes estrangeiro: o espaço de nomes que queremos acessar;
- Ponto montado (mount point): o nó do espaço de nomes atual contendo o identificador do espaço de nomes estrangeiro;
- Ponto de montagem (mounting point): o nó de um espaço de nomes estrangeiro onde iremos continuar a resolução de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

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



IMPLEMENTAÇÃO DE ESPAÇO DE NOMES

QUESTÃO BÁSICA

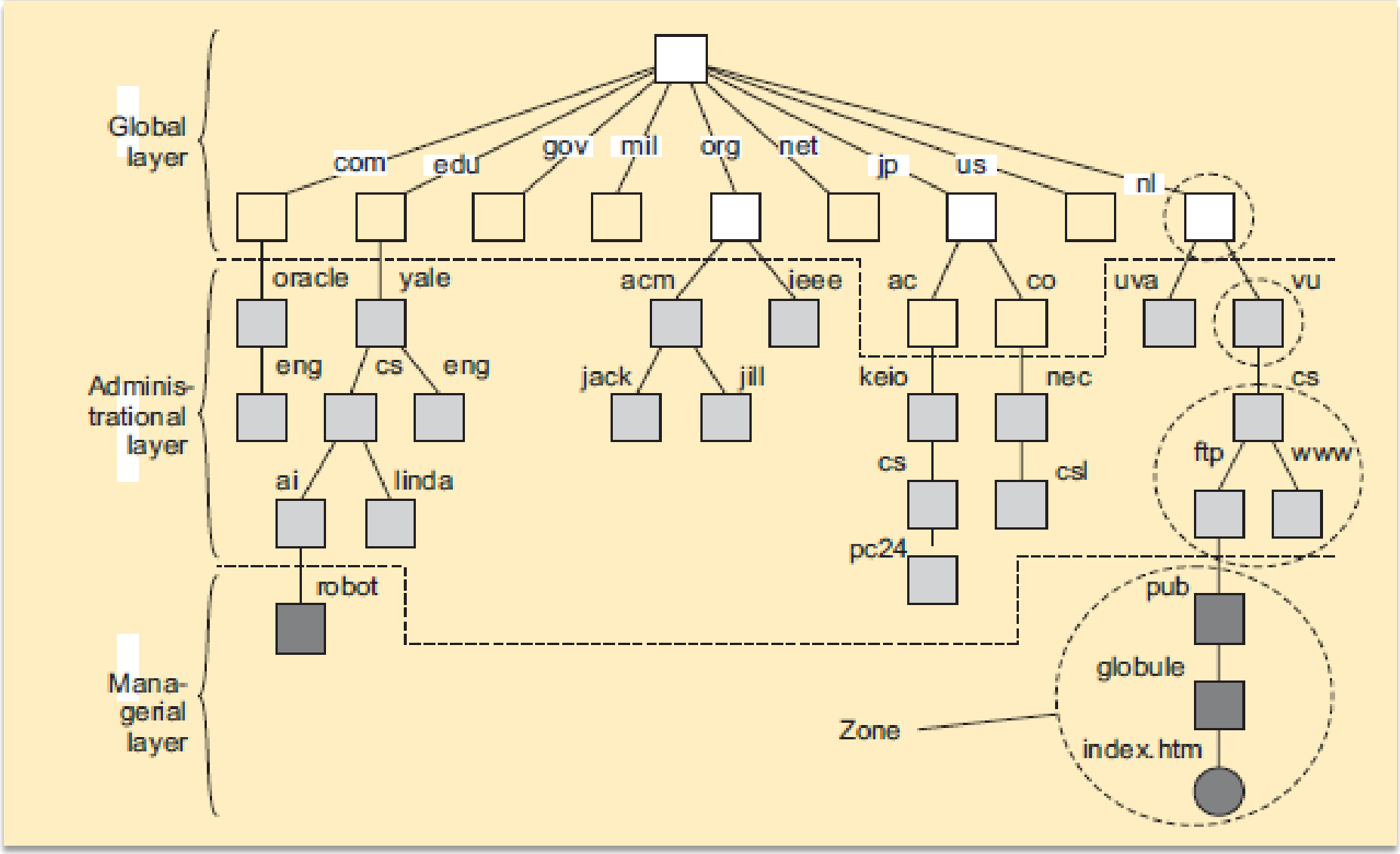
O processo de resolução de nomes distribuído bem como o gerenciamento do espaço de nomes através de máquinas múltiplas, através da distribuição de nós no grafo de nomes

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 adminsitrativo: 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 de uma única administração. A questão principal é o mapeamento efetivo de nós de diretório a servidores de nomes local.

