# 05 - Nombres

## IIC2523 - Sistemas Distribuidos

Cristian Ruz – cruz@ing.puc.cl

Departamento de Ciencia de la Computación
Pontificia Universidad Católica de Chile

Semestre 2-2020

# Contenidos

- Esquemas jerárquicos
- Esquemas de nombres estructurados
- Nombres basados en atributos

# Necesidad de nombres

## Necesidad

- Nombres permiten dar acceso a entidades.
- Entidad poseen **access points**
- Para encontrar un *access point* se necesita una dirección
- En un ambiente distribuido las direcciones debe ser únicas

# Identificadores

**Requisitos de un identificador**

- Debe referirse, como máximo a una única entidad
- Cada entidad debe tener solamente un identificador
- Un identificador no puede ser reasignado

Algunos identificadores pueden incluso tener contenido

# Broadcasting

¿Cómo encontrar la entidad a partir del nombre?

**Hacer *broadcast* del ID**

- No escala bien, más allá de redes LAN (ej: ARP)
- Requiere que todos los procesos estén escuchando

# Forwarding pointers

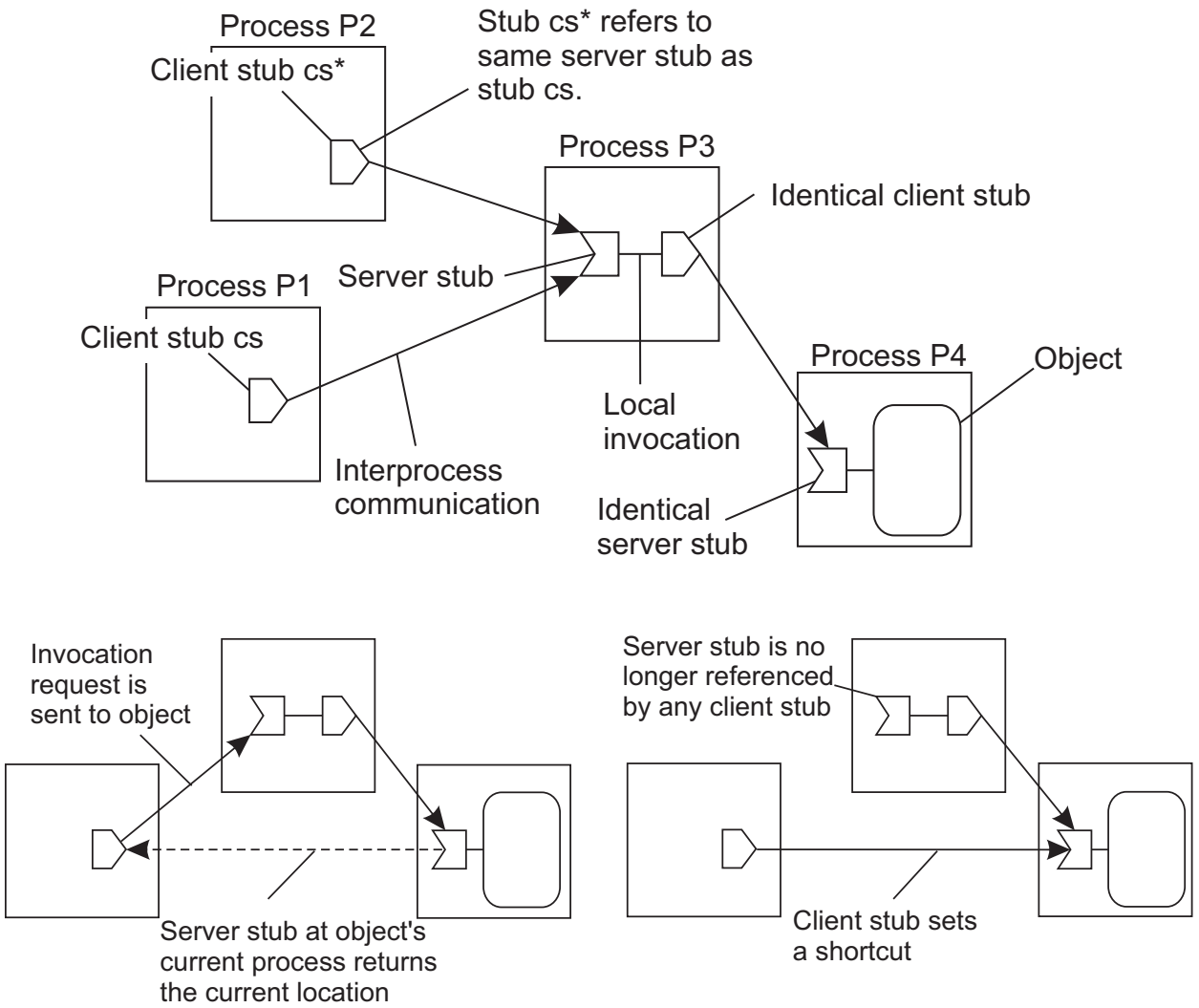
**¿Cómo hacerlo si una entidad cambia de ubicación?**

- Utilizar cadena de punteros
- Cuando una entidad cambia de ubicación, deja puntero a su nueva ubicación
- Una vez que se encuentra la ubicación nueva, se actualizan las referencias

Problemas:

- Tolerancia a fallas si hay un corte en la cadena
- Tiempo de la primera ubicación

