Fehér Gyula

Naming in Distributed Systems

TEMPUS S JEP-12495-97

1999

NAMING IN DISTRIBUTED SYSTEMS

INTRODUCTION

- The naming faciliy of a distributed operating system enables users and programs to assign character-string names to objects and subsequently use these names to refer to those objects.
- The locating faciliy, which is an integral part of the naming facility, maps an object's name to the object's location in a distributed system.
- The naming and locating facilities jointly form a naming system that provides the users with an abstraction of an object that hides the details of how and where an object is actually located in the network.
- It provides a further level of abstraction when dealing with object replicas. Given an object name, it returns a set of the locations of the object's replicas.
- The naming system plays a very important role in achieving the goal of
    - ❖ location transparency,
    - ❖ facilitating transparent migration and replication of objects,
    - ❖ object sharing.

DESIRABLE FEATURES OF A GOOD NAMING SYSTEM

1. Location transparency. Location transparency means that the name of an object should not reveal any hint as to the physical location of the object. That is, an object's name should be independent of the physical connectivity or topology of the system, or the current location of the object.

2. Location independency. For performance, reliability, availability, and security reasons, distributed systems provide the facility of object migration that allows the movement and relocation of objects dynamically among the various nodes of a system. Location independency means that the name of an object need not be changed when the object's location changes.

Furthermore, a user should be able to access an object by its same name irrespective of the node from where he or she accesses it (user migration).

Therefore, the requirement of location independency calls for a global naming facility with the following two features:

- ❖ An object at any node can be accessed without the knowledge of its physical location (location independency of request-receiving objects).
- ❖ An object at any node can issue an access request without the knowledge of its own physical location (location independency of request-issuing objects). This property is also known as user mobility.

3. Scalability. Distributed systems vary in size ranging from one with a few nodes to one with many nodes. Moreover, distributed systems are normally open systems, and their size changes dynamically.

Therefore, it is impossible to have an a priori idea about how large the set of names to be dealt with is liable to get. Hence a naming system must be capable of adapting to the dynamically changing scale of a distributed system that normally leads to a change in the size of the name space. That is, a change in the system scale should not require any change in the naming or locating mechanisms.

4. Uniform naming convention. In many existing systems, different ways of naming objects, called naming conventions, are used for naming different types of objects. For example, file names typically differ from user names and process names. Instead of using such nonuniform naming conventions, a good naming system should use the same naming convention for all types of objets in the system.

5. Multiple user-defined names for the same object. For a shared object, it is desirable that different users of the object can use their own convenient names for accessing it. Therefore, a naming system must provide the flexibility to assign multiple user-defined names to the same object. In this case, it should be possible for a user to change or delete his or her name for the object without affecting those of other users.

6. Group naming. A naming system should allow many different objects to be identified by the same name. Such a facility is useful to support broadcast facility or to group objects for conferencing or other applications.

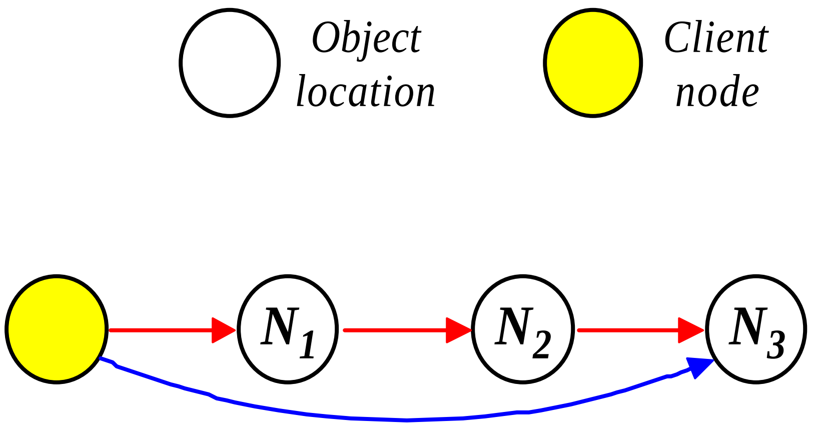
7. Meaningful names. A name can be simply any character string identifying some object. However, for users, meaningful names are preferred to lower level identifiers such as memory pointers, disk block numbers, or network addresses.

This is because meaningful names typically indicate something about the contents or function of their referents, are easily transmitted between users, and are easy to remember and use. Therefore, a good naming system should support at least two level of object identifiers, one convenient for human users and one convenient for machines.

8. Performance. The most important performance measurement of a naming system is the amount of time needed to map an object's name to its attributes, such as its location. In a distributed environment, this performance is dominated by the number of messages exchanged during the name-mapping operation. Therefore, a naming system should be efficient in the sense that the number of messages exchanged in a name-mapping operation should be as small as possible.

9. Fault tolerance. A naming system should be capable of tolerating, to some extent, faults that occur due to the failure of a node or a communication link in a distributed svstem network. That is, the naming system should continue functioning, perhaps in a degraded form, in the event of these failures. The degradation can be in performance. functionality, or both but should be proportional, in some sense, to the failures causing it.

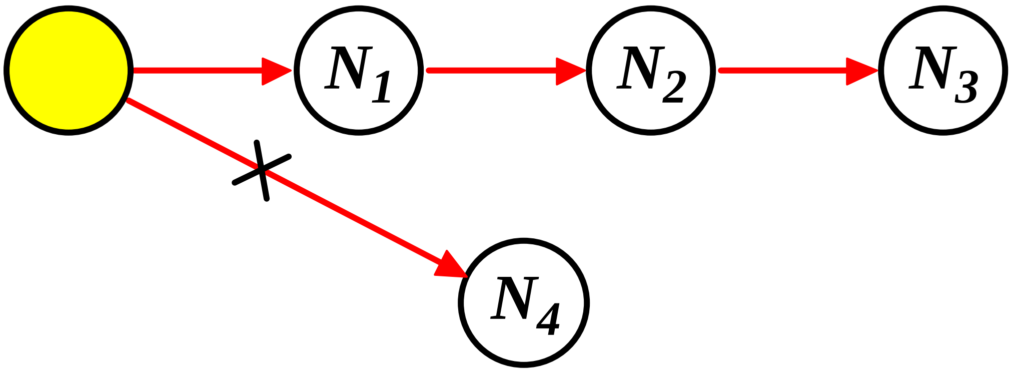
10. Replication transparency. In a distributed system, replicas of an object are generally created to improve performance and reliability. A naming system should support the use of multiple copies of the same object in a user-transparent manner. That is, if not necessary, a user should not be aware that multiple copies of an object are in use.



The cost is high if the object locating mechanism maps to node N_3 instead of node N_1.

11. Locating the nearest replica. When a naming system supports the use of multiple copies of the same object, it is important that the object-locating mechanism of the naming system should always supply the location of the nearest replica of the desired object. This is because the efficiency of the object accessing operation will be affected if the object-locating mechanism does not take this point into consideration.

This is illustrated by the example given below, where the desired object is replicated at nodes N_1, N_2, and N_3 and the object-locating mechanism is such that it maps to the replica at node N_3 instead of the nearest replica at node N_1. Obviously this is undesirable.



The importance of locating all the replicas.

FUNDAMENTAL TERMINOLOGIES AND CONCEPTS

- **Name**
- **Human-Oriented and System-Oriented Names**
- **Name space**
- **Flat Name Space**
- **Partitioned Name Space**
- **Name Server**
- **Name Agent**
- **Context**
- **Name Resolution**
- **Abbreviation/Alias**
- **Absolute and Relative Names**
- **Generic and Multicast Names**
- **Descriptive/Attribute-Based Name**
- **Source-Routing Name**

The naming system is one of the most important components of a distributed operating system because it enables other sewices and objects to be identified and accessed in a uniform manner. In spite of the imponance of names, no general unified treatment of them exists in the literature. This section defines and explains the fundamental terminologies and concepts associated with object naming in distributed systems.

Name

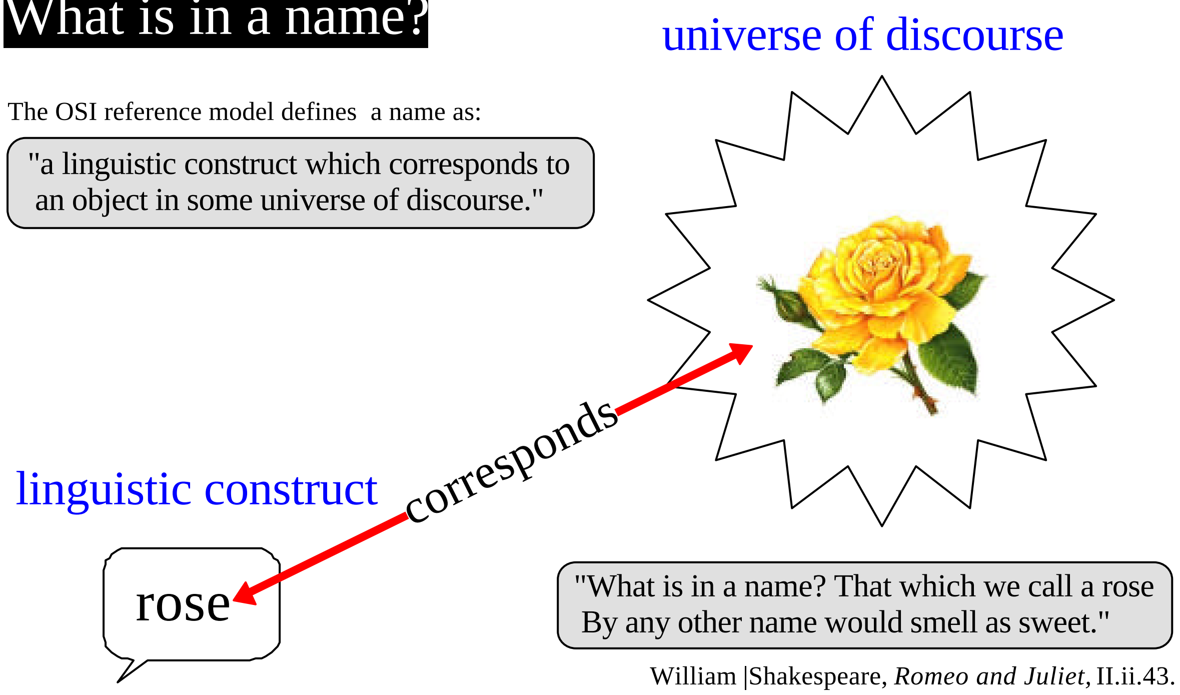
A name is a string composed of a set of symbols chosen from a finite alphabet. For example, TEMPUS, #173#4879#5965, node-1!node-2!node-3!sinha, /a/b/c, 25A2368DM197, etc. are all valid names composed of symbols from the ASCII character set. A name is also called an identifier because it is used to denote or identify an object.

A name may also be thought of as a logical object that identifies a physical object to which it is bound from among a collection of physical objects. Therefore, the correspondence between names and objects is the relation of binding logical and physical objects for the purpose of object identification.

What is in a name?

The OSI reference model defines a name as:

"a linguistic construct which corresponds to an object in some universe of discourse."



"What is in a name? That which we call a rose
By any other name would smell as sweet."

William |Shakespeare, *Romeo and Juliet*, II.ii.43.

Definitions

The OSI definition of a name is a very general one. It covers:

- ♣ Data items which identify objects by their location. Such names are called addresses.
- ♣ It also covers names which are assigned to objects. Such names are called titles.

Purpose

In the context of the name service, names allow users and the name service to communicate about objects.

Expectations

In the early days of naming standardisation, it was expected that the naming system would support "user-friendly naming" (*IFIP WG 6.5*).

By user-friendly naming we mean that it is easy for a user (specifically, a human user) to remember, guess or recognise a name for an object of interest.

The first *IFIP* document proposed a scheme wherein a name would be virtually any unordered collection of attribute (type, value) pairs which, following performance of a "name resolution algorithm" over the "global naming graph", turned out to describe one and only one object.

Problems of realisation.

Unfortunatelly, the naming scheme, based on unordered attribute (type, value) pairs, proved not to be efficiently implementable.

The fact that the components of names were unordered meant that, in the worst case, every naming system would have to be involved in performing the name resolution algorithm.

Solutions

It seems that there is trade-off between the user-friendliness of a naming scheme and the efficiency of its implementation. The naming system development has settled on a middle ground:

- ♣ Names can be formed from information about an object which a person would regard as a memorable and recognisable.
- ♣ On the other hand, once assigned, a name must be presented exactly by a would-be accessor. The various components of a name must appear in order. (This makes possible an efficient algorithm for determining whether a purpored nemeis actually a name or not.)

Name forms

Two forms of name are supported by the naming service:

- ♣ Distinguished names,
- ♣ Alias names (normally abbreviated to aliases.

(In principle other forms could be added in the future. However, there have been no significant extensions in this area during the last years.)

Names of both forms are constructed from attributes of relevant name system (directory) entries. More specifically, each such name is an ordered list of realtive distinguished names (RDNs). Every entry has exactly one RDN, a set of attribute (type, value) pairs from the entry chosen.