IMPLEMENTAÇÃO DE ESPAÇO DE NOMES

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



IMPLEMENTAÇÃO DE ESPAÇO DE NOMES

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

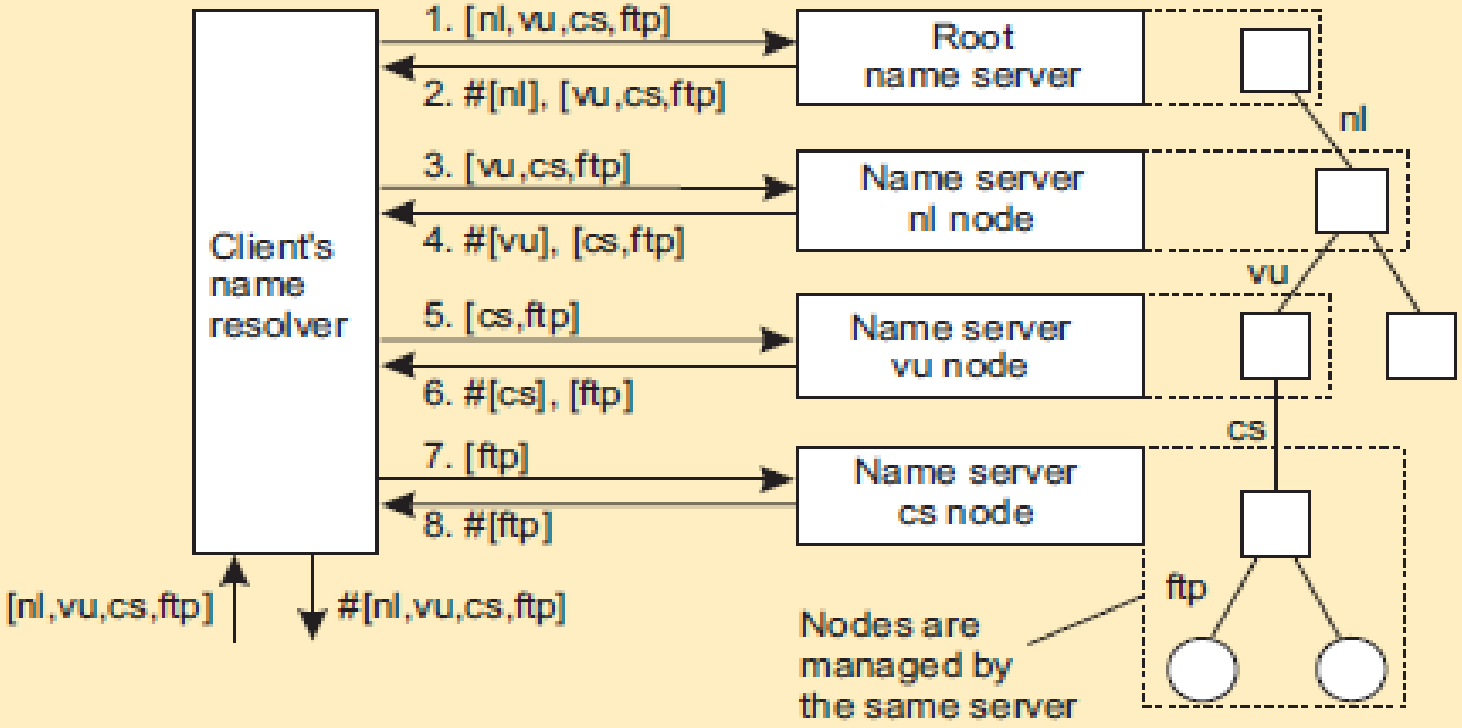
Item	Global	Administrational	Managerial
1	Worldwide	Organization	Department
2	Few	Many	Vast numbers
3	Seconds	Milliseconds	Immediate
4	Lazy	Immediate	Immediate
5	Many	None or few	None
6	Yes	Yes	Sometimes

1: Geographical scale 2: # Nodes 3: Responsiveness	4: Update propagation 5: # Replicas 6: Client-side caching?