# Ejemplo: SSP Chains



# Basado en *homes*

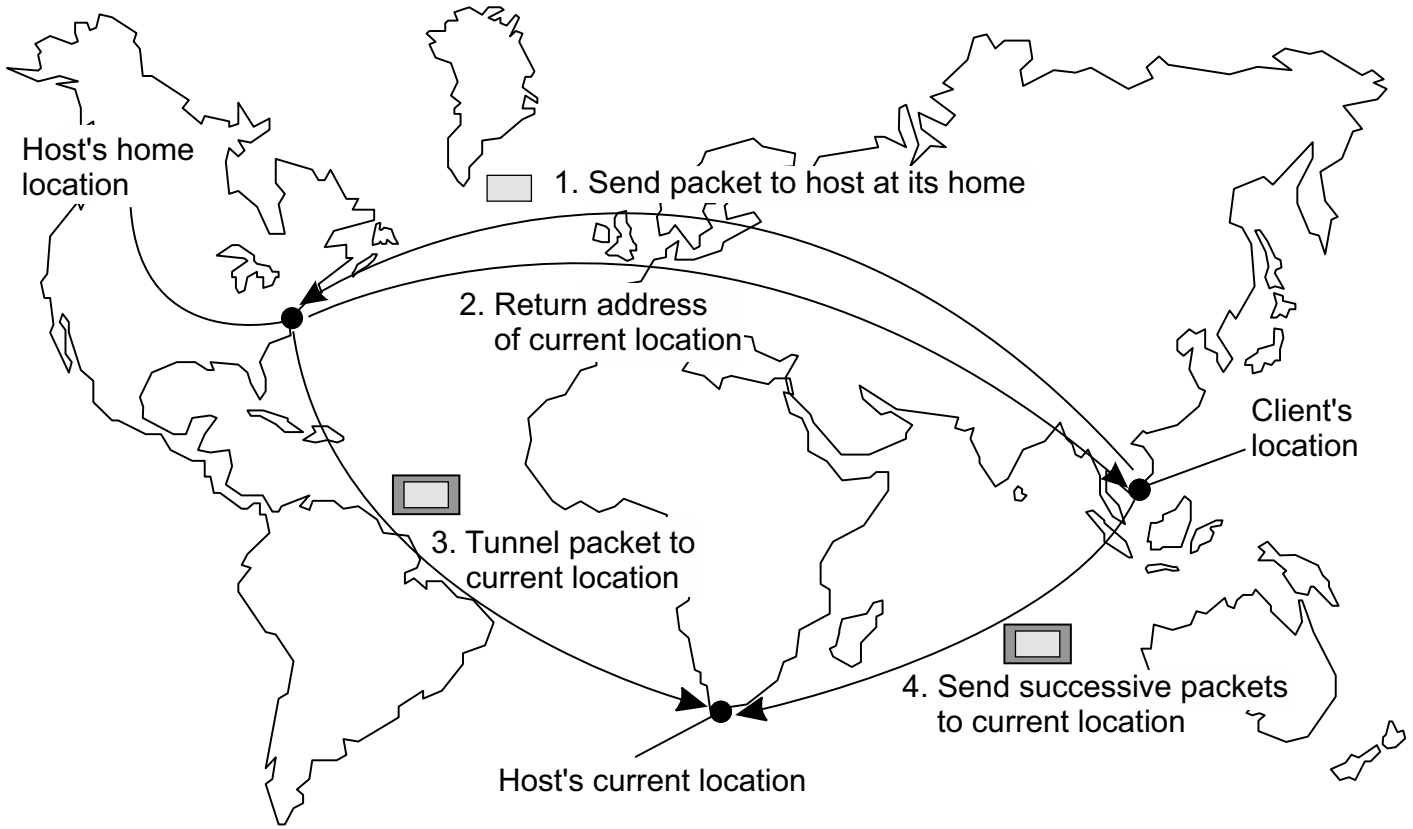
## Esquema basado en *homes*

Cada entidad posee una dirección *home* conocida o calculable por todos.

- Servicio de nombres puede mantener dirección de *homes* por entidad
- Dirección actual (real) es mantenida por el *home*
- Siempre el primer contacto es a través del *home*

# Basado en *homes*

## Mobile IP



# Basado en *homes*

**Desventajas**

- Se necesita un registro **permanente** de *homes*
- Ineficiente si la ubicación real es geográficamente cercana
- Si la entidad se mueve permanentemente, no permite actualizar los *homes*

Posible solución: usar (meta) esquema alternativo en nivel superior

# Distributed Hash Tables: Chord

## Organización en anillo lógico

- Cada nodo posse un identificador de $m$-bit
- Cada entidad posee una *key* de $m$-bit
- Entidad con *key* $k$ se encuentra bajo la "jurisdicción" de nodo con menor $id \geq k$, llamado **sucesor** succ($k$)

Cada noodo mantiene registro de su vecino, e inicia búsqueda lineal a lo largo del anillo.
¿Sirve como solución?

# Distributed Hash Tables: Chord

## Principio de DHT

- Cada nodo $p$ mantiene una **finger table** $FT_p[]$ con un máximo de $m$ entradas, donde:

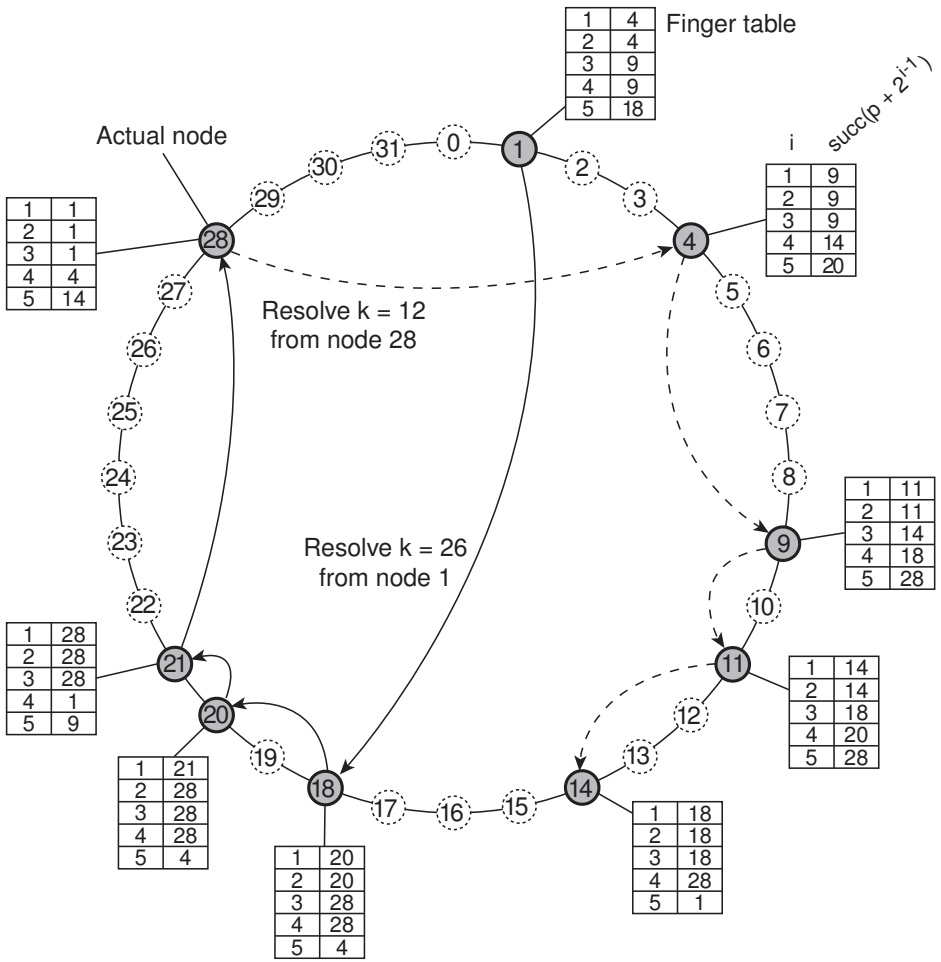
$$FT_p[i] = \text{succ}(p + 2^{i-1})$$

- Para encontrar una *key*, el nodo $p$ reenvía la solicitud al nodo $q$ que está en su entrada $j$ tal que:

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

- Si $p < k < FT_p[1]$, la solicitud también se envía a $FT_p[1]$

# Distributed Hash Tables: Chord




Búsqueda de *key* 26 desde 1, y *key* 12 desde 28.

