Chapter-4

Naming

Introduction

- names play an important role to:
  - share resources
  - uniquely identify entities
  - refer to locations
  - etc.
- an important issue is that a name can be resolved to the entity it refers
- to resolve names, it is necessary to implement a naming system
- in a distributed system, the implementation of a naming system is itself often distributed, unlike in non-distributed systems
- efficiency and scalability of the naming system are the main issues

1. Naming Entities

- Names, Identifiers, and Addresses
  - a name in a distributed system is a string of bits or characters that is used to refer to an entity
  - an entity is anything; e.g., resources such as hosts, printers, disks, files, objects, processes, users, ...
  - entities can be operated on; e.g., a resource such as a printer offers an interface containing operations for printing a document, requesting the status of a job, ...
  - to operate on an entity, it is necessary to access it through its access point, itself an entity (special)

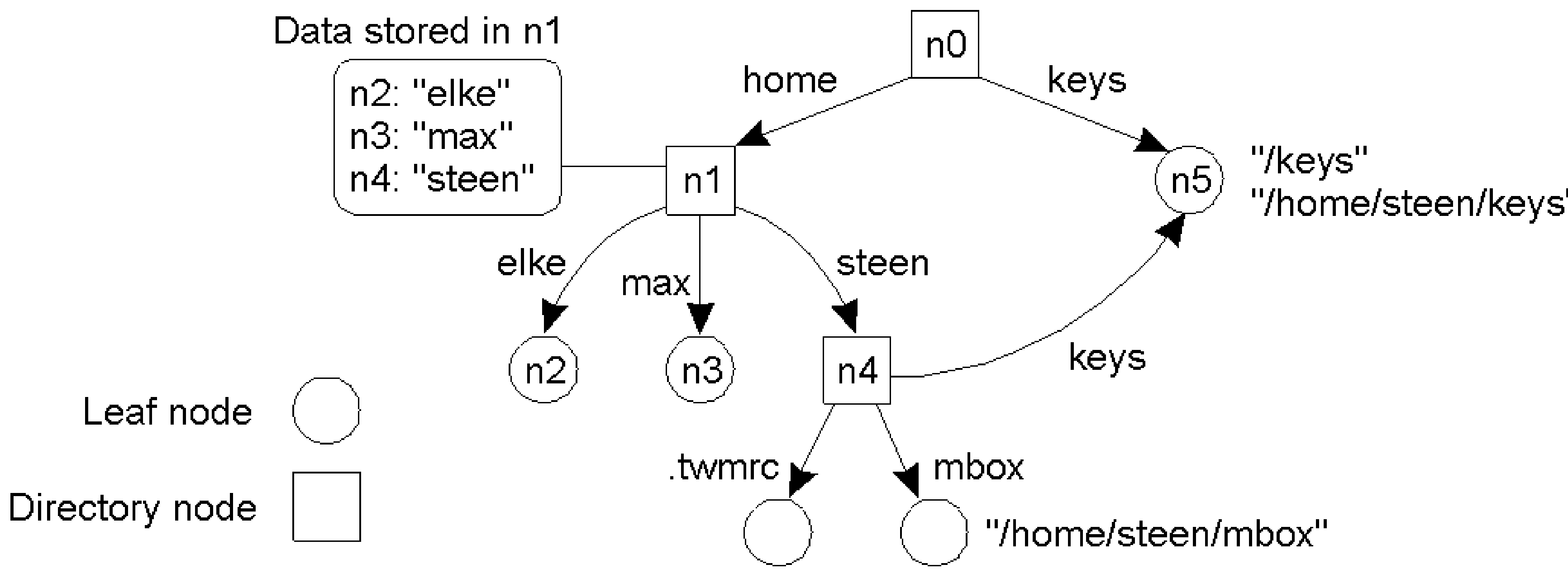
- access point
  - the name of an access point is called an address (such as IP address and port number as used by the transport layer)
  - the address of the access point of an entity is also referred to as the address of the entity
  - an entity can have more than one access point (similar to accessing an individual through different telephone numbers)
  - an entity may change its access point in the course of time (e.g., a mobile computer getting a new IP address as it moves)

- an address is a special kind of name
  - it refers to at most one entity
  - each entity is referred by at most one address; even when replicated such as in Web pages
  - an entity may change an access point, or an access point may be reassigned to a different entity (like telephone numbers in offices)
  - separating the name of an entity and its address makes it easier and more flexible; such a name is called location independent
- there are also other types of names that uniquely identify an entity; in any case an identifier is a name with the following properties
  - it refers to at most one entity
  - each entity is referred by at most one identifier
  - it always refers to the same entity (never reused)
- identifiers allow us to unambiguously refer to an entity

- examples
  - name of an FTP server (entity)
    - URL of the FTP server
  - address of the FTP server
    - IP number: port number
  - the address of the FTP server may change

