Module 8

Naming

Naming system

- A DCS supports several types of objects such as processes, files, I/O devices, mail boxes and processor nodes
- The Naming facility of DCS Enables users and programs to assign character-string names to objects and subsequently use these names to refer to these objects
- The Locating facility Maps an object's name to the object's location in a Distributed System
- Together they form a naming system that provides the users with an abstraction of an object that hides details of how & where an object is located in the network
- Naming system plays an important role in achieving the goal of location transparency in a DCS
- It facilitates transparent migration and replication of objects and object sharing

Desirable Features of a Naming System

1. Location transparency: means that the name of an object should not reveal any hint as to the physical location of the object
   - i.e., An object's name should be independent of the physical connectivity of topology of the system or the current location of the object
2. Location independency: For performance, reliability, availability, and security reasons, DS provides the facility of object migration that allows the movement of the objects dynamically among various nodes of a system
   - Location independency means name of an object need not be changed when the object's location changes
   - A user should be able to access an object by its same name, irrespective of the node from where he or she access it

Desirable Features of a Naming System (Cont'd)

- The requirement of location independency calls for a global naming facility which has:
  - An object at any node can be accessed without the knowledge of its physical location
  - To an object , any one can issue an access request without the knowledge of its own physical location (also known as user mobility)
- A location-independent naming system should support a dynamic mapping scheme so that it can map the same object name to different locations at two different instances of time
- Hence location independency is a stronger property than location transparency

3. Scalability: DCS vary in size from few nodes to many hundreds of nodes
   - They are open systems and their size changes dynamically

Desirable Features of a Naming System (Cont'd)

- Hence, 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 the naming system should be capable of adapting to this dynamically changing size of the DCS that leads to the change in the size of the name space
- i.e., 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
   - e.g., Filenames typically differ from user names and process names
   - A good naming system should use the same naming convention across all types of objects in the system

Desirable Features of a Naming System (Cont'd)

5. Multiple user-defined name for same object: For a shared object, it is desirable that different users of the object can use their own convenient names for accessing it
   - Should support multiple user defined names to the same object
   - 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 support many different objects to be identified by the common group name
   - Such a facility is useful to support broadcast facility, to group objects for group communication, for conferencing and so on
7. Meaningful names: A name can be simply any character string identifying some object

Desirable Features of a Naming System (Cont'd)

- However, Users prefer meaningful names rather than jumble of character string as it is easy to remember and use
- Hence a good naming system should support at least two levels 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 required to map an object's name to its attributes such as location
   - In a distributed environment, this performance is dominated by the number of messages exchanged during name mapping operation
   - Hence, a naming system should be efficient in the sense that the number of messages exchanged in a name mapping operation should be as minimum as possible

Desirable Features of a Naming System (Cont'd)

9. Fault tolerance: A naming system should be capable of tolerating, faults that occur due to failure of a communication link or a node
   - i.e., The naming system should continue functioning, may be in degraded form, in the event of these failures
   - The degradation can be in performance, functionality or both
10. Replication transparency: In a distributed system, the replicas of an object are created to improve performance and reliability
   - The naming system should support the use of multiple copies of the same object in a user transparent manner; i.e., user need not be aware of the existence of multiple copies of an object in use

Desirable Features of a Naming System (Cont'd)

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 nearest replica of the desired object
    - The efficiency of the object accessing operation will be affected if the object locating mechanism does not take this point into consideration
12. Locating all replicas: In addition to locating the nearest replica of an object, it is also important from a reliability point of view that all replicas of a desired object be located by the object locating mechanism.
    - Another replica can be used if nearest is inaccessible due to any reason, such as, communication link failure.

Fundamental Terminologies and Concepts

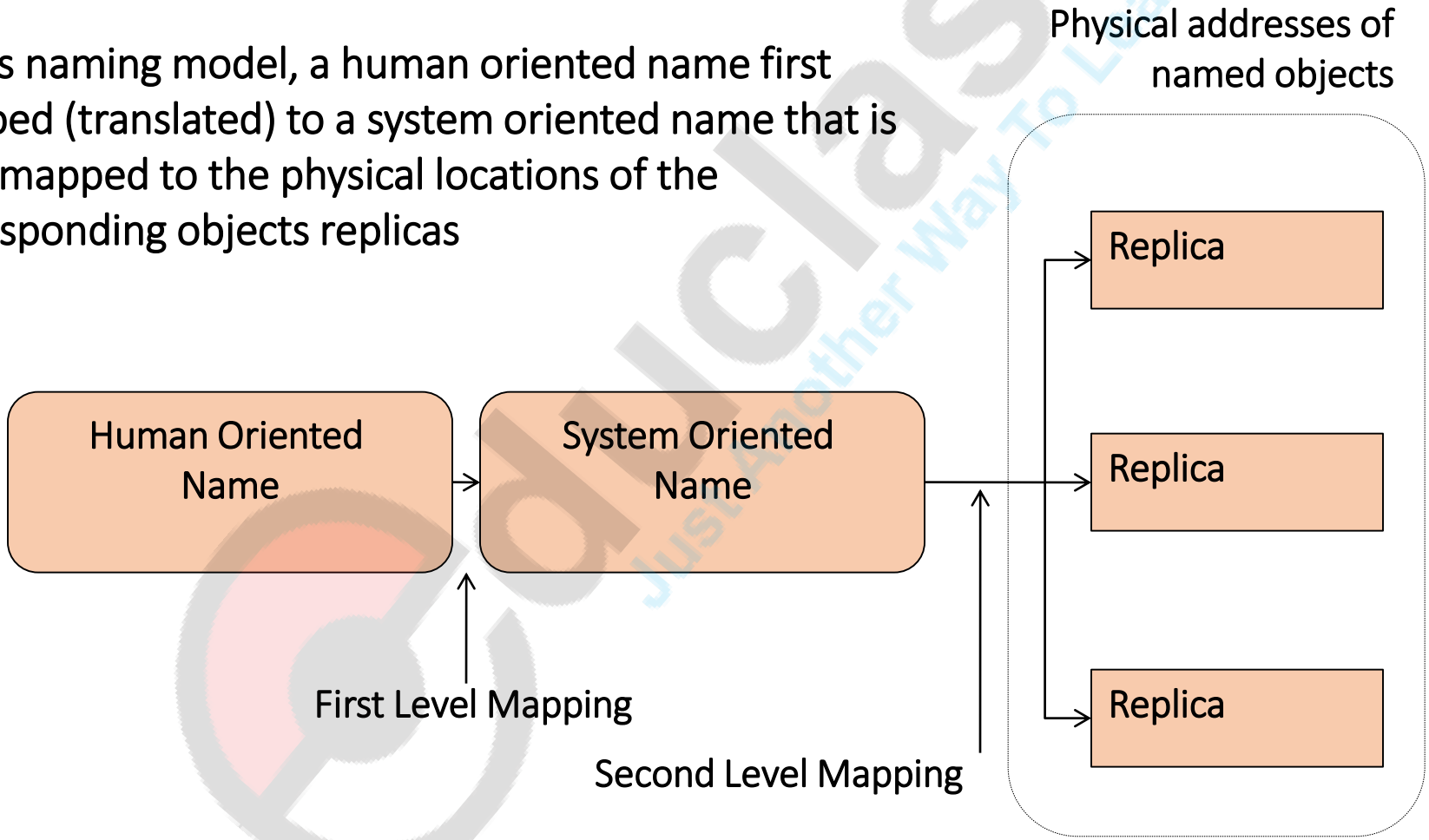
- **Name (Identifier)**
  - String composed of a set of symbols chosen from finite alphabet set. A logical object that identifies a physical object to which it is bound to.

- **Human oriented names (High level names)**
  - A character string that is meaningful to its users.
  - Different users can define their own names for shared objects.
  - Independent of physical location or structure of objects.
  - Not unique for an object & variable in length.

- **System-oriented names (Low level names)**
  - Uniquely identify every object in entire system
  - Bit patterns of fixed size automatically generated by system that can be easily manipulated and stored by machines

Fundamental Terminologies and Concepts (Cont'd)

In this naming model, a human oriented name first mapped (translated) to a system oriented name that is then mapped to the physical locations of the corresponding objects replicas



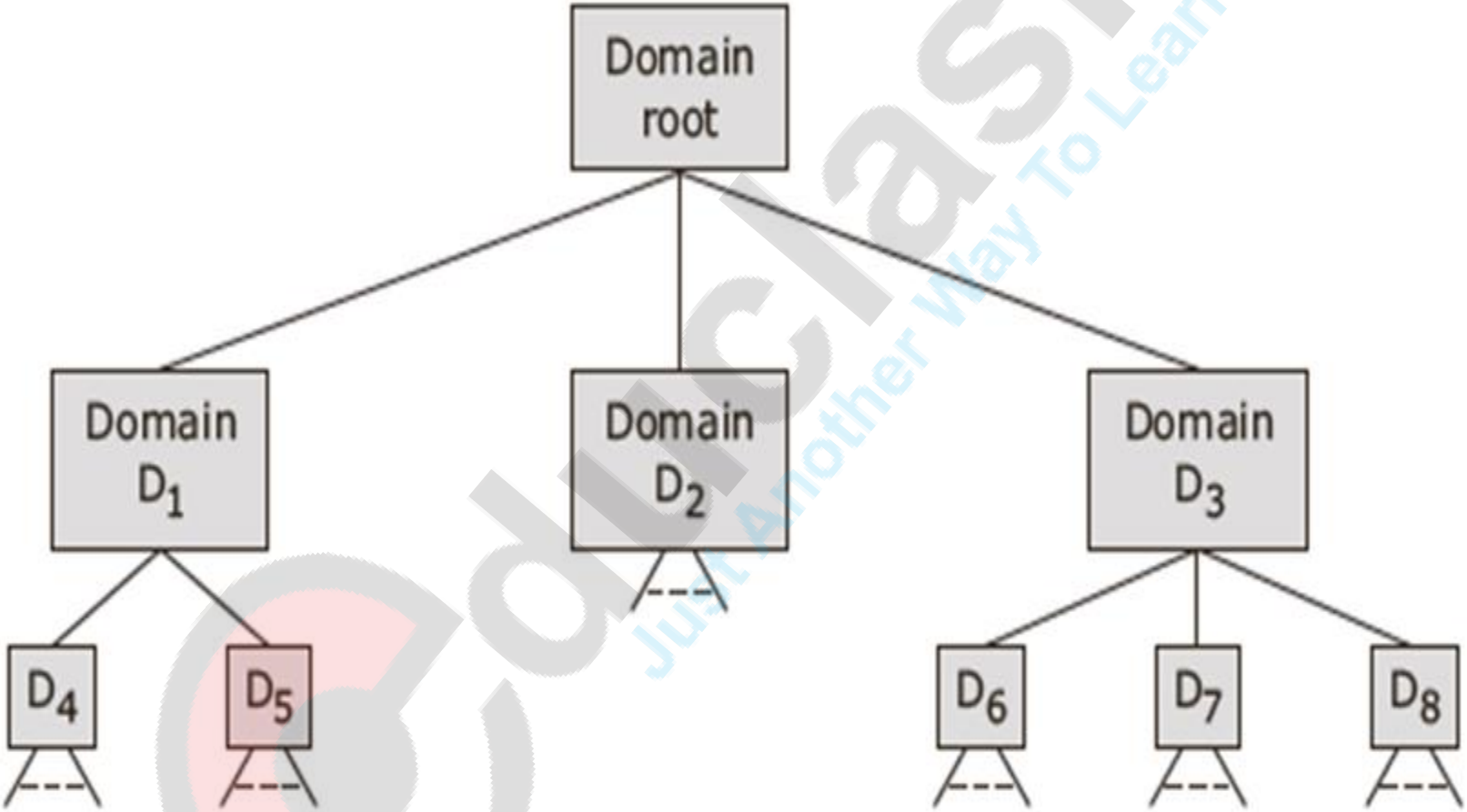
- **Name space**
  - Set of names complying with a given naming convention is said to form a name space.
  - An abstract container providing context for the names it holds and allowing disambiguation of items with same names residing in different namespaces.
  - Names in a namespace have to be unique.