RESOLUÇÃO INTERATIVA DE NOMES

PRINCÍPIO

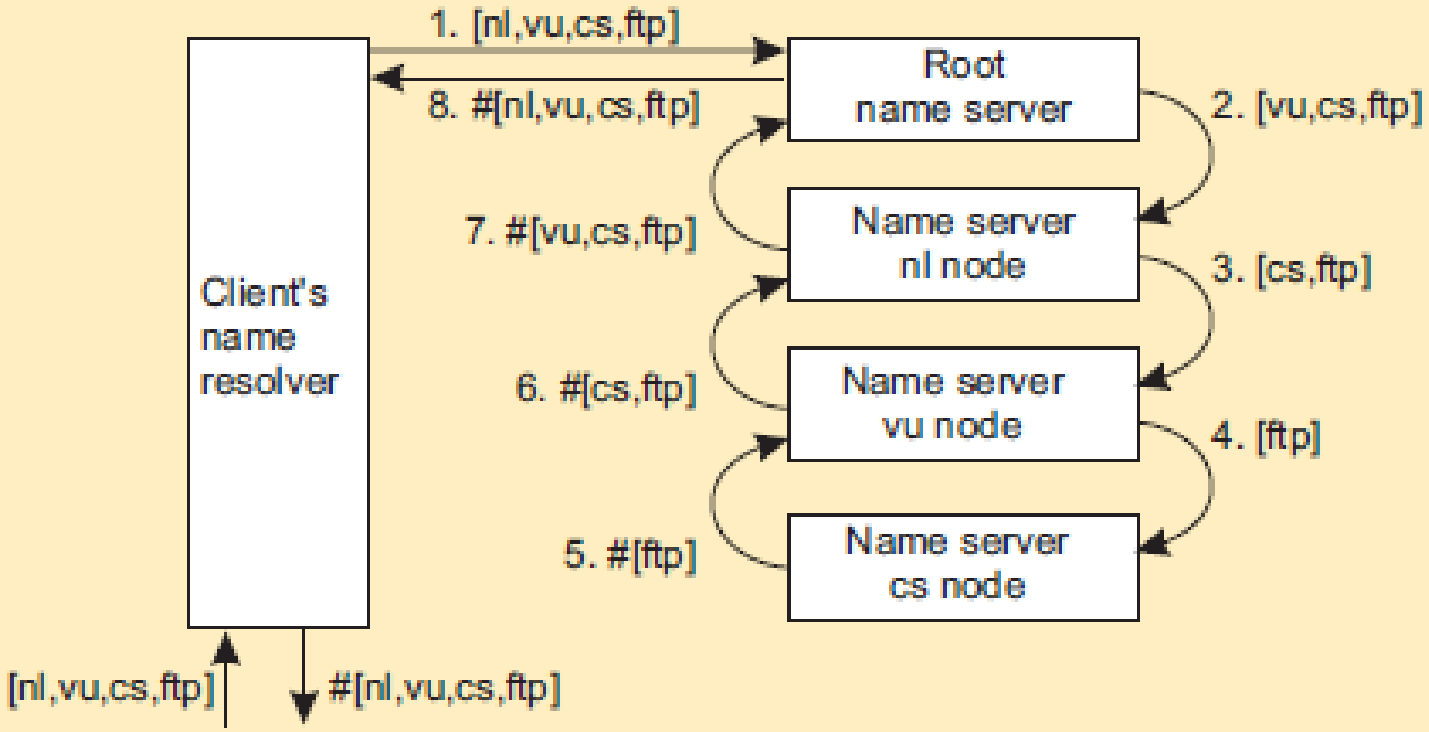
1. *resolve(dir, [nome$_1$, ... , nome$_k$])* enviado para *servidor$_0$* responsável por *dir*
2. *Servidor$_0$* resolve *resolve(dir, nome)* → *dir$_1$* , retornando o identificador (endereço) do *servidor$_1$* , que armazena *dir$_1$* .
3. Cliente envia *resolve(dir$_1$, [nome$_2$, ..., nome$_k$])*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



RESOLUÇÃO RECURSIVA DE NOMES

PRINCÍPIO

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



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*]