2. Name Spaces and Name Resolution

- names in a distributed system are organized into a name space
- a name space is generally organized as a labeled, directed graph with two types of nodes
  - leaf node: represents the named entity and stores information such as its address or the state of that entity
  - directory node: a special entity that has a number of outgoing edges, each labeled with a name



a general naming graph with a single root node

- each node in a naming graph is considered as another entity with an identifier
- a directory node stores a table in which an outgoing edge is represented as a pair (*edge label, node identifier*), called a directory table
- each path in a naming graph can be referred by the sequence of labels corresponding to the edges of the path and the first node in the path, such as

  N:<label1, label2, ..., labeln>, where N refers to the first node in the path
- such a sequence is called a path name
- if the first node is the root of the naming graph, it is called an absolute path name; otherwise it is a relative path name
- instead of the path name n0:<home, steen, mbox>, we often use its string representation /home/steen/mbox
- there may also be several paths leading to the same node, e.g., node n5 can be represented as /keys or /home/steen/keys

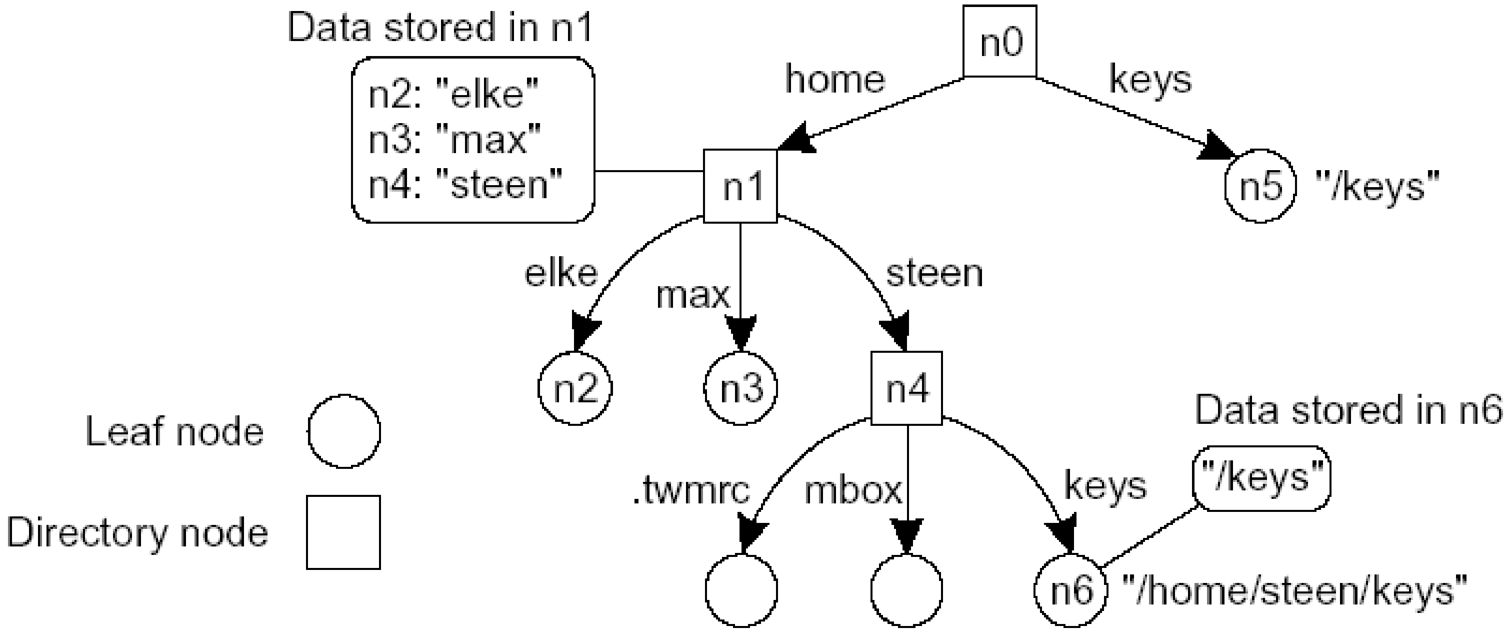
- although the above naming graph is directed acyclic graph (a node can have more than one incoming edge but is not permitted to have a cycle), the common way is to use a tree (hierarchical) with a single root (as is used in file systems)
  - in a tree structure, each node except the root has exactly one incoming edge; the root has no incoming edges
  - ➲ each node also has exactly one associated (absolute) path name

- Name Resolution
  - given a path name, the process of looking up a name stored in the node is referred to as name resolution; it consists of finding the address when the name is given (by following the path)
- Linking and Mounting
  - Linking: giving another name for the same entity (an alias)

    e.g., environment variables in UNIX such as HOME that refer to the home directory of a user
    - two types of links (or two ways to implement an alias):
      - hard link: to allow multiple absolute path names to refer to the same node in a naming graph

        e.g., in the previous graph, there are two different path names for node n5: */keys* and */home/steen/keys*