- **Flat name space**
  - Names are character strings exhibiting no structure. Primitive or flat names.
  - Suitable for small namespace

- **Partitioned name space**
  - Name space is partitioned into disjoint classes (domains).
  - Names defined in a domain must be unique within that domain.
  - Name structure reflects physical or organizational associations
  - Simple / Compound names
  - Ex. Hierarchical name space – Telephone System, IP Addresses
  - Number of levels may be fixed or arbitrary in hierarchical name space

- **Name server**
  - Manages namespace
  - Process that maintains information about named objects and provide facilities that enables users to access that information.
  - Binds object name to its location
  - Several name servers present in distributed systems with each server having information of a small subset of set of objects.
  - Authoritative name server
  - Partitioned namespaces easier to manage

- **Name agent**
  - Communication between clients & name servers happens via name agents.
  - Maintains knowledge of existing name servers.
  - Private name agent
    - Works for a single client and is structured as a set of subroutines linked to a client program.
  - Shared name agent
    - Structured as a part of the operating system kernel with system calls to invoke the naming service operations.



Name servers in hierarchical namespace

- **Context**
  - The environment in which name is valid.
  - Represent division of name space along natural geographical, organizational or functional boundaries.
  - Qualified name (context, name pair) uniquely identifies an object.
  - Useful for partitioning the name space.
  - In a partitioned namespace, each domain corresponds to context of namespace.

- **Name Resolution**
  - Process of mapping an object's name to object's properties, such as its location.
  - To resolve a name, client contacts its name agent, which contacts a known name server, which may in turn contact other name servers.
  - In partitioned namespace, name resolution mechanism travels a resolution chain from one context to another until the authoritative name servers of named object are encountered.

- **Abbreviation/ Alias**
  - Users can specify their own short-form substitutes for compound qualified names called abbreviations.
  - More than one simple name may be bound to same qualified name within a given context.(synonyms)

- **Absolute name**
  - An absolute name begins at the root context of the name space tree and follows a path down to the specified object.

- **Relative names**
  - It defines a path from the current context to the specified object.

- **Generic and multicast name**
  - Support one-to-many binding.
  - Naming system allow a simple name to be bound to a set of qualified names.
    - **Generic naming facility** - Name is mapped to any one of the set of objects to which it is bound.
    - **Group or multicast naming facility** - Name is mapped to all the members of the set of objects to which it is bound.

- **Descriptive/ Attribute Based Name**
  - An object is identified by a set of attributes or properties that describe the object & uniquely identify it among the group of objects in the name space.
  - All attributes together refer to a single object.
  - Ex. User = ?, Creation date = ?, file type = ?

- **Source – Routing Name**
  - Mirrors the structure of the underlying physical network. Eg, host1/host2/host3/file 1

Generating System-Oriented Names (Cont'd)

- One method is to treat each node of the system as a domain of the name space and treat a reading from the real-time clock called timestamp of a node as a unique identifier within the node
- Hence the global unique identifiers take the form of the pair (*node-id, timestamp*)
- Another method is to treat each server as domain and then to allow a server to generate object identifiers for the objects it serves in a server-specific manner
- Then global unique identifier take the form of the pair (*server-ID, server-specific-unique identifier*)
- The distributed approach has better efficiency and reliability as compared to centralized approach, However, it has the following drawbacks

Generating System-Oriented Names (Cont'd)

- In a heterogeneous environment, the form and the length of identifier may be different for different computers (nodes), resulting in non uniform identifiers
- It may become awkward or inefficient for applications to be prepared to deal with the non-uniformity of these low level identifiers
- Node or server boundaries explicitly visible in the scheme

- Generating Unique Identifiers in the Event of Crashes
  - Another important problem related with the creation of unique identifiers (global or local) is to be able create unique identifiers in the face of crashes
  - A crash my lead to the loss of state information of a unique identifier generator and hence on recovery may not function properly, which may result in the generation of non-unique identifiers

Generating Unique Identifiers in the Event of Crash

- Two basic approaches to solve this problem are:
  - Using a clock that operates across failures: is used at the location of the unique identifier generator
  - This clock is guaranteed to continue operating across failures and it will not recycle during the period within which the needed identifier must be unique
  - To implement this method one may require rather very long identifiers depending on the granularity of the clock interval needed
  - Using two or more levels of storage are used and the unique identifiers are structured in a hierarchical fashion with one field for each level
  - A counter associated with each level contains the highest value of the corresponding field assigned
  - The current values of these fields are cached in the main memory

Generating Unique Identifiers in the Event of Crash

- When a lower level field is about cycle or the associated storage device crashes, the next higher level counter is incremented and the lower level counters are reset
- If a stable storage (the information on which can survive crashes) is used, two levels storage (the upper level being the main memory storage and the lower level being the stable storage) are sufficient to creating a safe and efficient unique identifier generator
- As compared to the first method this method can yield shorter identifiers, but is more complex to implement
- Use RAID to avoid disk crash and to build stable storage

Object Locating Mechanism

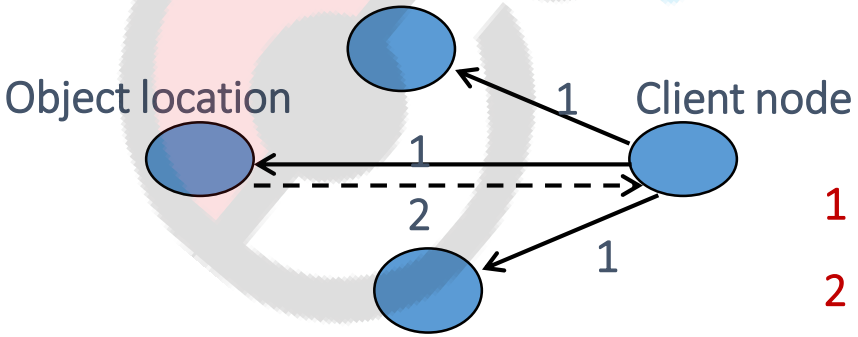
- Object locating is the process of mapping an object's system oriented unique identifier (UID) to the replica locations of the object
- The object locating operation is different and independent of the object accessing operation
- In DS, it is the only process of knowing the objects location, i.e., the node on which it is located
- The object accessing involves process of carrying out desired operation (e.g., read, write) on the object and it can only start after object locating operation has been carried out successfully
- Several object locating mechanism have been proposed and used by different distributed operating systems

Object Locating Mechanism (Cont'd)

- The suitability of a particular mechanism for a distributed system depends on various factors such as the scale of system, the type of UID being used for its naming system, whether the system supports object migration, local transparency of objects and so on
- These mechanisms are
  - Broadcasting
  - Expanding ring broadcast
  - Encoding location of objects within its UID
  - Searching creator node first and then broadcasting
  - Using forward location pointer
  - Using hint cache and broadcasting

Broadcasting

- An object is located by broadcasting a request for the object from a client node
- Request processed by all nodes & nodes currently having object reply back to the client node
- The process is very simple and has high degree of reliability because it supplies all the replica locations of the target object
- But poor efficiency & scalability, because of the vast amount of network traffic generated (directly proportional to the no of nodes present)
- Suitable for small networks with high communication speed and object locating requests are not so frequent



1 - Broadcast request message

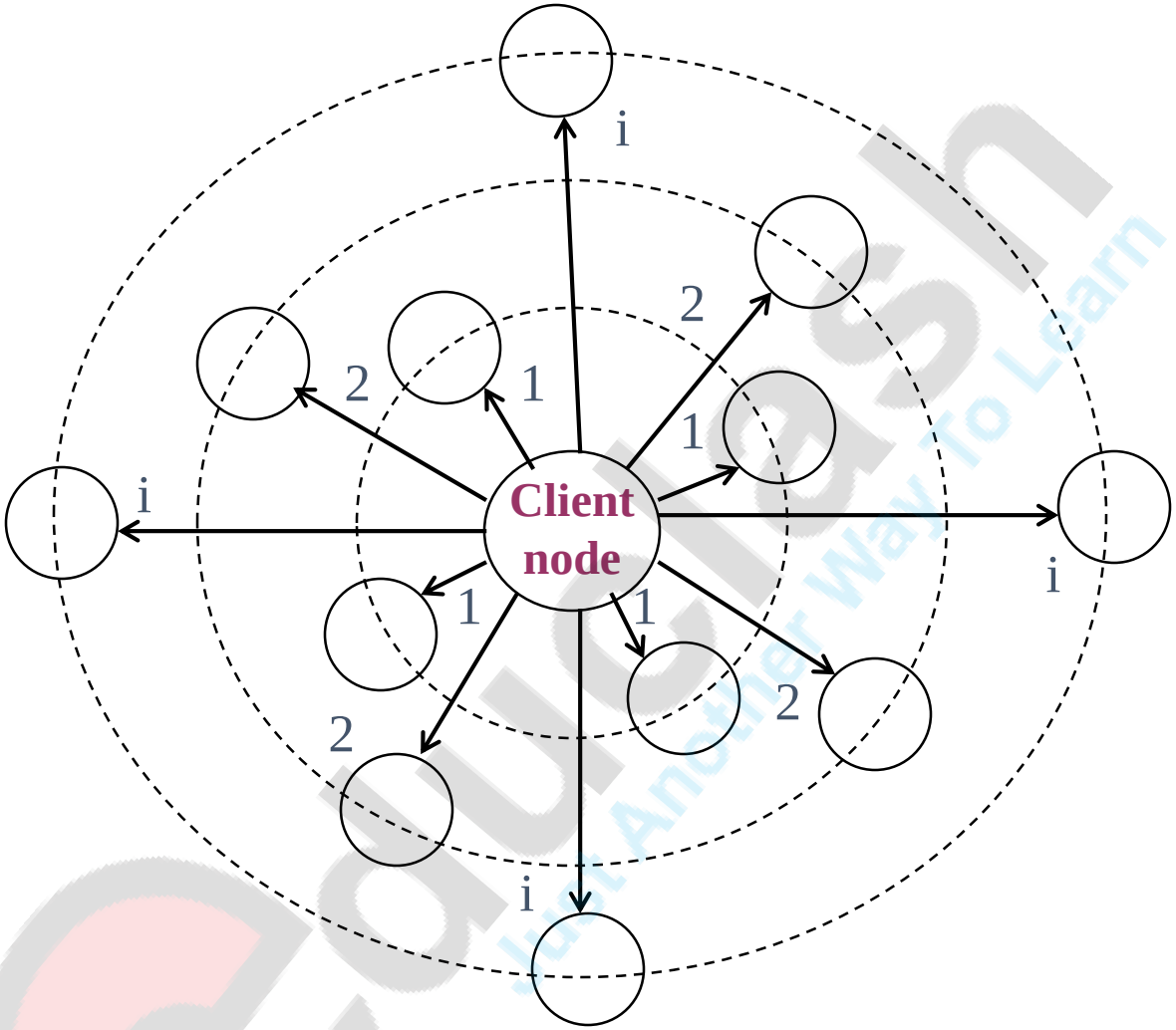
2 - Reply from node on which the object is located

Expanding Ring Broadcast

- Pure broadcasting is expensive for large networks and direct broadcast to all nodes may not be supported by wide area networks
- Hence a modified form of broadcasting is called expanding ring broadcast (ERB)
- It is normally employed in an internetwork that consists of LANs connected by gateways
- 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
- Hop - A hop corresponds to number of gateways in between processors

Expanding Ring Broadcast (Cont'd)

- If a message from processor A to processor B must pass through at least two gateways, A and B are two hops distant
- The processors in the same LAN are zero hop distant
- Ring - A ring is a set of LANs, a certain distance away from a processor
- $Ring_i[A]$ – set of LANs i hop away and $Ring_0[A]$ is a local network
- The ERB search works as follows to locate an object X
- Beginning with i =0, request message is broadcast to all LAN's in $Ring_i[A]$ (fig. in the next slide)
- If response received, search ends or else i is incremented by 1 and broadcast again



1 - Searching nodes at 0 hop distance

2 - Searching nodes at 1 hop distance if search of 0 hop fails

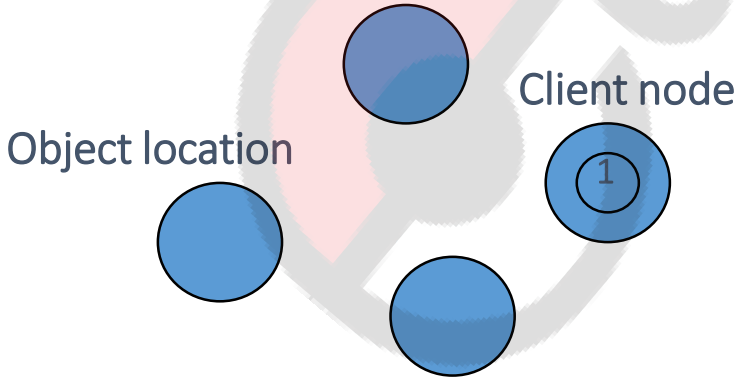
i + 1 - Searching nodes at i hops distance if searches up to i -1 hops fail