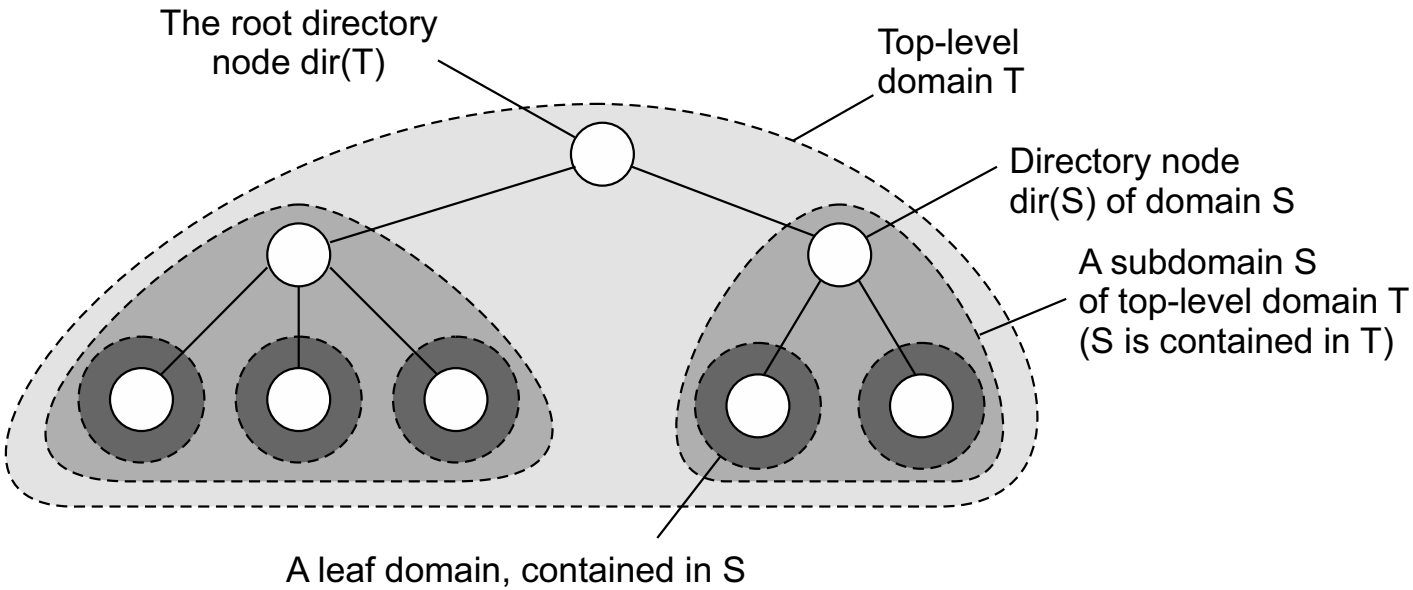
# Distributed Hash Tables: Chord

```
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])
```

# Hierarchical Location Services (HLS)

## Hierarchical Location Services (HLS)

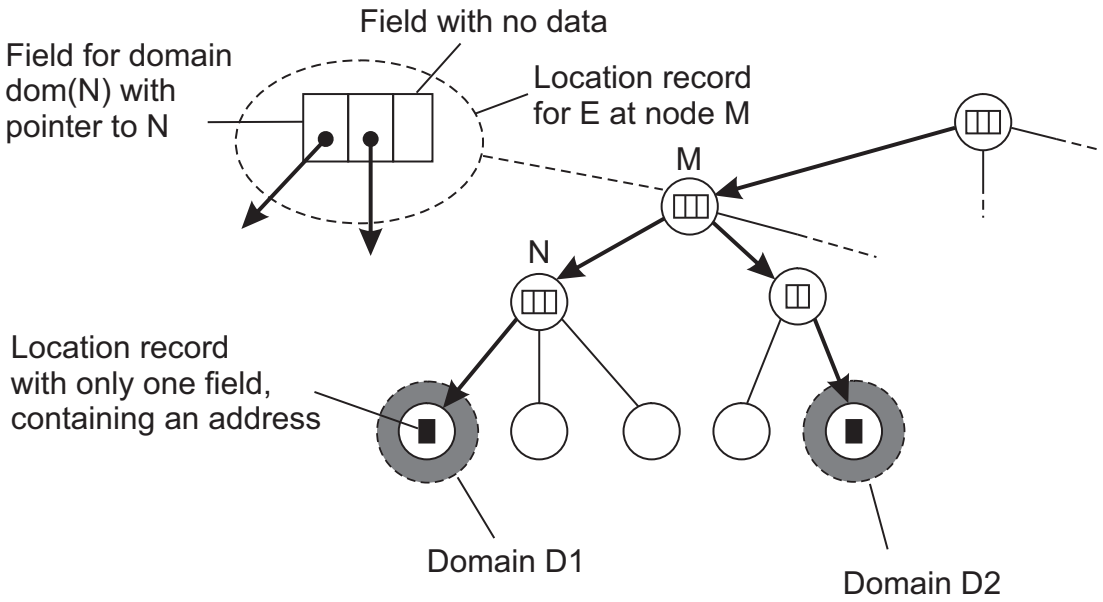
Constuir un árbol para guiar la búsqueda mediante dominios jerárquicos. Cada *root* es un "administrador de directorio".