- symbolic link: representing an entity by a leaf node and instead of storing the address or state of the entity, the node stores an absolute path name



the concept of a symbolic link explained in a naming graph

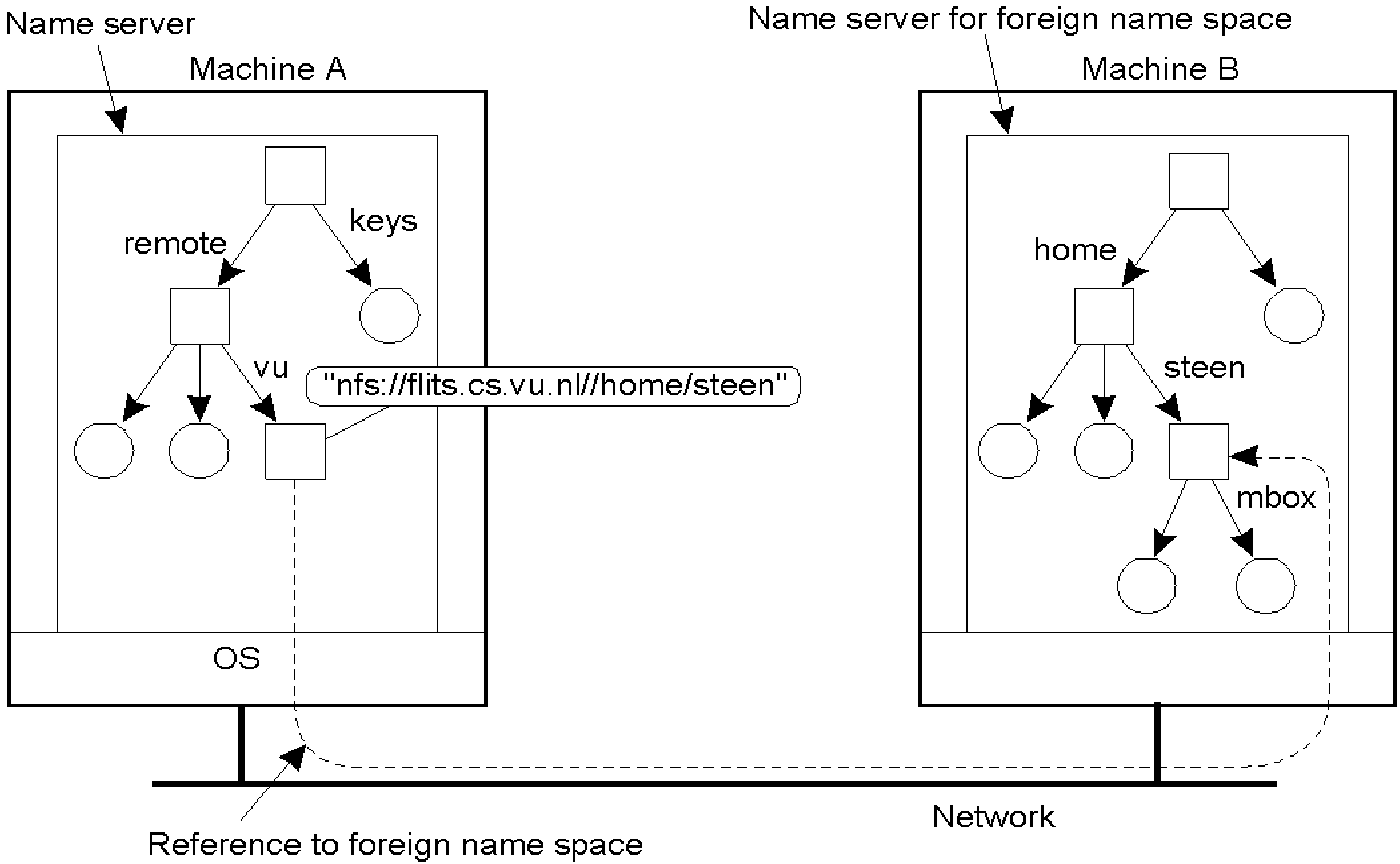
- when first resolving an absolute path name stored in a node (e.g., /home/steen/keys in node n6), name resolution will return the path name stored in the node (/keys), at which point it can continue with resolving that new path name

- so far name resolution was discussed as taking place within a single name space
- name resolution can also be used to merge different name spaces in a transparent way
- two methods: mounting and adding a new root node and making the existing root nodes its children

1. Mounting
    - as an example, consider a mounted file system, which can be generalized to other name spaces as well
    - let a directory node store the directory node from a different (foreign) name space
    - the directory node storing the node identifier is called a mount point
    - the directory node in the foreign name space is called a mounting point, normally the root of a name space
    - during name resolution, the mounting point is looked up and resolution proceeds by accessing its directory table