Expanding Ring Broadcast (Cont'd)

- The ring size is bounded from above by the diameter of the internetwork
- The method does not supply all the replica locations of an object simultaneously, but it supplies the nearest replica
- The efficiency of the search 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
  - The scheme uses structured object identifiers
  - One field of the structured UID is the location of the object
  - Given an UID, it extracts the corresponding object's location from its UID by examining the appropriate field of the structured ID
  - Extracted location is the node on which the object resides

Encoding Location of Object within its UID

- This is a straightforward and efficient scheme
- One restriction of the scheme is the object is not permitted to move once it is assigned to a node, as it would require change of its UID
- Hence the object is fixed to a node throughout its lifetime
- Another limitation is the scheme is that it is not clear how to support multiple replicas of an object → no migration, no replication



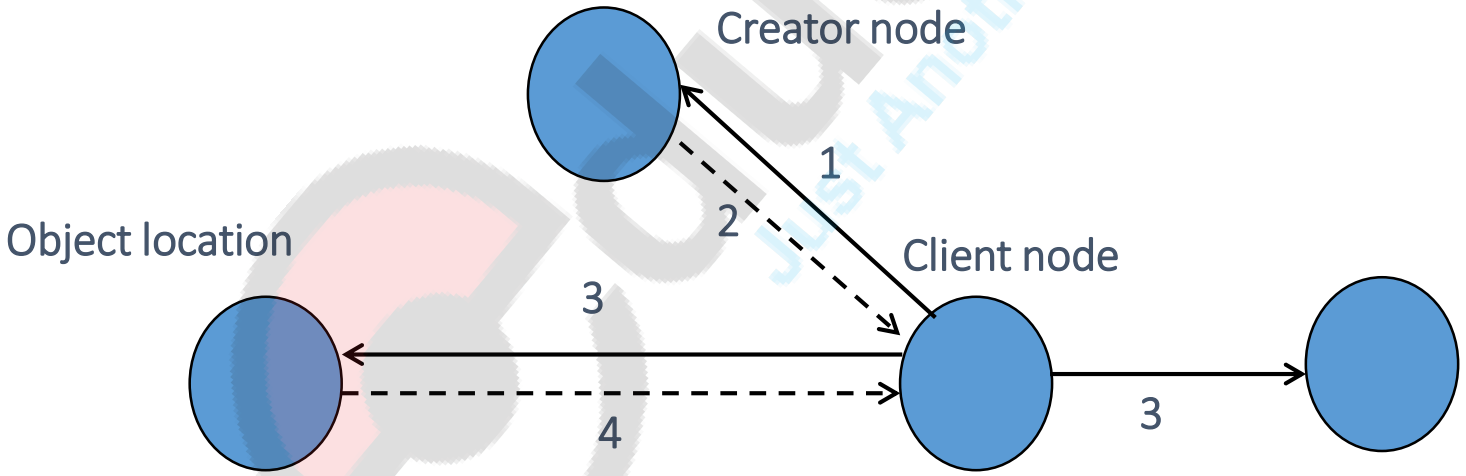
1 - Extracting object's location from its UID. No message exchange with any other node is required for locating the object.

Searching Creator Node First and then Broadcasting

- The scheme is a simple extension of the previous scheme to support 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(though may not be true always)
- It is because the object migration is not frequent as it is an expensive operation
- Creator node information is extracted from the UID and request is sent to that node
- If object no longer resides on its creator node, a failure reply is replied back to the client node

Searching Creator Node First and then Broadcasting (Cont'd)

- In case of failure object is located by broadcasting the request from the client node
- As compared to simple broadcasting scheme, this method helps in reducing the network traffic to a great extent



1 - Searching creator node; 2 – Negative reply from creator node

3 - Broadcast request; 4 - Reply from object's current location

Searching Creator Node First and then Broadcasting (Cont'd)

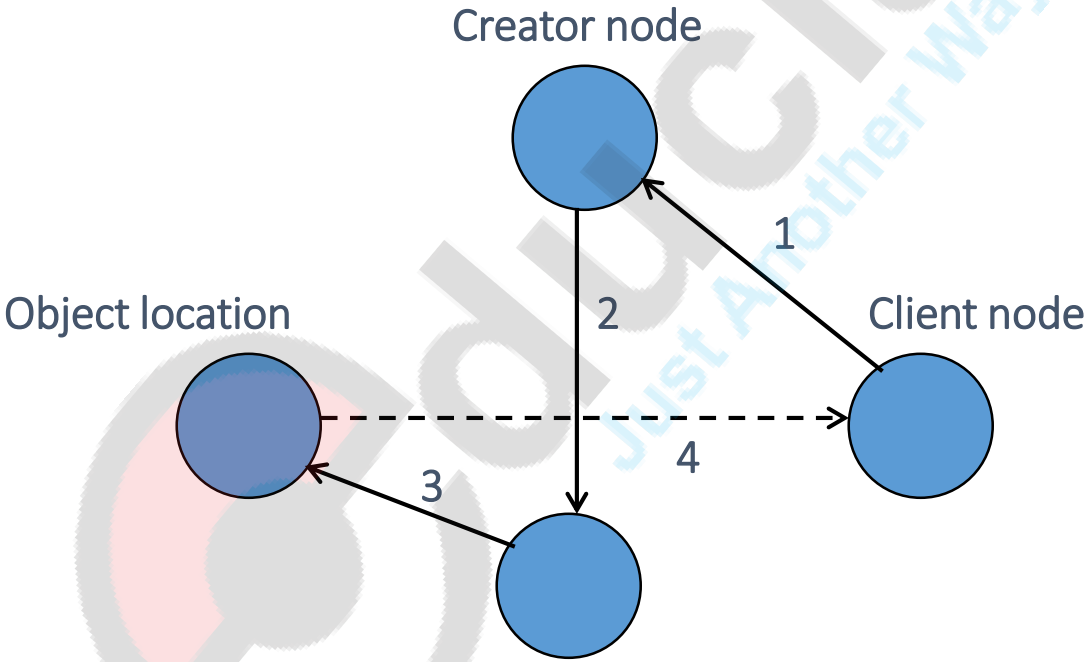
- Further, the scheme is more flexible than the method of encoding the location of an object within its UID because it allows the system to support object migration
- It does not support locating of replicas
- The use of broadcast protocol to locate those objects that are not found in its creator nodes limits the scalability of the mechanism

Using Forward Location Pointer

- The scheme is an extension of the previous scheme and the aim is to avoid broadcast protocol
- Whenever an object is migrated from one node to another, a forward location pointer is left at its previous node
- To locate a node, system first contacts creator node and then simply follows forward pointers or chain of pointers to the node on which the object currently resides
- Its main advantage is that, it avoids use of broadcast protocol totally and also supports object migration facility
- It has the following drawbacks
  - Chain of pointers can be long and cost of locating is proportional to chain length

Using Forward Location Pointer

- It impossible to locate an object if an intermediate pointer is lost or not available because of node failure and hence reliability is poor
- It introduces additional system overhead to upkeep



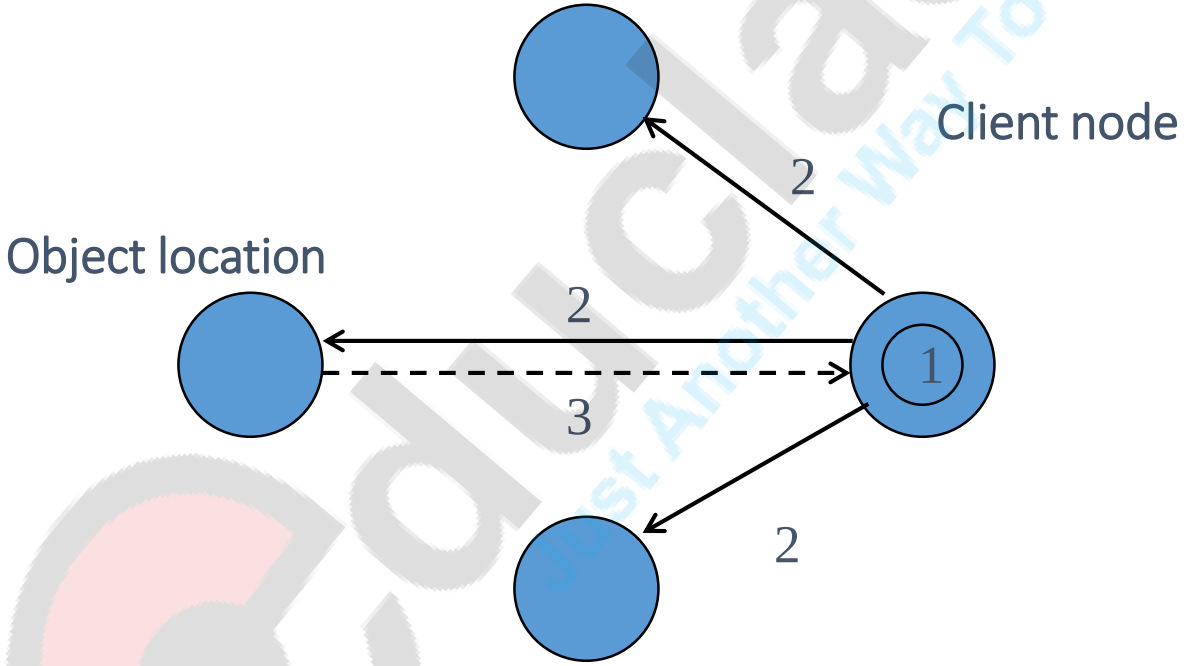
1, 2, 3 – Path of message forwarding

4 - Reply from the node on which the object is located

Using Hint Cache & Broadcasting

- Another commonly used approach is the cache-broadcast scheme
- In this scheme, 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, Local cache is searched for the UID; If found, location information is extracted from the cache
- Request is then send to the node specified in the extracted location information
- If object no longer resides at that node, negative reply is sent back
- If UID info is not found in the cache or cache info is outdated, location is then searched through broadcasting and is recorded in the client's node cache

Using Hint Cache & Broadcasting (Cont'd)



1 - Searching of local cache

2 - Broadcast request message

3 - Reply from the node on which the object is located.

Using Hint Cache & Broadcasting (Cont'd)

- Note that cache entry serves as only a hint because it is not always correct
- The 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
- One problem with this scheme, is that 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

Using Hint Cache & Broadcasting (Cont'd)

- It is the most commonly used object-locating mechanism in modern distributed operating systems
- Efficient if high degree of locality is exhibited
- Supports object migration
- Broadcasting involved
- Schemes broadcasting, expanding ring broadcast and using hint cache broadcasting can work with both unstructured and structured UIDs
- While encoding location of object within its UID, searching creator node first and broadcasting and using forward location pointer require structured UID

Using Hint Cache & Broadcasting (Cont'd)

- As summation schemes encoding the location of an object with in UID, searching the creator node and then broadcasting and using forward location pointers require structured UID
- Schemes broadcasting, expanding ring broadcast and using hint cache and broadcasting can work with both unstructured and structured UIDs
- Also notice that encoding location of object within its UID works without any exchange of messages with any other node, but it is inflexible as it does not support object migration and replication facilities

Human Oriented Names

- System oriented names, though useful and appropriate for machine handling, are not suitable for use by users
- Users will have tough time remembering these names, and each object has only single system oriented name and all users have to remember and use this only one name
- To overcome this limitation, almost all the naming systems provide the facility to the user to define and use their own suitable names for various objects in the system
- These user defined object names, which form a name space on top of the system oriented names' name space are called human oriented names which have the following characteristics:
  - These are character strings that are meaningful to their users

Human Oriented Names (Cont'd)

- Defined and used by users
- Different users can use their own suitable names for a shared object
- i.e., the facility of aliasing is provided to the users
- Human oriented names are variable in length not only in names for different objects but even in different names for the same object
- Due to facility of aliasing, same name be used by two different users at the same time to refer to different objects
- Furthermore, a user may use the same name to represent at different instances of time to refer to different objects
- Hence, human oriented names are not unique in space or time