# Hierarchical Location Services (HLS)

## Invariantes

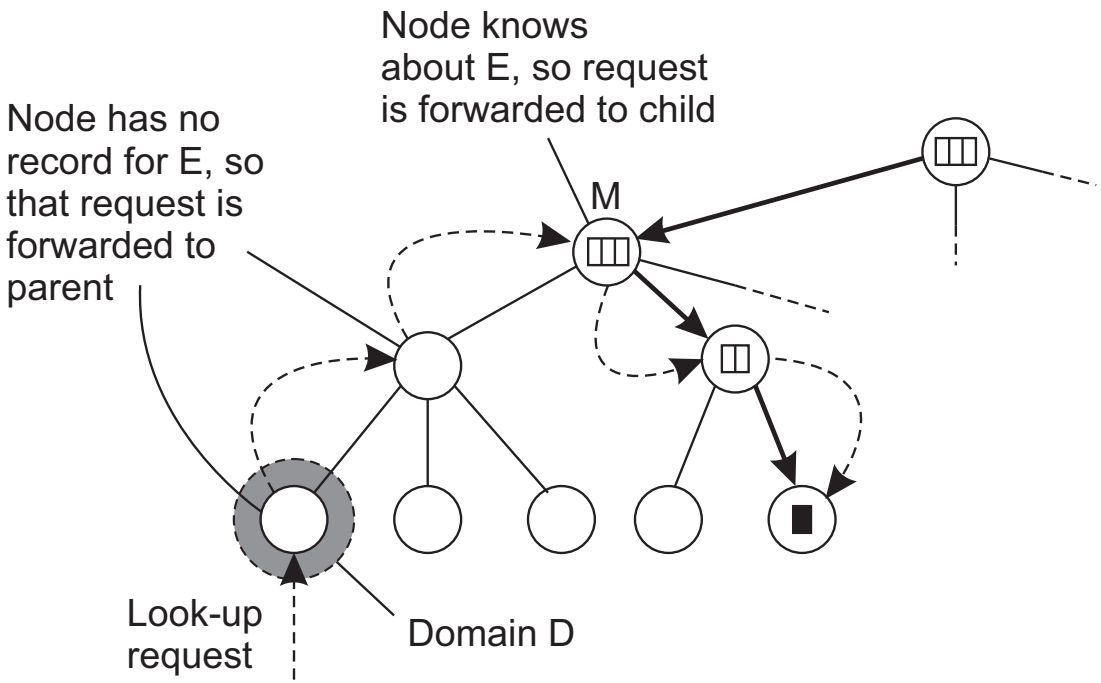
- La dirección de la entidad *E* se almacena en una hoja o nodo intermedio
- Nodos intermedios tienen punteros a hijos sí y solo ese subárbol contiene la dirección de la entidad
- El *root* conoce todas las entidades



# Hierarchical Location Services (HLS)

## HLS *Lookup*

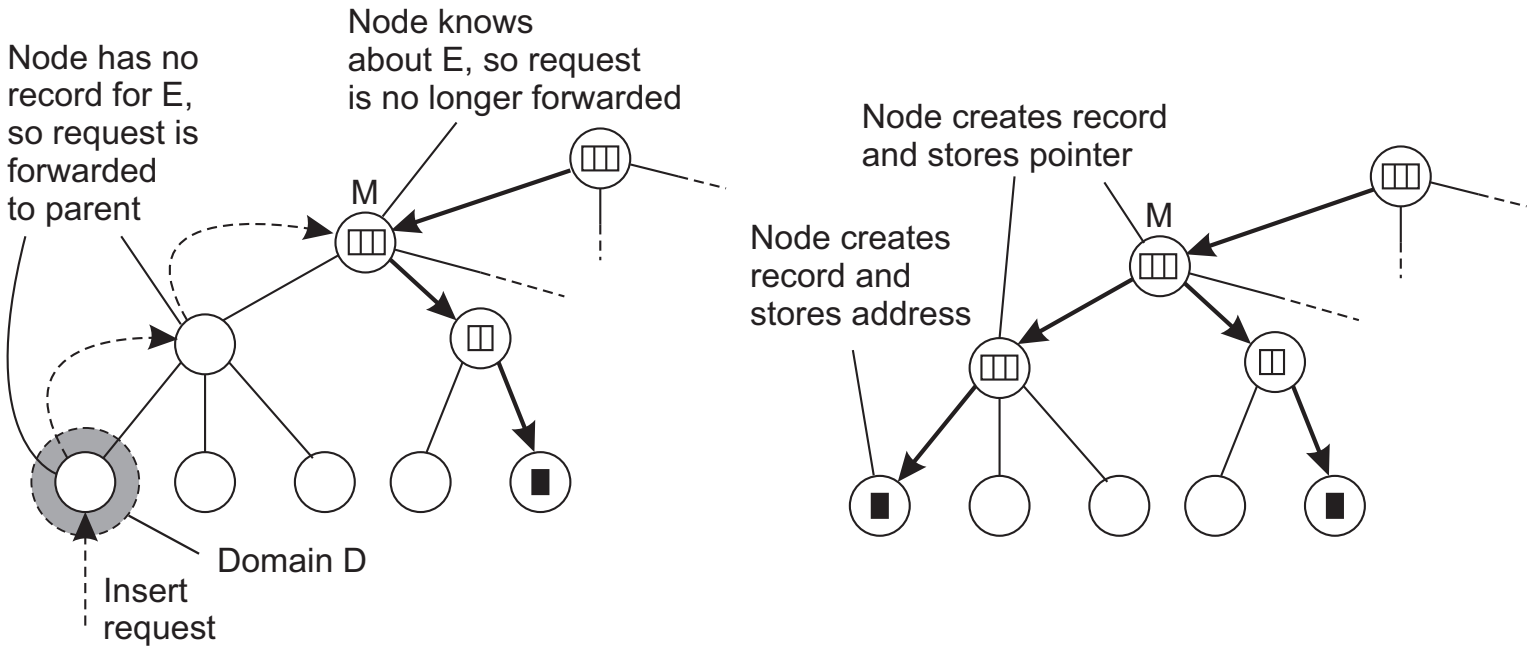
- Empezar búsqueda en la hoja local
- Si el nodo conoce acerca de $E$, entonces descender por el puntero.
- Si el nodo conoce acerca de $E$, entonces ascender
- Límite superior es el *root*



# Hierarchical Location Services (HLS)

## HLS *Insert*

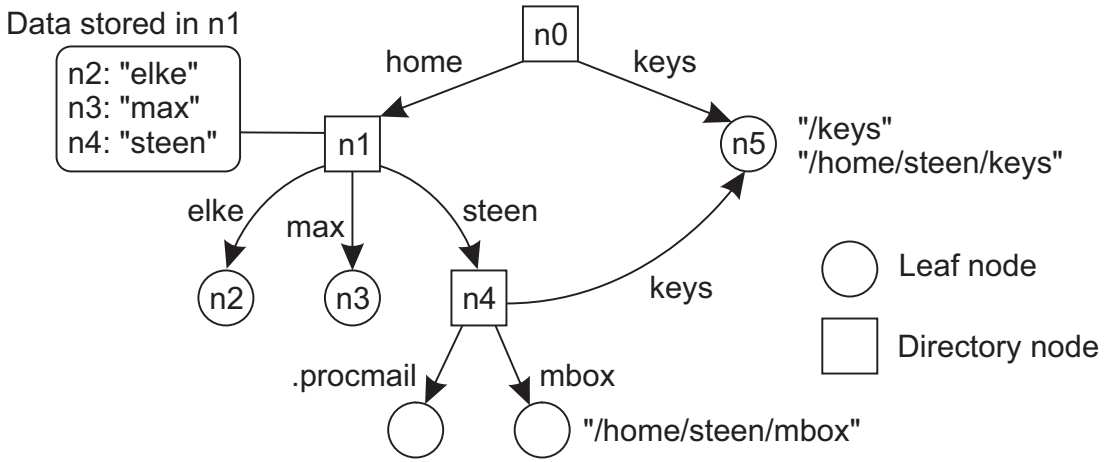
- Solicitud de *insert* se dirige al primer nodo que sepa acerca de *E*
- Se crea una cadena de puntero al nodo hoja



# Espacios de nombres

## Grafos de nombres

Nodos hojas representan una **entidad con nombres**
Un nodo **directorio** apunta a otros nodos.