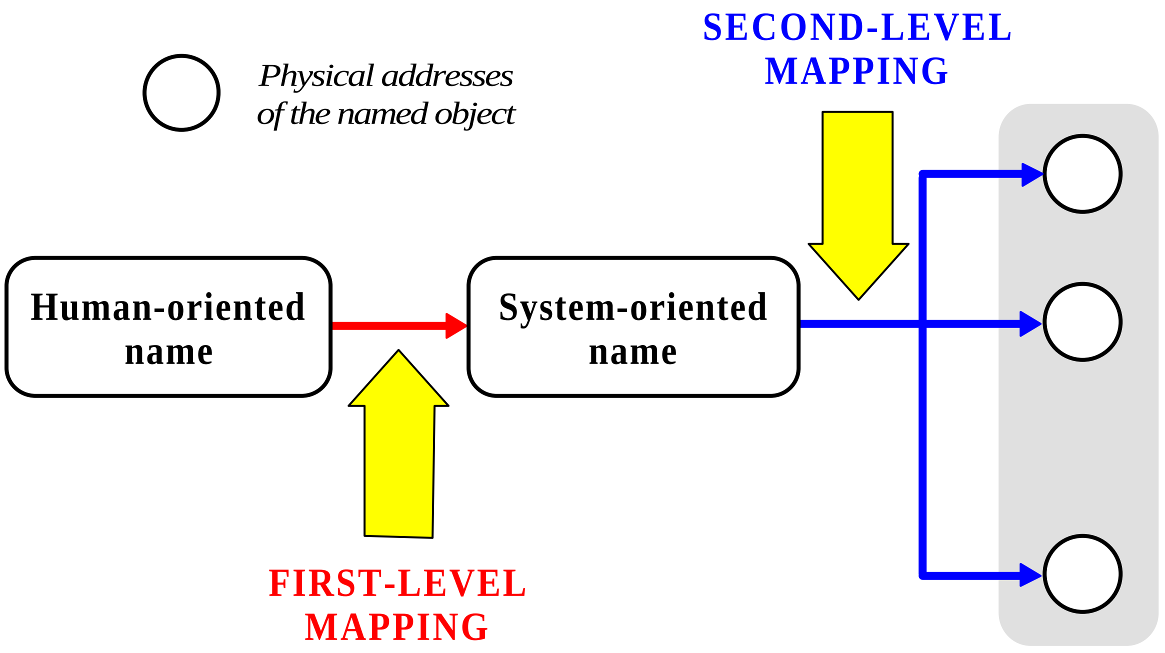
Human-Oriented and System-Oriented Names

Names are used to designate or refer to objects at all levels of system architecture. They have various purposes, forms, and properties depending on the levels at which they are defined. However, an informal distinction can be made between two basic classes of names widely used in operating systems human-oriented names and system-oriented names.

A human-oriented name is generally a character-string that is meaningful to its users. For example, /user/sinha/project-file-1 is a human-oriented name. Human-oriented names are defined by their users. For a shared object, different users of the object must have the flexibility to define their own hunan-oriented names for the object for accessing it. Flexibility must also be provided so that a user can change or delete his or her own name for the object without affecting those of other users. For transparency, human-oriented names should be independent of the physical location or the structure of objects they designate. Human-oriented names are also known as high-level names because they can be easily remembered by their users.

Human-oriented names are not unique for an object and are normally variable in length not only for different objects but also for different names for the same object. Hence, they cannot be easily manipulated, stored, and used by the machines for identification purpose. Moreover, it must be possible at some level to uniquely identify every object in the entire system.

Therefore, in addition to human-oriented names, which are useful for users, system-oriented names are needed to be used efiiciently by the system. These names generally are bit patterns of fixed size that can be easily manipulated and stored by machines. They are automatically generated by the system. They should be generated in a distributed manner to avoid the problems of efficiency and reliability of a centralized unique identifier generator. They are basically meant for use by the system but may also be used by the users. They are also known as unigue identifiers and low-level names.



A simple naming model based on the use of human-oriented and system-oriented names in a distributed system.

Figure above shows a simple naming model based on these two types of names. In this naming model, a human-oriented name is first mapped (translated) to a system-oriented name that is then rnapped to the physical locations of the corresponding object's replicas.

Name space

A naming system employs one or more naming conventions for name assignment to objects.

For example, a naming system may use one naming convention for assigning human-oriented names to objects and another naming convention for assigning system-oriented names to objects. The syntactic representation of a name as well as its semantic interpretation depends on the naming convention used for that name. The set of names complying with a given naming convention is said to form a name space

Flat Name Space

The simplest name space is a flat name space where names are character strings exhibiting no structure. Names defined in a flat name space are called primitive or flat names. Since flat names do not have any structure, it is difficult to assign unambiguous meaningful names to a large set of objects. Therefore, flat names are suitable for use either for small name spaces having names for only a few objects or for system-oriented names that need not be meaningful to the users.

Partitioned Name Space

When there is a need to assign unambiguous meaningful names to a large set of objects, a naming convention that partitions the name space into disjoint classes is normally used.

When partitioning is done syntactically, which is generally the case, the name structure reflects physical or organizational associations. Each partition of a partitioned name space is called a domain of the name space.

Each domain of a partitioned name space may be viewed as a flat name space by itself, and the names defined in a domain must be unique within that domain. However, two different domains may have a common name defined within them.

A name defined in a domain is called a simple name. In a partitioned name space, all objects cannot be uniquely identified by simple names, and hence compound names are used for the purpose of unique identification.

A compound name is composed of one or more simple names that are separated by a special delimiter character such as /, $, @, 9~, and so on (following the UNIX file system convention, the delimiter character / will be used for the examples and discussion presented here). For example, /a/b/c is a compound name consisting of three simple names a, b, and c.

A commonly used type of partitioned name space is the hierarchical name space, in which the name space is partitioned into multiple levels and is structured as an inverted tree.

Each node of the name space tree corresponds to a domain of the name space. In this type of name space, the number of levels may be either fixed or arbitrary.

For instance, the Grapevine system manages a name space tree having two levels, while in the Xerox Clearinghouse, a name space tree has three levels.

On the other hand, the DARPA Internet Domain Naming System and the Universal Directory Service allow a name space tree to have arbitrarily many levels. Names defined in a hierarchical name space are called hierarchical names.

Hierarchical names have been used in file systems for many years and have recently been adopted for naming other objects as well in distributed systems. Several examples of hierarchical name spaces are also found in our day-to-day life.

For instance, telephone numbers fully expanded to include country and area codes form a four-level hierarchical name space and network addresses in computer networks form a three-level hierarchical name spaee where the three levels are for network number, node number, and socket number.

Distinguished names

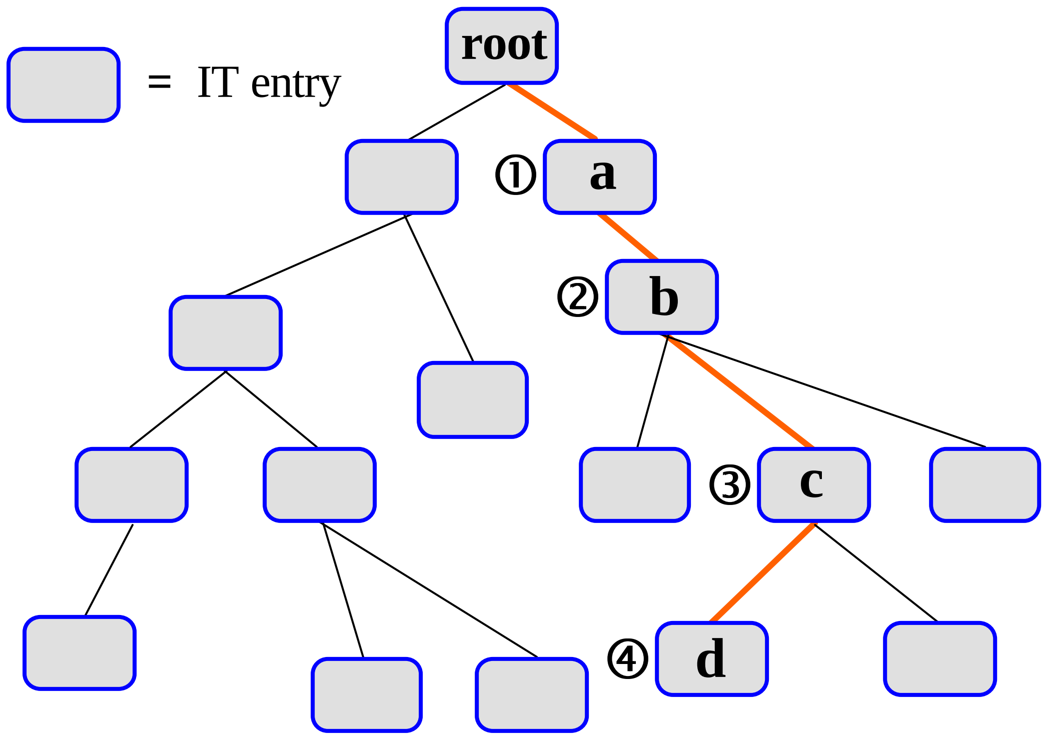
While every name is unambiguous, distinguished names (DNs) are also unique, each object having precisely one.

The distinguished name of an object, is a sequence of RDNs of the object's entry and all of its superiors, starting from the root.

Because the naming data base is arranged as a tree, there is a unique path from the root to the object's entry (which does not involve alias entries.)

Distinguished name as an IT path.

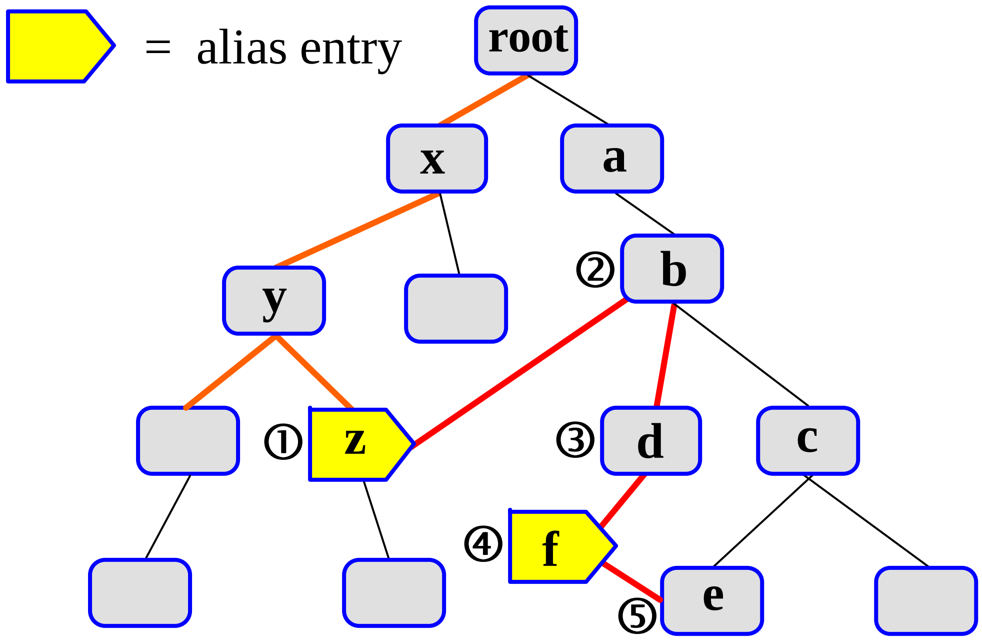
While every name is unambiguous, distinguished names (DNs) are also unique. (There is a unique path from the root to the object's entry.)



If, in the figure, the RDNs of entries ①,②,③ and ④ are represented by **a,b,c** and **d** respectivelly, then, for example, the distinguished name of entry ③ would be the sequence: [**a,b,c**].

Alias name as an IT path.

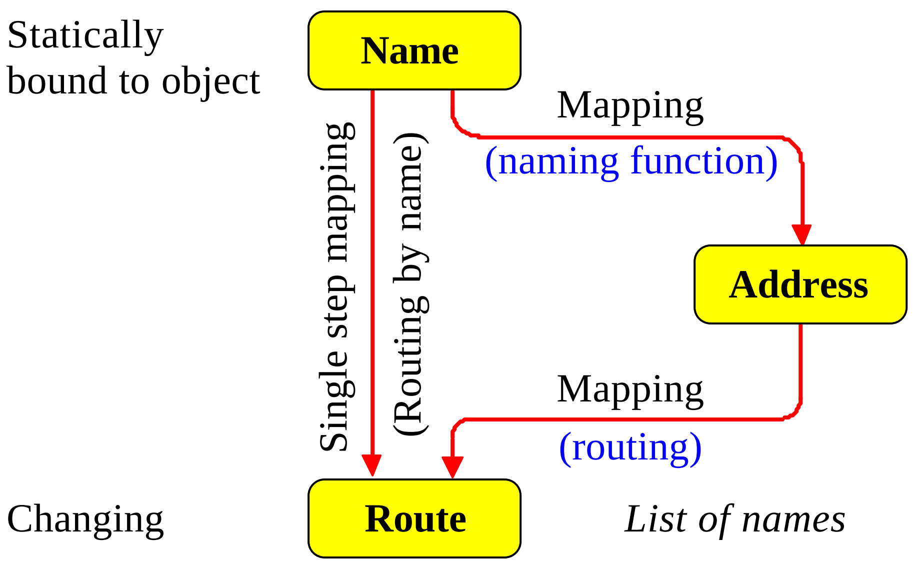
Like a distinguished name (DN), alias is a path through the IT from the root to the object's entry. However, unlike the DN, an alias name involves one or more alias entries:



In the figure, two alias entries are depicted, ① and ④. As a result, there are alternative paths to several of the entries, including one to entry ⑤ which involves both alias entries. This path is shown by the red line. The alias corresponding to this path consists of the list of the RDNs encountered along it, excluding those of the entries pointed directly by the alias entries. This name for ⑤ is [**x**,**y**,**z**,**d**,**f**].