- consider a collection of name spaces distributed across different machines (each name space implemented by a different server)
- to mount a foreign name space in a DS, the following are at least required
  - the name of an access protocol (for communication)
  - the name of the server
  - the name of the mounting point
- each of these names needs to be resolved
  - to the implementation of the protocol
  - to an address where the server can be reached
  - to a node identifier in the foreign name space
- the three names can be listed as a URL

- example: Sun’s Network File System (NFS) is a distributed file system with a protocol that describes how a client can access a file stored on a (remote) NFS file server
  - an NFS URL may look like nfs://flits.cs.vu.nl/home/steen
    - nfs is an implementation of a protocol
    - flits.cs.vu.nl is a server name to be resolved using DNS
    - /home/steen is resolved by the server
  - e.g., the subdirectory /remote includes mount points for foreign name spaces on the client machine
    - a directory node named /remote/vu is used to store nfs://flits.cs.vu.nl/home/steen
    - consider /remote/vu/mbox
    - this name is resolved by starting at the root directory on the client’s machine until node /remote/vu, which returns the URL nfs://flits.cs.vu.nl/home/steen
    - this leads the client machine to contact flits.cs.vu.nl using the NFS protocol
    - then the file mbox is read in the directory /home/steen



mounting remote name spaces through a specific process protocol

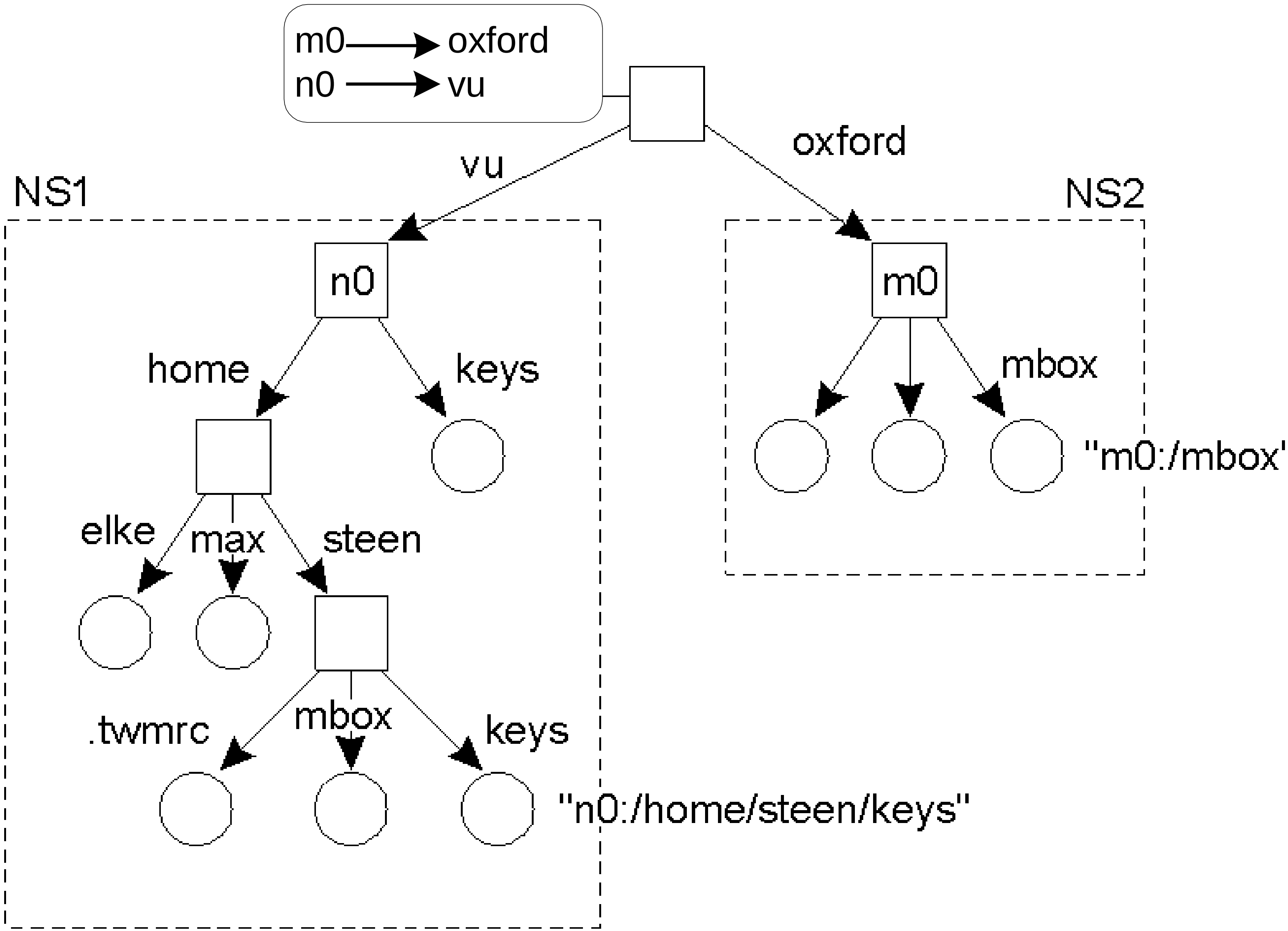
- distributed systems that allow mounting a remote file system also allow to execute some commands
- example commands to access the file system

  cd /remote/vu
- by doing so the user is not supposed to worry about the details of the actual access; the name space on the local machine and that on the remote machine look to form a single name space