Nodo directorio contiene una tabla de pares $\langle \text{nodeID}, \text{edgeLabel} \rangle$
Nodos pueden almacenar otros **atributos**: dirección física, tipos, alias, etc

# Resolución de nombres

## Problema

Para resolver un nombre, necesitamos un nodo directorio.
¿Cómo encontrarlo?

## Mecanismo de clausura: ¿cómo encontrar nodo inicial?

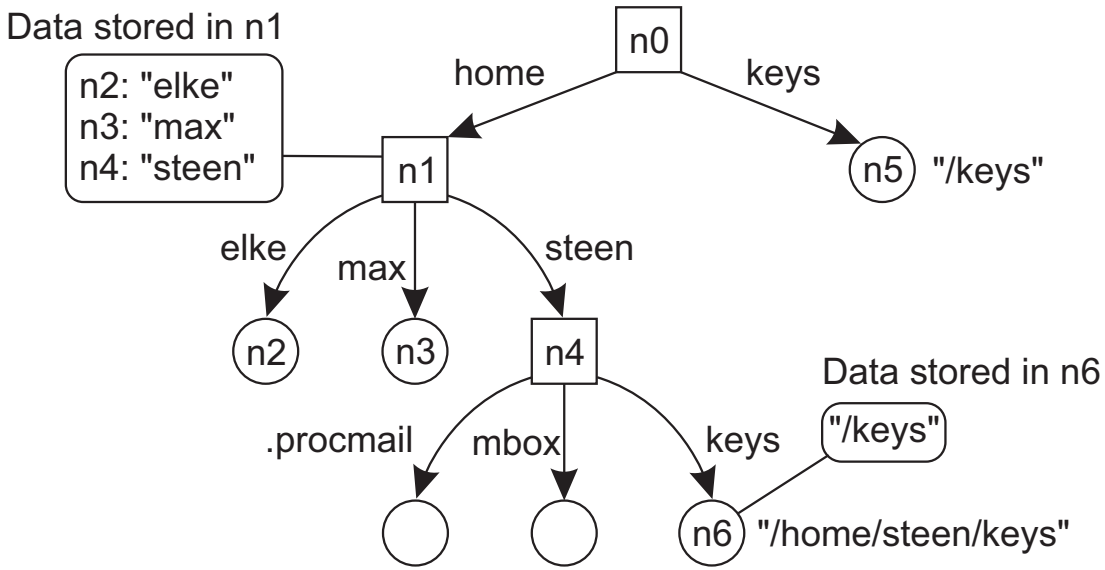
Mecanismo predefinido de acuerdo al contexto
Ejemplos:

- Si es `www.ing.puc.cl`, buscar mediante DNS
- Si es `/home/cruz/T1`, buscar el directorio raíz en el sistema de archivos
- Si es `+56 9 88123465`, utilizar marcador telefónico
- Si es `146.155.13.45`, buscar dirección IP

# *Links*

## Tipos de *links*

- **Hard link**. Apunta al mismo nodo del destino (al contenido).
- **Soft/symbolic link**. Apunta al mismo nombre del destino.



# *Mount*

## Problema

Mezclar los contenidos de **espacios de nombres**, sin perder las referencias en los contextos originales.

## Mounting

Permite asociar un identificador con un nodo de otro **espacio de nombres**. Permite **transparencia de acceso**

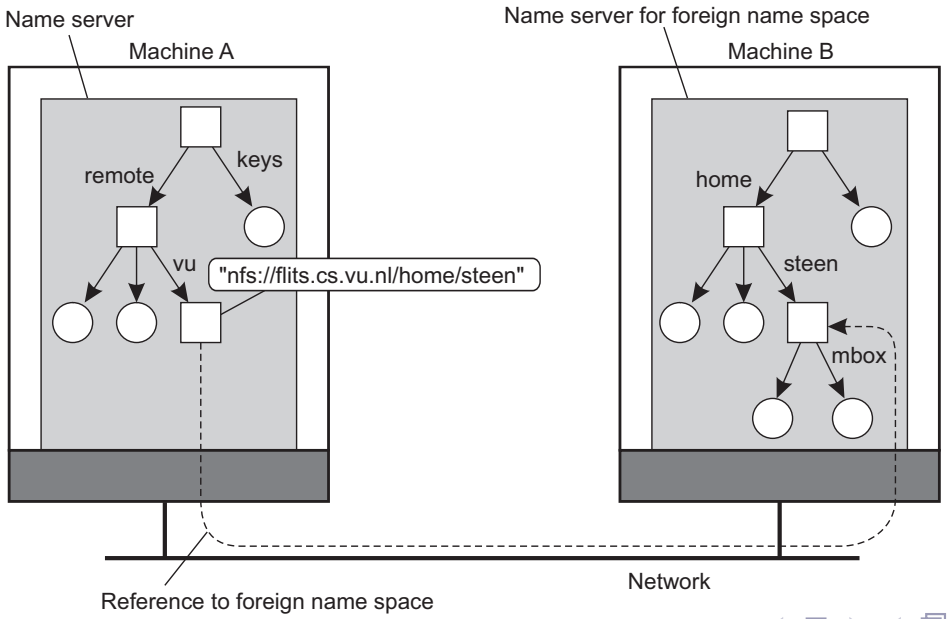
- **Foreign name space**: espacio que se desea acceder
- **Mount point**: nodo del espacio actual que contiene al identificar del nodo en el *foreign name space*
- **Mounting point**: nodo del *foreign name space* donde se continúa la resolución

# *Mount*

## Mounting

Permite asociar un identificador con un nodo de otro **espacio de nombres**. Permite **transparencia de acceso**