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

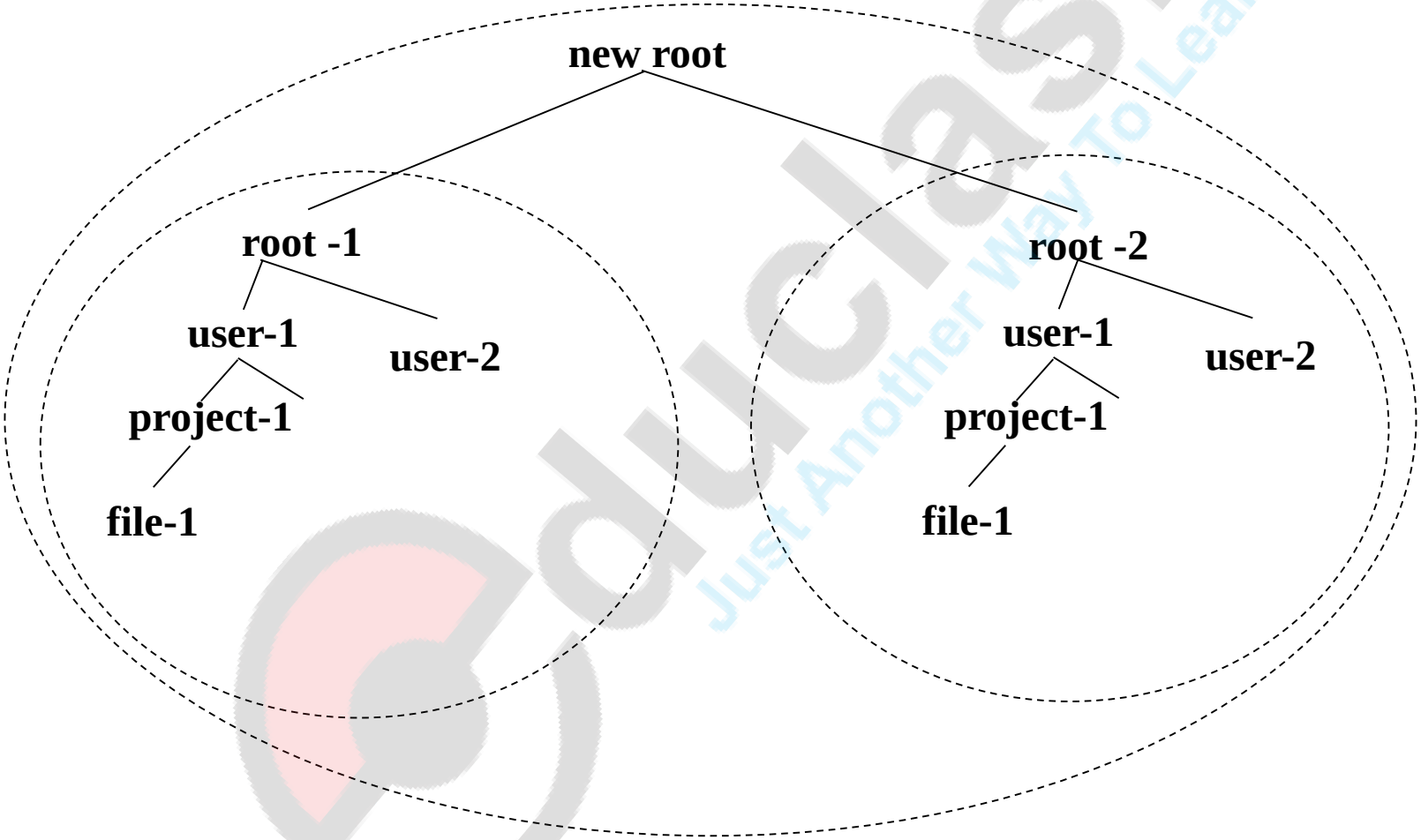
Constant vs. Arbitrary Number of Levels

- Hierarchically partitioned namespace can have either constant number of levels or arbitrary number of levels
- However, it is important to decide in advance of the two possibilities
- Both schemes have their advantages and disadvantages
- Constant number of levels
  - Simpler and easier to implement as compared to arbitrary number of level scheme
  - The software for manipulating names are less complicated
  - Difficult to decide number of levels in advance
  - All Algorithms for name manipulation systems have to be changed if new levels are added later

Constant vs. Arbitrary Number of Levels (Cont'd)

- Arbitrary number of levels
  - Expansion is easy by combining independently existing name spaces into a single name space by creating new root and making the existing roots as its children
  - The figure in the next slide illustrate this diagrammatically
  - Name that was unambiguous within its old namespace, is unambiguous in new namespace, even if the name also appeared in some other name space that was combined
  - No need to change any of the algorithms for name manipulation when the number of levels change
  - Its main disadvantage is that software for manipulating names tend to be more complicated when compared to constant level scheme

Constant vs. Arbitrary Number of Levels (Cont'd)



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

Major Issues in Human-Oriented Names

- In overall the arbitrary level scheme is:
  - More flexible than constant level schemes
  - This is used in all the recent distributed systems
  - Hence all discussions here will be based on arbitrary level naming system
- Major Issues in Human-Oriented Names
  - Selecting a proper scheme for global object naming
  - Selecting a proper scheme for partitioning a name space into contexts
  - Selecting a proper scheme for implementing context binding
  - Selecting a proper scheme for name resolution

Human Oriented Hierarchical Naming Schemes

- Combining an object's local name with its host name
  - The naming scheme uses a name space that comprises of several isolated name spaces
  - Each isolated name space corresponds to a node of a distributed system and a name in this name space uniquely identifies an object of the node
  - In a global system the objects are named by some combination of their - host name, local name which guarantees a system wide unique name where host name is the name of the node on which the object resides
  - This naming system is simple and easy to implement
  - Its main drawback is that it is neither location transparent nor location independent

Human Oriented Hierarchical Naming Schemes

- It is inflexible in the sense that object's absolute name changes every time the object migrates from one node to another
- Hence not suitable for modern distributed operating system

- Interlinking isolated name spaces into a single namespace
  - In this scheme, also the global name space is comprised of several isolated name spaces
  - However, unlike the previous scheme in which the isolated name spaces remain isolated throughout, the isolated namespaces are joined together to form a single naming structure the global name space
  - The position of these component name spaces in the naming hierarchy is arbitrary

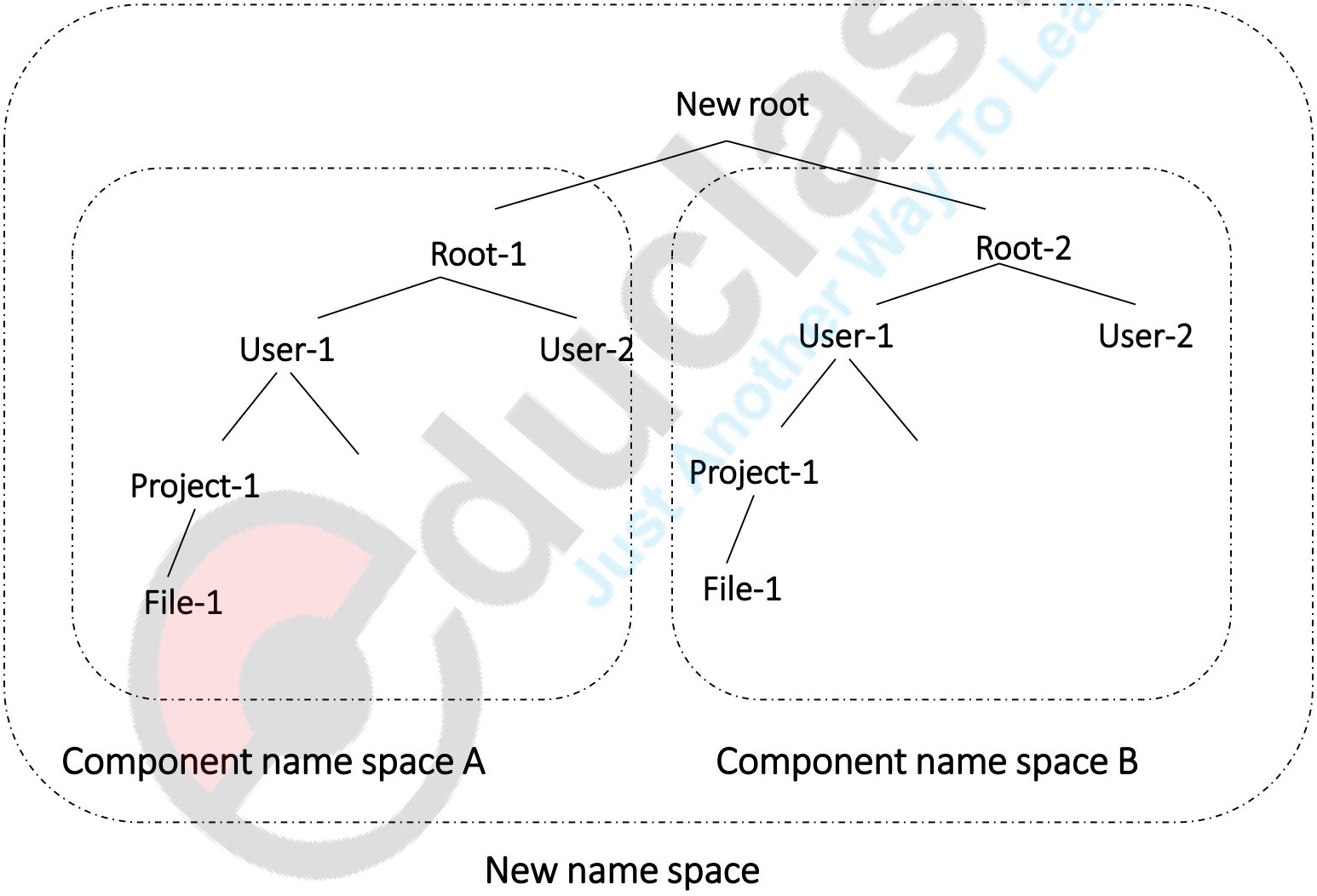
Human Oriented Hierarchical Naming Schemes

- In the naming structure, a component name space may be placed below any other name space either directly or through other component name spaces
- In this scheme there is no notion of absolute pathname
- Each pathname is relative to some context, either to the current working context or to current component name space
- This naming system is used in UNIX United to join number of UNIX systems to compose a UNIX United system
- In the single naming structure of UNIX United, each component system is a complete UNIX directory tree belonging to a certain node
- The root of each component name space is assigned a unique name so that they become accessible and distinguishable externally

Human Oriented Hierarchical Naming Schemes

- A component's own root is still referred to as / and still serves as the starting point of all pathnames starting with a /
- The UNIX notation ../is used to refer to the parent of a component's own root
- Hence, there is only one root that is its own parent and that is not assigned a string name, namely the root of the composite structure, which is just a virtual root needed to make the whole structure a single tree
- A simple example of such a naming structure is shown in the next slide, in which the composite structure is comprised of two component name spaces A and B whose roots are named *root-1* and *root-2*, respectively

Human Oriented Hierarchical Naming Schemes



Human Oriented Hierarchical Naming Schemes

- Note that within the component name space A, file-1 of the two component name spaces A and B will be referred to as /user-1/project-1/file-1 and ../root-2/user-1/project-1/file-1 respectively
- Similarly if the current working directory is if client is project-1 of component name space B, file-1 of the two component name spaces A and B will be refer to as ../../root-1/user-1/project-1/file-1 and file-1 respectively
- The main advantage of this naming scheme is that it is simple join existing name spaces into a single global name space
- However, In naming system that employ this scheme for global object naming, an important issue that need to be handled is to allow clients to use names that were valid in independent namespaces without need to update them to match new namespace structure

Human Oriented Hierarchical Naming Schemes

- Sharing Remote Name Spaces on Explicit Request
  - This scheme, popularized by Sun Microsystems' NFS, is also based on the idea of attaching the isolated name spaces of various nodes to create a new name space
  - In this scheme users are given the flexibility to attach a context of a remote name space to one of the contexts of their local name spaces
  - Once a remote context is attached locally, its objects can be named in a location transparent manner Ex. mount protocol
  - The goal is to allow some degree of sharing among the name spaces of various nodes in a transparent manner on explicit request by the users
  - Therefore, the global view of the resulting name structure is a forest of trees, one for each node, with some overlapping due to the shared subtrees

Sharing Remote Name Spaces on Explicit Request

- Unlike the previous scheme in which the entire name space of each node is attached to a single naming structure, in this scheme the users have the flexibility to attach to their local name space tree either a complete remote name space or a part of it (subtree)
- Request to share a remote namespace affects only the client from which the request was made and no other node
- This ensures node independence
- In NFS, mount protocol is used to attach a remote name space directory to a local name space
- Hence to make a remote directory accessible in a transparent manner from a node, a mount operation has to be performed from that node
- The node that performs mount operation is called the client node and the node whose name space directory is mounted is called a server node