	Attribute		
	Structure	*Time*	*Number*
Name	Primitive Partitioned Descriptive	Static Dynamic	Individual Group
Mapping (Routing)	*Scheme*	*Function*	*Time*
	Source Hop-by-hop Broadcast	Centralized Distributed	Static Dynamic
Route	List of names		

Names, routes and routing.



A name, an address and a route.

Distribution of the naming information .

Conceptually, the naming/directory abstract service is provided by the directory as a whole. In reality, the directory is a collection of individual computer systems, directory system agents (DSAs), which cooperate to provide the service.

The services has been designed so that its users don't need to know how the directory is disributed in order to use it.

There are various ways that the directory could be distributed to meet this requirement.

Two extreme possibilities, neither of which is remotely feasible for the global solution, although practical in a single domain, are total centralisation (having a single DSA hold all of the information the DIB) and total replication (having it replicated on every DSA).

Any practical solution for the global naming system will require each DSA to hold only some of the DIB and to route requests which involve other information to the DSAs which hold it.

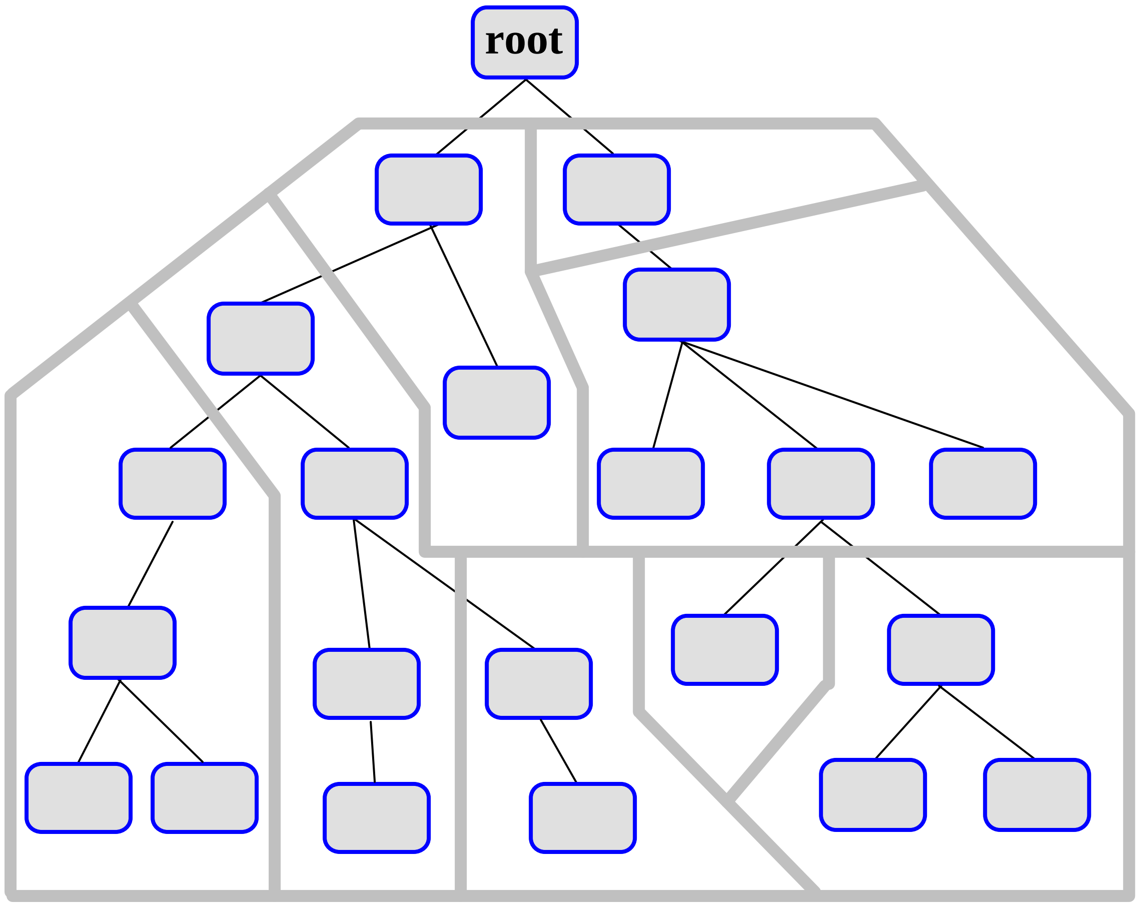
Naming contexts.

Naming contexts are the primary units of naming information distribution. Each DSA holding one or more of them.

A naming context is a set of entries constituting a subtree of the DIT. It includes at least one entry, the root of the subtree.

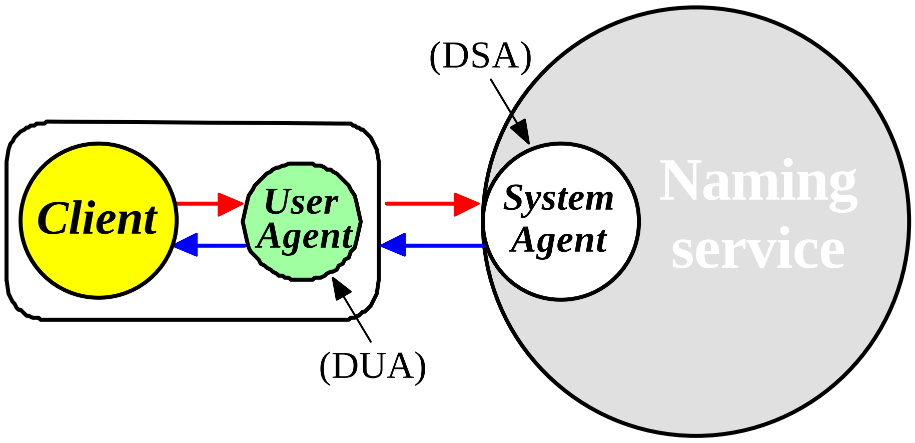
A naming context is partially described by the distinguished name of its root entry. This name is called the *context prefix* because it is a prefix to the distinguished names of each entry in the naming context.

Every entry is in one, and only one, naming contexts, so that the naming contexts partition the DIT. Such partition is depicted:



Service

The naming solution, introduced earlier, shows one object, the naming system itself, providing service to other objects, the users, via directory user agents (DUAs). This service is called the *naming abstract service*.



The abstract service is so called because it is capable of being realised in many ways. The realisation which the directory standard supports directly is the directory access protocol (DAP), an application protocol which allows the DUA and the naming/directory service to be in different computer systems.

The abstract service is quite simple one, intended to provide the building blocks which designers of particular applications and DUA software cab use to build the higher level services which their customers need. The service is defined in terms of a number of operations which the naming/directory can perform at the DUA request.

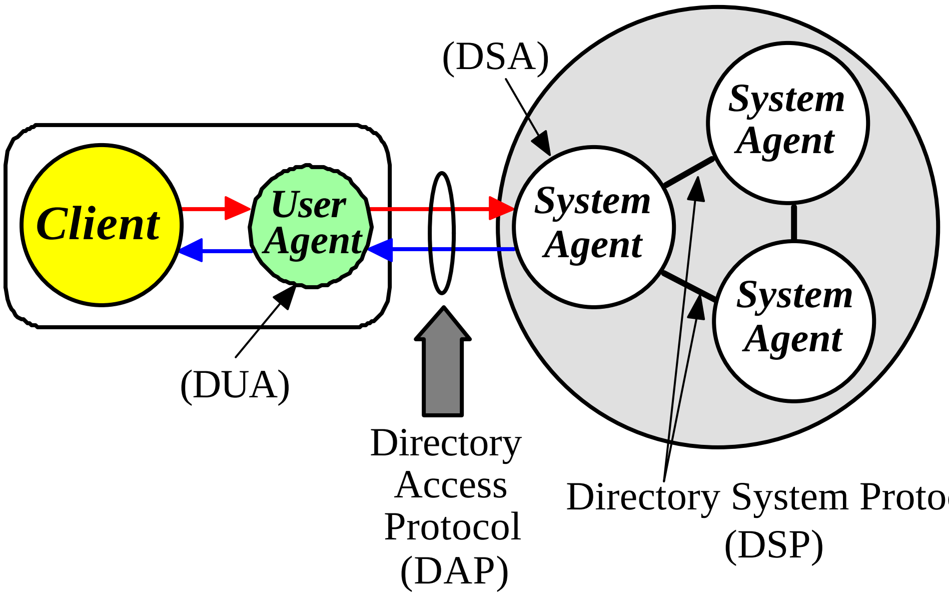
Abstract service

The abstract service is so called because it is capable of being realised in many ways. The realisation which the directory standard supports directly is the directory access protocol (DAP), an application protocol which allows the DUA and the naming/directory service to be in different computer systems.

The abstract service is quite simple one, intended to provide the building blocks which designers of particular applications and DUA software cab use to build the higher level services which their customers need. The service is defined in terms of a number of operations which the naming/directory can perform at the DUA request.

Functional model of the distributed naming system

The naming information-base is distributed throughout the word-wide collection of directory system agents that form the whole directory.


(DSA)
Client
User Agent
System Agent
System Agent
System Agent
(DUA)
Directory Access Protocol (DAP)
Directory System Protocol (DSP)

Name Server

Name spaces are managed by name servers. A name server is a process that maintains information about named objects and provides facilities that enable users to access that information. It acts to bind an object's name to some of its properties, including the object's location.

In practice, several name servers are normally used for managing the name space of object names in a distributed system.

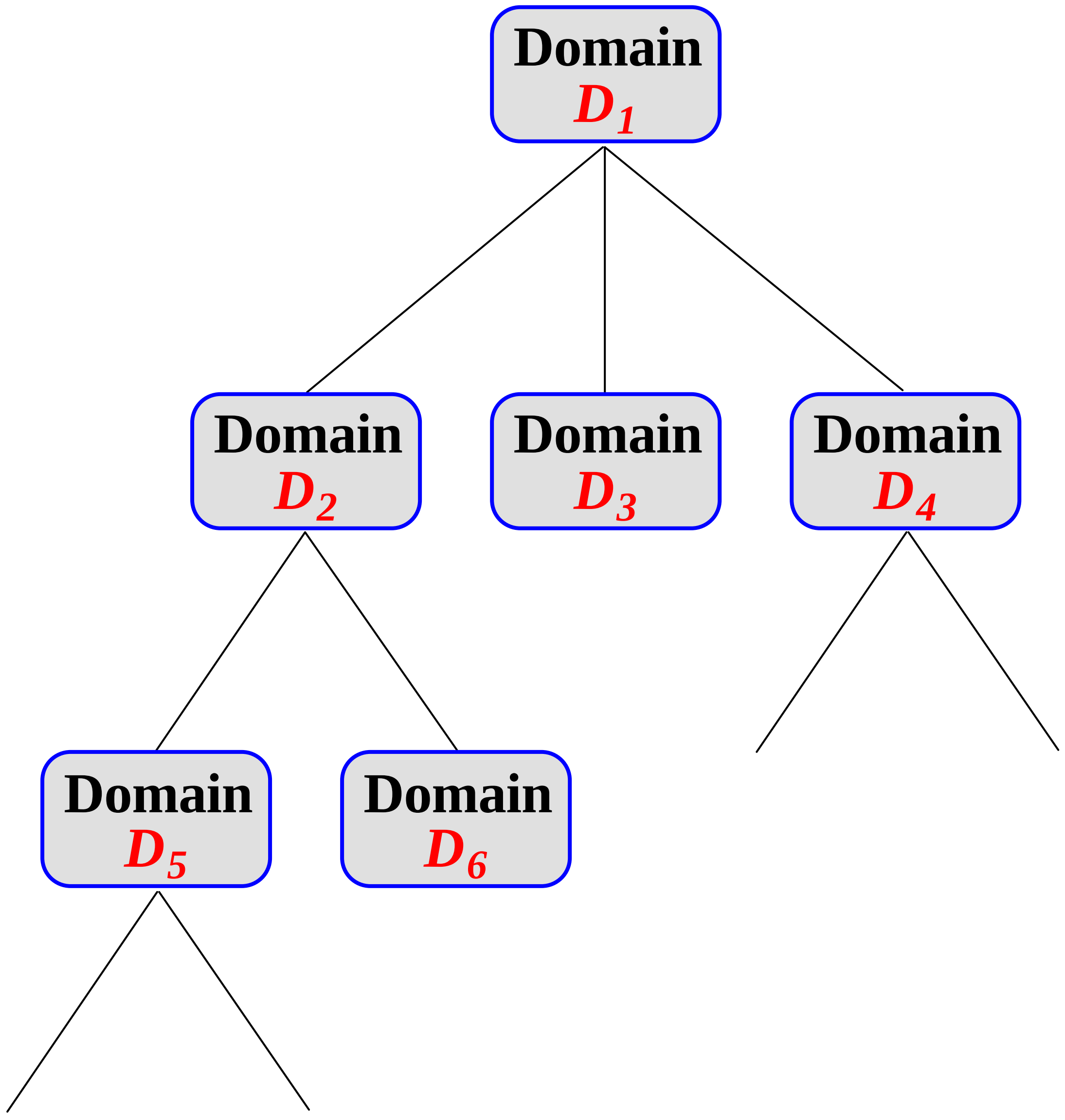
Each name server normally has information about only a small subset af the set of objects in the distributed system. The name servers that store the informatlon about an object are called the authoritative name servers of that object.