- **Foreign name space**: espacio que se desea acceder
- **Mount point**: nodo del espacio actual que contiene al identificar del nodo en el *foreign name space*
- **Mounting point**: nodo del *foreign name space* donde se continúa la resolución



# Implementación de espacios de nombres

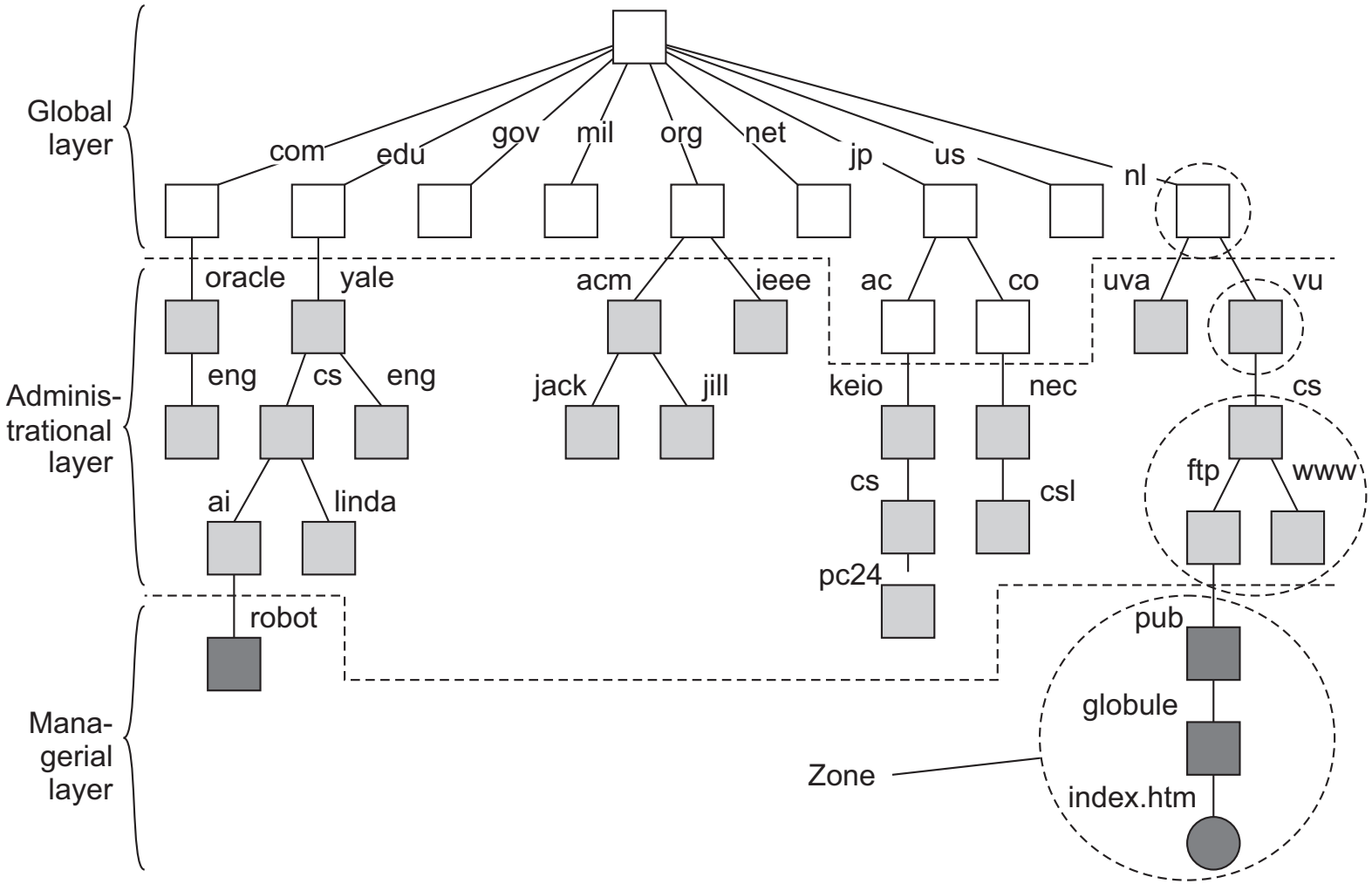
## Idea

Distribuir el proceso de resolución particionando el grafo

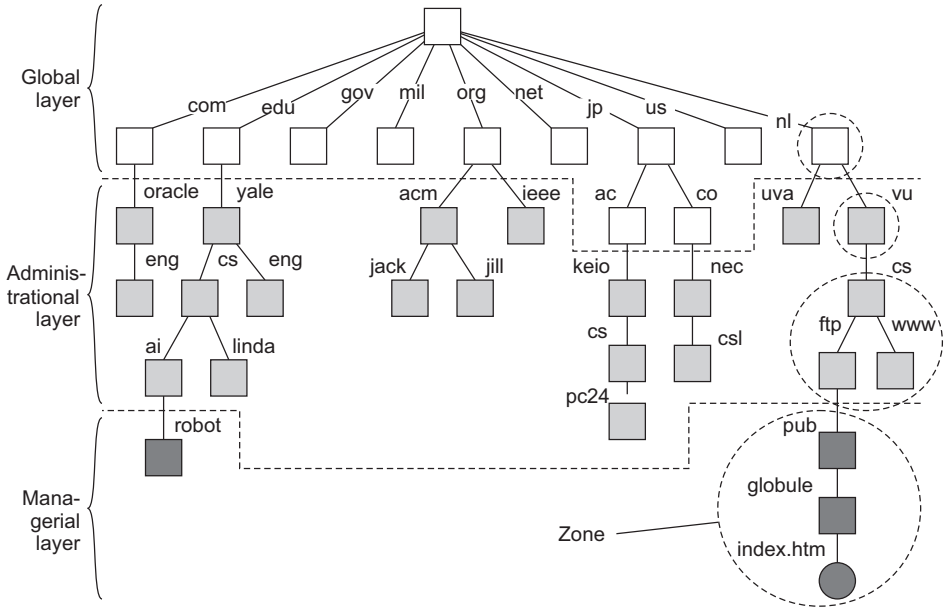
## Niveles

- **Nivel global**: nodos directorio de alto nivel
- **Nivel administrational**: administran dominios separados
- **Nivel managerial**: nodos de bajo nivel que contienen *mappings* a servidores de nombres.

