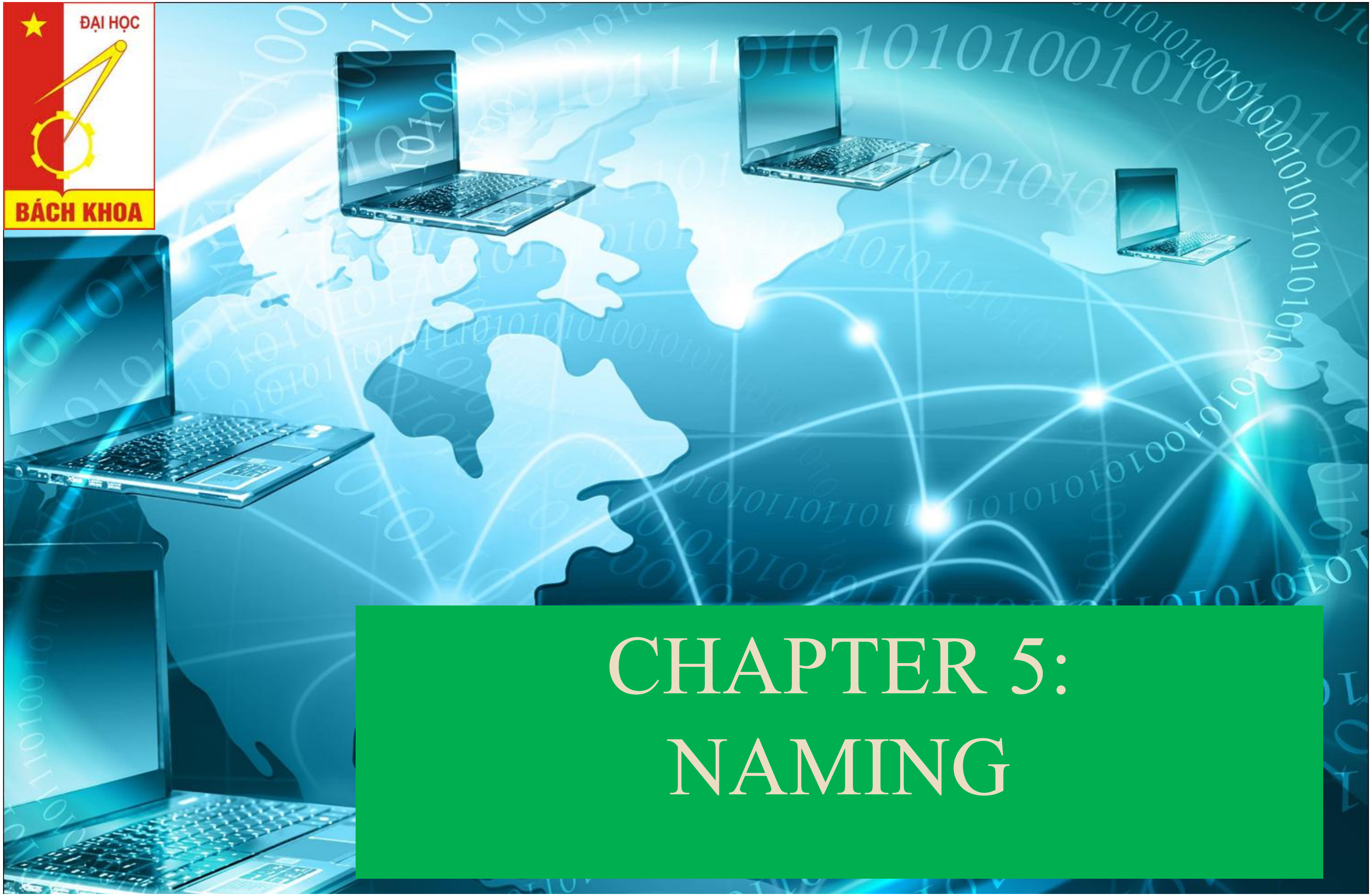


# CHAPTER 5: NAMING

DR. TRẦN HẢI ANH

# Outline

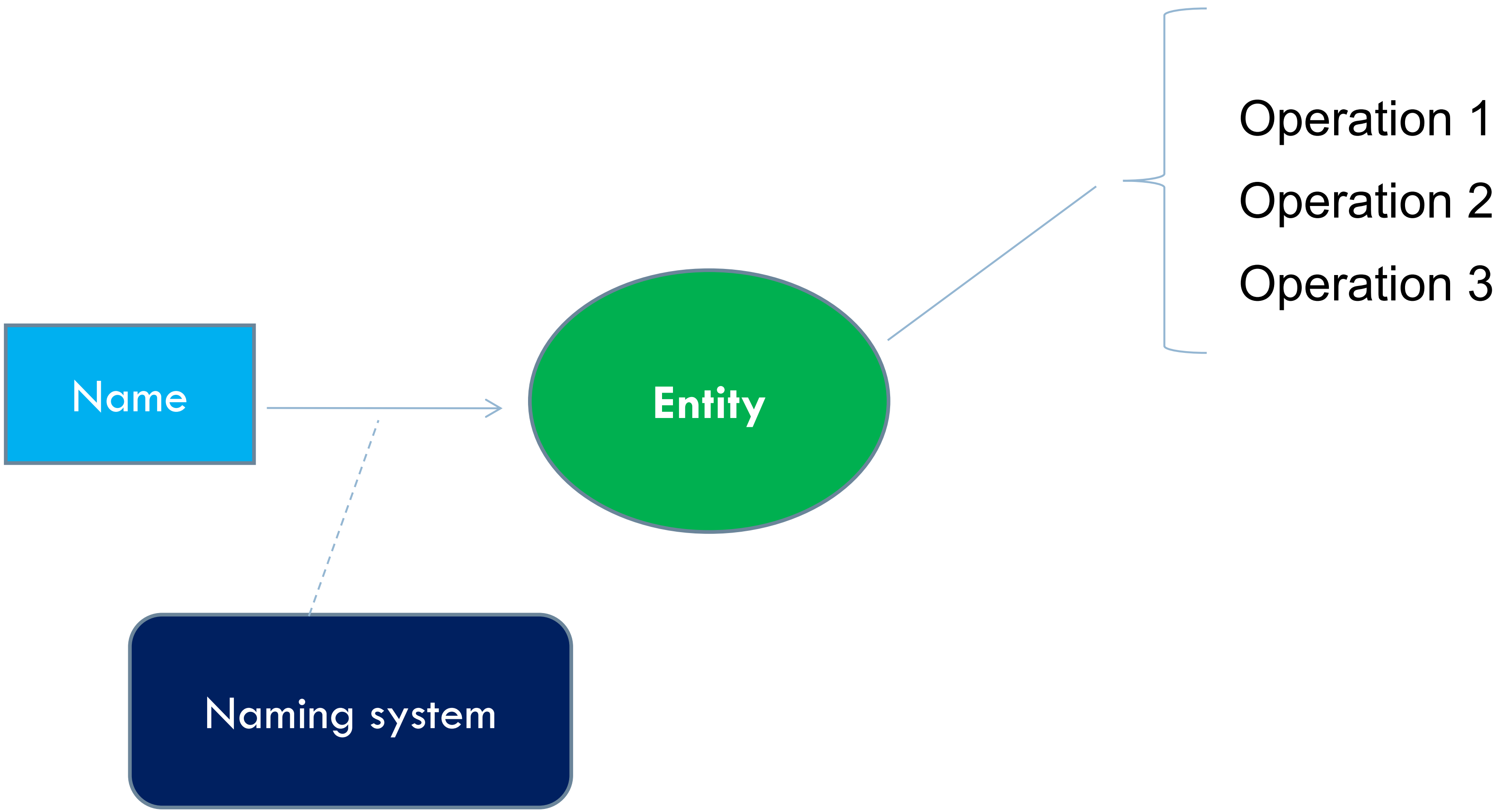


1. **Names. Identifiers and Address**
2. **Flat Naming**
3. **Structured Naming**

# 1. Names. Identifiers and Address

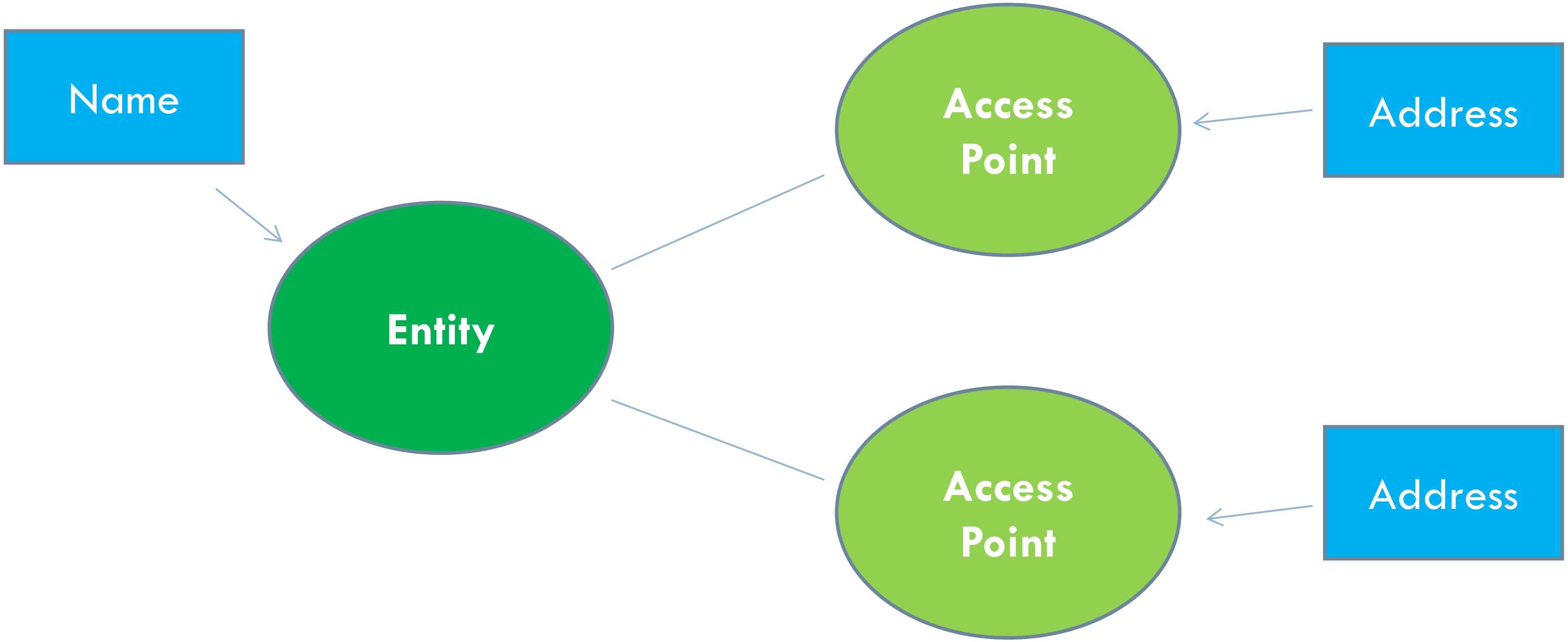
# Entity & Name




Operation 1
Operation 2
Operation 3
Name
Entity
Naming system

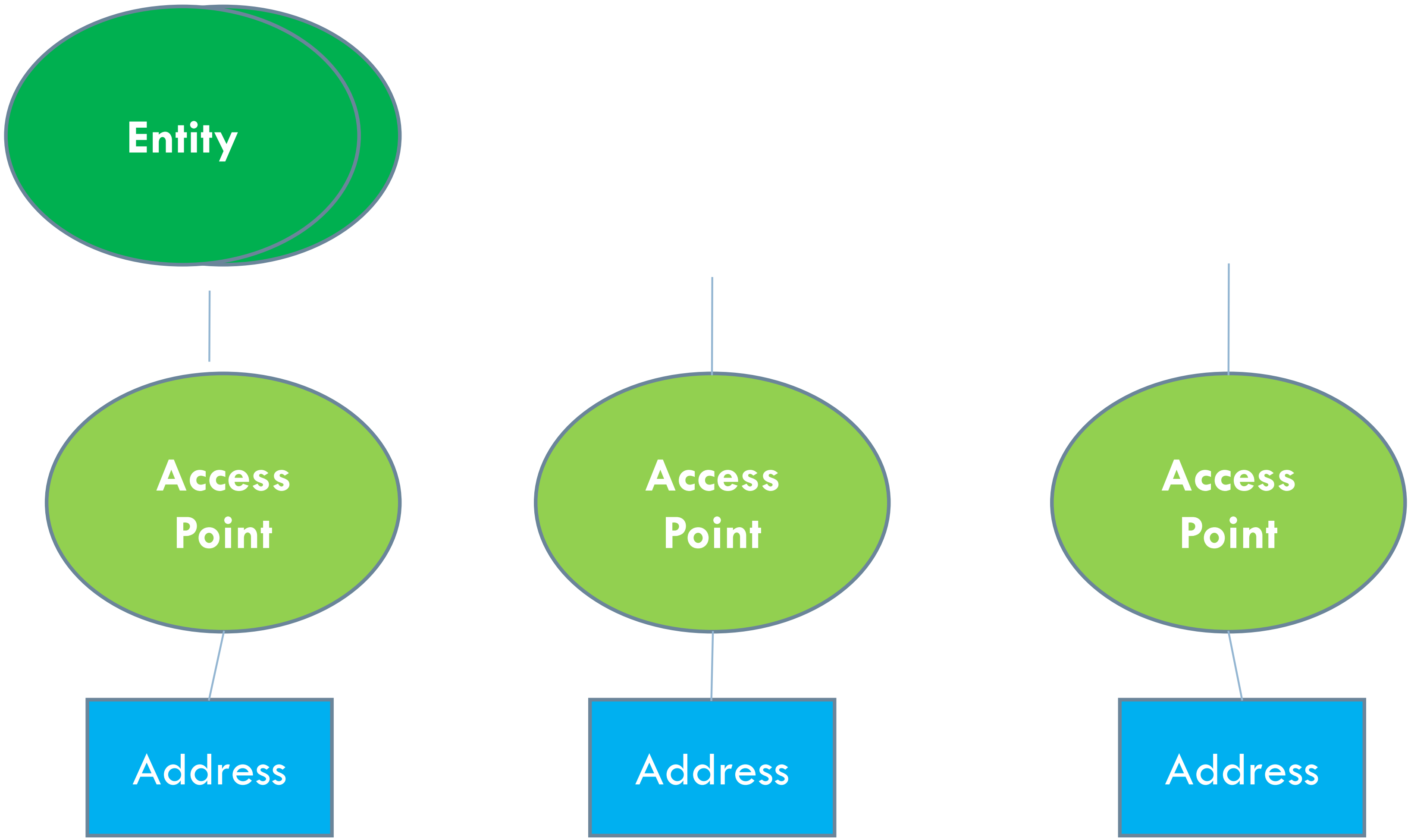
# Entity, A.P





# Location independent





# Identifier



- **An identifier refers to at most one entity.**
- **Each entity is referred to by at most one identifier.**
- **An identifier always refers to the same entity (it is never reused)**