To determine the authoritative name servers for every named object, the name service maintains authority attributes that contain a list of the authoritative name servers for each object.

Partitioned name spaces are easier to manage efficiently as compared to flat name spaces because they enable the amount of configuration data required in each name server to be reduced since it need only be maintained for each domain and not for each individual object.

For example, in a hierarchical name space, it is sufficient that each name server store only enough information to locate the authoritative narne servers for the root domain of the name tree.

The authoritative name servers of the root domain, in turn, should know the locations of the authoritative name servers of the domains that branch out from the root domain. In general, the authoritative name servers of a domain should know the locations of the authoritative name servers of only those domains that branch out from that domain.



Domains of a hierarchical name space

For example, in the name space tree of Figure above all name servers must know the locations of the authoritative name servers of domain D_1; the authorative name servers of domain D_1 need only know the locations of the authoritative name servers of domains D_2, D_3, and D_4; and the authoritative name servers of domain D_2 need only know the locations of the authoritative name servers of domains D_5 and D_6.

The amount of configuration data that must be maintained by name servers at the various levels of the hierarchy is proportional to the degree of branching of the name space tree. For this reason, hierarehical naming conventions with several levels are often better suited for naming large number of objects.

Context

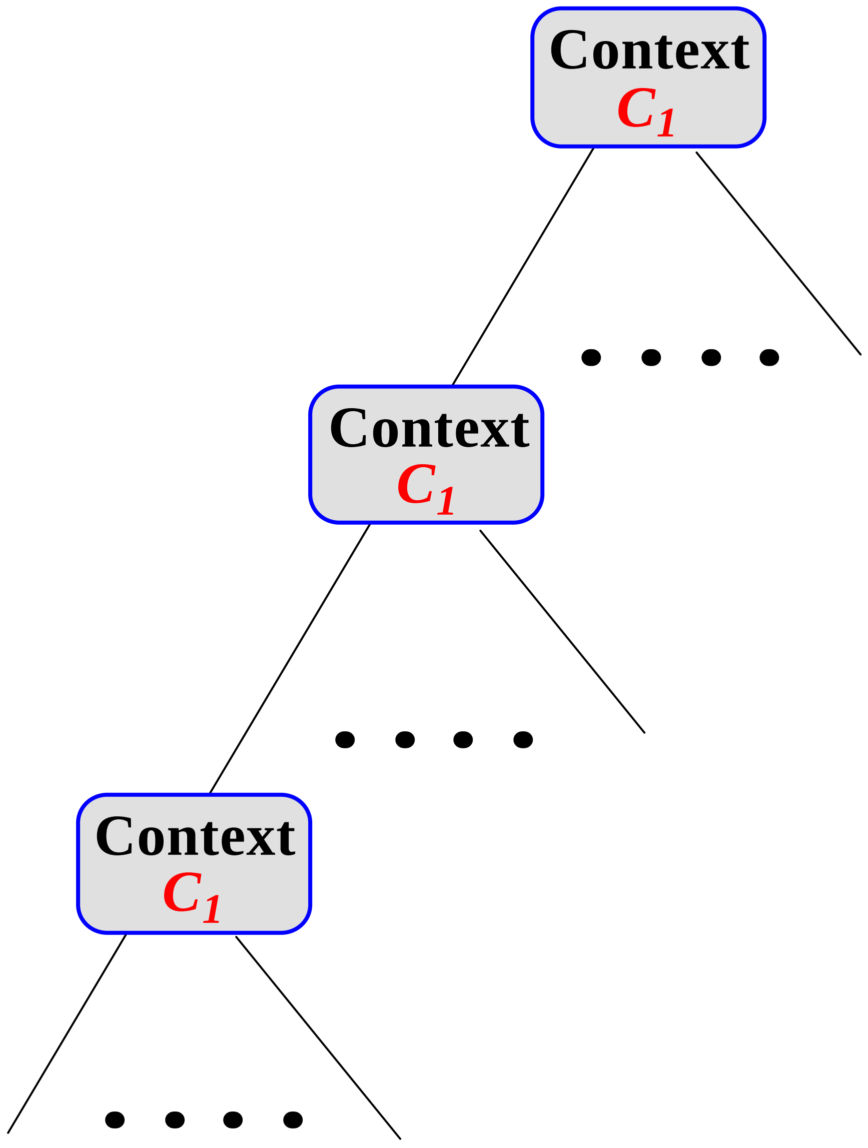
Names are always associated with some context. A context can be thought of as the environment in which a name is valid.

Because all names are interpreted relative to some context, a context/name pair is said to form a gualified name that can be used for uniquely identifying an object.

The notion of context has proved to be very useful for partitianing a name space into smaller components. Often, contexts represent a division of the name space along natural geographical, organizational, or functional boundaries.

In a partitioned name space, each domain corresponds to a context of the name space. Names in a context can be generated independently of what names exist in any other context.

Therefore, a name may occur in more than one context. Contexts rnay also be nested, as in the case of hierarchical name spaces. For example. in the name space tree of figure below,



Nested context

context C_3 is nested within context C_2, which in turn is nested within context C_1. In nested contexts, a qualified name consists of a series of names identifying, respectively, a context, a subcontext, a sub-subcontext followed by a name inside the last sub-sub- . . . context.

For the purpose of name management, contexts provide a means of partitioning the naming information database so that it may be distributed among multiple name servers.

Contexts represent indivisible units for storage and replication of information regarding named objects.

Name Resolution

Name resolution is the process of mapping an object's name to the object's properties ,such as its location.

Since an object's properties are stored and maintained by the authoritative name servers of that object, name resolution is basically the process of mapping an object's name to the authoritative name servers of that object.

Once an authoritative name server of the object has been located, operations can be invoked to read or update the object's properties.

Each name agent in a distributed system knows about at least one name server apriori.

To get a name resolved, a client first contacts its name agent, which in turn contacts a known name server, which may in turn contact other name servers.

Absolute and Relative Names

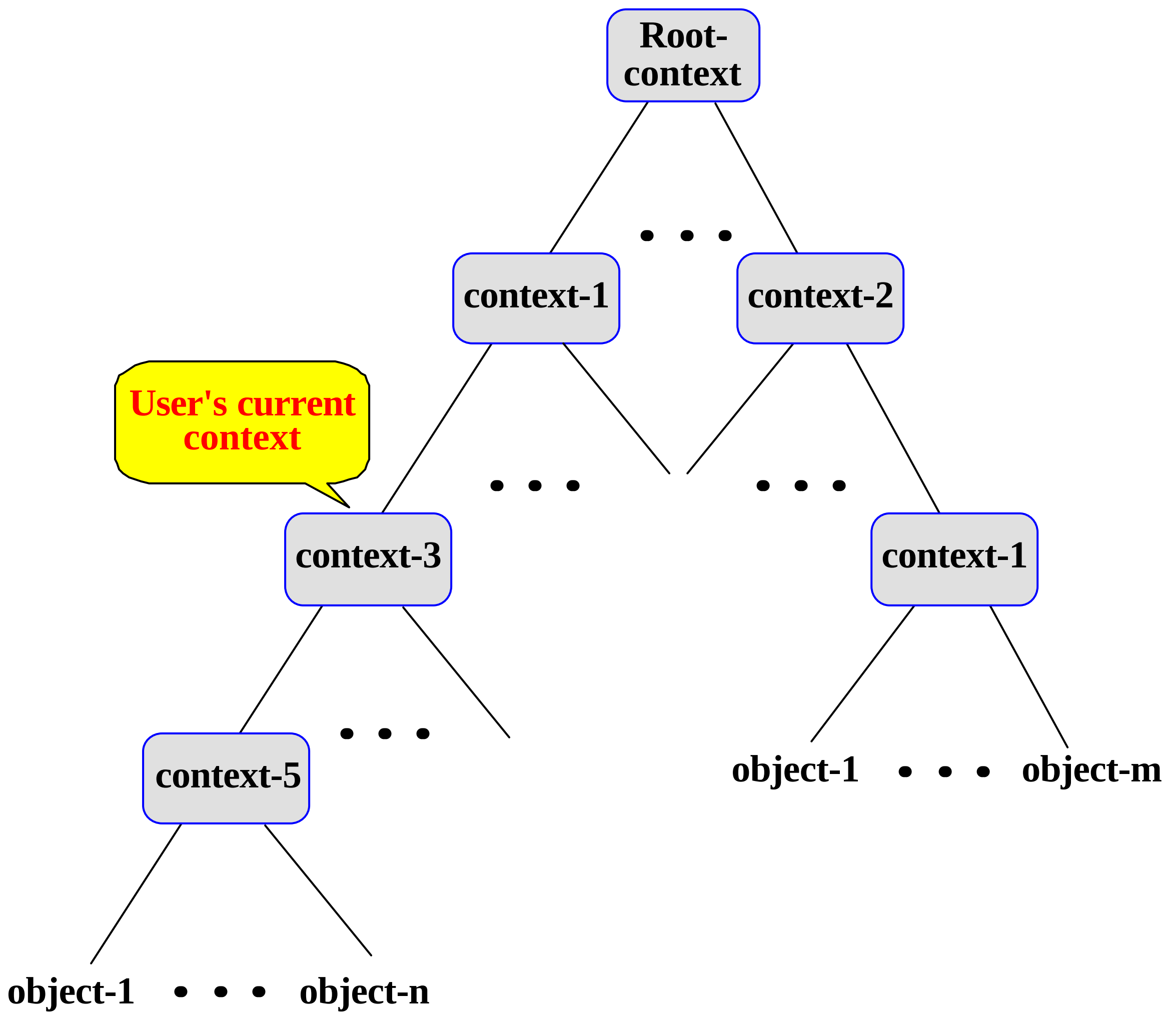
Another method to avoid the necessity to specify the full qualified name of an object in tree-structured name spaces is the use of the concept of current working context.

A current working context is also known by the shorter names current context or working context. According to this concept, a user is always associated with a context that is his or her current context. A user can change his or her current context whenever he or shedesires.

An absolute name begins at the root context of the name space tree and follows a path down to the specified object, giving the context names on the path.

On the other hand, a relative name defines a path from the current context to the specified object. It is called a relative name because it is "relative to” (start from) the user's current context.

For example, in the tree-structured name space of figure below,



A tree-structured name space
to illustrate absolute and relative names.

if the user's current eontext is

root-context/context-l/context-3,

the relative name

context-5/object-1

refers to the same object as does the absolute name

root-context/context-1/context-3/context-/object-1.

Means are also provided in this method to refer to the parent context of a context. For example, in the UNIX file-naming convention.

In this method, a user may specify an object in any of the following ways:

1. Using the full (absolute) name
2. Using a relative name '
3. Changing the current context first and then using a relative name

Generic and Multicast Names

The use of generic and multicast names requires the naming system'to support one-to-many bindings. That is, the naming system must allow a simple name to be bound to a set of qualified names.

In a generic naming facility, a name is mapped to any one of the set of objects to which it is bound. This type of facility is useful in situations such as when a user wants a request to be serviced by any of the servers capable of servicing that request and the user is not concerned with which server services his or her request.

SYSTEM-ORIENTED NAMES

- **Centralized Approach for Generating**
- **System-Oriented Names**
- **Distributad Approach for Generating**
- **System-Oriented Names**
- **Generating Unique Identifiers in the Event of Crashes**

OBJECT LOCATING MECHANISMS

- Broadcasting
- Expanding Ring Broodcast
- Encoding location of Object within Its UID
- Searching Creator Node First and Than Broadcasting
- Using Forward Location Pointers
- Using Hint Cache and Broadcasting

Object locating is the process of mapping an object's system-oriented unique identifier (UID for short) to the replica locations of the object.

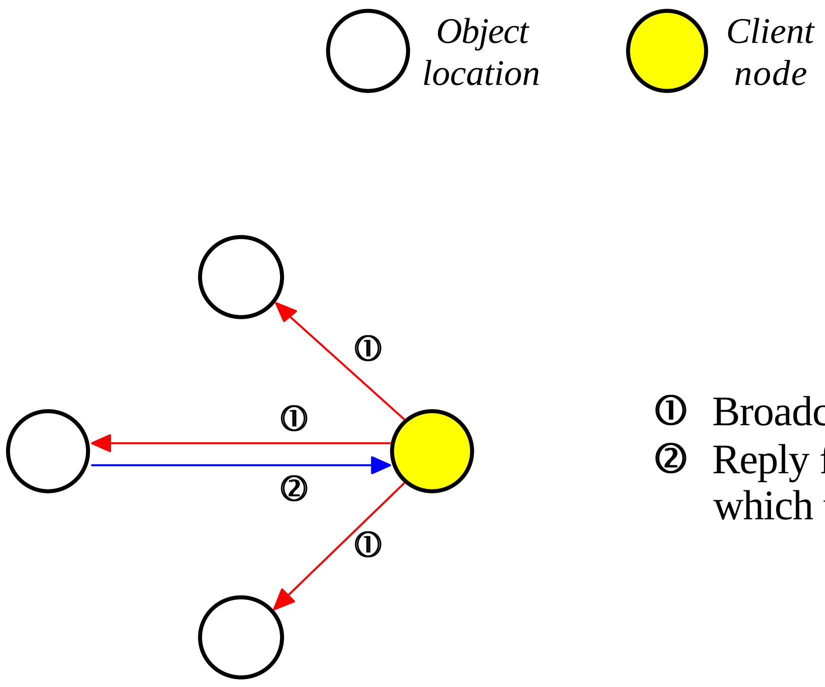
In a distributed system, object locating is only the process of knowing the object's location, that is, the node on which it is located.