# Implementación de espacios de nombres



# Implementación de espacios de nombres



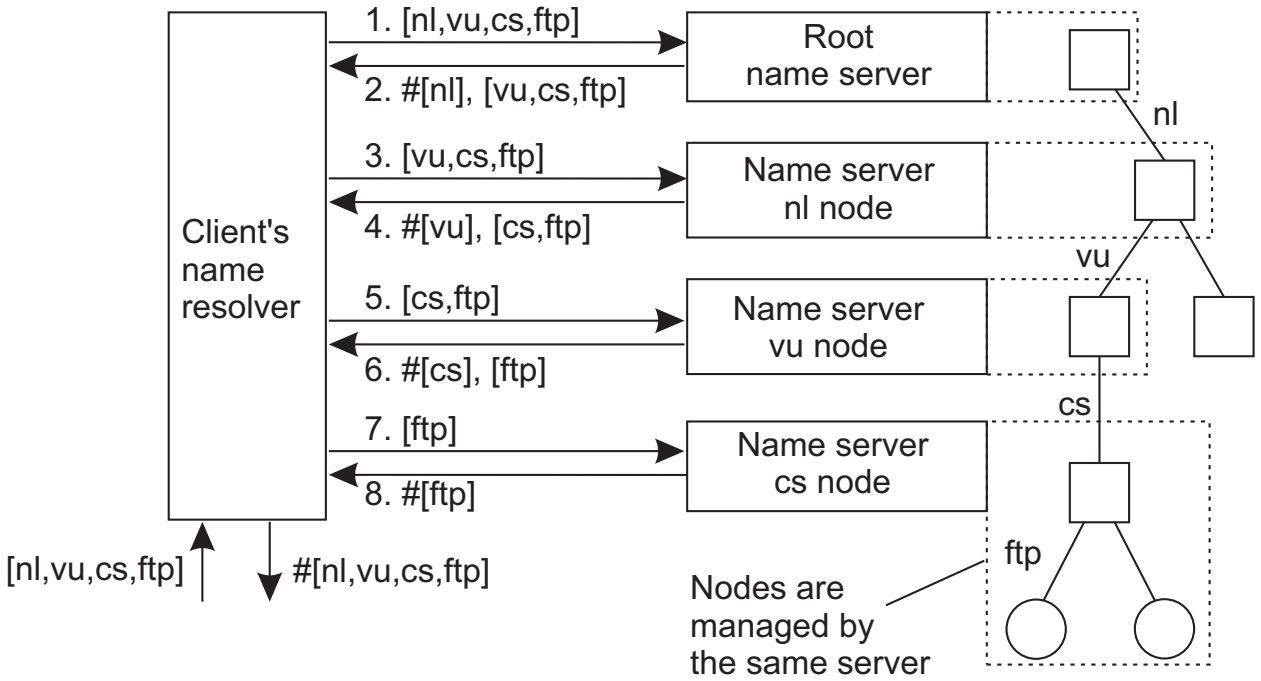

| Item | Global | Administrational | Managerial |
|---|---|---|---|
| 1 | Worldwide | Organization | Department |
| 2 | Few | Many | Vast numbers |
| 3 | Seconds | Milliseconds | Immediate |
| 4 | Lazy | Immediate | Immediate |
| 5 | Many | None or few | None |
| 6 | Yes | Yes | Sometimes |

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

# Resolución iterativa

## Resolución iterativa

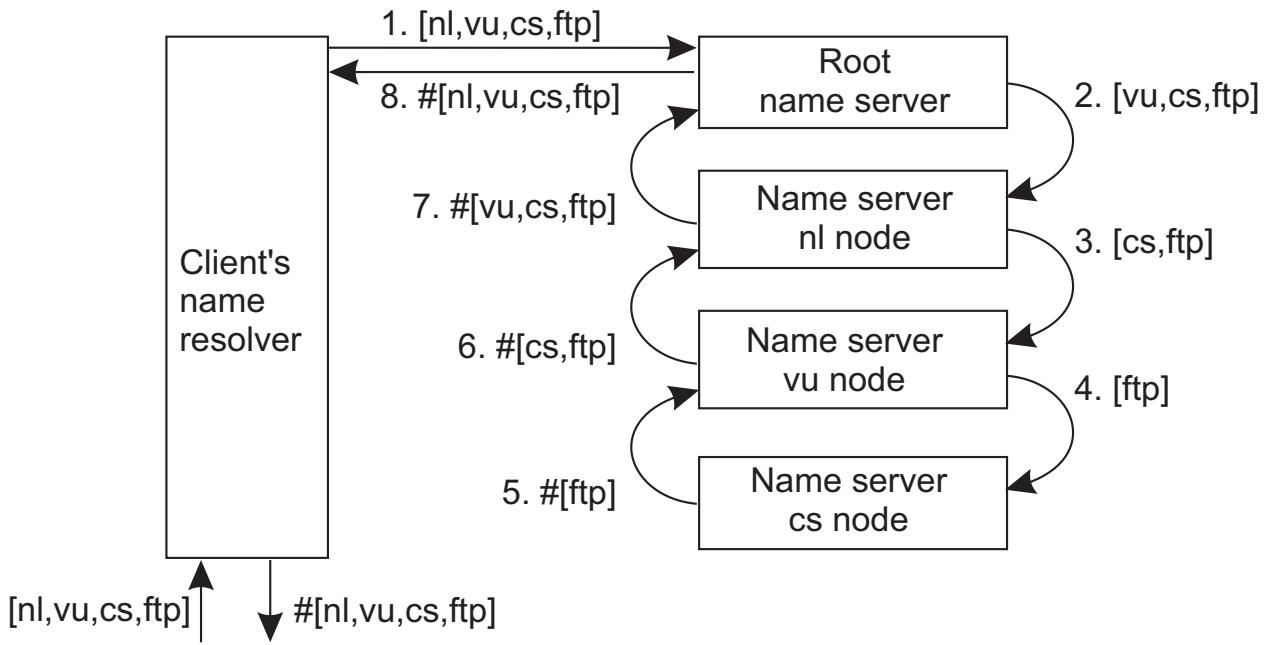
1. $resolve(dir, [name_1, \ldots, name_k])$ se envía a $Server_0$, responsable de $dir$
2. $Server_0$ resuelve $resolve(dir, name_1) \rightarrow dir_1$, con la identificación de $Server_1$, responsable de $dir_1$
3. Cliente envía $resolve(dir_1, [name_2, \ldots, name_k])$ se envía a $Server_1$, etc



# Resolución recursiva

## Resolución iterativa

1. $resolve(dir, [name_1, \ldots, name_k])$ se envía a $Server_0$, responsable de $dir$
2. $Server_0$ resuelve $resolve(dir, name_1) \rightarrow dir_1$, y envía $resolve(dir_1, [name_2, \ldots, name_k])$ se envía a $Server_1$, responsable de $dir_1$ responsable de $dir_1$
3. $Server_0$ espera respuesta de $Server_1$ y retorna respuesta al cliente



# DNS

## Implementación de espacio estructura jerárquico

- Cada nodo posee solamente un eje de entrada
- Dominio: subárbol
- Nombre de dominio: ruta hasta la raíz

Información por nodo:

| 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 |

# Esquemas basados en atributos

## Esquemas basados en atributos

**Servicios de directorio**. Búsqueda de elementos de acuerdo a uno más atributos.

Pero, ¿cómo buscar sin tener que recorrer todos los miembros?