# Resolving names and identifiers to addresses



- Name-to-address binding
- Problem: not appropriate to large network
- Naming systems
-

# URI, URL và URN



- **URI:**
  - a string of characters used to identify a resource.
  - interact with representations of the resource over a network
  - URL and URN
  - It comprises 5 parts: scheme, authority, path, query and fragment

```
  foo://example.com:8042/over/there?name=ferret#nose
  \_/   \______________/\_________/ \_________/ \__/
   |           |            |            |        |
scheme     authority       path        query   fragment
```

- **URN:**
  - ISBN 0486275574 (run:isbn:0-486-27557-4)
- **URL:**
  - file:///home/username/RomeoAndJuliet.pdf

# 2. Flat naming

# 2.1. Definition



- Identifiers are simply random bit strings (unstructured)
- It does not contain any information of location
- Goal: how flat names can be resolved
  1. Simple solutions
  2. Home-based Approaches
  3. Distributed Hash Tables
  4. Hierachical Approaches

# 2.2. Simple Solutions



- **2.2.1. Broadcasting and Multicasting**
- **2.2.2. Forwarding pointers**

# 2.2.1. Broadcasting and Multicasting



- **Condition: System supports broadcasting facilities:**
  - A message containing the identifier of the entity is broadcast to each machine.
  - Each machine is requested to check whether it has that entity.
  - Only the machines that can offer an access point for the entity send a reply message containing the address of that access point.

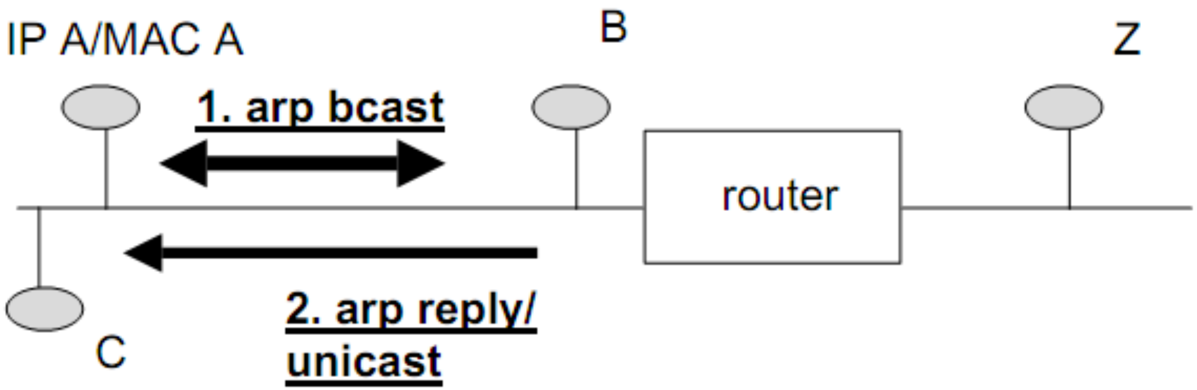
# 2.2.1. Broadcasting and Multicasting

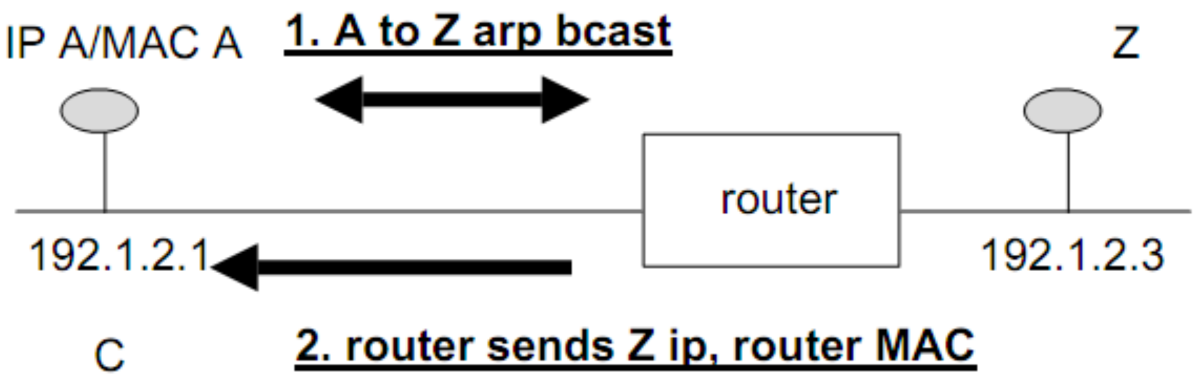


- **Inefficient when the network grows**
  - Wast network bandwidth by request messages
  - Too many hosts may be interrupted by requests they cannot answer.
- **→ multicasting**

# Example: ARP

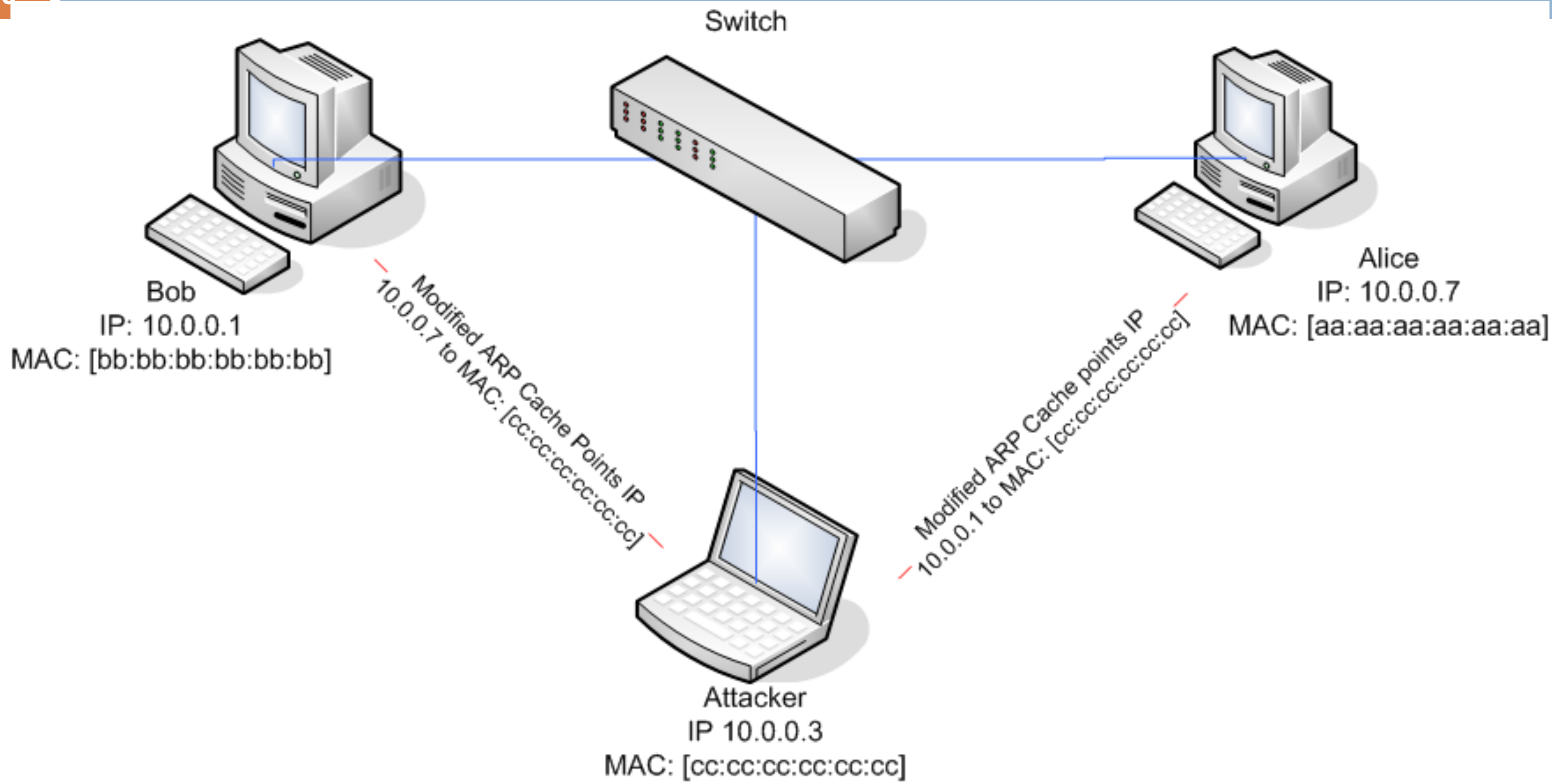



IP A/MAC A
B
Z
1. arp bcast
router
2. arp reply/
unicast
C


IP A/MAC A
1. A to Z arp bcast
Z
router
192.1.2.1
192.1.2.3
C
2. router sends Z ip, router MAC

# ARP-Spoofing





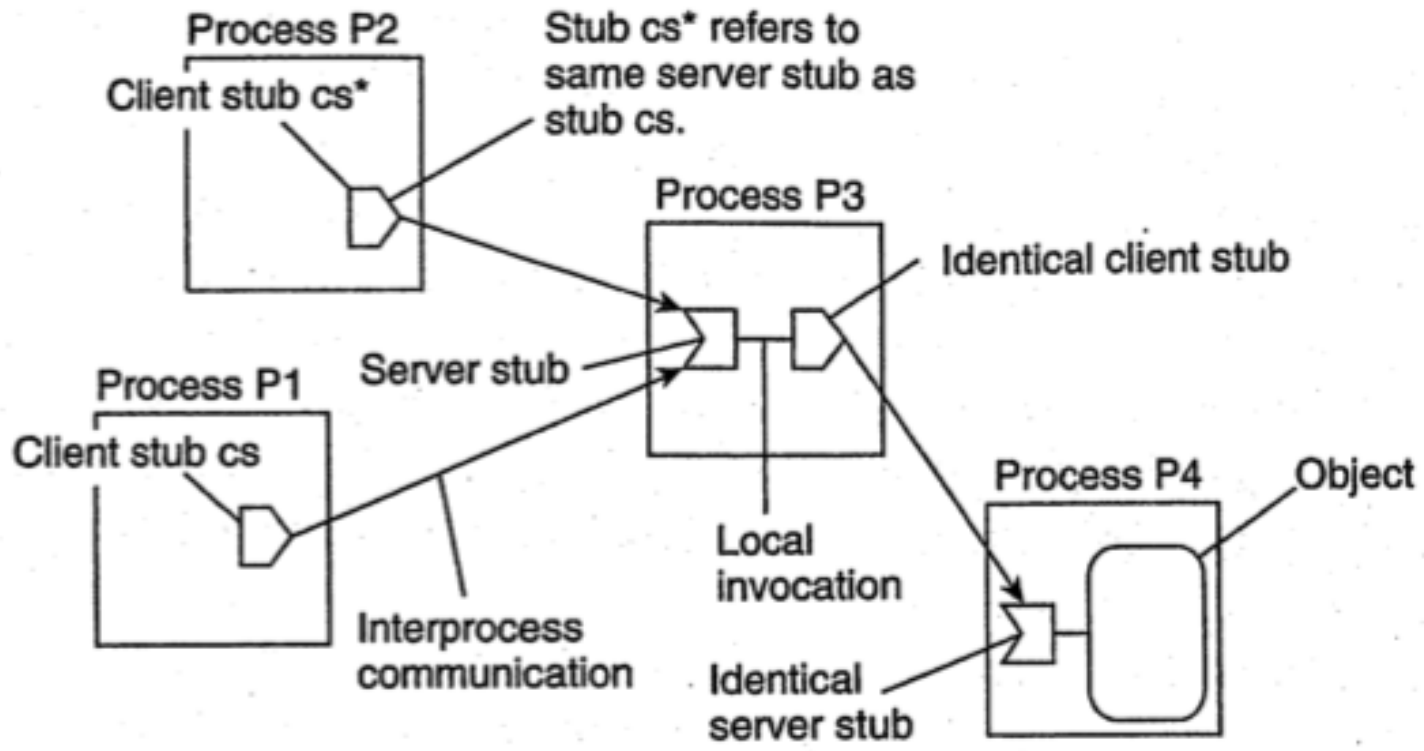
# 2.2.2. Forwarding Pointer



- **When an entity moves from A to B, it leaves behind in A a reference to its new location at B.**
- **Advantage:**
  - Simplicity: By using a traditional naming service, a client can look up the current address by following the chain of forwarding pointers.
- **Drawbacks**
  - A chain of FP can become so long → locating that entity is expensive.
  - All intermediate nodes have to maintain their part of the chain.
  - Broken links → cannot reach the entity