On the other hand, object accessing involves the process of carrying out the desired operation (e.g., read, write) on the object. Therefore, the object-accessing operation starts only after the object-locating operation has been carried out successfully.

Several object-locating mechanisms have been proposed and are being used by various distributed operating systems.

Broadcasting

In this method, an object is located by broadcasting a request for the object from a client node. The request is processed by all nodes and then the nodes currently having the object reply back to the client node. Amoeba uses this method for locating a remote port.



① Broadcast request message.
② Reply from the node on which the object is located.

Broadcasting object-locating mechanism

The method is simple and enjoys a high degree of reliability because it supplies all replica locations of the target object.

However, it suffers from the drawbacks of poor efficiency and scalability because the amount of network traffic generated for each request is directly proportional to the number of nodes in the system and is prohibitive for large networks.

Therefore, this method is suitable only when the number of nodes is small, communication speed is high, and object-locating requests are not so frequent.

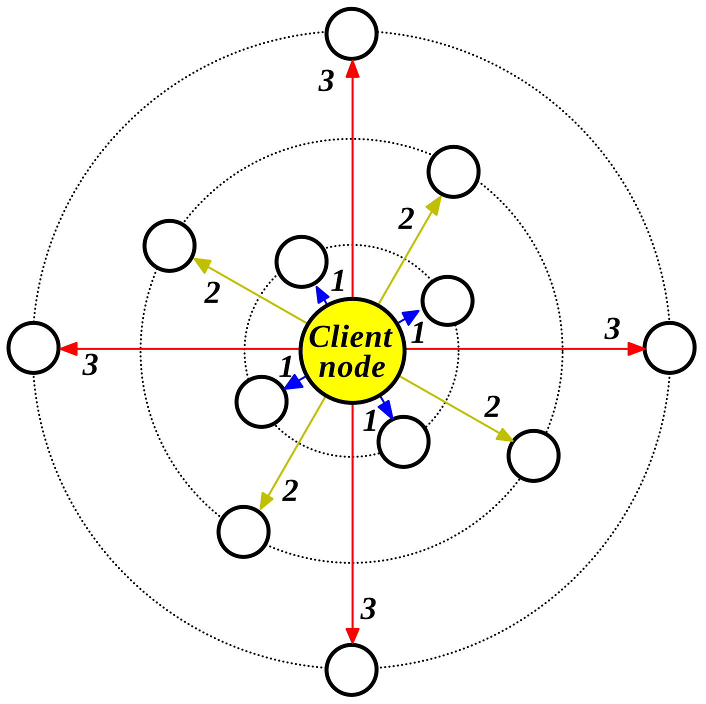
Expanding Ring Broodcast

Pure broadcasting is expensive for large networks. Moreover, direct broadcasting to all nodes may not be supported by wide-area networks. Therefore, a modified form of broadcasting, called expanding ring broadcast is normally employed in an internetwork that consists of local area networks (LANs) connected by gateways.

In this method, increasingly distant LANs are systematically searched until the object is found or until every LAN has been searched unsuccessfully.

The distance metric used is a hop. A hop corresponds to a gateway between processors.

For example, if a message from processor A to processor B must pass through at least two gateways, A and B are two hops distant. Processors on the same LAN are zero hop distant. A ring is the set of LANs a certain distance away from a processor. Thus, Ring_0[A] is A's local network, Ring_1[A] is the set of LANs one hop away, and so on.



① Searching nodes at 0 hope distance.
② Searching nodes at 1 hope distance if the search of 0 hop fails.
③ Searching nodes at *i* hope distance if the searches up to *i* -1 hopes fail.

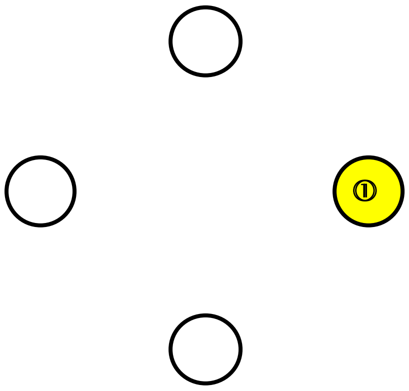
Broadcasting object-locating mechanism

An ERB search works as follows. Suppose that processor A needs to locate object X. Beginning with *i* = 0, a request message is broadcast to all LANs in Ring$_0$[A]. If a response is received, the search ends successfully. Otherwise, after a timeout period has elapsed, *i* is incremented by 1 and the request broadcast is repeated. The ring size *i* is bounded from above by the diameter of the internetwork.

This method does not necessarily supply all the replica locations of an object simultaneously, although it does supply the nearest replica location.

The efficiency of an object-loeating operation is directly proportional to the distance of the object from the client node at the time of locating it.

Encoding location of Object within Its UID



① Extracting object's location from its UID. No message exchange with any other node is required for locating the object.

Object-locating mechanism encods the location of an object within its UID.

This scheme uses structured object identifiers. One field of the structured UID is the location of the object. Given a UID, the system simply extracts the corresponding object's location from its UID by examining the appropriate field of the structured UID. The extracted location is the node on which the object resides.

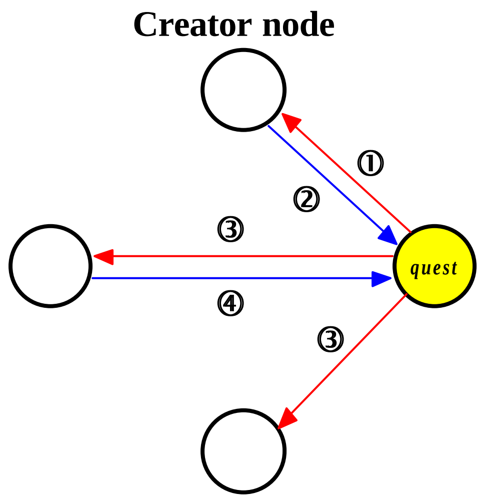
This is a straightforward and efficient scheme. One restriction of the scheme.however, is that an object is not permitted to move once it is assigned to a node, since this would require its identifier to change. Consequently, an object is fixed to one node throughout iks lifetime.

Another limitation of the scheme is that it is not clear how to support multiple replicas of an object. Therefore, the use of this object-locating scheme is limited to those distributed systems that do not support object anigration and object replication. ,

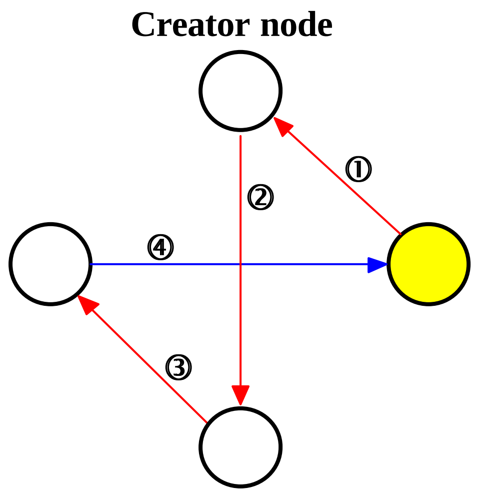
Searching Creator Node First and Than Broadcasting

This scheme is a simple extension of the previous scheme. The included extension is basically meant for supporting object migration facility. The method is based on the assumption that it is very likely for an object to remain at the node where it was created (although it may not be always true). This is because object migration is an expensive operation and objects do not migrate frequently.

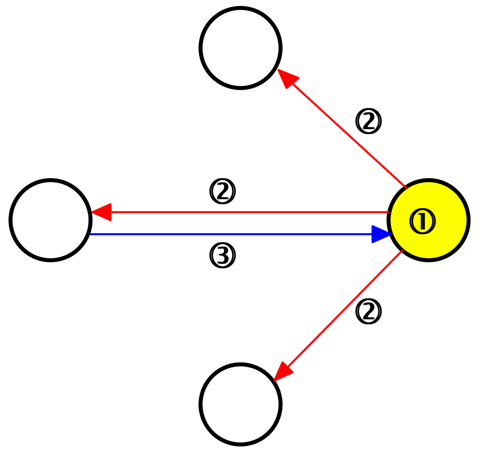
In this scheme. the location field of the structured UID contains the identifier of the node on which the object was created. Given a UID, the creator node information is extracted from the UID and a request is sent to that node. If the object no longer resides on its creator node, a failure reply is returned back to the client node. In case of failure, the object is located by broadcasting the request from the client node. This method of object locating is used in.



① Searching the creator node.
② Negativ reply.
③ Broadcast request message.
④ Reply from the node on which the object is located.



①, ②, ③ Path of message forwarding.
④ Reply from the node on which the object is located.



① Searching of local cache.
② Broadcast request message.
③ Reply from the node on which the object is located.

As compared to the broadcasting scheme, this method helps in reducing the network traffic to a great extent.

The scheme is more flexible than the method of encoding the location of an object within its UID because it allows the system to support object migration. However, the use of broadcast protocol to locate those objects that are not found on their creator nodes limits the scalability of this mechanism.

Using Forward Location Pointers

This scheme is an extension of the previous scheme. The goal of this extension is to avoid the use of broadcast protocol. A forward location pointer is a reference used at a node to indicate the new location of an object.

Whenever an object is migrated from one node to another, a forward location pointer is left at its previous node. Therefore, to locate an object, the system first contacts the creator node of the object and then simply follows the forward pointer or chain of pointers to the node on which the object currently resides.

The method has the advantages of totally avoiding the use of broadcast protocol and allowing the support of object migration facility.

However, the method practically has the following major drawbacks:

1. The object-locating cost is directly proportional to the length of the chain of pointers to be traversed and grows considerably as the chain becomes longer.
2. It is difficult, or even impossible, to locate an object if an intermediate pointer has been lost or is unavailable due to node failures. Therefore, the reliability of the mechanism is poor.
3. The method introduces additional system overhead for upkeep.

Using Hint Cache and Broadcasting

Another commonly used approach is the cache-broadcast scheme. In this method, a cache is maintained on each node that contains the (UID, last known location) pairs of a number of recently referenced remote objects.

Given a UID, the local cache is examined to determine if it has an entry for the UID. If an entry is found, the corresponding location information is extracted from the cache. The object access request is then sent to the node specified in the extracted location information.

If the object no longer resides at that node, however, the request is returned with a negative reply, indicating that the location information extracted from the cache was outdated.

If the specified UID is not found in the local cache or if the localion information of the object in the local cache is found to be outdated, a message is broadcast throughout the network requesting the current location of the object.

Each node that receives the broadcast request performs an internal search for the specified object. A reply message is returned to the client node from the node on which the object is found. This location of the object is then recorded in the client node's cache. Notice that a cache entry only serves as a hint because it is not always correct.

This scheme can be very efficient if a high degree of locality is exhibited in locating objects from a node. It is also flexible since it can support object migration facility.

The method of on-use update of cached information avoids the expense and delay of having to notify other nodes when an object migrates.

The broadcast requests will clutter up the network, disturbing all the nodes even though only a single node is directly involved with each object-locating operation.

This is the most eommonly used object-locating mechanism in modern distributed operating systems. It is used in .Amoeba, V System, DOMAIN, NEXUS, Mach, and Chorus.

HUMAN-ORIENTED NAMES

- **Human-Oriented Hierarchical Naming Schemes**
    - ♦ Combining an Object's Local Name with Its Host Name
    - ♦ Interlinking Isolated Name Spaces into a Single Name Space
    - ♦ Sharing Remote Name Spaces on Explicit Request
- **Context Binding**
    - ♦ Table-Based Strategy
    - ♦ Procedure-Based Strategy
- **Distribution of Contexts and Name Resolution Mechanisms**
    - ♦ Centralized Approach
    - ♦ Fully Replicated Approach
    - ♦ Distribution Based on Physical Structure of Name Space
    - ♦ Structure-Free Distribution of Contexts
    - ♦ Interacting with Name Servers During Name Resolution.

System-oriented names such as 31A5 285F AD19 B1C8, though useful for machine handling, are not suitable for use by users. Users will have a tough time if they are required to remember these names or type them in.

Furthermore, each object has only a single system-oriented name, and therefore all the users sharing an object must remember and use its only name.

To overcome these limitations, almost all naming systems provide the facility to the users to define and use their own suitable names for the various objects in the system.

These user-defined object names, which form a name space on top of the name space for system-oriented names, are called human-oriented names.