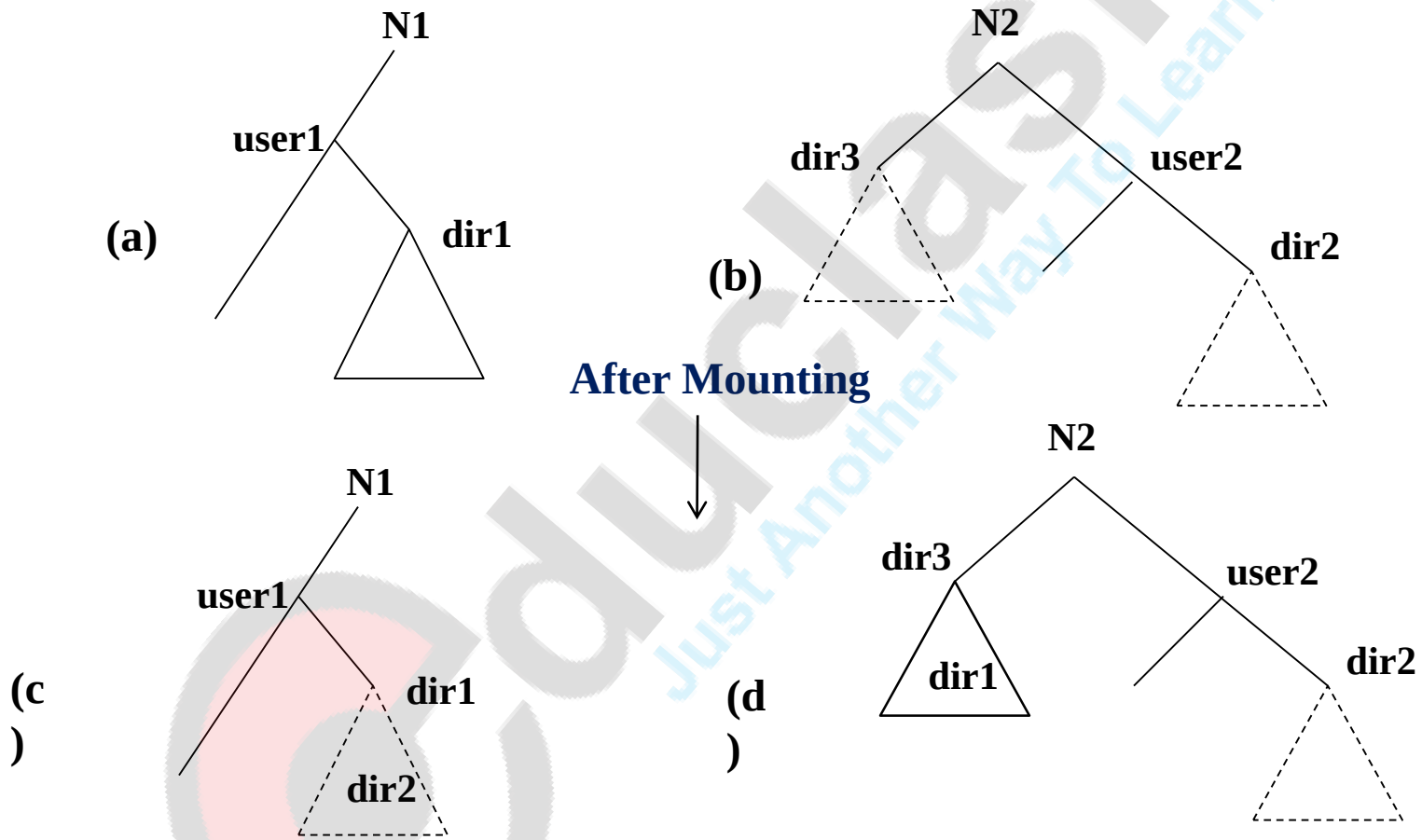
Sharing Remote Name Spaces on Explicit Request

- NFS allows every machine to be both a client and a server at the same time
- A server must export all those directories of its name space tree that are to be made accessible to remote clients and complete subtree rooted at an exported directory is exported as a unit
- Hence, all directories below an exported directory automatically become accessible to client as the mounting process sets up a link between mount point of client's local namespace & exported directory of server's namespace
- The semantics of the mount operation are that a server's file system directory exported by the server is mounted over a directory of the client's file system

Sharing Remote Name Spaces on Explicit Request

- The mounted directory's subtree of the server file system appears to be an integral part of the client's file system, replacing the subtree descending from the client's local directory on which the server's directory was mounted
- There is no difference between the file located on remote file server and a file located in the local disk
- A client can mount a directory in one of following ways:
- Manual mounting: mount command is to be used every time to access server directory is to be mounted on the client's local name space
  - umount allows to dismount the directory
  - The user must be a super user to use the mount and umount commands
  - This allows flexibility to the clients to dynamically mount and unmount servers' directories depending on the changing needs

Sharing Remote Name Spaces on Explicit Request



(a) A part of the name tree of node N_1; (b) a part of the node N_2; (c) structure of the name tree of Node N_1 after mounting dir2 of node N_2 over dir1; (d) structure of the name tree N_2 after mounting dir1 of node N_1

Sharing Remote Name Spaces on Explicit Request

- Static mounting: clients mount automatically the required directories of their choice out the directories exported by the server without manual intervention
  - The required commands to mount the desired files are written into a file called /etc/rc file, which is automatically executed when the client machine is booted
  - Hence all mount commands take place automatically at boot time
- Automounting: allows server directories to mounted only when required if the file is not accessed for a particular period of time, it is automatically dismounted

- Advantages
  - Provides flexibility to attach only necessary portions of remote name space

Sharing Remote Name Spaces on Explicit Request

- Instead of a single server, a set of servers' directories may also be simultaneously associated with a directory of client's local name space
- In this case, when the client accesses a file in the name space below the mount directory for the first time, the OS sends a message to each servers whose directories are associated
- The first one to reply wins, and its directory is mounted
- Automounting scheme helps to achieve fault tolerance because a server need to be up and working only when client actually needs it and when set of servers have the same file

- Disadvantages
  - It does not provide a consistent global name space for all nodes i.e. same object may have different absolute names when viewed from different nodes

Sharing Remote Name Spaces on Explicit Request

- The only way ensure that all nodes use the same name for the same object is by careful manual management of each nodes mount table
- Such systems do not scale well
- Require management of each node's mount table→ administrative complexity
- Directories exported by failed nodes become unavailable on its client nodes
- Similarly migrating objects from node to another requires changes in the name spaces of all the affected nodes
- Does not provide location transparency & user mobility as the specification of remote directory as an argument for mount operation is done in a non-transparent fashion

A single global name space

- A single global name structure spans all the objects of all the nodes in a DS
- Same namespace is visible to all users & object's absolute name is always the same irrespective of its location
- Hence the scheme supports location independency of - accessing as well as accessed object
- Most commonly used approach in modern Distributed Operating System
- Storing naming info in a centralized node or replicating it every node is not desirable because of the storage overhead involved
- Several distributed operating Systems use single global name space approach for object naming

A single global name space

- In this scheme, a single global name structure spans all the nodes in the system
- Hence the same name space is visible to all users and an object's absolute name is always the same irrespective of the location of the object or the user accessing it
- Thus the scheme supports both types of location independency – that of the location of the accessing object (e.g. process) and the location of the accessed object
- The single global name space approach is the most commonly used approach by modern Distributed Operating Systems
- Hence subsequent discussion will be based on this approach

Distributed Computing

Lecture 46

Human Oriented Names - II

Partitioning a Name Space into Contexts

- Name space management involves storage and maintenance of naming information, which consists of object names, object attributes, and the bindings between the two
- Due to the reliability and space overhead problems in distributed system, storing the complete naming information at a central node or replicating it at every node is not desirable
- Hence, the naming information is decentralized and replicated by having several naming servers, each having a copy of a portion of the complete naming information
- These name servers interact among themselves to maintain the naming information and to satisfy the name resolution requests of the users

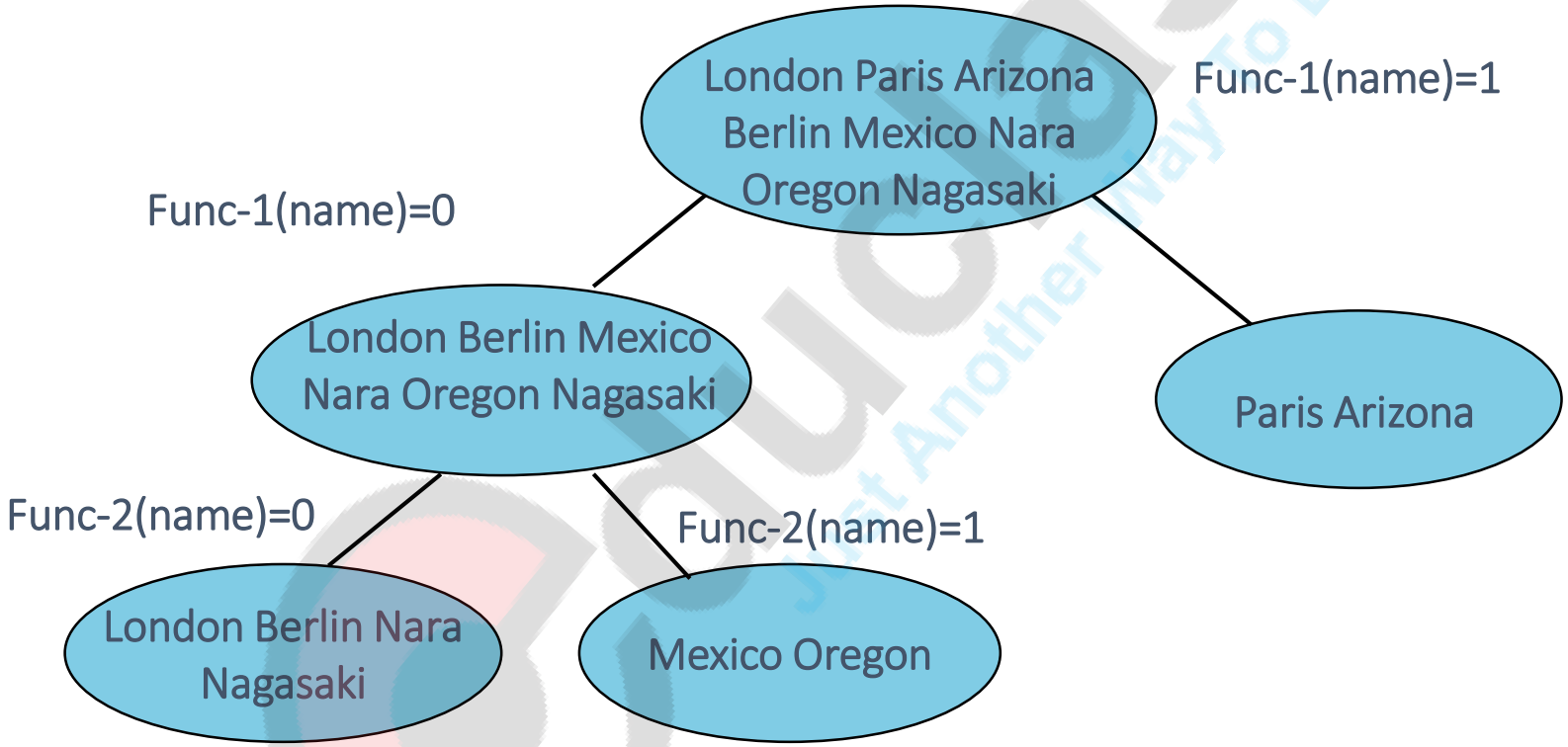
Partitioning a Name Space into Contexts

- A basic problem is how to decompose the naming information database to be among the name servers
- The main goal of the decomposition mechanism for this purpose is to minimize the overhead involved in the maintenance of naming information and the resolution of names
- The concept of context is used for partitioning a name space into smaller components
- Contexts represent indivisible units of storage and replication of information regarding the named objects
- A name space is partitioned into contexts by using clustering conditions
- A clustering condition is an expression that, when applied to a name, returns either a true or false value, depending on whether the given name exists in the context designated by the clustering condition