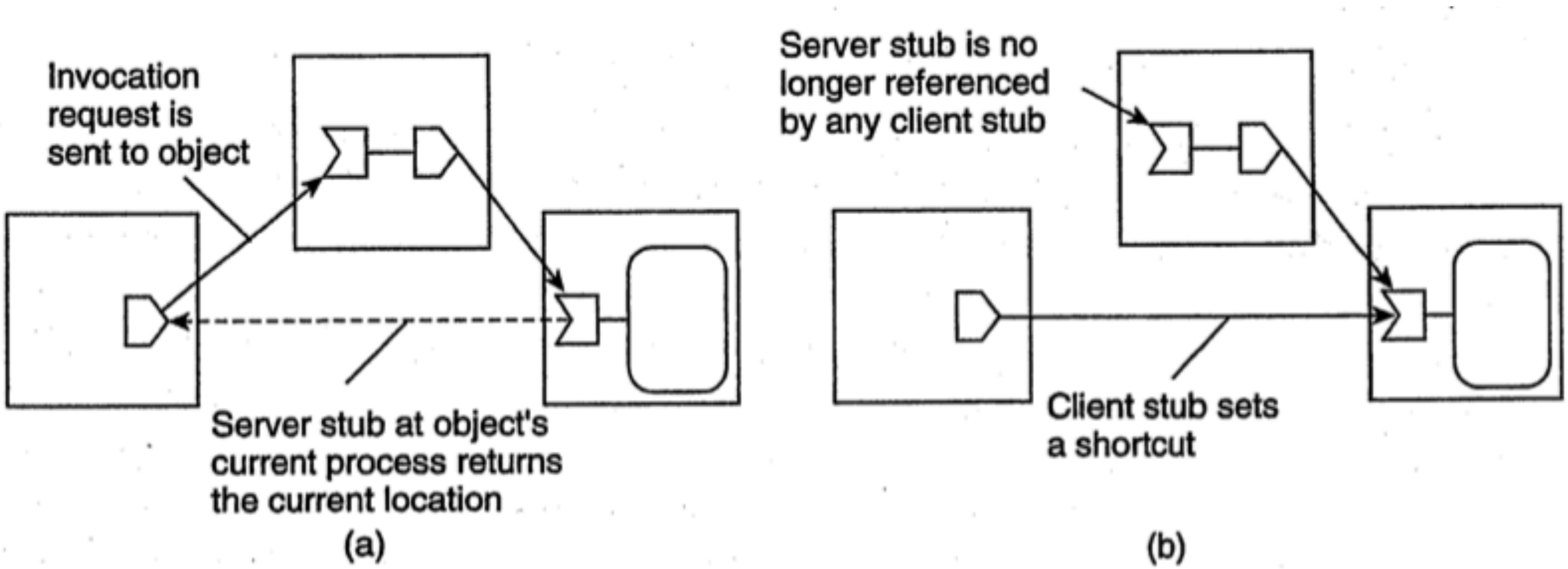
# Forwarding Pointer mechanism





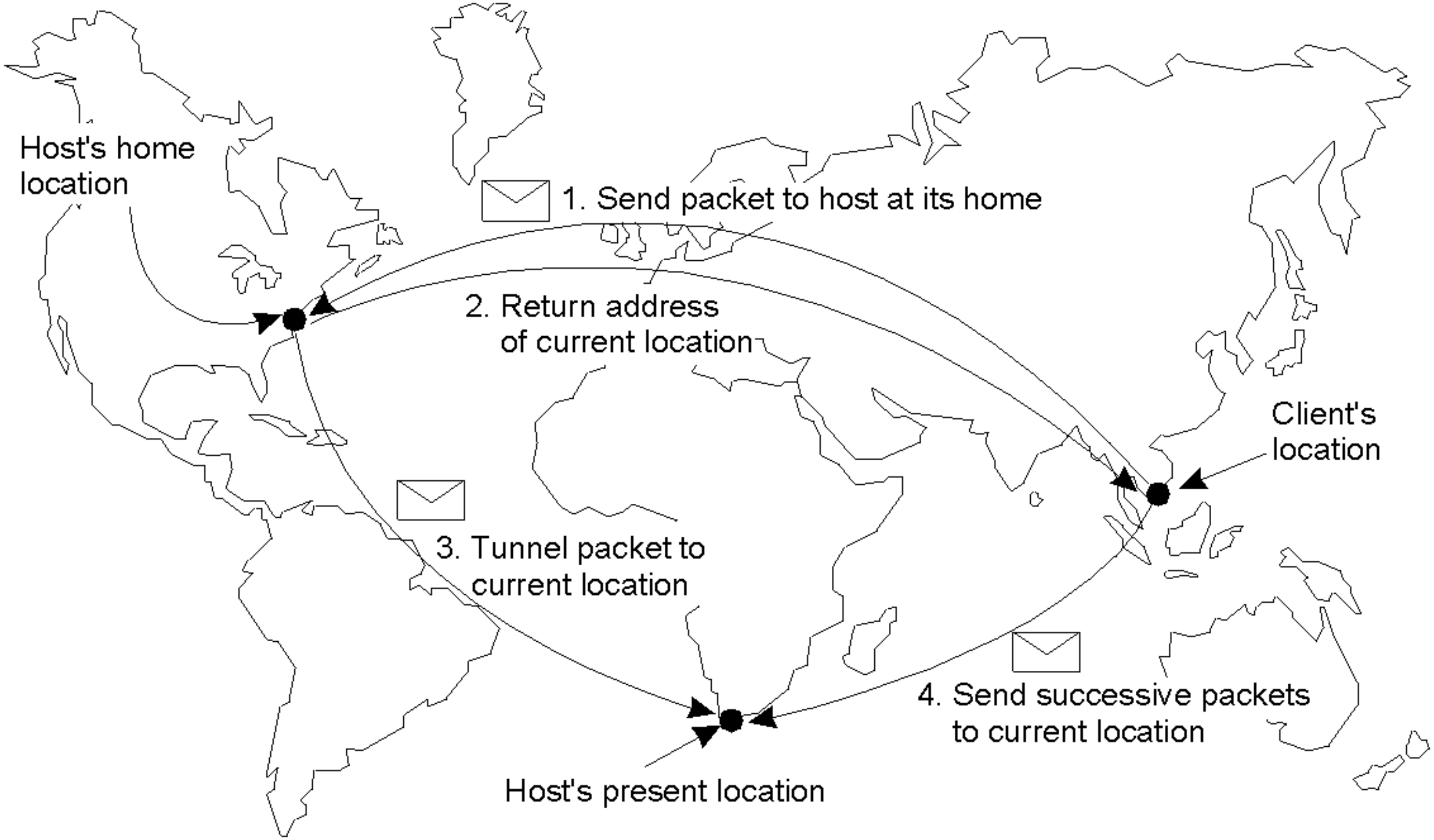
# Redirecting a FP





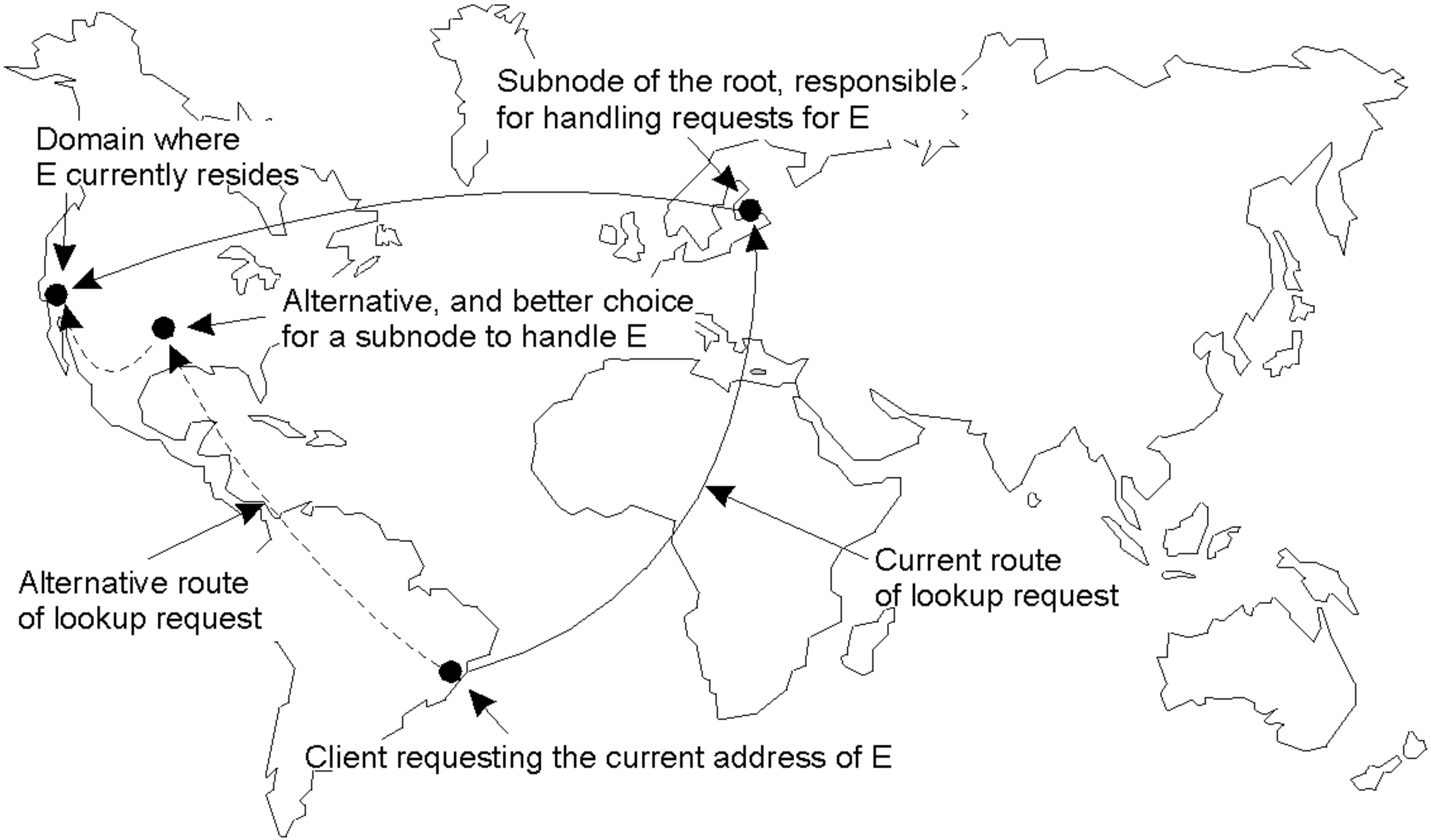
# 2.3. Home-based Approaches





# Solution for stable home problem





# 2.4. Distributed Hash Tables



- Chord system
- Create the ring with prev(n) and succ(n)
- Use finger table to determine the succ(k) of key k
- FTp is the finger table of node p:

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

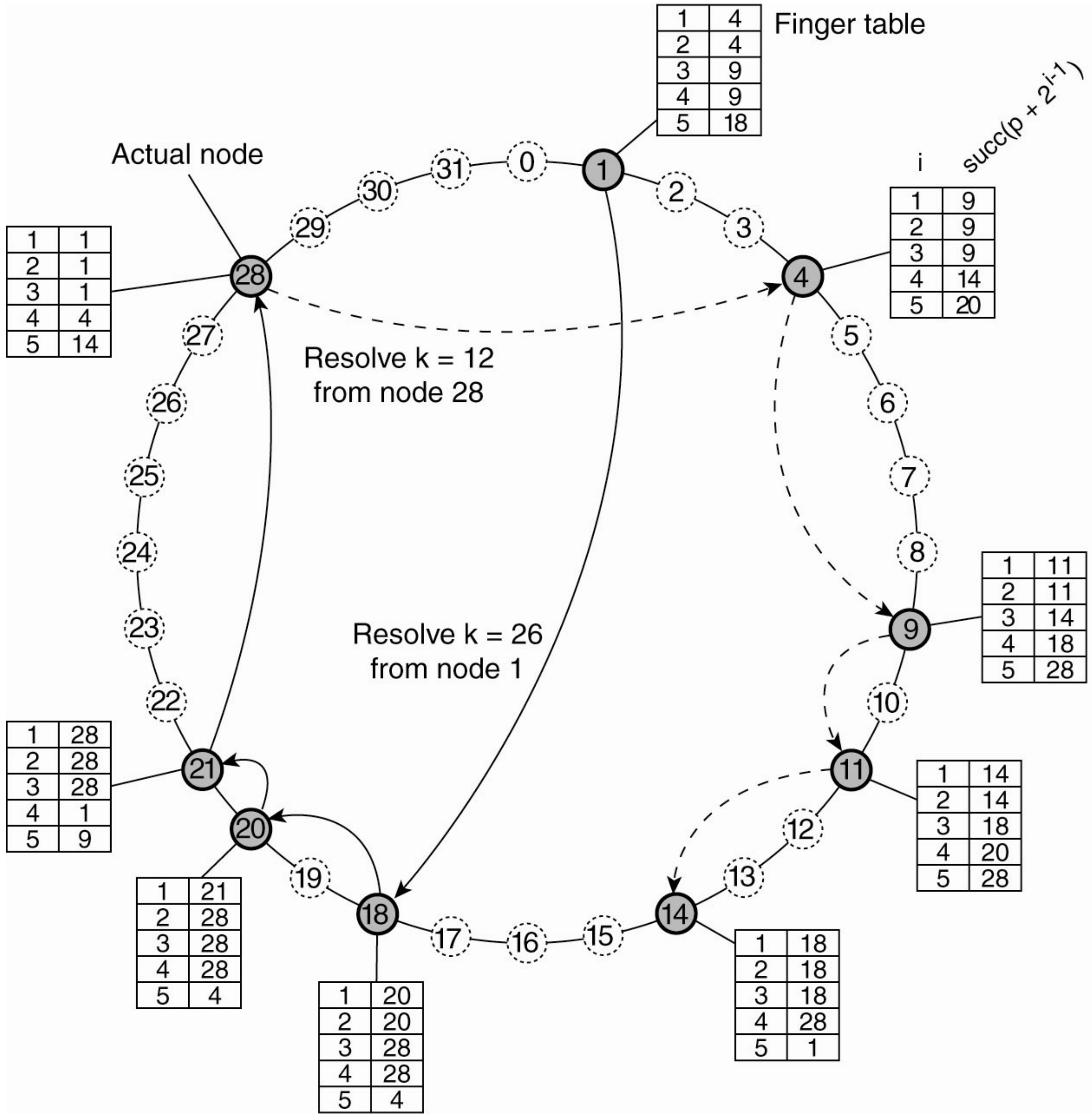
- To look up a key *k*, node *p* will then immediately forward the request to node *q*:

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

- Update the finger tables after inserting a new node

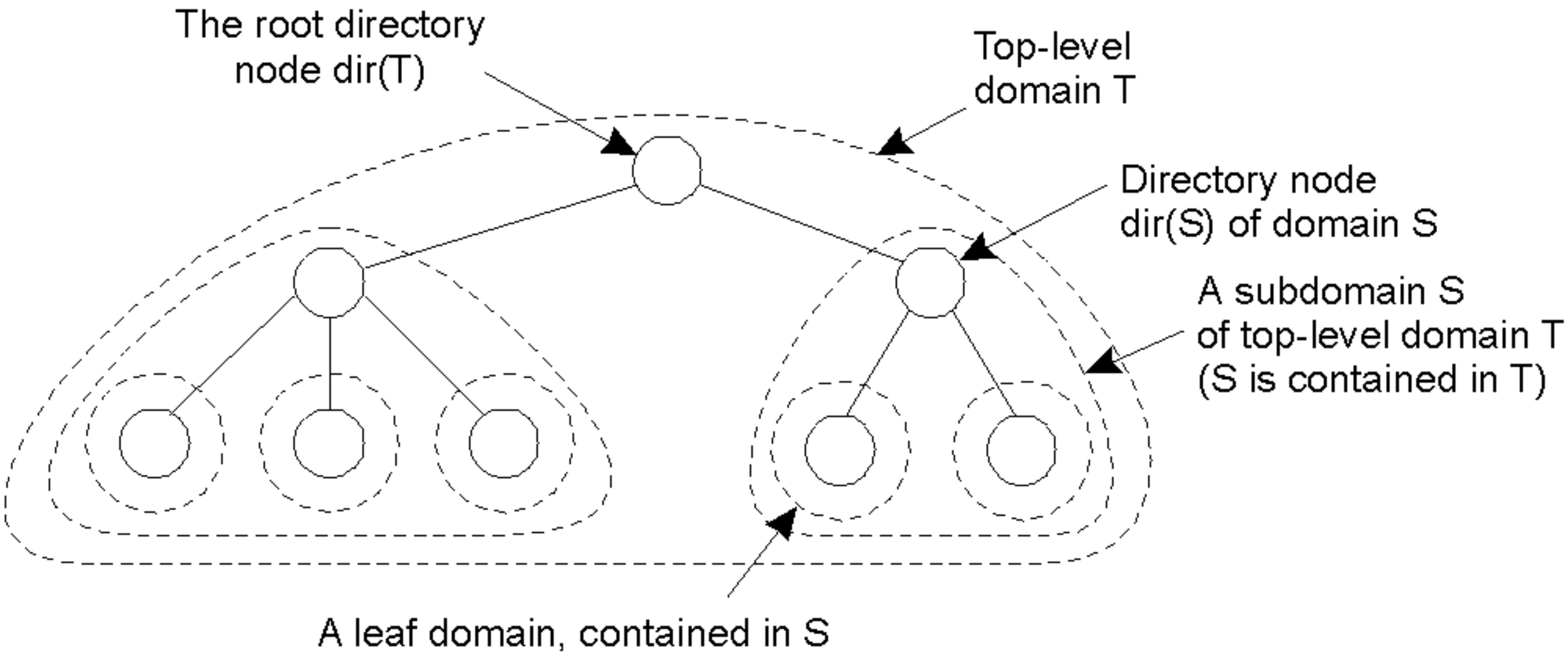
# Chord system with finger tables



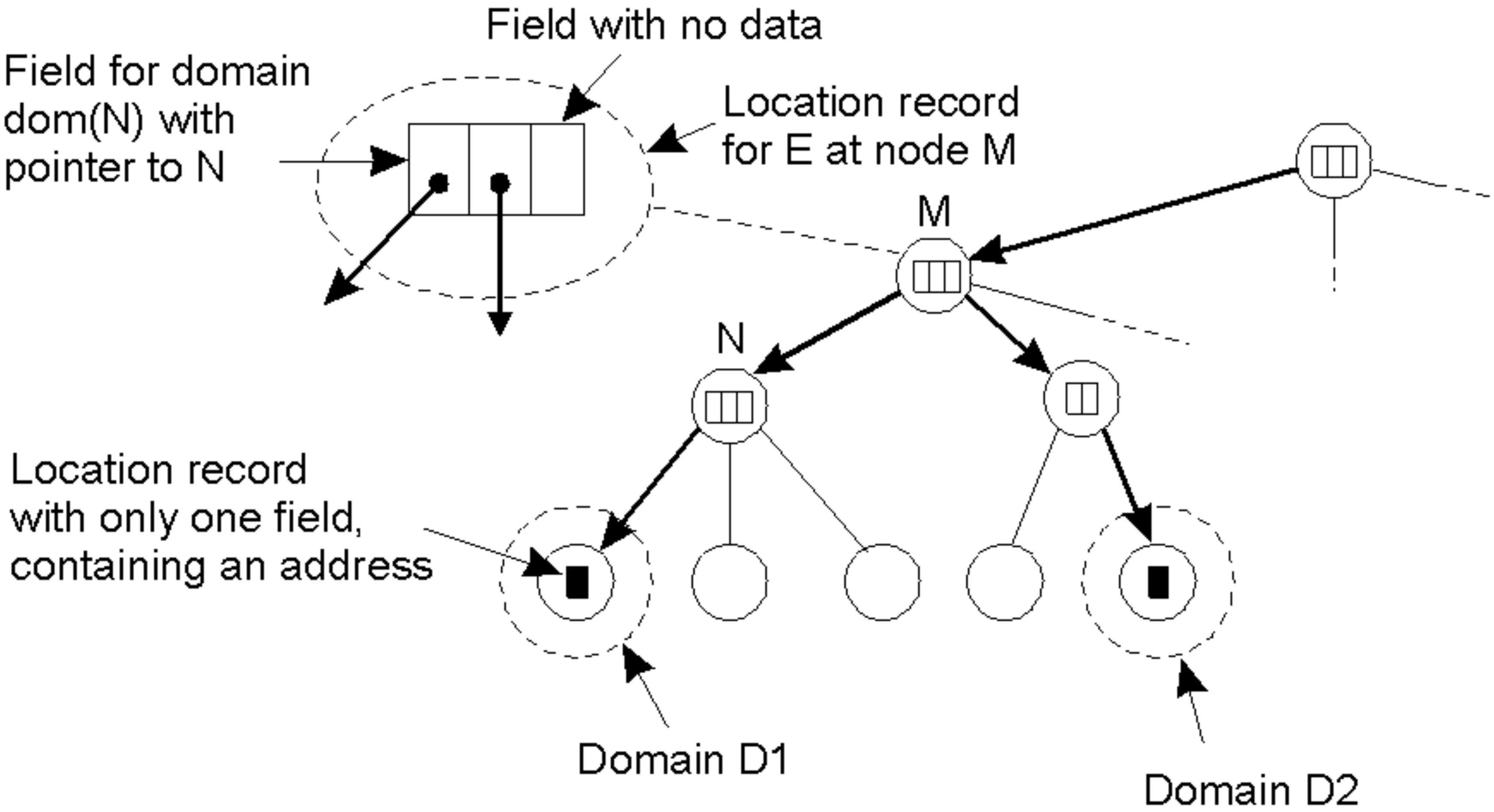


# 2.5. Hierarchical Approaches



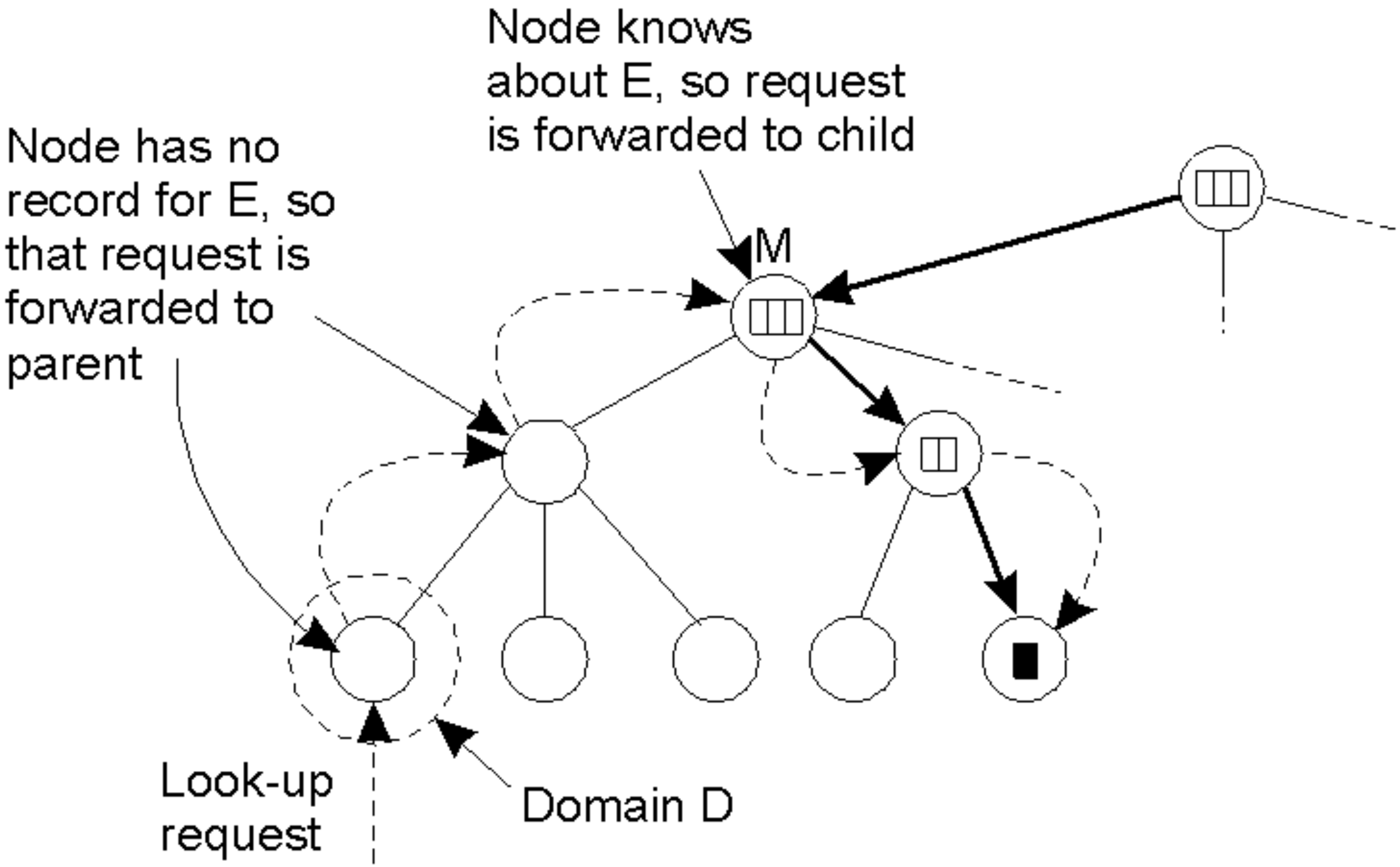


# An entity having two addresses in different leaf domains



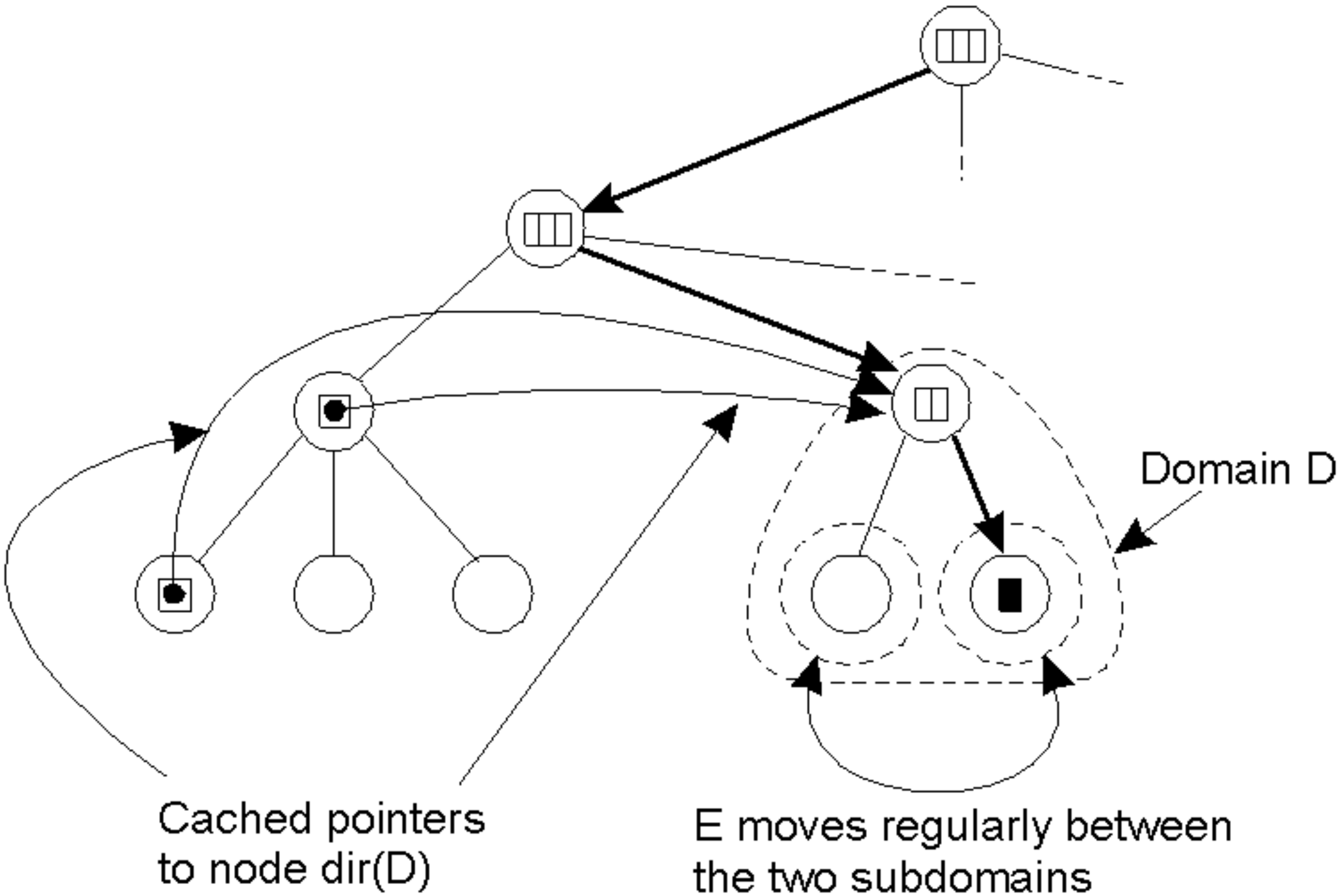
# Looking-up




Node knows about E, so request is forwarded to child
Node has no record for E, so that request is forwarded to parent
M
Look-up request
Domain D