Human-oriented names normally have the following eharacteristics:

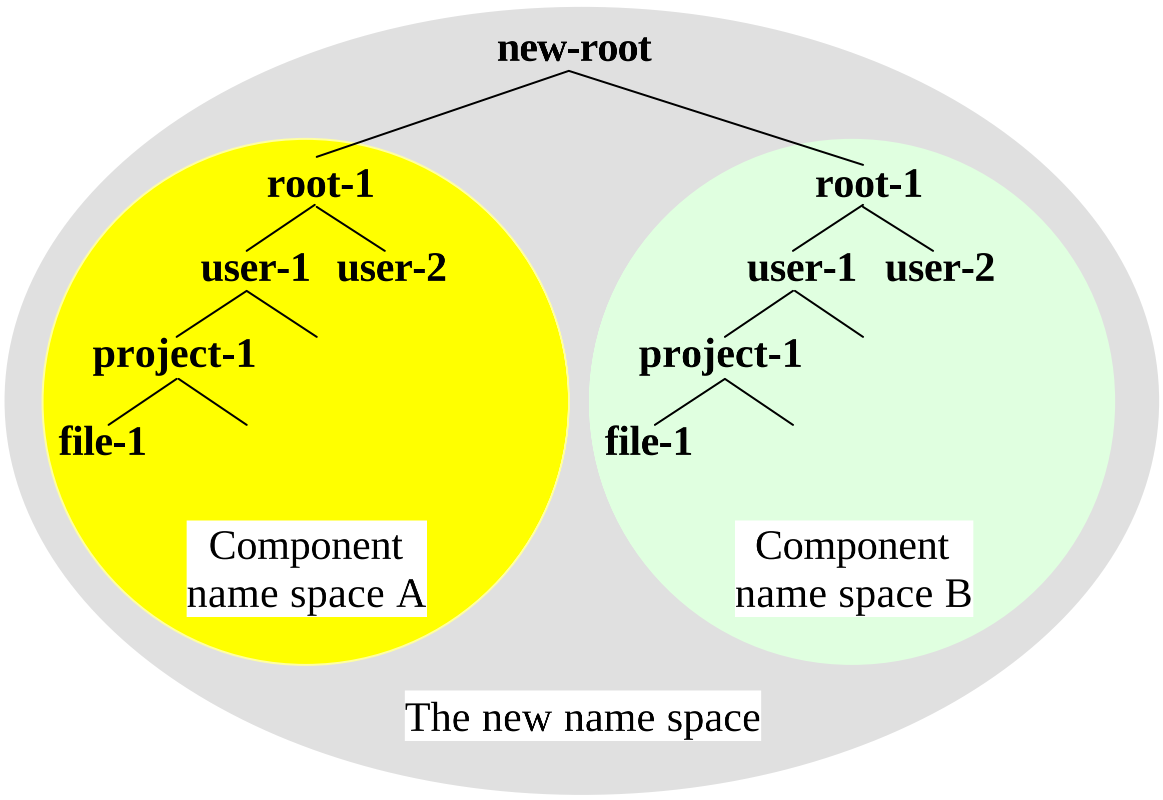
1. They are character strings that are meaningful to their users.
2. They are defined and used by the users.
3. Different users can define and use their own suitable names for a shared object. That is, the facility of aliasing is provided to the users.
4. Human-oriented names are variable in length not only in names for different objects but even in different names for the same object.
5. Due to the facility of aliasing, the same name may be used by two different users at the same time to refer to two different objects. Furthermore, a user may use the same name at different instances of time to refer to different objects. Therefore, human-oriented names are not unique in either space or time.

Because of the advantages of easy and efficient management of name space, hierarchically partitioned name spaces are commonly used for human-oriented object names.

The names of a hierarchical narning system are of the form cl/c2/. . . /cj, should *i* be constant or arbitrary?

The main advantage of the constant-level scheme is that it is simpler and easier to implement as compared to the arbitrary-level scheme. This is because all software in the arbitrary-level scheme must be able to handle an arbitrary number of levels, so software for manipulating names tend to be more complicated than those for constant-level scheme. The disadvantage of the constant-level scheme is that it is diffcult to decide the number of levels, and if new levels are to be added later, considerably more work has to be done because all the algorithms for name manipulation must be properly changed to take care of the new levels.

On the other hand, arbitrary-level schemes have the advantage of easy expansion by combining independently existing name spaces into a single name space. For example, as shown in figure below, two independent name spaces may be combined into one name space by adding a new root, making the existing roots its children. The major advantage is that if a name was unambiguous within its old name space, it is still unambiguous within its new name space, even if the name also appeared in some other name space that was combined. There is no need to change any of the algorithms for manipulating names.



Combining two name spaces to form a single name space by adding a new root

Arbitrary-level schemes, being more flexible than constant-level schemes, are used by most recent distributed systems. Hence, subsequent sections of this chapter deal with arbitrary-level hierarchical name spaces.

The major issues associated with human-oriented names are as follows:

1. Selecting a proper scheme for global object naming
2. Selecting a proper scheme for partitioning a name space into contexts
3. Selecting a proper scheme for implementing context bindings
4. Selecting a proper scheme for name resolution
   These isue are described in the next section.

Interacting with Name Servers During Name Resolution.

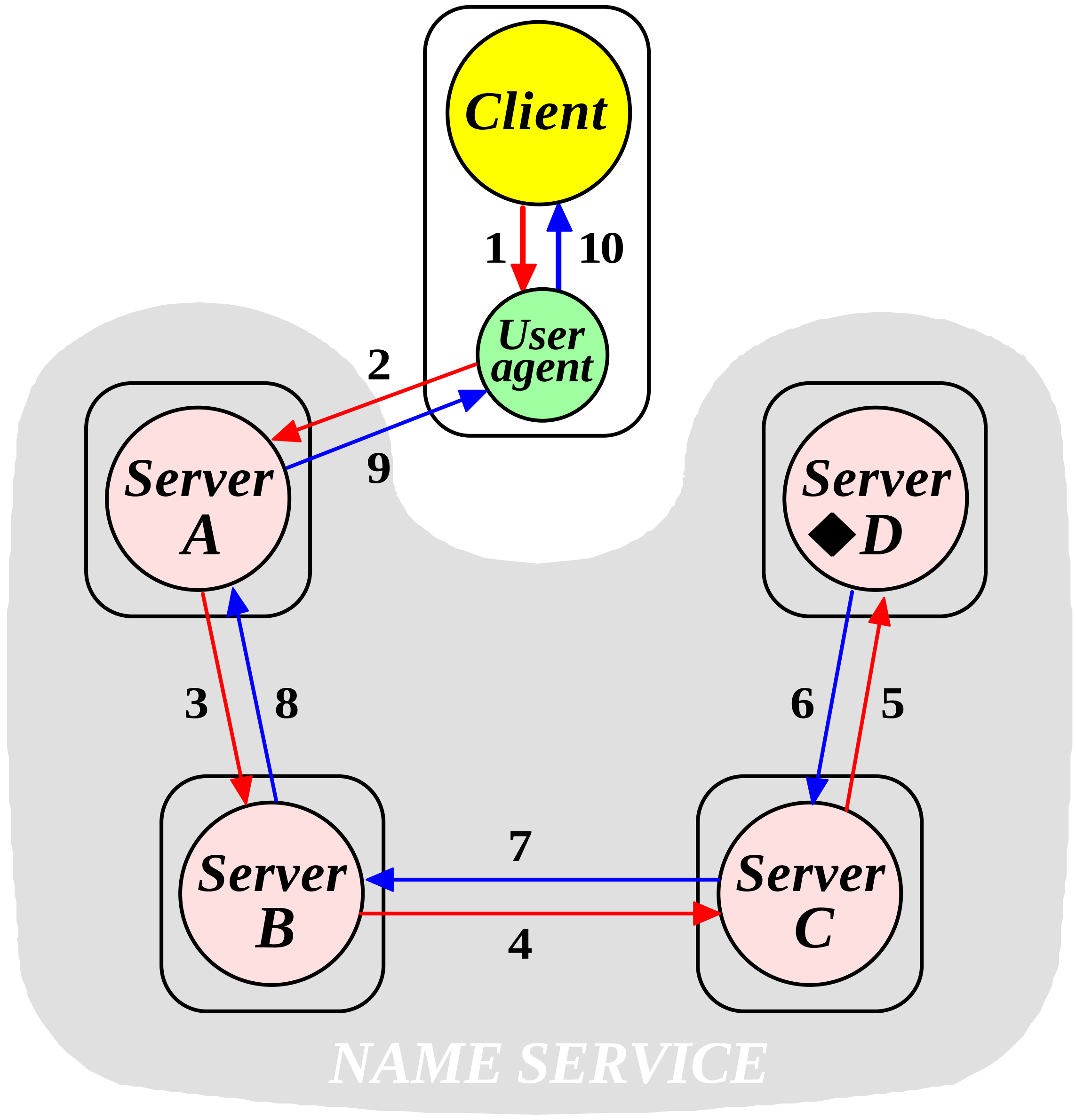
Various contexts of a given pathname may be stored at different name servers. Therefore, the resolution of a pathname in such a situation will involve interacting with all the name servers that store one or more contexts of the pathname. During name resolution, a name agent may interact with the name servers in one of the following manners:

1. *Recursive*. In this method, the name agent forwards the name resolution request to the name server that stores the first context needed to start the resolution of the given name.

   After this, the name servers that store the contexts of the given pathname are recursively activated one after another until the authority attribute of the named object is extracted from the context corresponding to the last component name of the pathname.

   The last name server returns the authority attribute to its previous name server, which then returns it to its own previous name server, and so on.

   Finally, the fast name server that received the request from the name agent returns the authority attribute to the name agent.



◆ = *required data*

Recursive resolution protocol

As an example, if the name */a/b/c/d* is to be resolved, the name agent sends it to the name server (say S_A) of the root context (/) and waits for a reply.

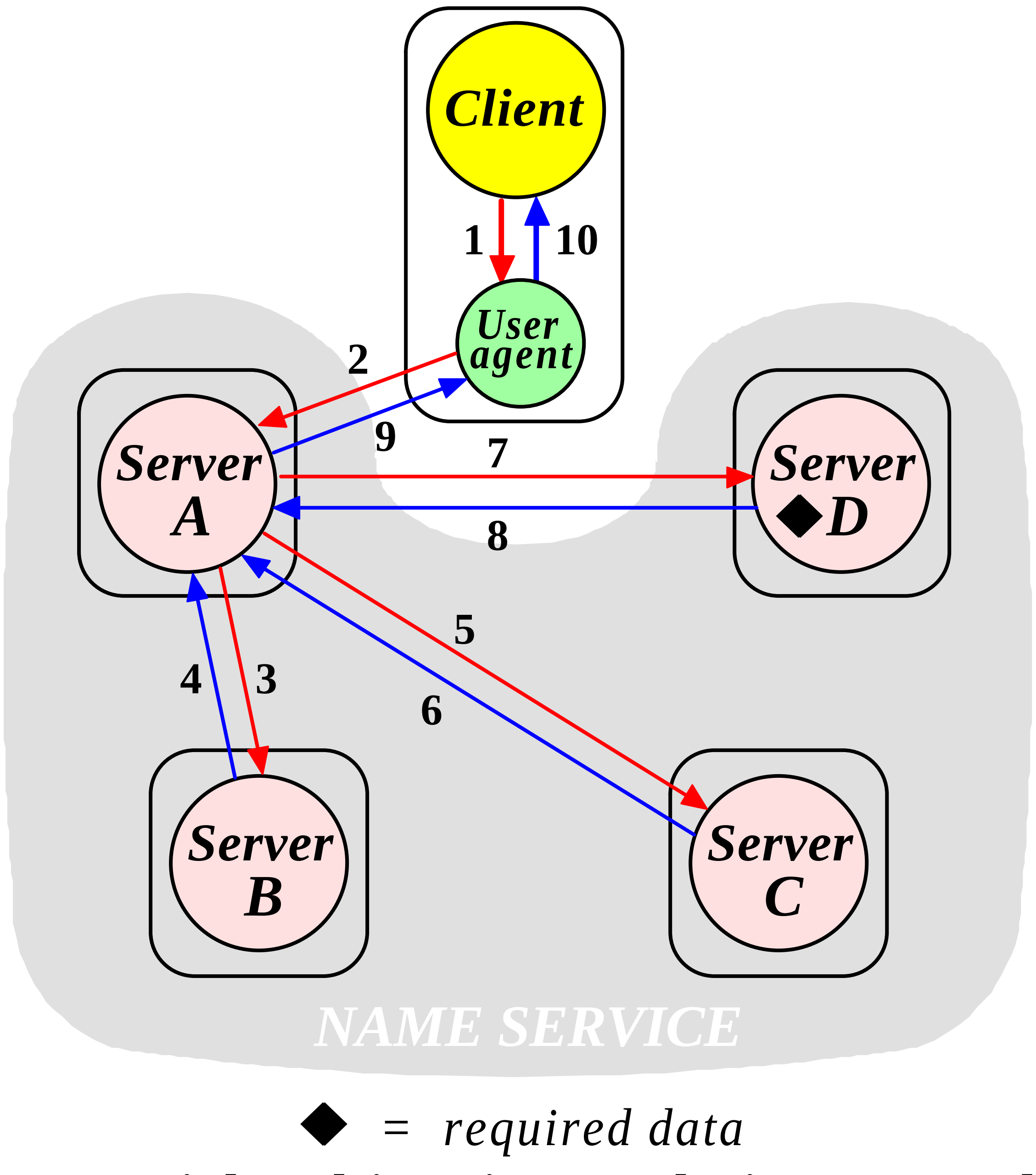
Then S_A searches for the component name *a* in the root context, extracts the corresponding binding information, sends the remaining pathname *b/c/d* to the name server (say S_B) of the next context (/a), and waits for a reply. Then S_B extracts from context */a* the binding information corresponding to the component name *b*, sends the remaining pathname *c* to the name server (say S_C) of the next context (*/a/b*), and waits for a reply. Then S_C extracts from context */a/b* the authority attribute corresponding to the component name c and returns it to S_B,. which in turn returns it to S_A, and finally S, retums it to the name agent.