# Esquemas basados en atributos: LDAP

Mezcla de sistemas basados en servicio de directorio (atributos), y esquema estructurado.

## *Lightweight Directorio Access Protocol* (LDAP)

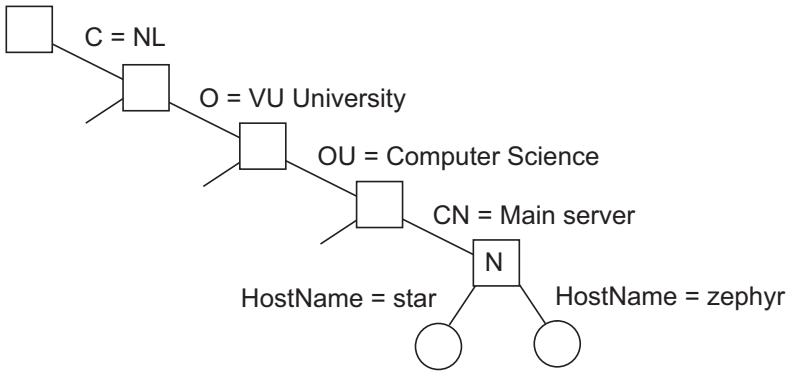
Cada entrada de directorio consiste de pares $\langle$atributo, valor$\rangle$, y posee un nombre único para facilitar las búsquedas.

| 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 |

# Esquemas basados en atributos: LDAP

## Elementos LDAP

- **Directory Information Base**. Conjunto de todas las entradas de directorio en LDAP
- **Relative Distinguished Name (RDN)**. Secuencia de *naming attributes*
- **Directory Information Tree**. Grafo de LDAP donde cada nodo es una entrada de directorio.




| 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* |

# Índices descentralizados

Atributos distribuidos entre servidores

## Atributos descentralizados

- Conjunto de $N$ atributos: $\{a^1, a^2, \ldots, a^N\}$
- Cada atributo $a^k$ toma su valor desde un conjunto $R^k$
- Cada atributo $a^k$ tiene asociado un conjunto de $n_k$ servidores $S^k = \{S_1^k, S_2^k, \ldots, S_{n_k}^k\}$
- Servidor para atributo $a_k$ contiene conjunto de pares $\langle E, \text{val} \rangle$. El servidor referencia a todas las entidades que poseen valor val para el atributo $a_k$.

Se construye en *mapping* global $F$ de atributos y valores a servidores:

$$F(a^k, v) = S_j^k, \text{con} S_j^k \in S, v \in R^k$$

Para hacer una consulta por todas las entidades que cumplen $a^1 = v^1, a^2 = v^2, \ldots, a^N = v^N$, se debe hacer $N$ consultas, una a cada servidor, y luego procesar las entidades resultantes.

# Índices descentralizados

Si llamamos $L(a^k, v)$ al conjunto de respuesta que entrega el servidor $S^k$, entonces una consulta expresada como:

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

puede calcularse como:

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

## Desventajas

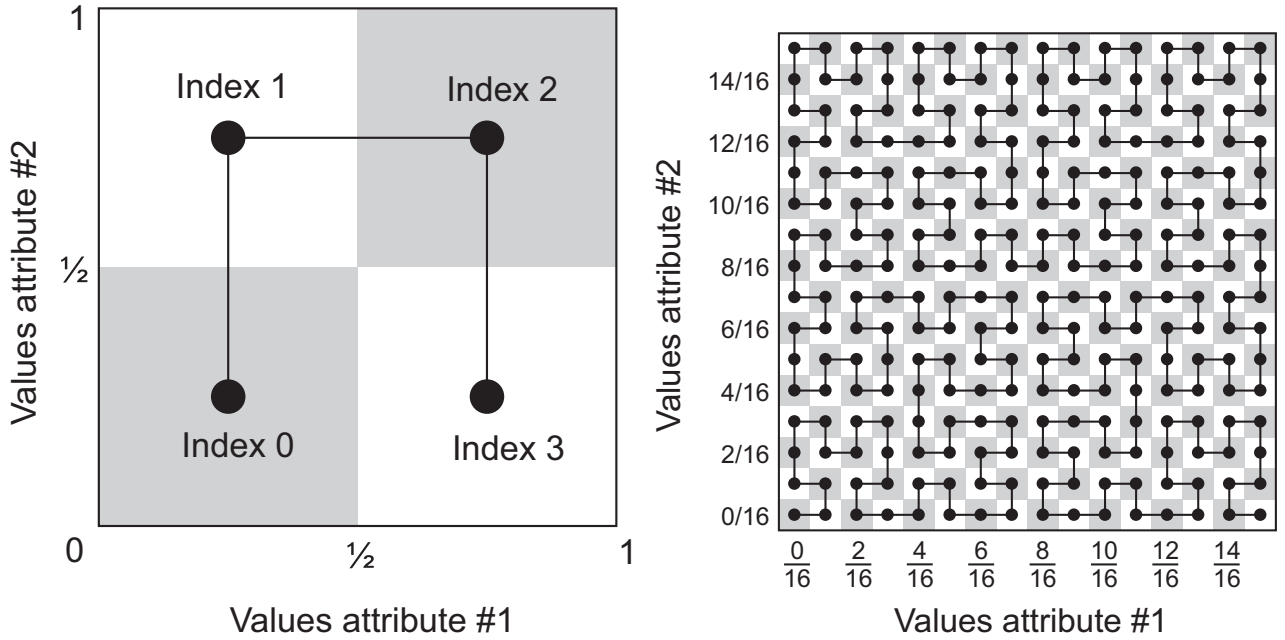
- Una consulta con $k$ atributos se convierte en consultas a $k$ servidores
- Algunas respuestas pueden ser muy grandes (ej, buscar *firstName* = Eleuterio, *lastName* = Rojas), y no pueden ser previamente filtradas en el servidor.

# *Space-filling curves*

## Idea

- Mapear un conjunto de atributes $N$-dimensional, a una sola dimensión
- Espacio se divide entre *index-servers* de manera unidimensional

Ejemplo con dos atributos en el rango $[0, 1]$




Cada índice está asociado a un *index server*.

*Index servers* están conectados por cercanía en el espacio multidimensional.