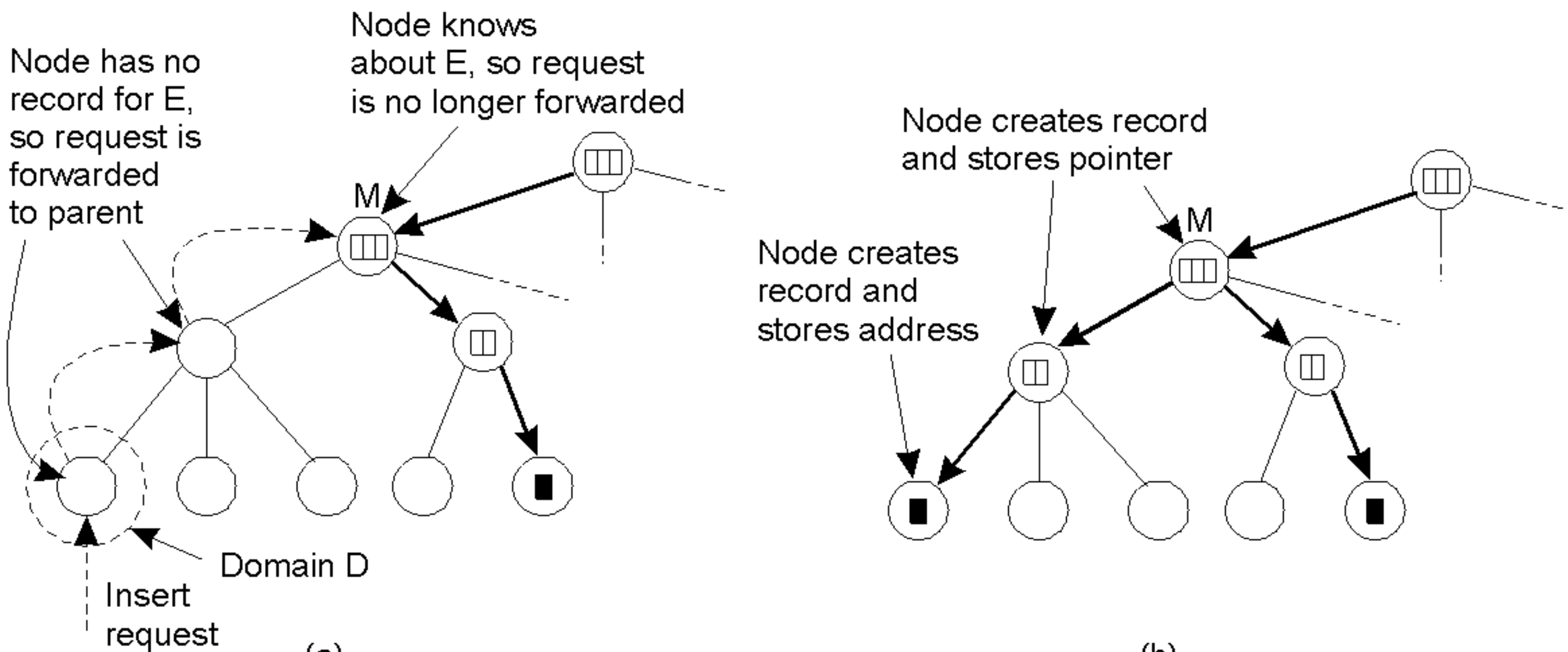
# Caching




Domain D
Cached pointers
to node dir(D)
E moves regularly between
the two subdomains

# Updating




Node has no
record for E,
so request is
forwarded
to parent
Node knows
about E, so request
is no longer forwarded
M
Domain D
Insert
request
Node creates record
and stores pointer
M
Node creates
record and
stores address

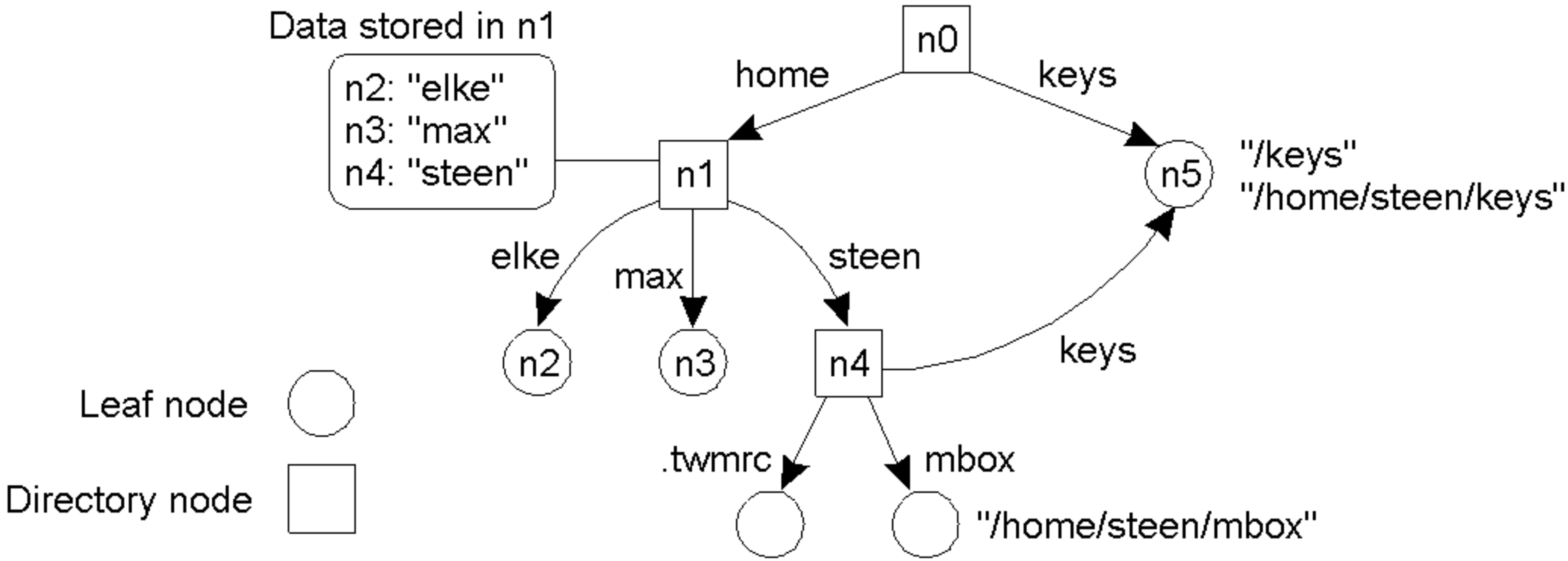
(a)

(b)

29

# 3. Structured Naming

# Structured Name Spaces




**A general naming graph**

# Name Spaces



- **Leaf node:**
  - No outgoing edge
  - Store information of its address
- **Directory node:**
  - Outgoing edge
  - Store a table with info (edge label, node identifier)
- **Path name: N: <label1, label2, label3, label4, ...>**
- **Absolute path name/Relative path name**

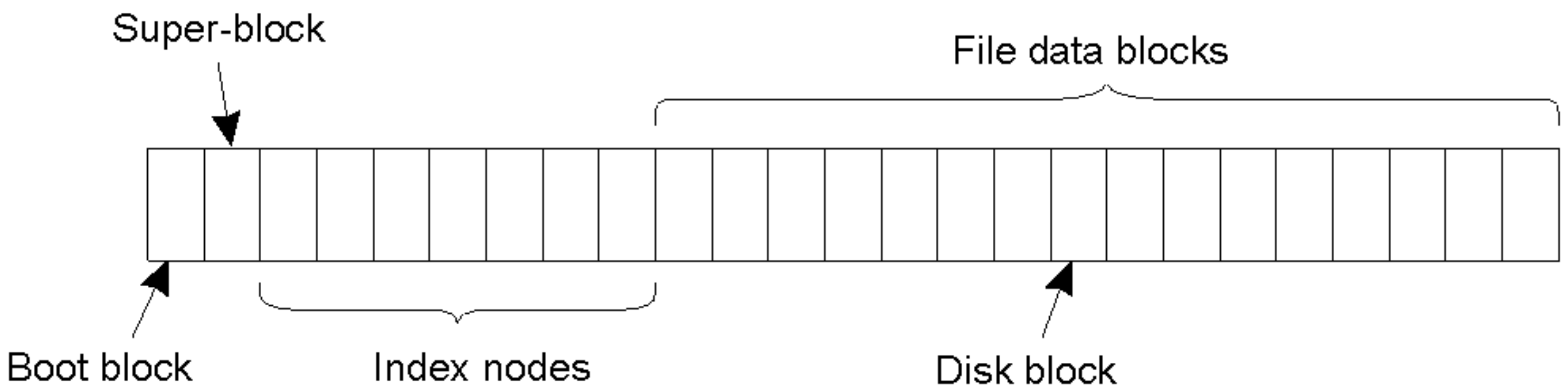
# Name resolution



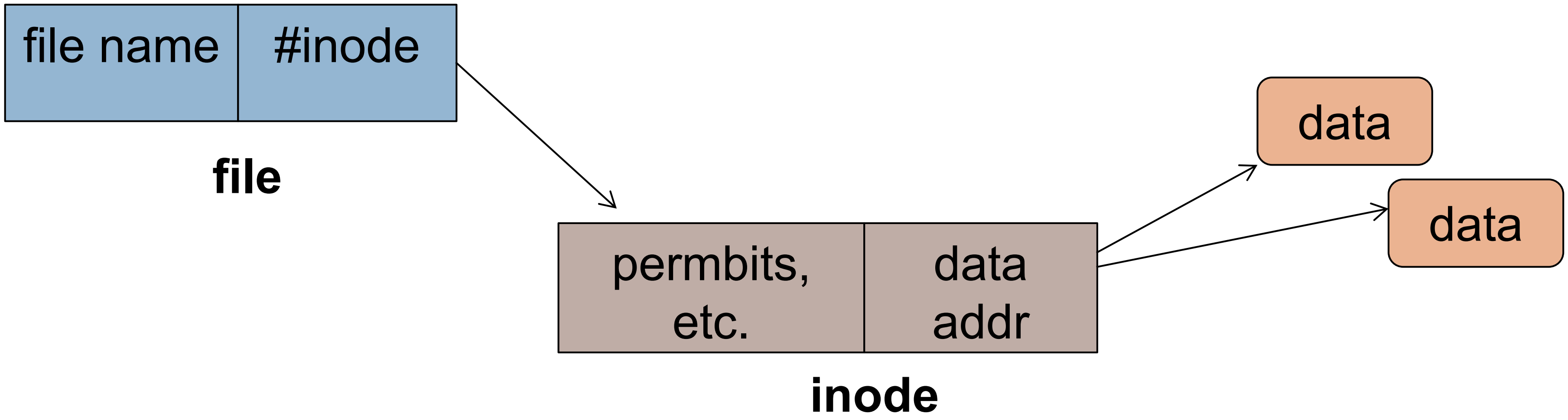
- Consider a path name: N:<label1, label2, ..., labeln>
- Start at node N of the naming graph, where the name label1 is looked up in the directory table, and which returns the identifier of the node to which label1 refers.
- Continue at the identified node by looking up the name label2
- So on ...
- Relatively with the UNIX file system