QUESTÕES DE ESCALABILIDADE

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

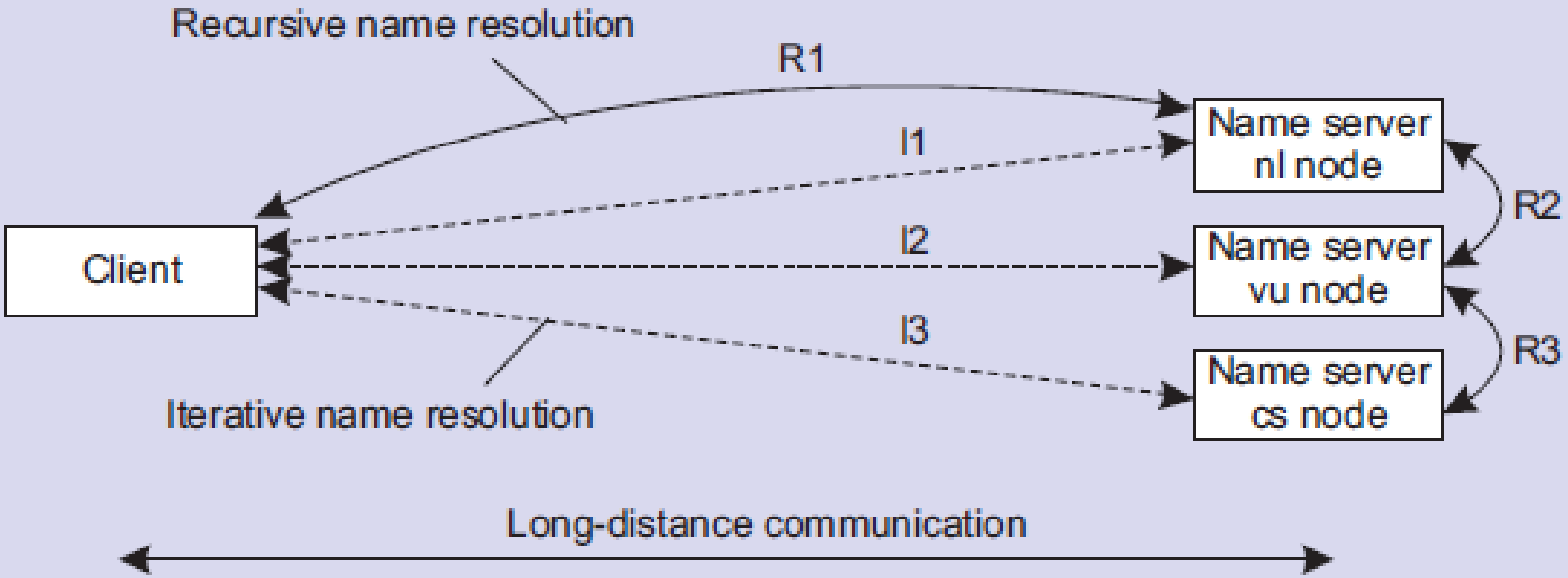
Assuma (pelo mesmo nos níveis global e administrativo) que o conteúdo de nós dificilmente muda. Podemos então aplicar replicação extensiva mapeando nós a múltiplos servidores e iniciar a resolução de nomes no servidor mais próximo.

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

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

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.

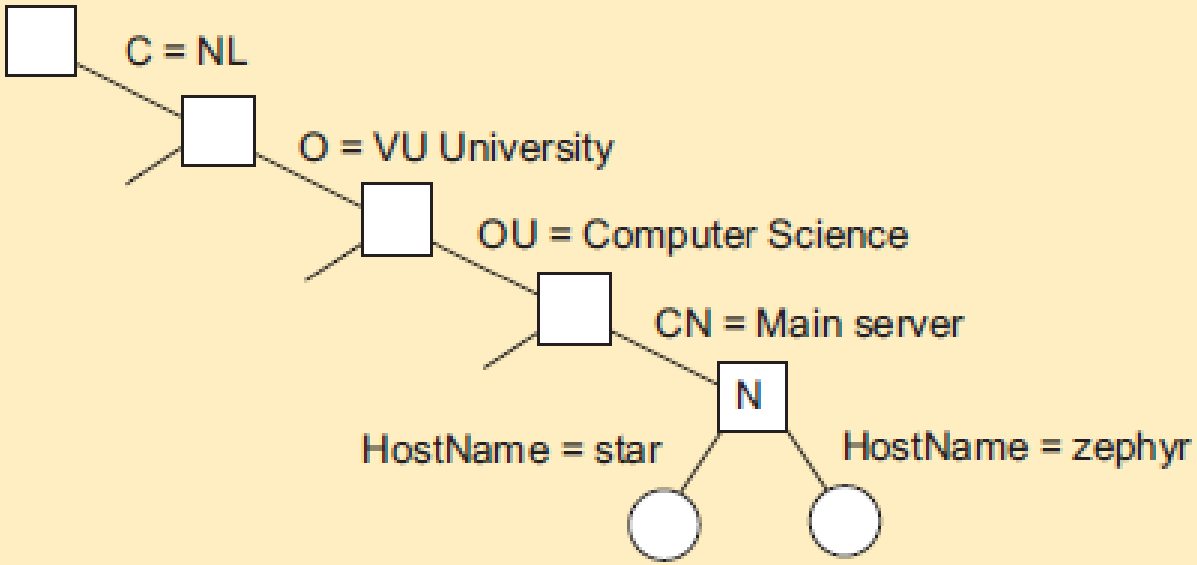
Attribute	Abbr.	Value
Country	*C*	NL
Locality	*L*	Amsterdam
Organization	*O*	VU University
OrganizationalUnit	*OU*	Computer Science
CommonName	*CN*	Main server
Mail_Servers	–	137.37.20.3, 130.37.24.6, 137.37.20.10
FTP_Server	–	130.37.20.20
WWW_Server	–	130.37.20.20