2. Add a new root node and make the existing root nodes its children

- a method followed in GNS (Global Name Service by DEC)
- problem: existing names need to be changed

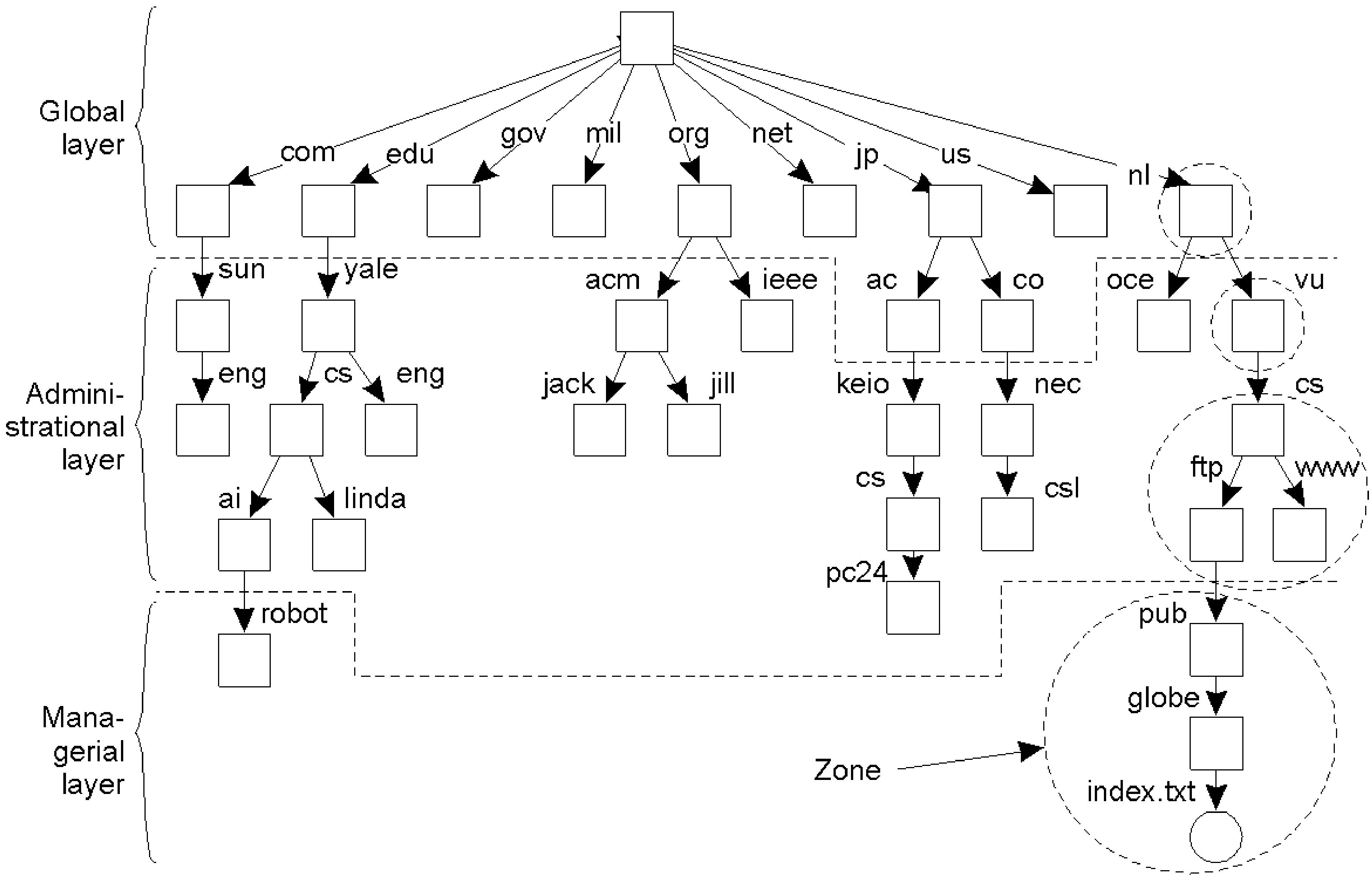
  e.g., the absolute path name /home/steen has now changed to a relative path name and corresponds to the absolute path name /vu/home/steen
- hence the system must expand no:/home/steen to /vu/home/steen without the awareness of users
- this requires storing a mapping table (with entries such as $n0 \rightarrow vu$) when a new root node is added
- merging thousands of name spaces may lead to performance problems



organization of the DEC Global Name Service

- The Implementation of a Name Space
  - a name space forms the heart of a naming service
  - a naming service allows users and processes to add, remove, and lookup names
  - a naming service is implemented by name servers
  - for a distributed system on a single LAN, a single server might suffice; for a large-scale distributed system the implementation of a name space is distributed over multiple name servers
- Name Space Distribution
  - in large scale distributed systems, it is necessary to distribute the name service over multiple name servers, usually organized hierarchically
  - a name service can be partitioned into logical layers
  - the following three layers can be distinguished (Cheriton and Mann)