# General organization of the UNIX file system

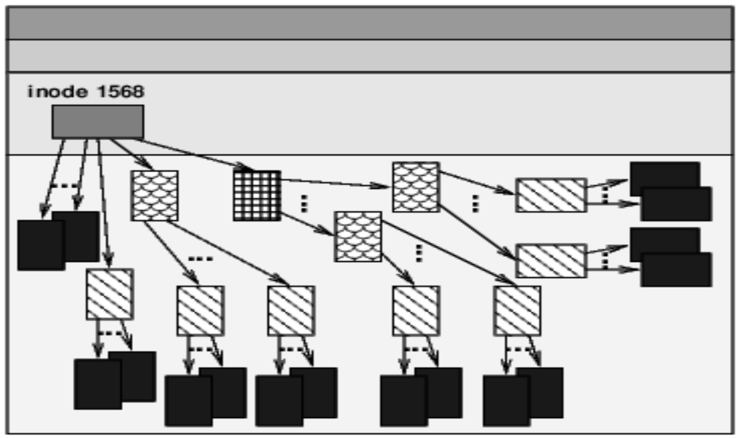


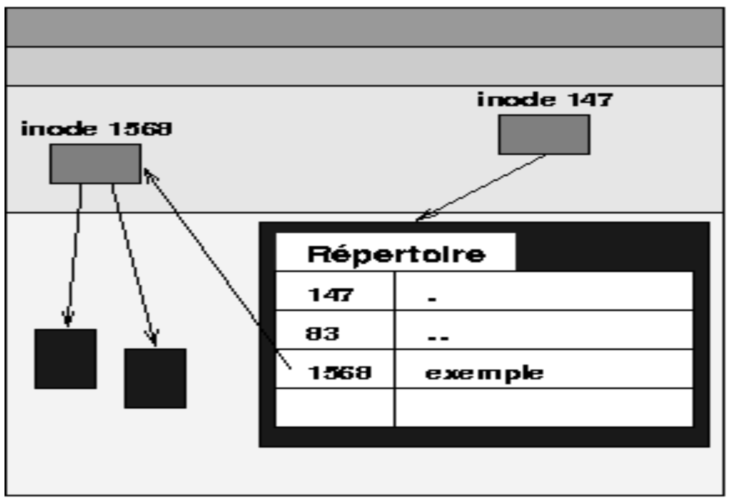


# File system in UNIX

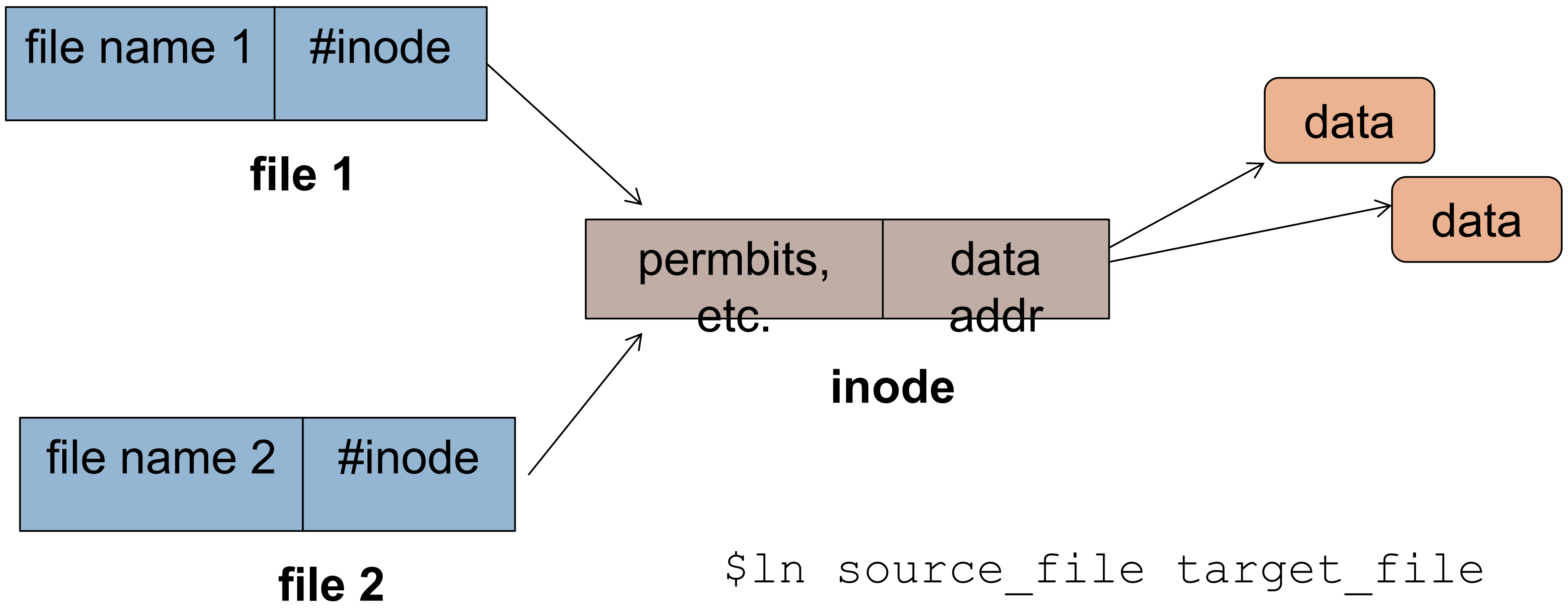


# Directory node (folder)



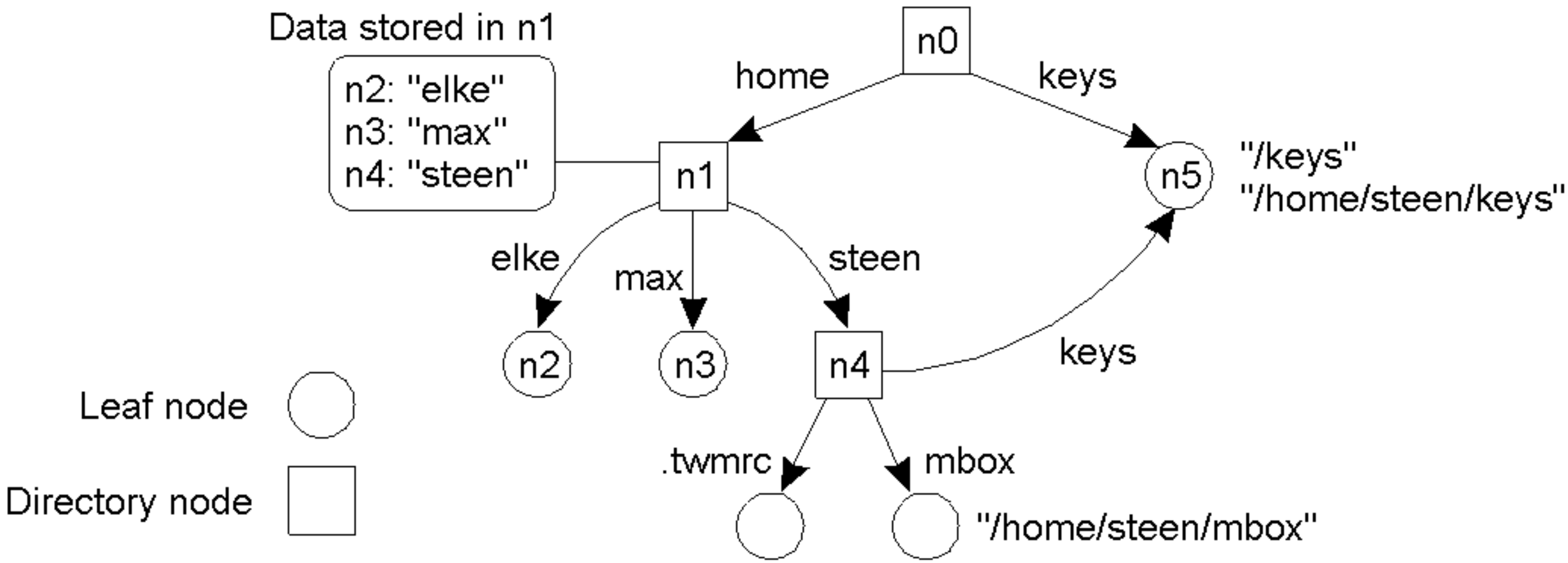
# Hard link



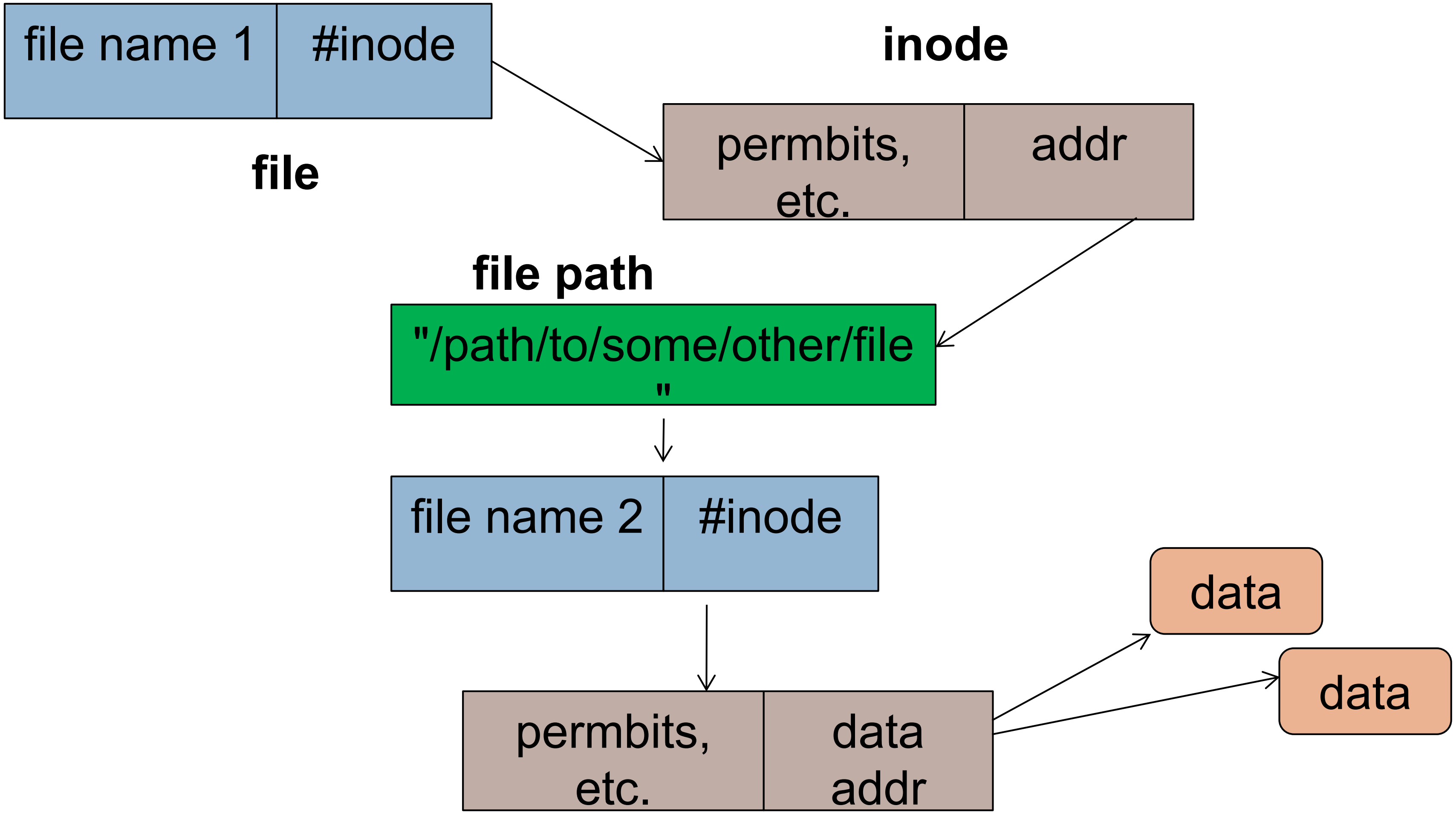

`$ln source_file target_file`

# Hard link (cont.)





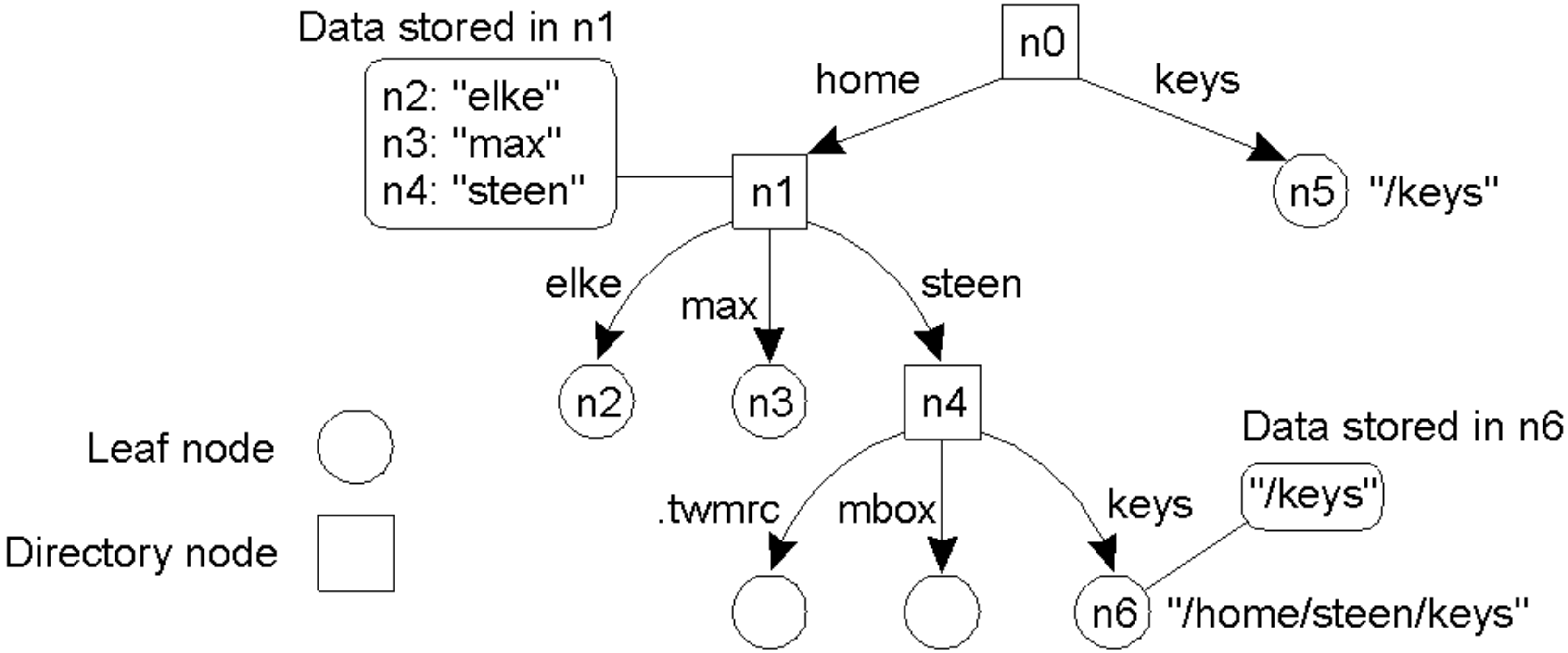
# Soft link



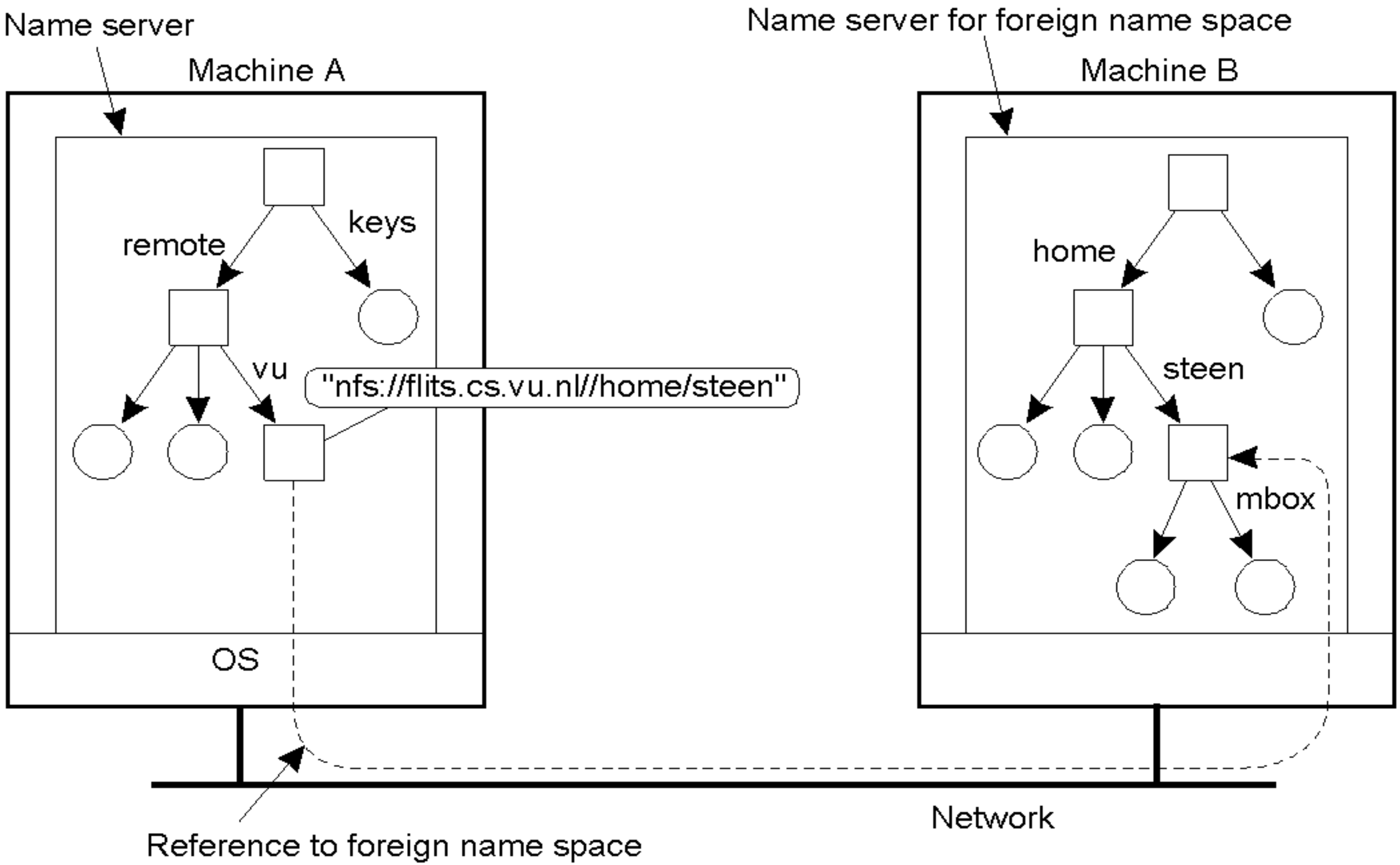

`$ln -s source_file target_file`

# Soft link



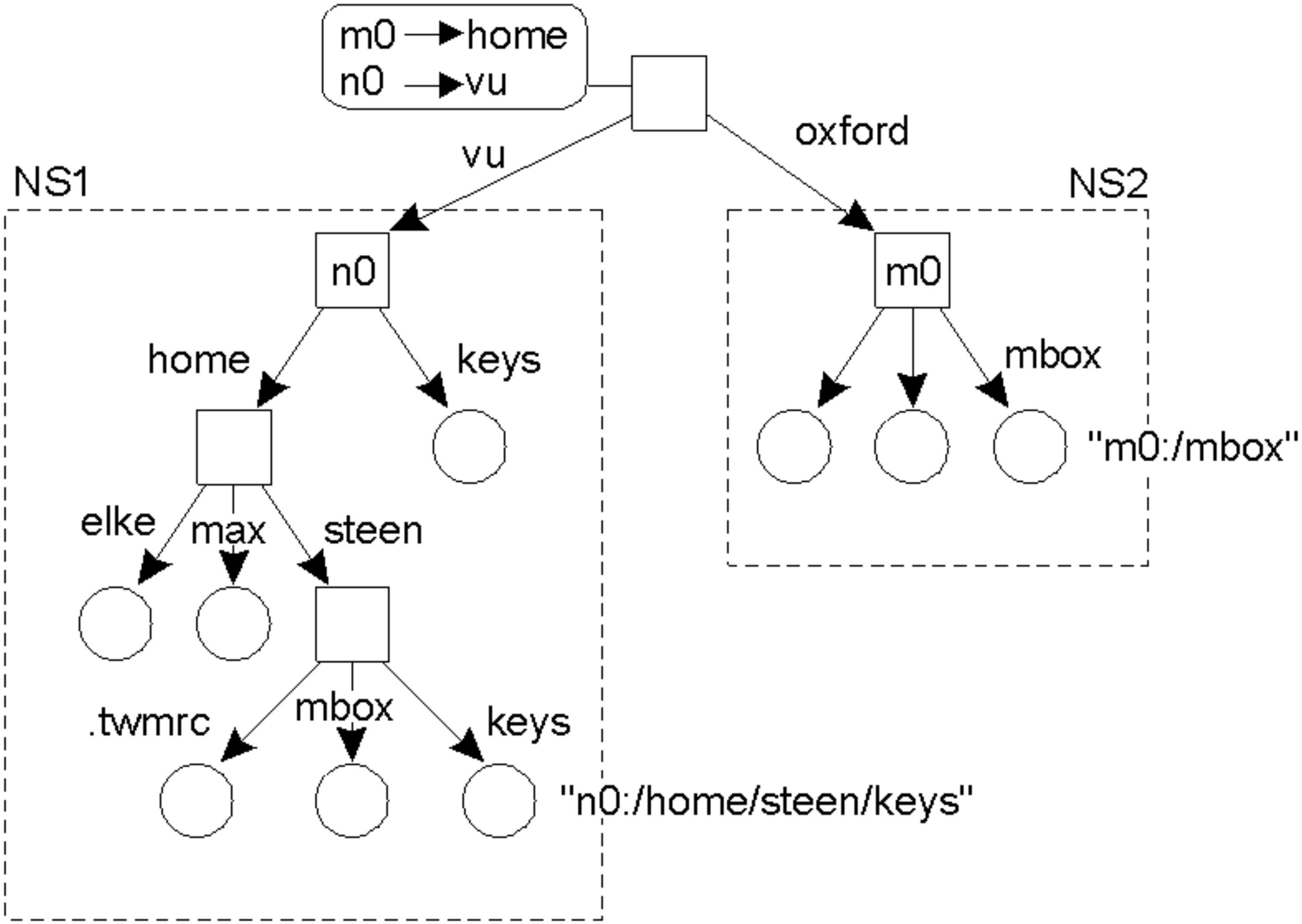

Data stored in n1
n2: "elke"
n3: "max"
n4: "steen"
n0
home
keys
n1
n5
"/keys"
elke
max
steen
n2
n3
n4
Data stored in n6
"/keys"
.twmrc
mbox
keys
n6
"/home/steen/keys"
Leaf node
Directory node

# Mounting


Name server
Machine A
remote
keys
vu
"nfs://flits.cs.vu.nl//home/steen"
OS
Name server for foreign name space
Machine B
home
steen
mbox
Network
Reference to foreign name space

# Merging





m0 → home
n0 → vu
vu
oxford
NS1
NS2
n0
m0
home
keys
mbox
"m0:/mbox"
elke
max
steen
.twmrc
mbox
keys
"n0:/home/steen/keys"

# Naming service



- **Functions:**
  - Add names