The name agent has little work to do but the name servers may be involved in processing several requests at the same time. Therefore, the name servers may get overloaded in situations where the number of name agents is too large as compared to the number of name servers.

Hence, this mechanism is not suitable for use in those systems in which the ratio of name agents to name servers is very high. Furthermore, to allow a name server to start another job when waiting for a response, the name servers have to be multiprogrammed.

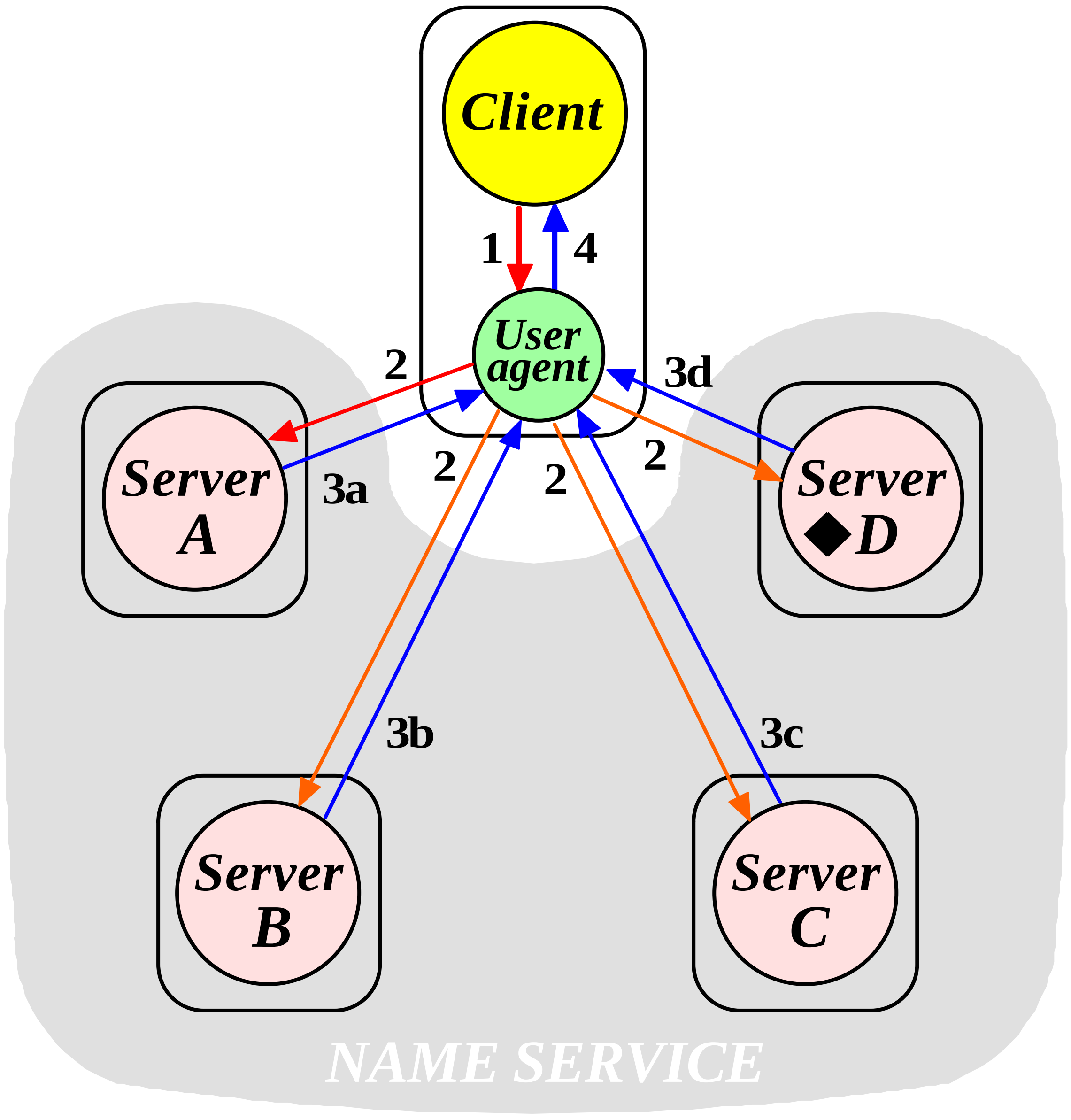
2. *Iterative*.

In this method, name servers do not call each other directly. Rather, the name agent retains control over the resolution process and ane by one calls each of the servers involved in the resolution process.



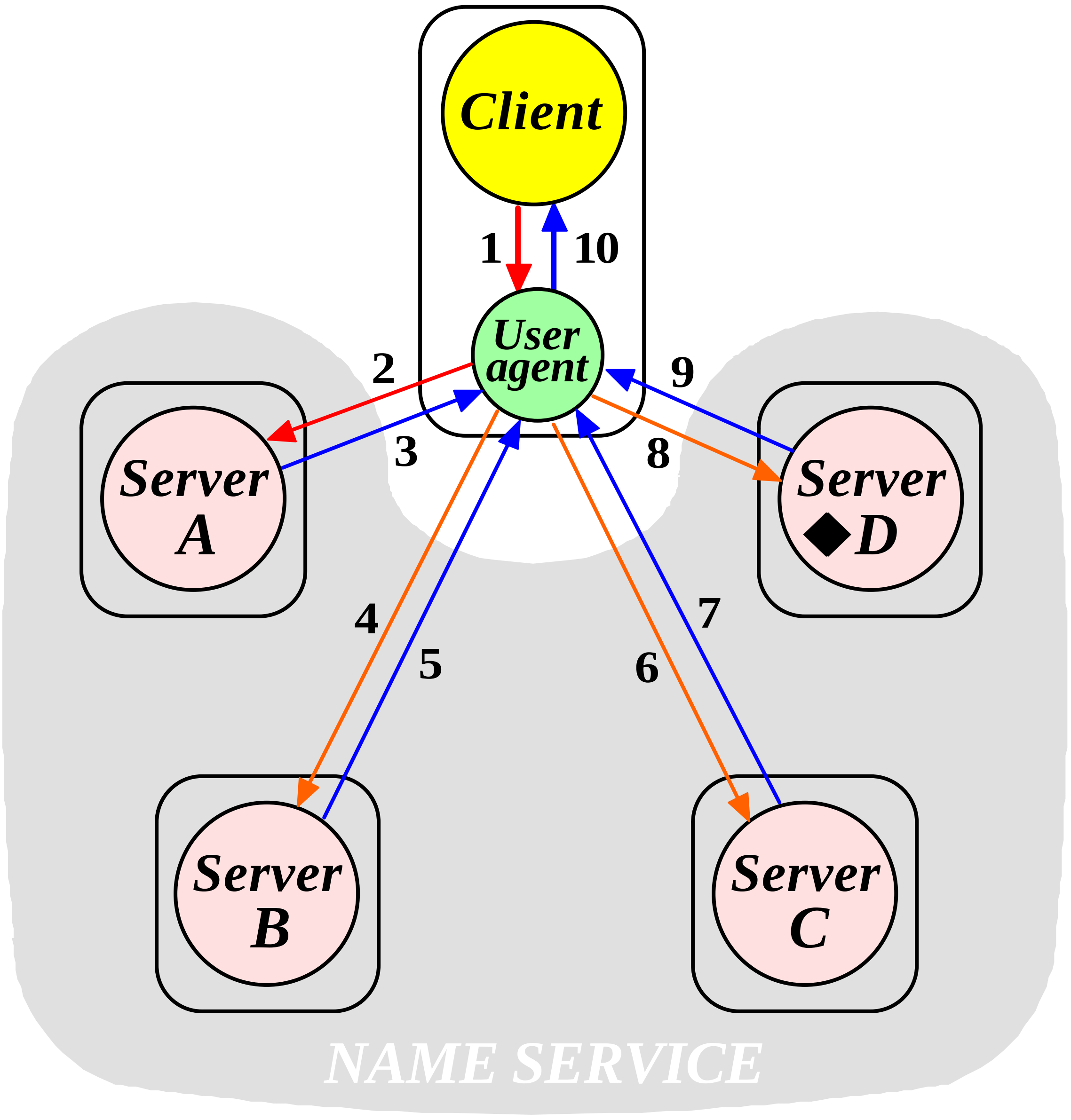
Sequential multicasting resolution protocol

As in the recursive process, the name agent first sends the name to be resolved to the name server that stores the first contest needed to start the resolution of the given name.



◆ = *required data*

Parallel iterative resolution protocol



◆ = *required data*

Sequential iterative resolution protocol

The server resolves as many components of the name as possible. If the name is completely resolved, the authority attribute of the named object is returned by the server to the name agent.

Otherwise. the server returns to the name agent the unresolved portion of the name along with the location information of another name server that the name agent should contact next.

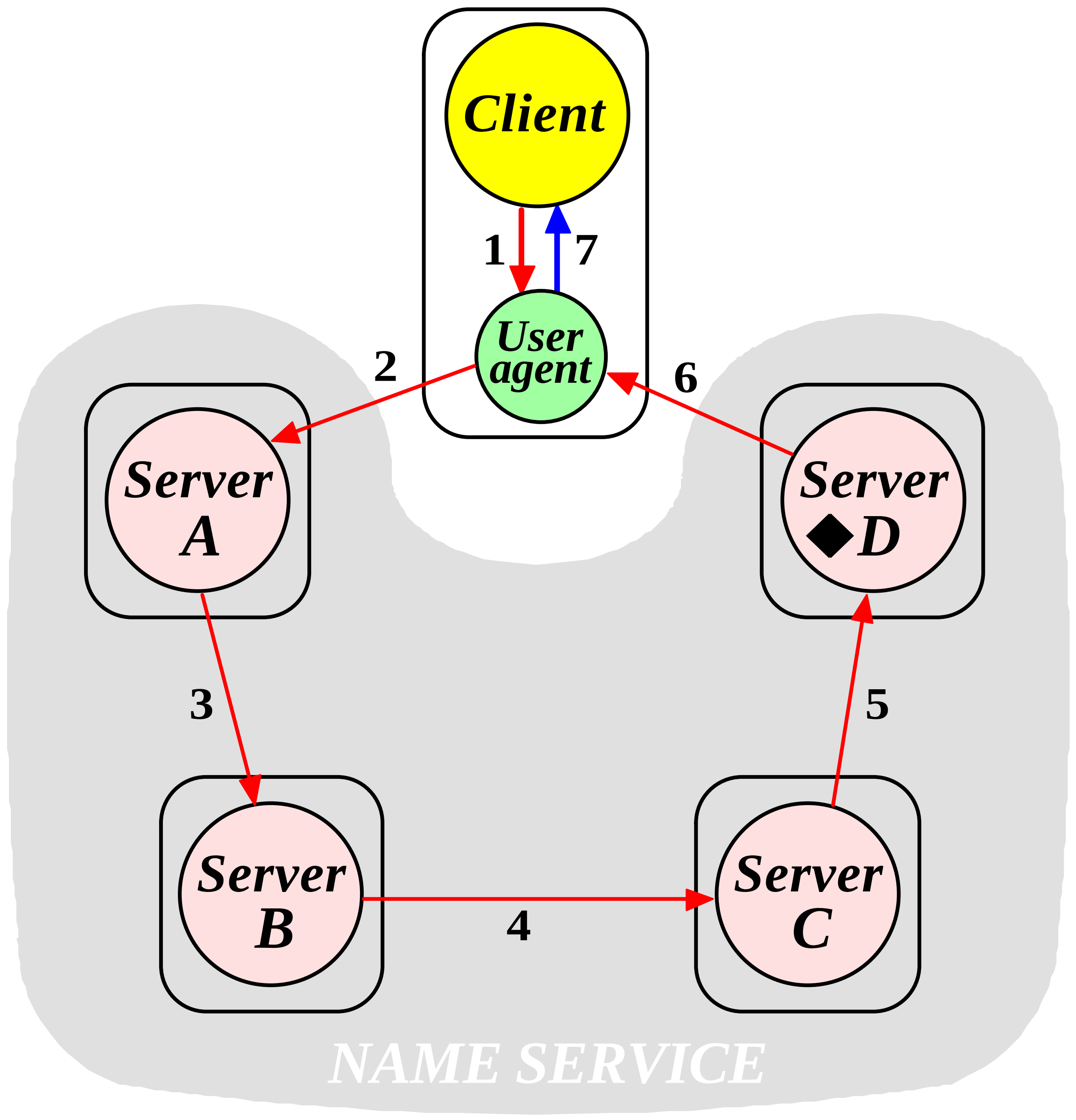
To continue the name resolution. The name agent sends a name resolution request along with the unresolved portion of the name to the next name server. The process continues until the name agent receives the authority attribute of the named object.

3. *Transitive*.

The name agent first sends the resolution request to the name server that stores the first context needed to start the resolution process.

The server resolves as many components of the name as possible. It then passes on the unresolved portion of the name to the name server that stores the next context needed to proceed with the resolution process.

This name server resolves as many components of the name as possible and then passes on the unresolved portion of the name to the next name server. The process continues and the name server that encounters the authority attribute of the named object returns it directly to the name agent.