- global layer
  - formed by highest level nodes (root node and nodes close to it or its children)
  - nodes on this layer are characterized by their stability, i.e., directory tables are rarely changed
  - they may represent organizations, groups of organizations, ..., where names are stored in the name space
- administrational layer
  - groups of entities that belong to the same organization or administrational unit, e.g., departments
  - relatively stable
- managerial layer
  - nodes that may change regularly, e.g., nodes representing hosts of a LAN, shared files such as libraries or binaries, …
  - nodes are managed not only by system administrators, but also by end users



an example partitioning of the DNS name space, including Internet-accessible files, into three layers

- the name space is divided into nonoverlapping parts, called zones in DNS
- a zone is a part of the name space that is implemented by a separate name server
- some requirements of servers at different layers
  - performance (responsiveness to lookups), availability (failure rate), etc.
  - high availability is critical for the global layer, since name resolution cannot proceed beyond the failing server; it is also important at the administrational layer for clients in the same organization
  - performance is very important in the lowest layer, since results of lookups can be cached and used due to the relative stability of the higher layers
  - they may be enhanced by client side caching (global and administrational layers since names do not change often) and replication; they create implementation problems since they may introduce inconsistency problems.

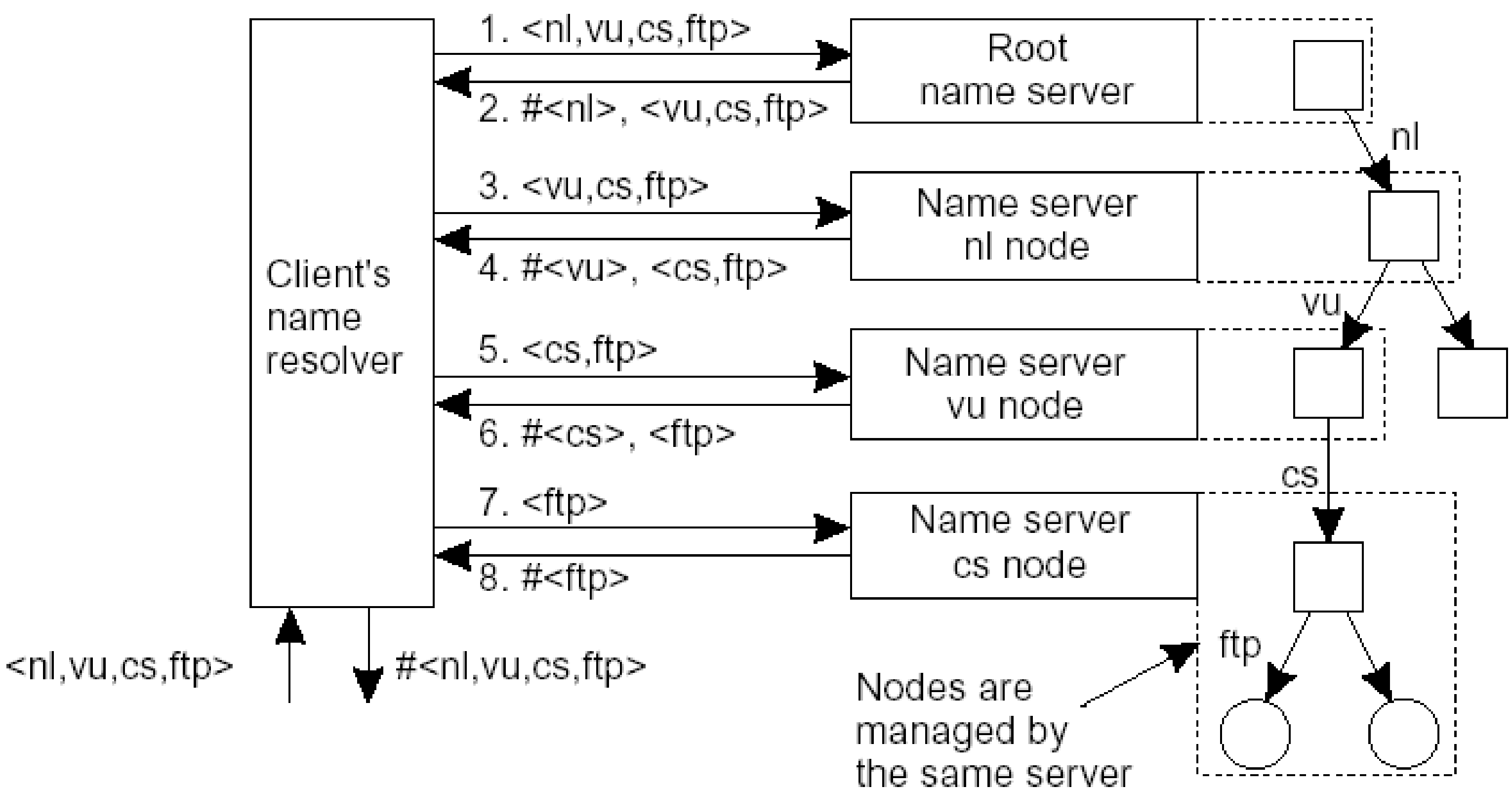
- Implementation of Name Resolution
  - recall that name resolution consists of finding the address when the name is given
  - assume that name servers are not replicated and that no client-side caches are allowed
  - each client has access to a local name resolver, responsible for ensuring that the name resolution process is carried out
  - e.g., assume the path name