  - Remove names
  - Look up names
- **Naming service is implemented by name servers**
- **In large-scale distributed systems (many entities, large geographical area) → distribute the implementation of a name space over multiple name servers**

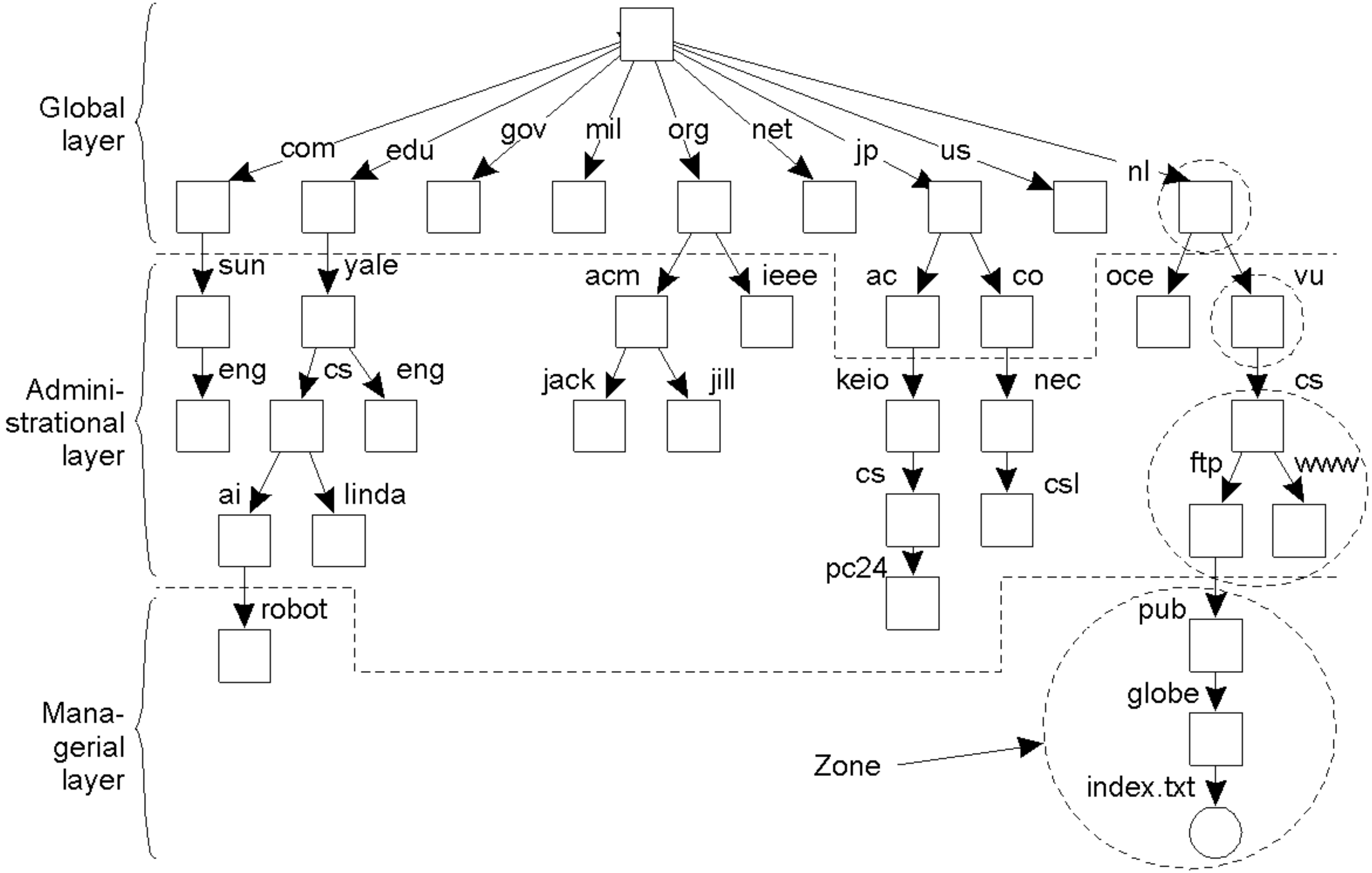
# Hierarchical organization



- **Global layer**
  - root node + directory nodes logically close to the root (children)
  - Stability (rarely changed)
  - represent organization, or group of organization
- **Administrational layer**
  - represent groups of entities that belong to the same organization
- **Managerial layer**
  - consist of nodes that may change regularly

# DNS name space





# Comparison of three layers



| Item | Global | Administrational | Managerial |
|---|---|---|---|
| Geographical scale of network | Worldwide | Organization | Department |
| Total number of nodes | Few | Many | Vast numbers |
| Responsiveness to lookups | Seconds | Milliseconds | Immediate |
| Update propagation | Lazy | Immediate | Immediate |
| Number of replicas | Many | None or few | None |
| Is client-side caching applied? | Yes | Yes | Sometimes |