Algorithmic Clustering

- The three basic clustering methods are algorithmic clustering, syntactic clustering and attribute clustering which are discussed here
- Algorithmic Clustering
  - Names are clustered according to the value that results from applying a function to them
  - Therefore in algorithmic clustering, the clustering condition is a predicate of the form function(name) = value
  - For example a hash function that clusters the names into buckets is a good example of algorithmic clustering
  - Supports structure free name distribution i.e. does not have any restriction on the administrative control over parts of name space

Algorithmic Clustering (Cont'd)



Func-1(name) : if (total characters (name) = even)

Func-2(name) : if (total vowels (name) = even)

Algorithmic Clustering (Cont'd)

- Simple example of the clustering of names is given in the figure
- At first func-1 is applied to names in the name space to partition them into two contexts
- One of the context is still found to be large
- Hence a second function func-2 is applied to the names of this context to further partition the context into smaller contexts
- Thus starting with a complete name space as a single context, a sequence of clustering conditions can be applied to yield reasonably sized contexts
- Can be used for flat name spaces also

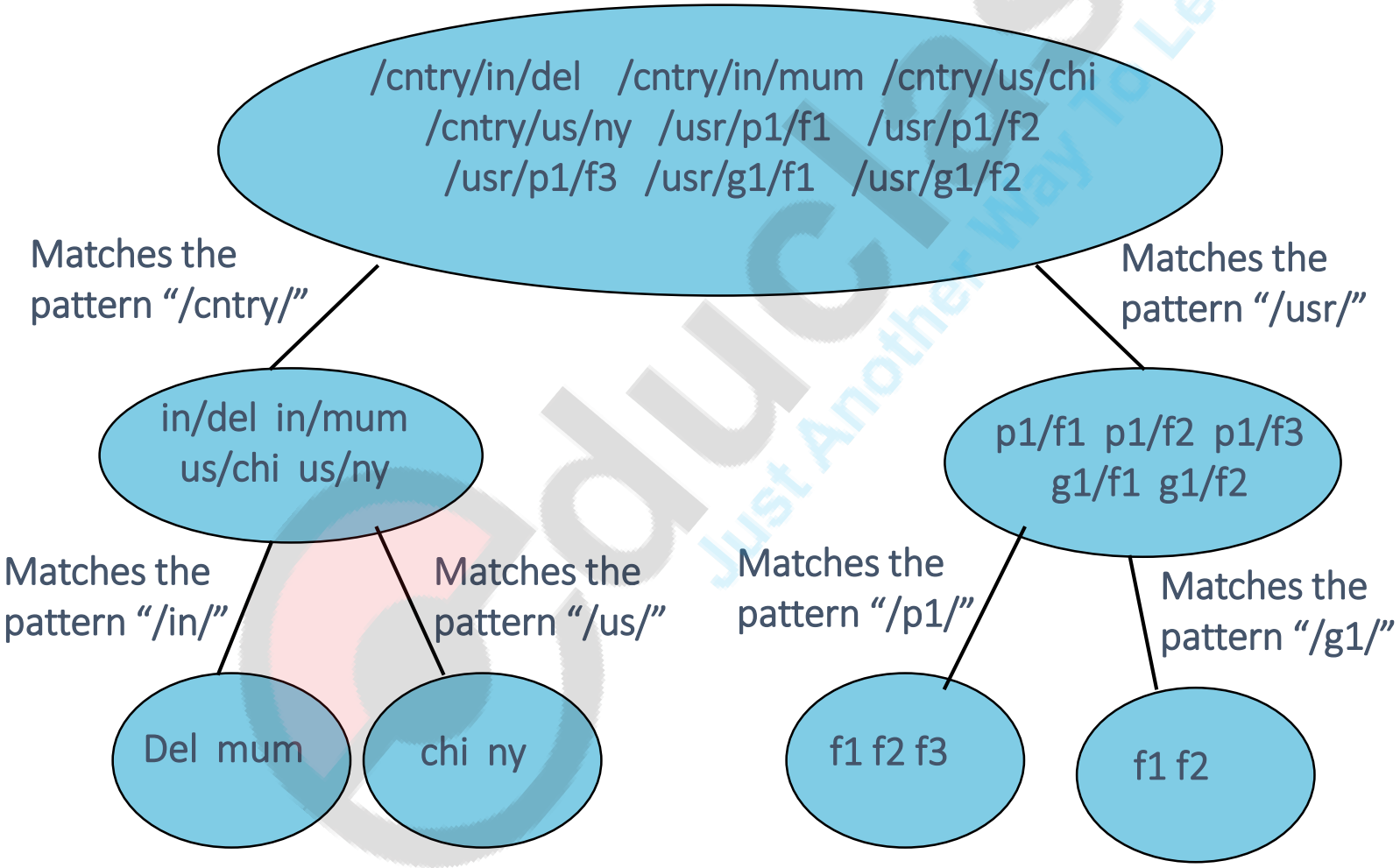
Algorithmic Clustering (Cont'd)

- The main advantage of algorithmic clustering mechanism is that it supports structure free name distribution that places no restriction on the administrative control over parts of name space
- The partitions of such a name space do not correspond to the structure of the names, such as their sizes, or the number component names or the order of component names or characters with in a name, and so on
- In particular the owner of an object may choose its authoritative name servers, subject to administrative constraints, independent of the object name
- Algorithmic clustering also has the advantage that it allows a healthy name resolution tree to be built even for flat name space
- Difficult to devise proper clustering functions; if not selected properly end up with some large and some too small clusters

Syntactic Clustering

- This is the most commonly used approach for partitioning name space into contexts
- Syntactic clustering is done using pattern matching techniques
- Patterns are the templates against which a name is compared, wild card search which are denoted by asterisks are supported and match any sequence of characters
- Thus a syntactic clustering condition that is meant for a particular pattern, when applied to a name, returns TRUE if the name matches the pattern
- All names that match particular pattern, such as names with a common prefix are considered part of the same context
- As an illustration a simple example of syntactic clustering of names are given in the next slide

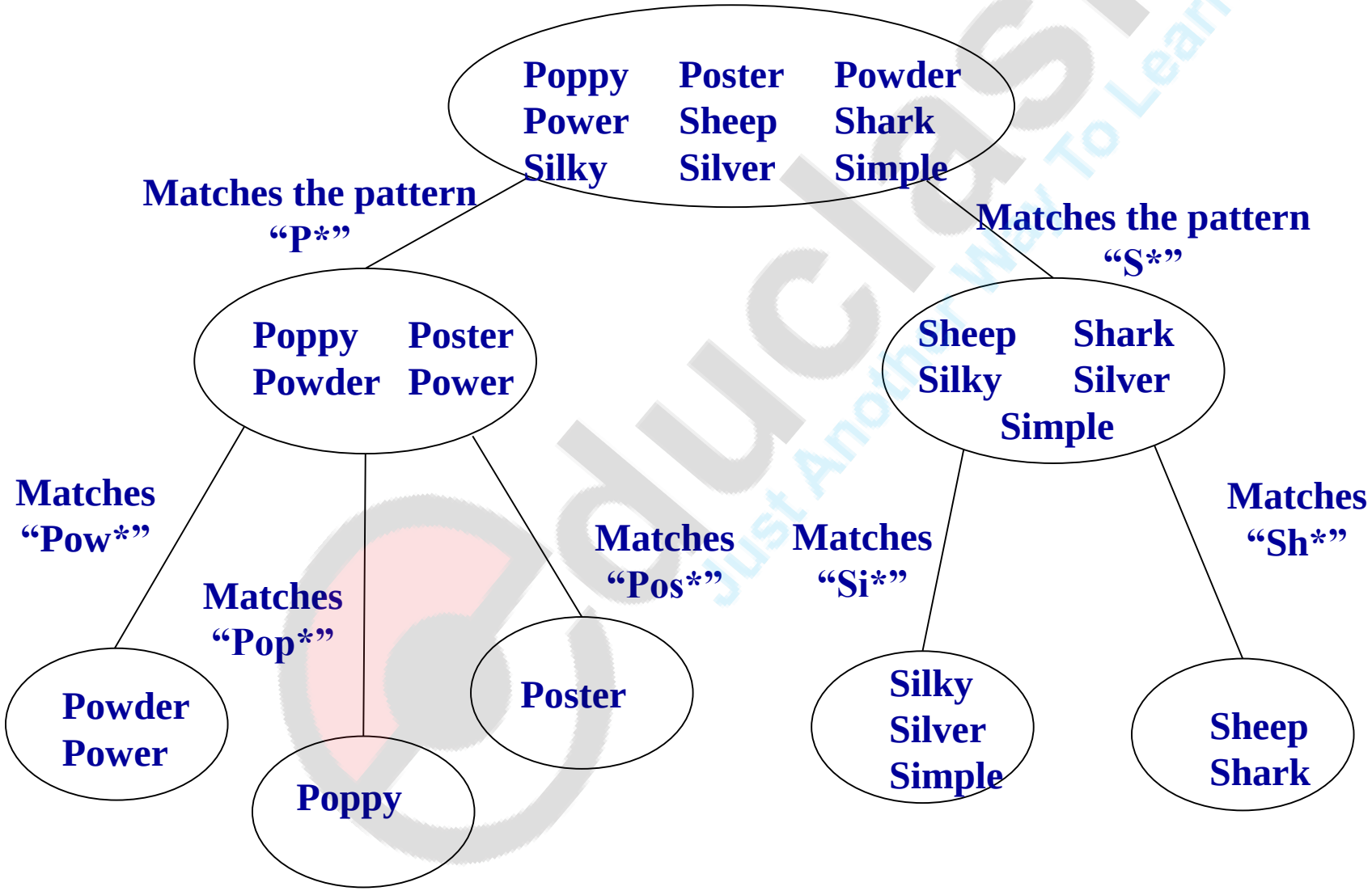
Syntactic Clustering (Cont'd)



Syntactic Clustering (Cont'd)

- It can be used for structured or unstructured names
- Figure shows the partitioning of structured names and figure in the following slide shows partitioning of flat names
- It can be seen from the first figure that hierarchically structured name spaces pattern matching is usually performed on a component-by component basis and resulting contexts usually contain only the unmatched part of the names
- Hence at subsequent levels, a new clustering condition is applied only on the truncated part of the names
- Syntactic clustering mechanism allows names to be resolved in a manner similar to their structure as is done by all management systems
- This means simple matching suffices as a clustering technique

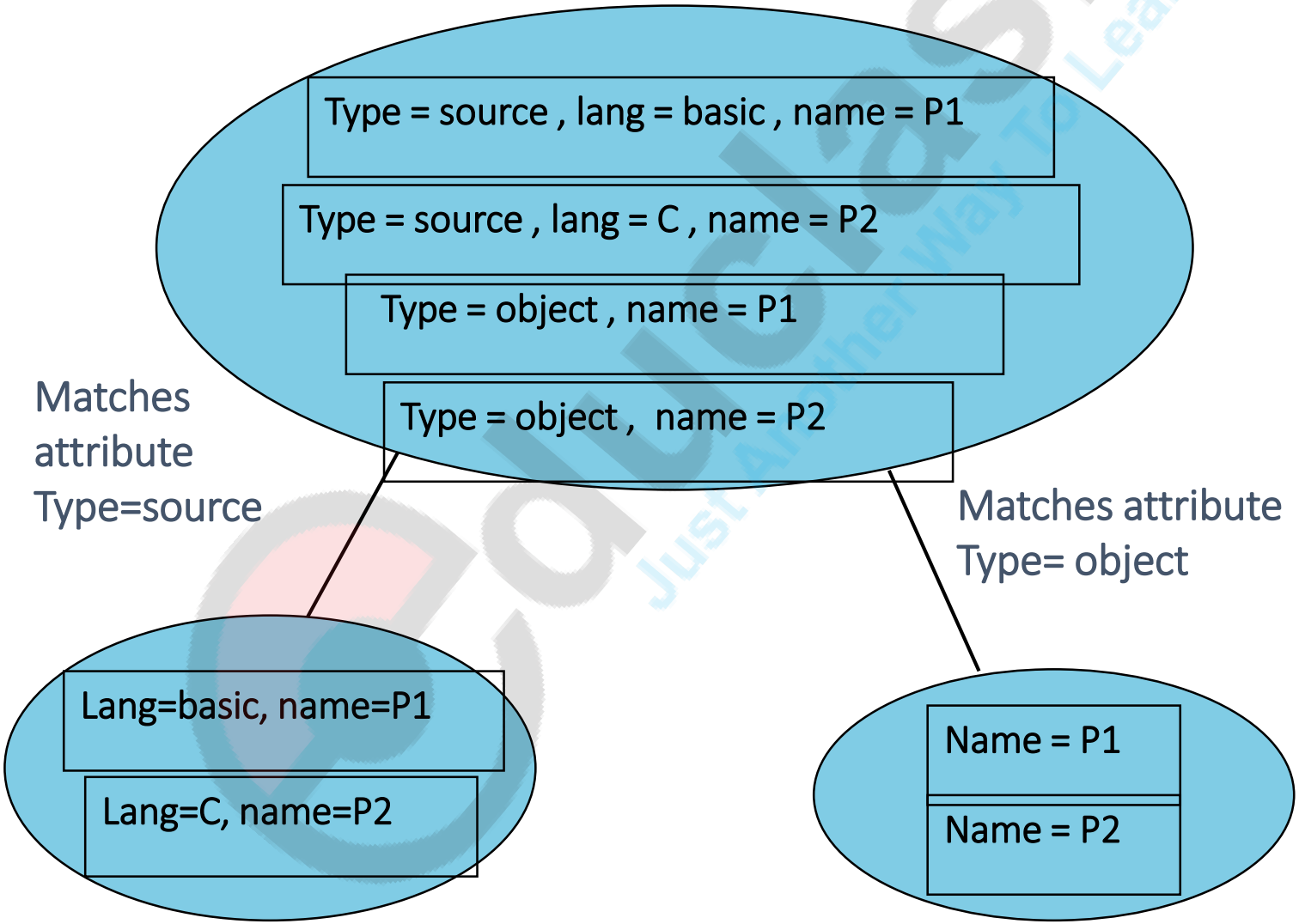
Syntactic Clustering (Cont'd)