    root:<nl, vu, cs, ftp, pub, globe, index.txt>

    is to be resolved

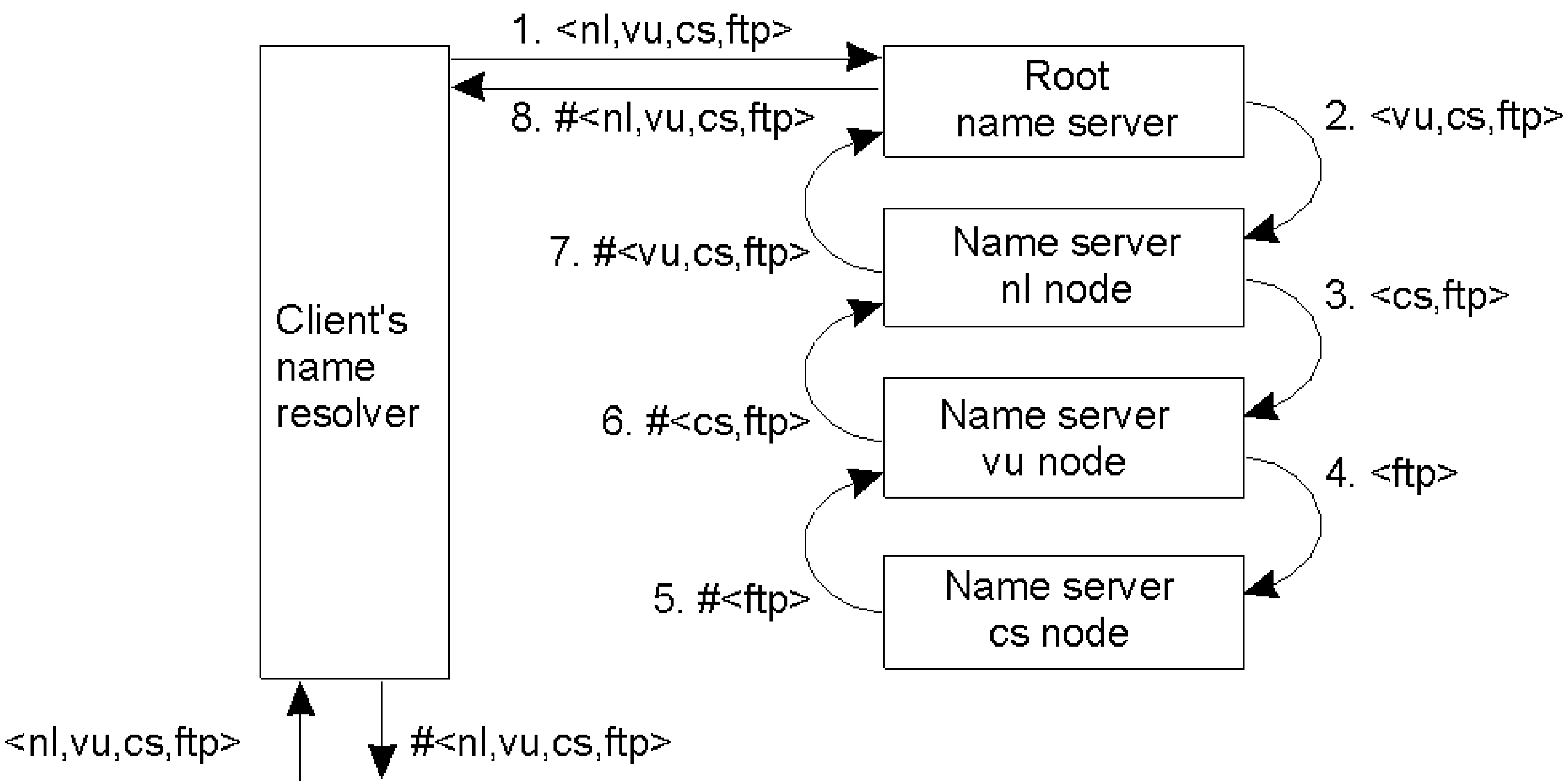
    or using a URL notation, this path name would correspond to
    ftp://ftp.cs.vu.nl/pub/globe/index.txt
  - two ways of implementing name resolution
    - iterative name resolution
    - recursive name resolution