LDAP

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)'')`

ÍNDICE DISTRIBUÍDO

IDÉIA BÁSICA

- Assuma um conjunto de atributos $\{a^1, ..., a^N\}$
- Cada atributo a^k pega valores de um conjunto R^k
- Para cada atributo a^k associe um conjunto $S^k = \{S_1^k, \ldots, S_{n_k}^k\}$ *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

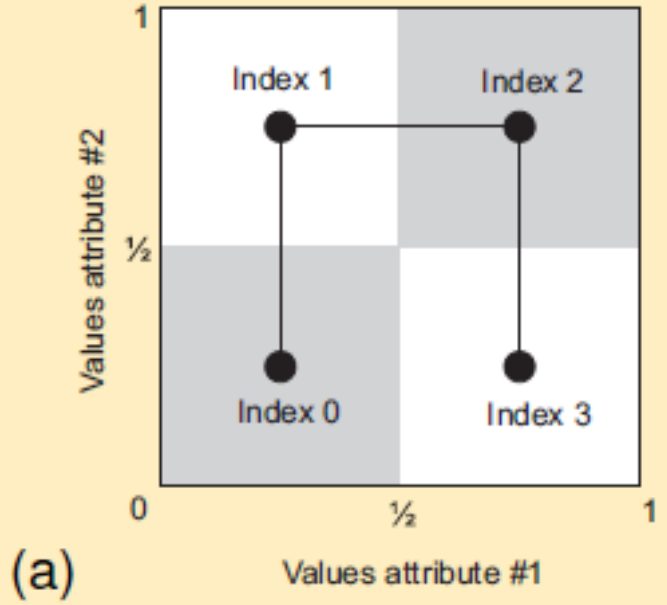
- Uma *query* envolvendo *k* atributos requer o contato em *k* servidores
- Imagine *looking* up "*lastName = Smith ∧ firstName = Pheriby*": o cliente pode precisar processar muitos arquivos porque existe muitas pessoas com nome "*Smith*"
- Não facilmente suportado para consulta de intervalos (range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

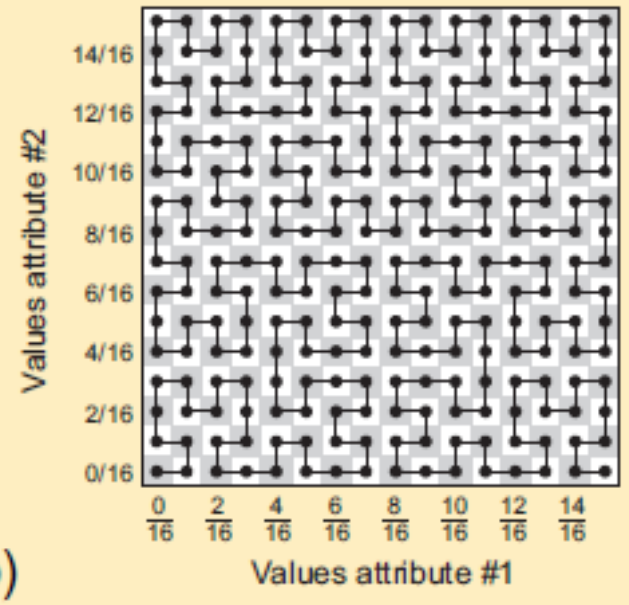
ALTERNATIVA: MAPEAR TODOS ATRIBUTOS EM 1 DIMENSÃO E ENTÃO INDEXAR

CURVAS DE PREENCHIMENTO DE ESPAÇO: PRINCÍPIO

1. Mapeie o espaço N dimensional coberto por N atributos $\{a^1, ..., a^N\}$ 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)*