Attribute Clustering

- The method of attribute clustering is used by attribute based naming conventions
- In this method names are grouped based on the attributes possessed by the names
- An attribute has both type and a value, where the type indicates the format and meaning of the value field
- Hence, all names having the same attribute (type, value) pair are placed in the same partition in the attribute clustering mechanism
- As an illustration, a simple example of attribute clustering of names is given in the slide

Attribute Clustering (Cont'd)



Attribute Clustering (Cont'd)

- Note that in a hierarchically structured name space, attribute clustering is usually performed on an attribute-by-attribute basis, and the resulting contexts contain only the unmatched attributes of the names
- At subsequent levels, a new clustering condition is applied only on the remaining attributes of the names
- However, attribute clustering conditions are not restricted to matching a single additional attribute in each step and several attributes may be matched in a single step

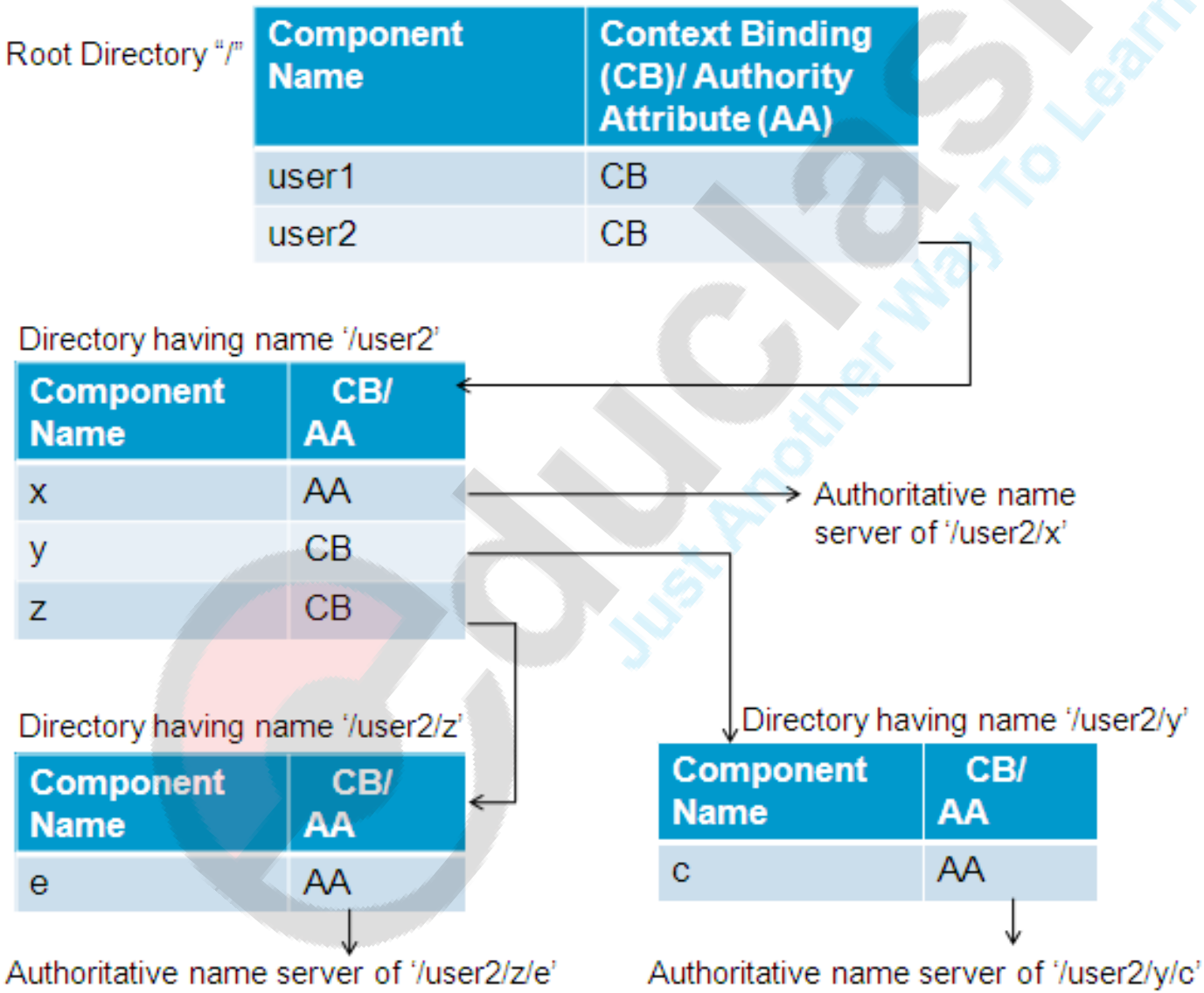
Context Binding

- The contexts of a name space are distributed among the various name servers managing that name space
- i.e., a name server normally stores only a small subset of the set of contexts of the name space
- When a name is to be resolved, a server first looks in local contexts for an authority attribute for the named object
- Authority attribute contains a list of the authoritative name servers for the named object
- If the authority attribute is not found in a local context, checks context binding
- Context Binding associates the context within which it is stored to another context, that has more information about named object & name servers that store that context

Context Binding (Cont'd)

- Two Strategies normally used for context binding are:
- Table - Based Strategy
  - Most commonly used approach for implementing context binding in hierarchical tree structured name spaces
  - Each context is a table having two fields: the first field stores a component name of the named object and the second field either stores context binding information or authority attribute information
  - The context bindings reflect the delegation of authority for managing parts of the name space
  - The contexts of table based strategy is also called as directories
  - As an illustration an example of table based strategy for implementing context bindings is given in the next slide

Table - Based Strategy



Context Binding (Cont'd)

- Procedure - Based Strategy
  - Context binding done by procedure, which, when executed, supplies information about the next context to be consulted for the named object
  - The syntactic clustering condition used by each context can also be used as the procedure for supplying context binding for the names defined with in context
  - Less flexible when compared to table based strategy as changing of context binding will require changes in clustering procedures
  - No configuration data is required & hence no data management required

Distribution of Contexts and Name Resolution Mechanisms

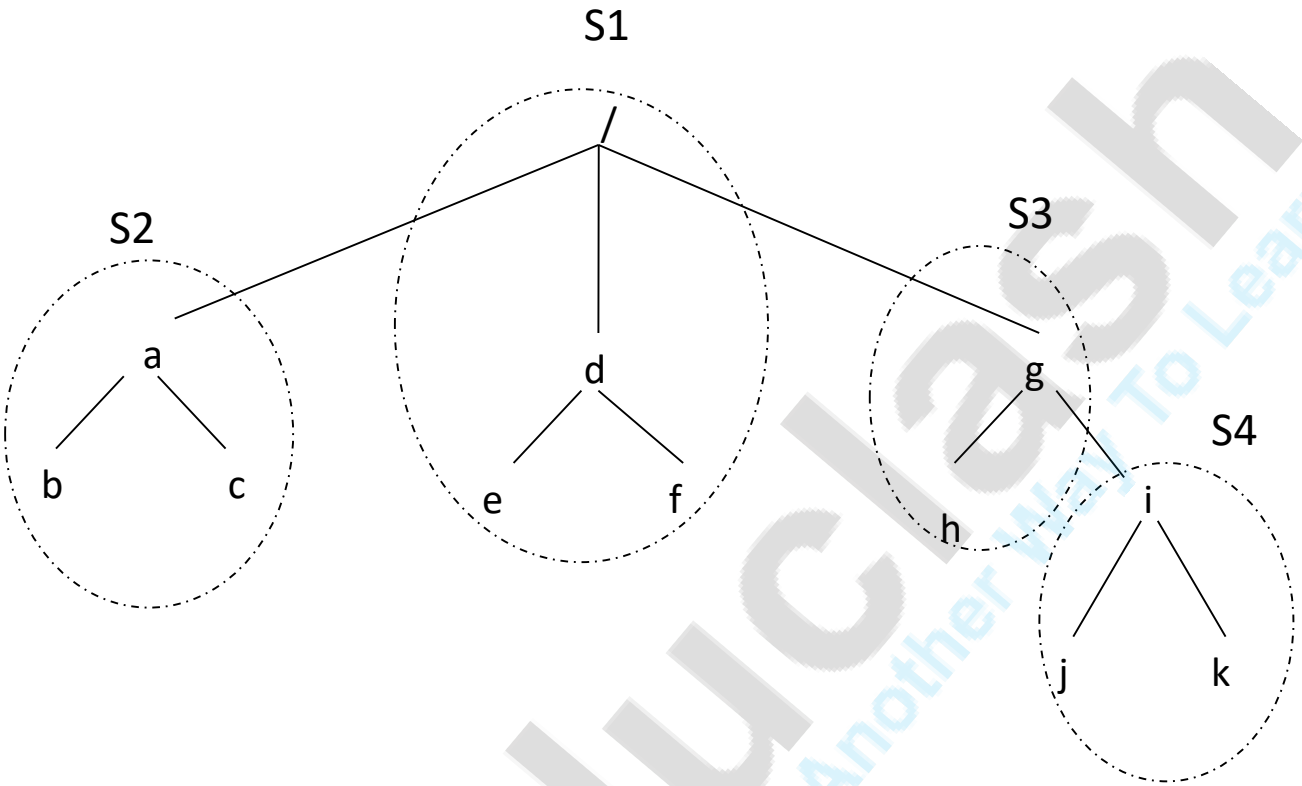
- Name resolution is the process of mapping an object's name to the authoritative name servers of the object
- It involves transversal of a resolution chain of contexts until the authority attribute of the named object is found
- Traversal of the resolution chain of contexts for name is greatly influenced by the location of these contexts in a DS
- Some of the commonly used name resolution mechanisms:
  - Centralized approach
  - Fully replicated approach
  - Distribution based on physical structure of name space
  - Structure free distribution of contexts

Distribution of Contexts and Name Resolution Mechanisms (Cont'd)

- Centralized approach
  - A single name server in the entire distributed system is located at a centralized node
  - It is responsible for storing all contexts and maintaining the name mapping information in them up to date
  - Location of centralized server known to all nodes
  - Simple, easy to implement and efficient but not scalable & reliable
- Fully replicated approach
  - A name server is located on each node of distributed system & all contexts are replicated at every node
  - Simple & efficient as resolution requests are serviced locally without the need of any communication with other nodes
  - Large resource overhead especially for large name space and in maintaining consistency over all the nodes
  - Not suitable for large name spaces

Distribution of Contexts and Name Resolution Mechanisms (Cont'd)

- Distribution based on physical structure of name space
  - Most commonly used approach for hierarchical tree-structured name spaces
  - The name space is divided into several subtrees that are known by different names in different systems like domains, zones etc
  - There are several name servers, with each server providing storage for one or more zones (domains)
  - Each client maintains a name prefix table
  - Consistency of name prefix table entries is maintained be detecting & updating stale table entries on use
- Advantages
  - Matching name prefix rather than full names allows a small number of table entries to cover large number of names resulting in good performance