- Iterative
  - a name resolver hands over the complete name to the root name server
  - the root server will resolve the name as far as it can and return the result to the client; at the minimum it can resolve the first level and sends the name of the first level name server to the client
  - the client calls the first level name server, then the second, ..., until it finds the address of the entity



the principle of iterative name resolution

- Recursive
  - a name resolver hands over the whole name to the root name server
  - the root server will try to resolve the name and if it can’t, it requests the first level name server to resolve it and to return the address
  - the first level will do the same thing recursively



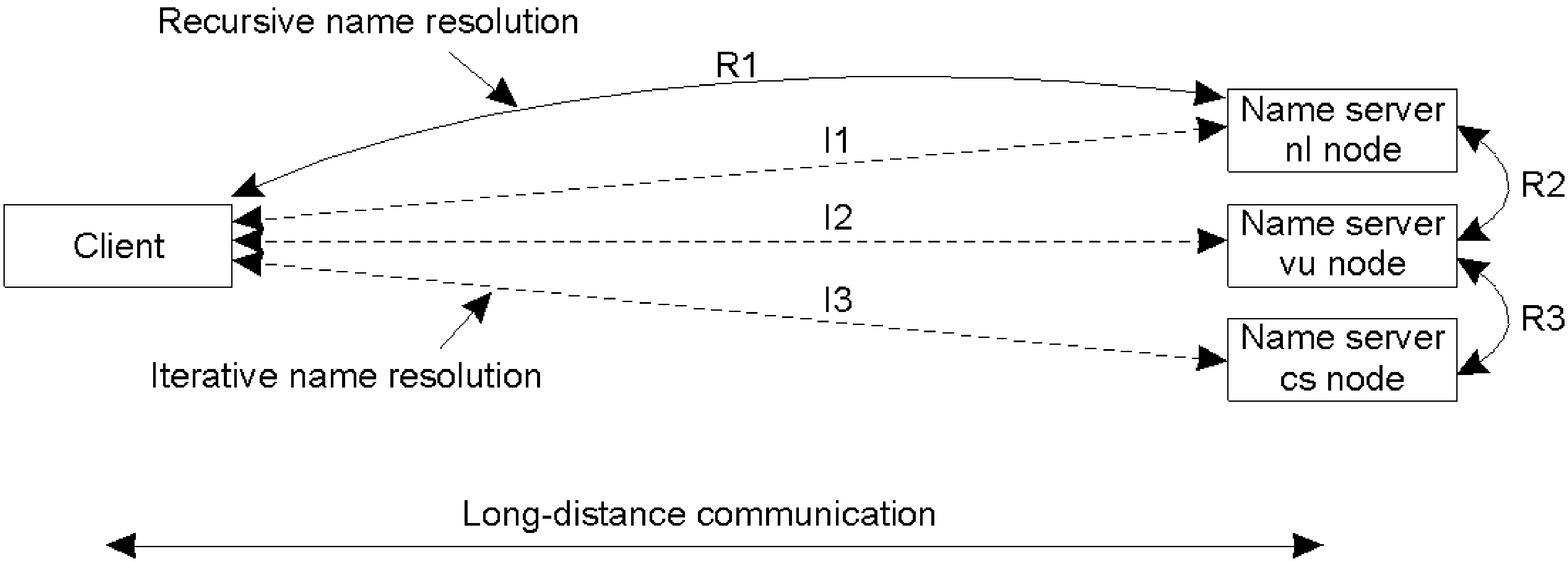
the principle of recursive name resolution

- Advantages and drawbacks
  - recursive name resolution puts a higher performance demand on each name server; hence name servers in the global layer support only iterative name resolution
  - caching is more effective with recursive name resolution; each name server gradually learns the address of each name server responsible for implementing lower-level nodes; eventually lookup operations can be handled efficiently

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>

recursive name resolution of <nl, vu, cs, ftp>; name servers cache intermediate results for subsequent lookups

- communication costs may be reduced in recursive name resolution



the comparison between recursive and iterative name resolution with respect to communication costs; assume the client is in Ethiopia and the name servers in the Netherlands

- Summary

Method	**Advantage(s)**
Recursive	Less Communication cost; Caching is more effective
Iterative	Less performance demand on name servers

- Example 1 - The Domain Name System (DNS)
  - one of the largest distributed naming services is the Internet DNS
  - it is used for looking up host addresses and mail servers
  - hierarchical, defined in an inverted tree structure with the root at the top
  - the tree can have only 128 levels