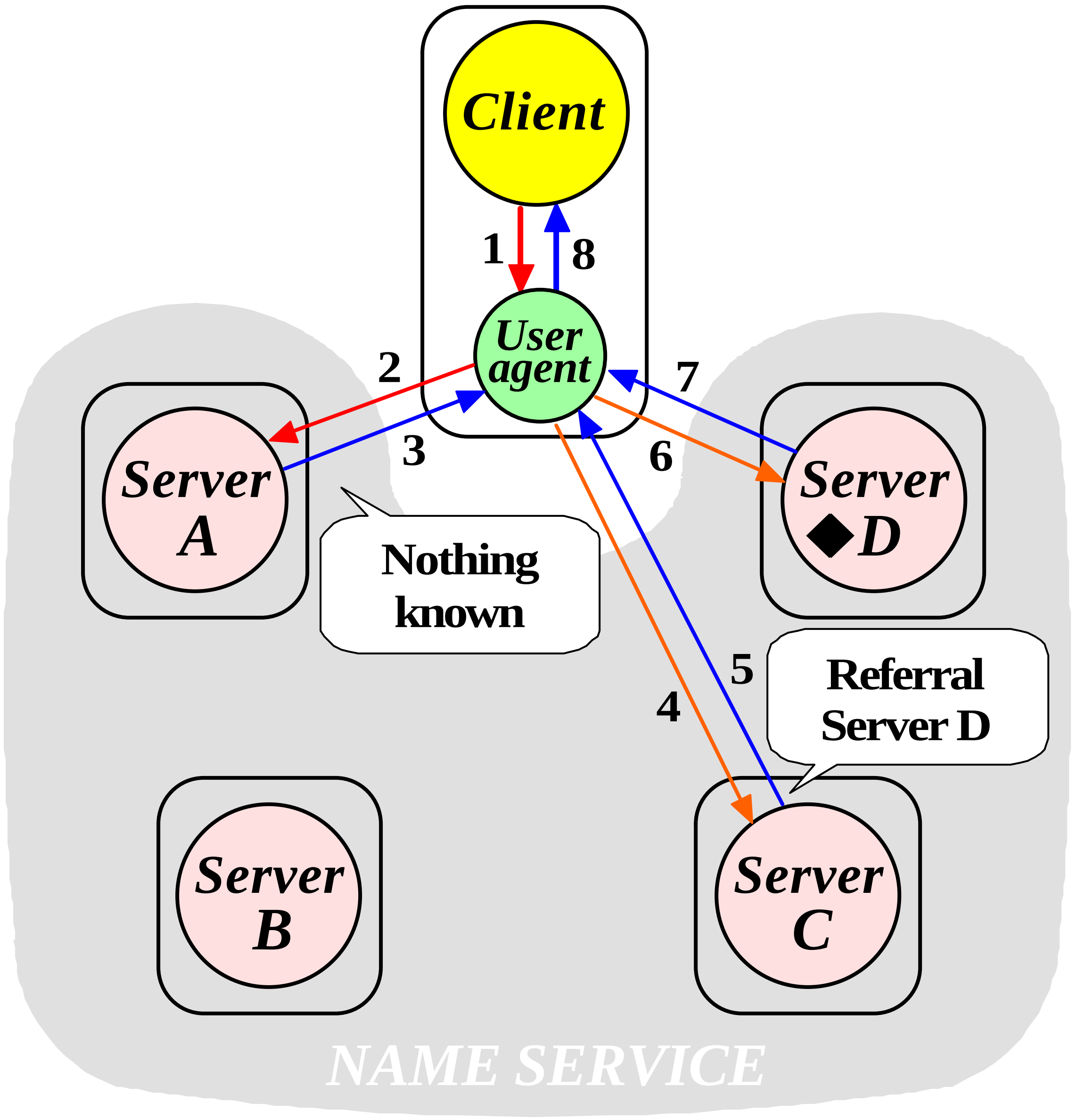
◆ = *required data*

Transitive resolution protocol

Notice that, as in the recursive method, in this method also the name agent has little work to do.

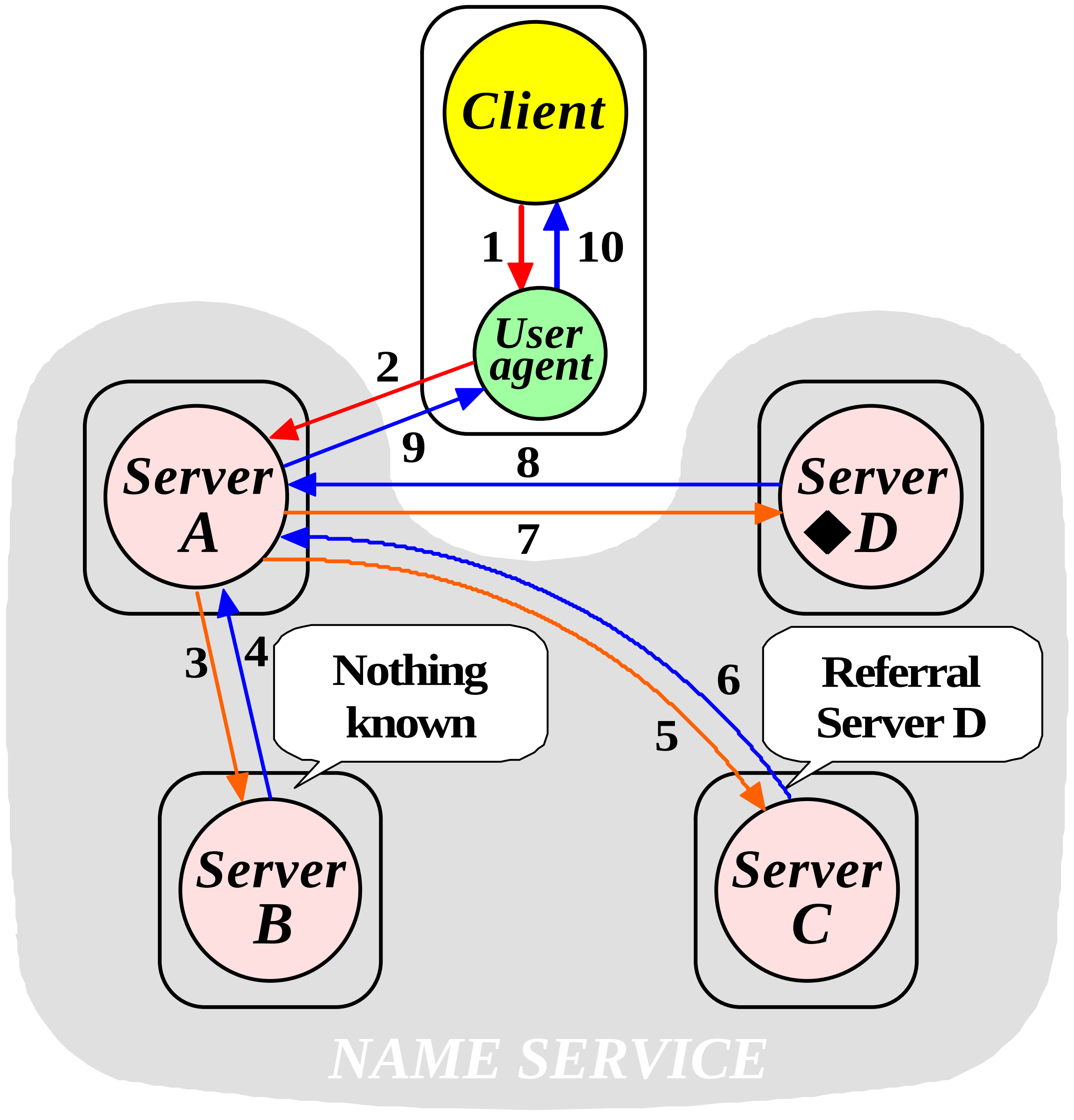
Also notice from that the transitive approach requires the fewest number of messages. However, a sender does not receive any acknowledgment message once it passes on the resolution operation to another server.

Therefore, this approach should be used in systems with reliable communication. On the other hand, recursivie and iterative approaches can be efficiently supported by RPC-based communication systems because they use a "call-response" model.



◆ = *required data*

Agent based referral resolution protocol



◆ = *required data*

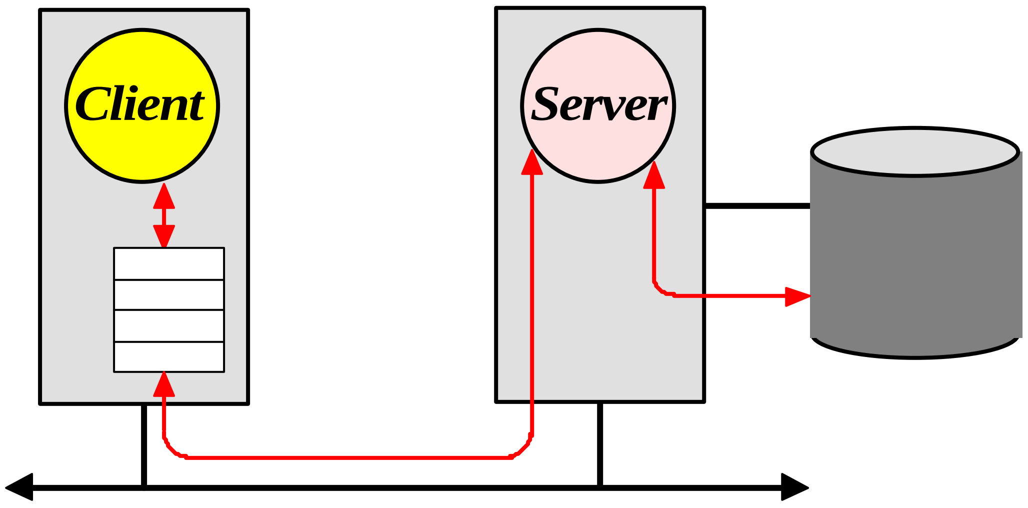
Server based referral resolution protocol

NAME CACHES

- **Types of Name Caches**
- **Approaches for Name Cache Implamentation**
- **Multicache Consistency**
- **Immediate Invalidate**
- **On-Use Update**

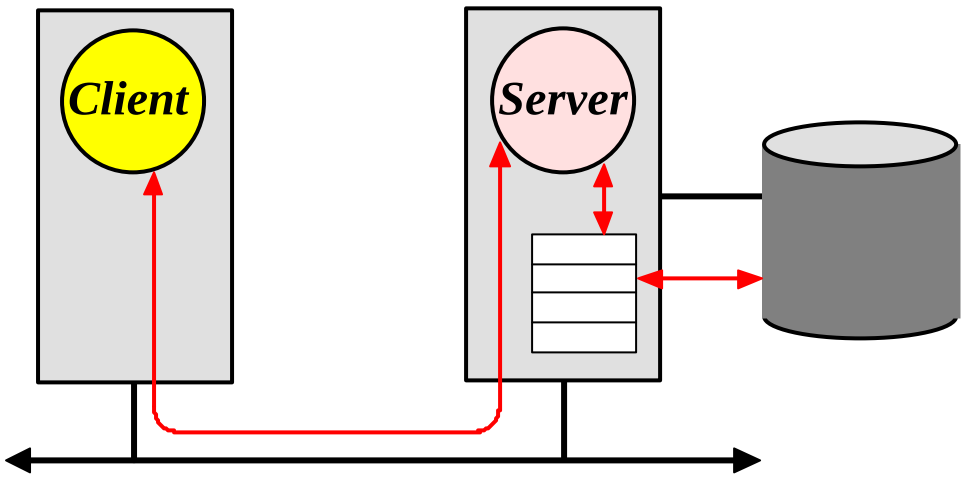
Client based cache

Caching is an important technique for building responsive, scalable distributed systems. A cache can be maintaned either by the client or the server or by both.



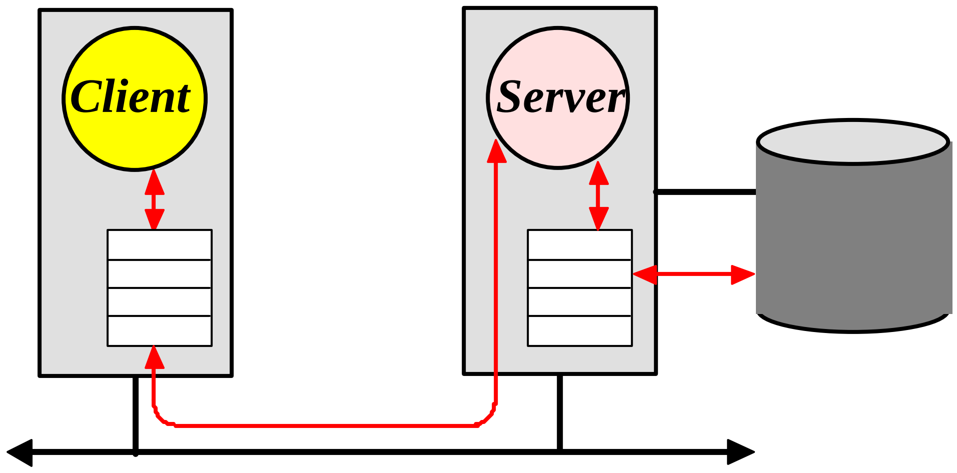
Caching at the client is an effective way of pushing the processing workload from the server out to client devices,if a client has the capacity.

Server based cache



If results data likely to be reused by multiple clients or if the client devices do not have the capacity then caching at the server is more effective.

Multi-level caching



Caches can be miantaned at multiple levels. For example, caches can be maintaned at all clients and all servers. Use of a cache at one level reduces the number of requests handled at the levels below.

SUMMARY