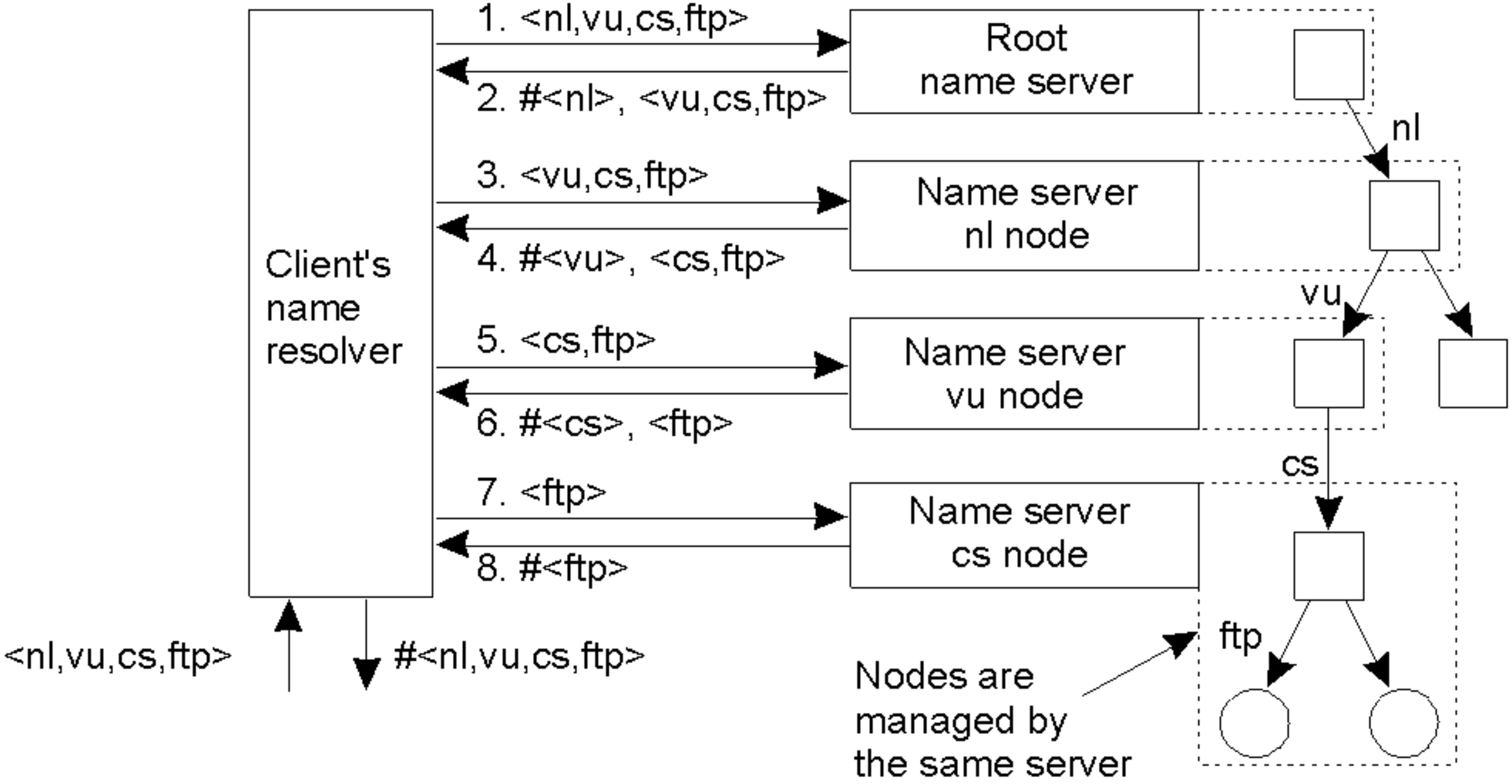
# Implementation of Name Resolution



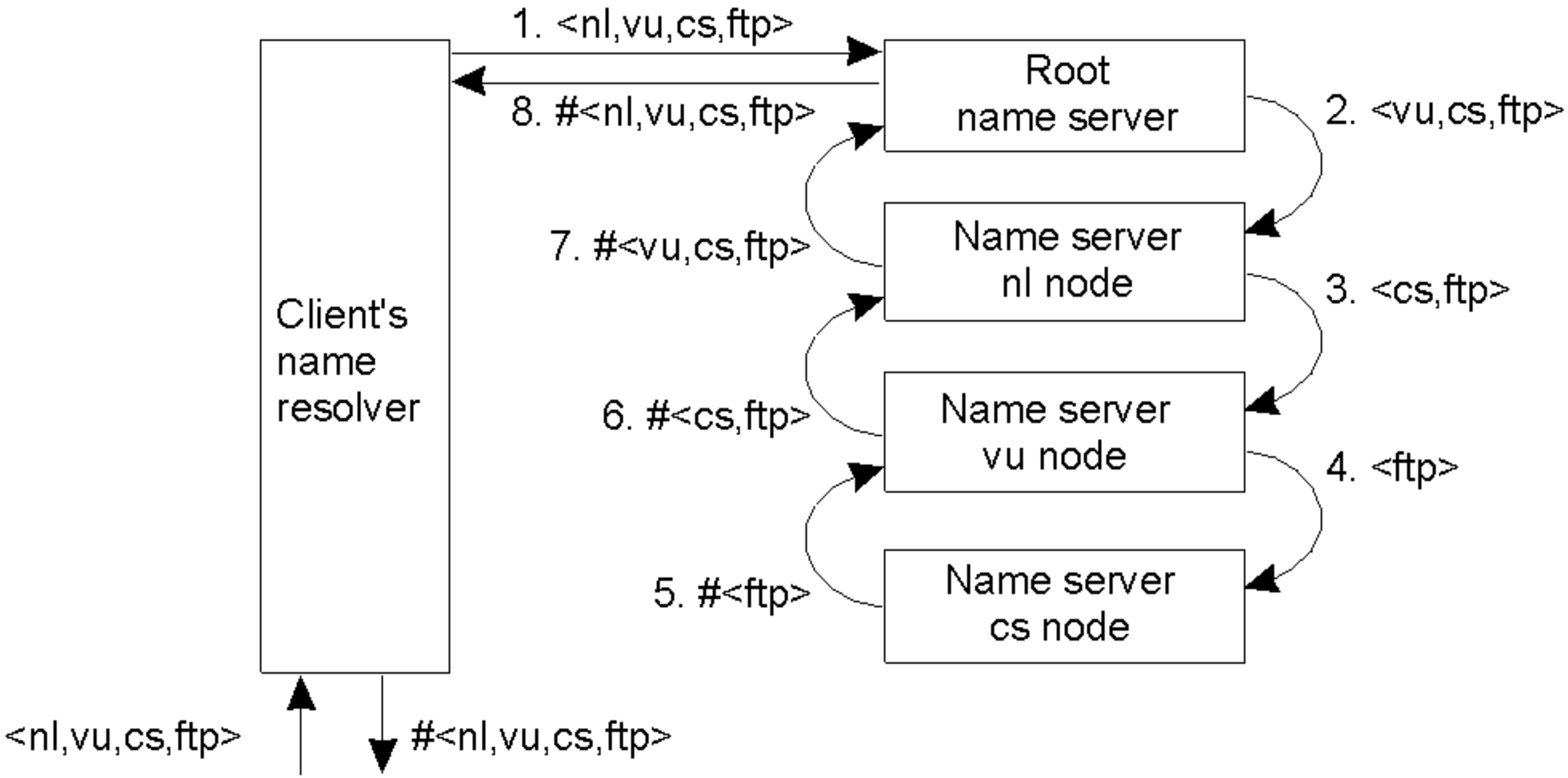
- **Depend on the distribution of a name space across multiple name servers**
- **Each client has a name resolver**
- **2 ways of implementation of name resolution:**
  - Iterative name resolution
  - Recursive name resolution

# Iterative name resolution



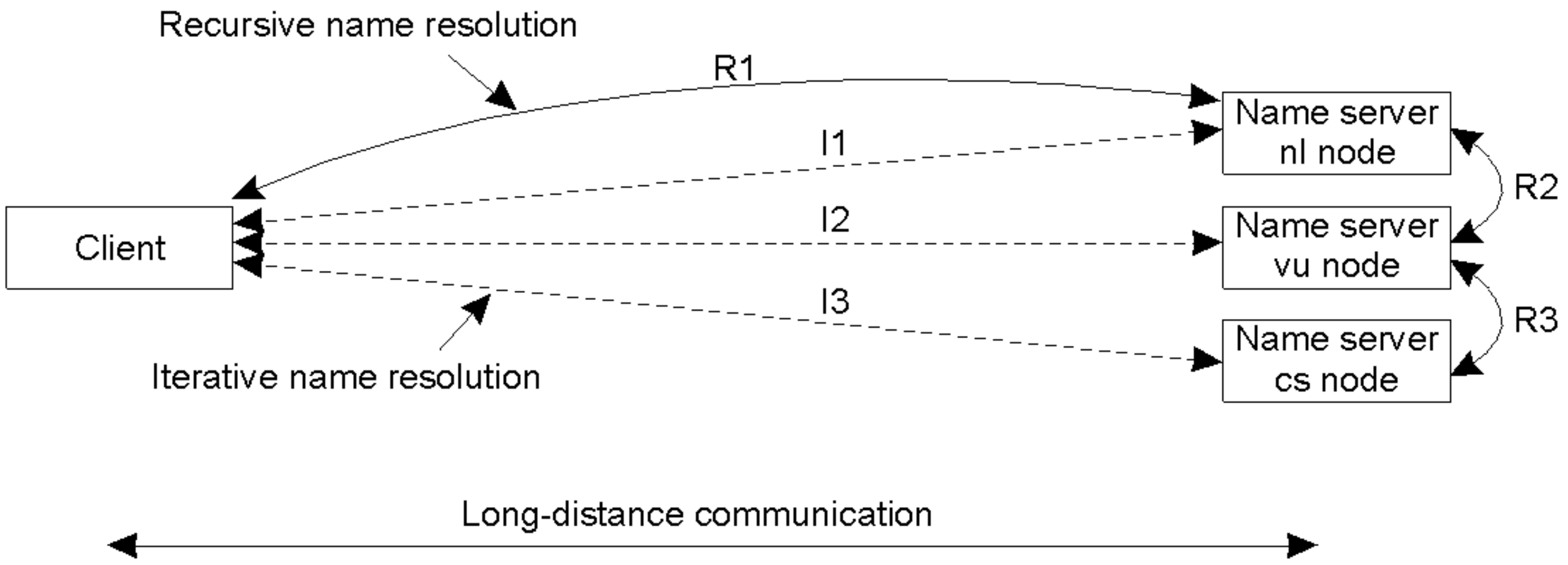


# Recursive name resolution



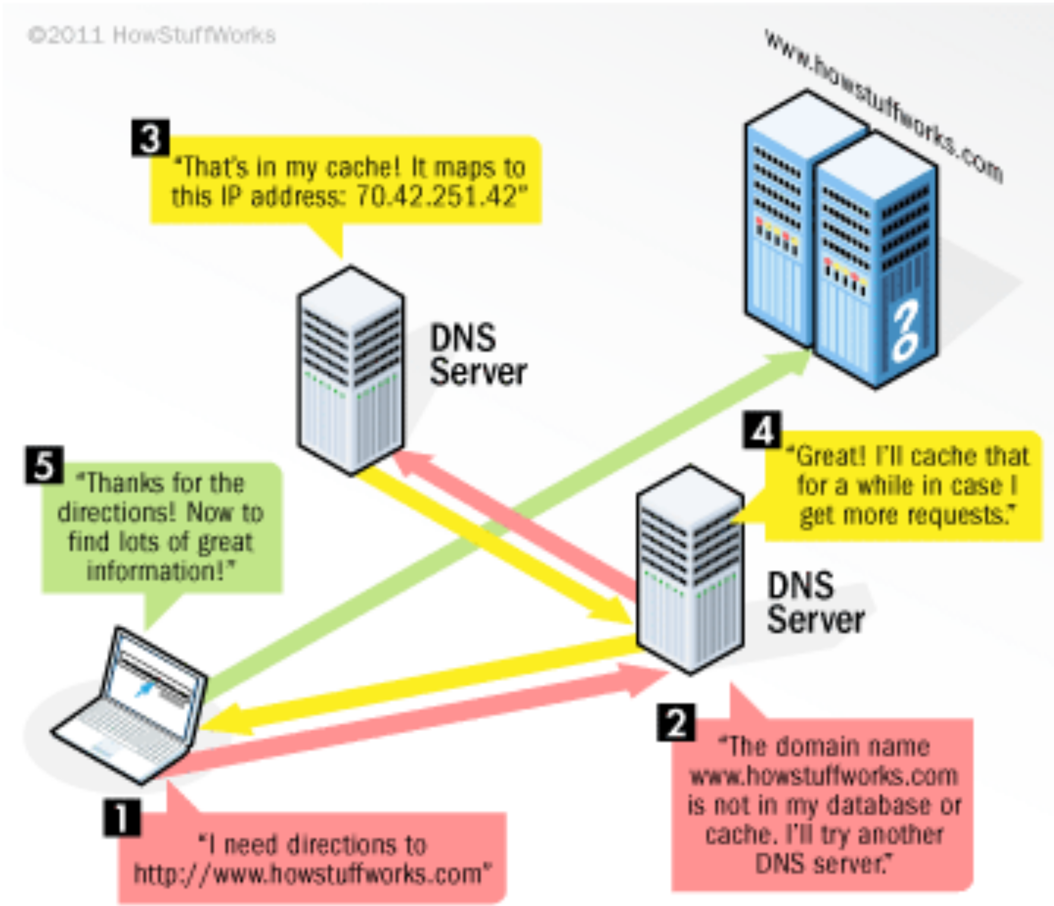
# Recursive vs. iterative name resolution





# Example: DNS






[source: howstuffworks.com]

# DNS Terminology, Components, and Concepts

**Top-Level Domain**

**Hosts**

**SubDomain**

**Fully Qualified Domain Name (FQDN)**

**Name Server**

**Zone File**

**Records**

# Record types

## Start of Authority (SOA)

```
domain.com.   IN SOA ns1.domain.com. admin.domain.com. (
                                          12083    ; serial number
                                          3h       ; refresh interval
                                          30m      ; retry interval
                                          3w       ; expiry period
                                          1h       ; negative TTL
)
```

## A and AAAA Records

```
host      IN      A       IPv4_address
host      IN      AAAA    IPv6_address
```

## CNAME records

```
server1       IN   A        111.111.111.111
www           IN   CNAME    server1
```

# Record types

## MX records

```
        IN   MX   10   mail1.domain.com.
        IN   MX   50   mail2.domain.com.
mail1   IN   A         111.111.111.111
mail2   IN   A         222.222.222.222
```

## NS records

```
        IN   NS        ns1.domain.com.
        IN   NS        ns2.domain.com.
ns1     IN   A         111.222.111.111
ns2     IN   A         123.211.111.233
```