Name Prefix	Zone Identifier	
	Server Identifier	Specific Zone Identifier
/	S1	12
/a	S2	33
/g	S3	24
/g/i	S4	56

Distribution of Contexts and Name Resolution Mechanisms (Cont'd)

- Name prefix table helps in bypassing part of directory lookup mechanism
- So performance & reliability enhances because a crash on one name server does not prevent clients from resolving names in zones on other servers
- On use consistency checking saves consistency control overheads

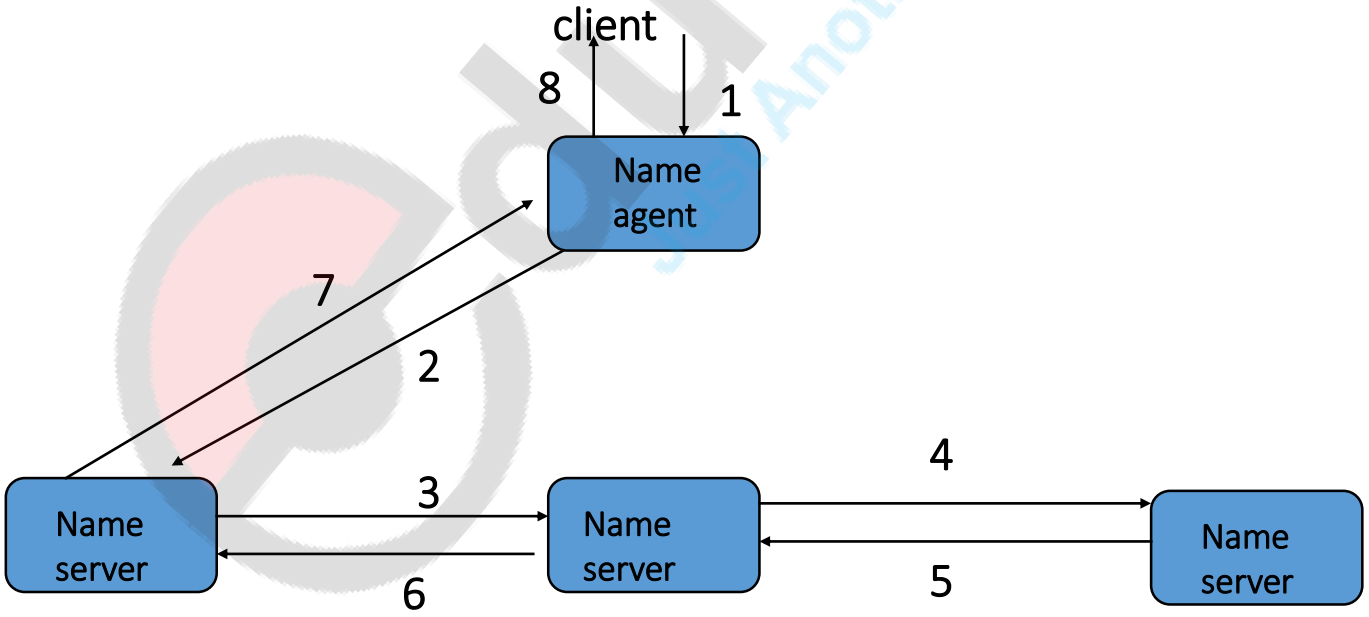
- Structure Free Distribution of Contexts
  - Name services should be able to be reconfigured if present servers become overworked or if system scale changes
  - Reconfiguration might require changing an object's authoritative name server
  - Use Structure Free Distribution: Context of a namespace can be freely stored/ moved at any name server independently of any other context
  - Permits maximum flexibility

Issues in Structure Free Distribution

- Locating Context objects during Name Resolution
  - Name is always consists of a context/name pair; i.e., name is always associated with a context
  - Using metacontext
    - It contains name & authority pairs for all context objects in namespace and size of the meta context depends on the no of contexts in name space
    - To resolve a name first metacontext is referred to obtain authority attribute of context
  - Always start resolution at root context which is replicated at all name servers; used when the name space is structured as single global hierarchical name tree
- Interacting with name servers during Name Resolution
  - Various contexts of a given pathname may be stored at different name servers and hence the resolution of a pathname is such a situation will involve interacting with all the name servers that store one or more contexts of pathname

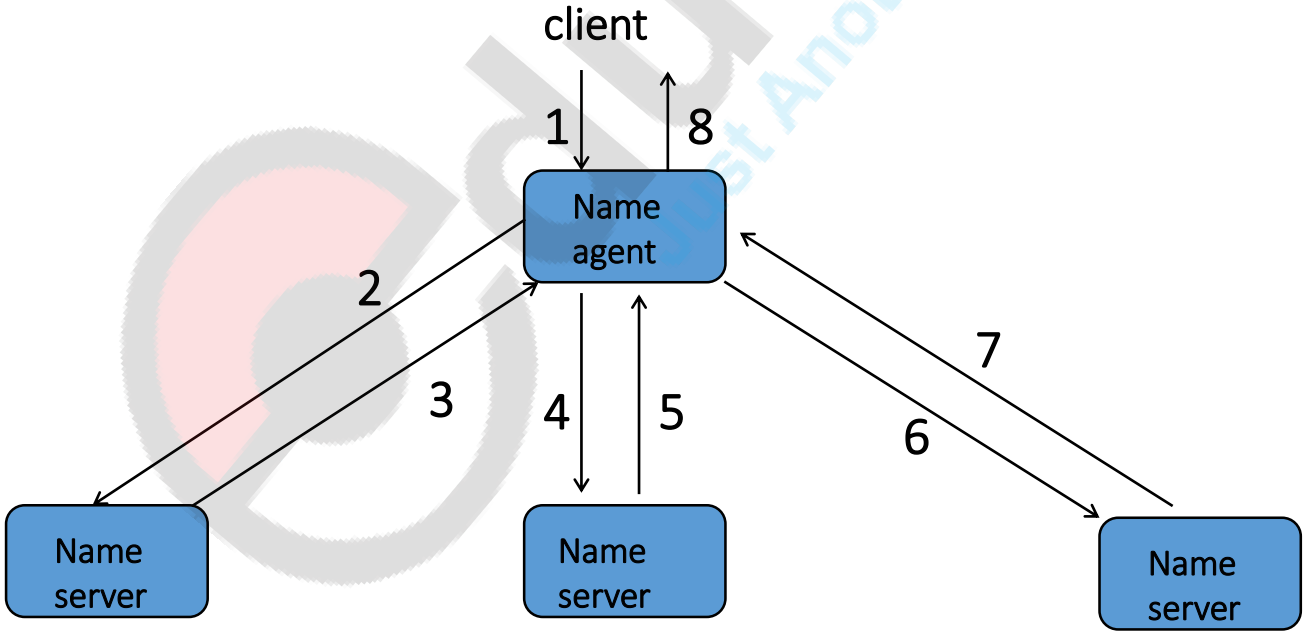
Recursive

- The name agent forwards the name resolution request to the name server and stores the first context needed to start resolution of the given name
- Then the name servers storing the contexts of the given pathname are recursively activated on after another until the authority attribute of the named object is extracted
- Recursively in the opposite direction the authority attribute to the name agent



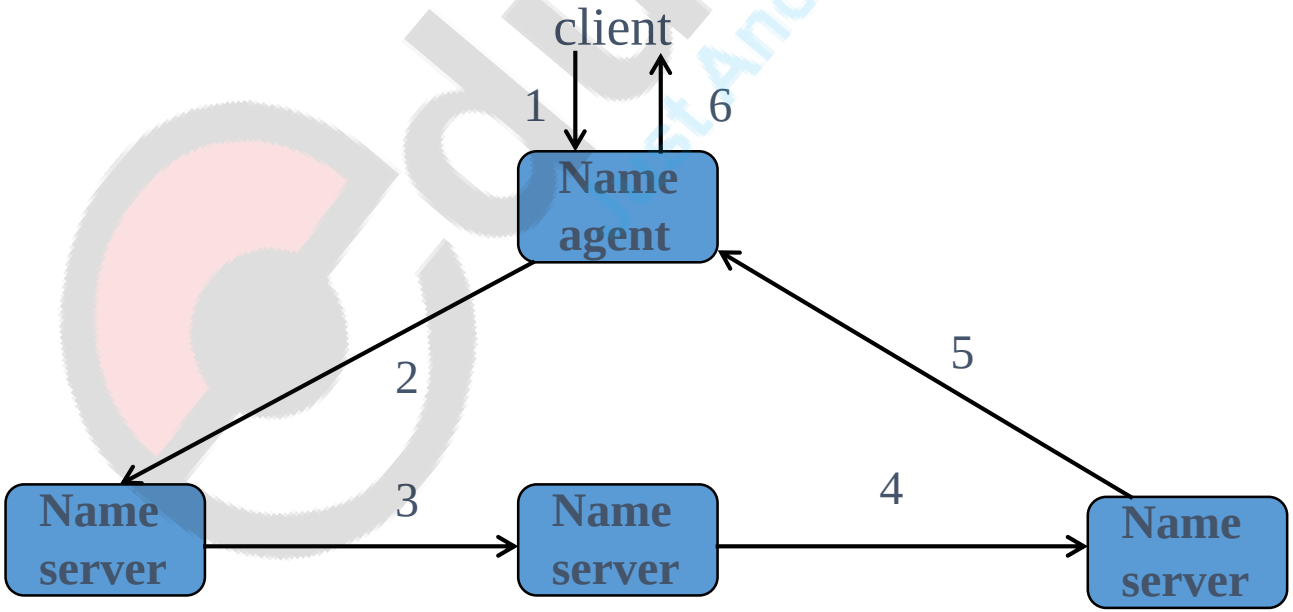
Iterative

- In this method name server does not call each other directly; rather the name agent retains the control over the resolution process and calls each of the servers involved in the resolution process
- If resolution, the first server sends the authority attribute of the object else the next server to be contacted
- The process continues until the name is fully resolved and receives the authority attribute of the named object



Transitive

- The name agent sends the resolution request to the name server that stores the first context required to start the resolution process
- It resolves as many components as possible and passes on the unresolved portion to the next server
- This process continues until authority attribute of the named object is encountered and it is returned directly to he name agent



Distributed Computing

Lecture 47
Name Caches

Name Caches

- It has been found that in OS that provide a flexible hierarchical name space, the system overhead involved in name resolution operations is considerably large
- Studies have shown that nearly 40% of the system call overhead in UNIX is for file name resolution
- Even in case of network traffic, large portion is naming related
- Investigations have found that a simple distributed name cache can have substantial positive effect on distributed system performance
- Client caches the result of a name resolution operation for a while, rather than repeating it every time the value is needed
- Characteristics of name service related activities:

Name Caches (Cont'd)

- High degree of locality of name lookup (spatial locality):
  - The property of "locality of reference" has been observed in program execution, file access , as well as database access
  - Measurements clearly show that a high degree of locality also exists in the use of pathnames for accessing objects
  - Due to this locality feature a reasonable size name cache used for caching recently used naming information provide excellent hit ratios
- Slow update of name information database:
  - It has been observed that naming data does not change very fast, so inconsistencies are rare
  - The activity of most users is usually confined to a small, slowly changing subset of the entire name information database
  - Most of the naming data have a high read-to-modify ratio

Name Caches (Cont'd)

- Hence the cost of maintaining the consistency of cached data is very low

- On-use consistency of cached information is possible
  - An attractive feature of name service related activity is that it is possible to find that some thing does not work if one tries to use obsolete naming data, so that it can be attended to at the time of use
  - i.e., Name cache consistency can be maintained by detecting and discarding stale cache entries on use
  - With on-use consistency checking, there is no need to invalidate all related cache entries when a naming data update occurs, yet stale data never causes a name to be mapped to a wrong object

Types of Name Caches

- Depending on the type of info stored in each entry name cache can be:
- Directory cache
  - Each entry consist of a directory page and used in those systems that use iterative method of name resolution
  - All recently used directory pages that are brought to the client node during name resolution are cached for a while
  - Operations like listing the contents of a directory, expanding wild card arguments, and accessing parent directories all use information found in directory pages
  - Entire page cached for only one useful entry and hence blocks a large area of precious cache space
  - Large size cache required

Types of Name Caches (Cont'd)

- Prefix cache
  - This type of name cache is used in naming systems that use the zone-based context distribution mechanism
  - In this name cache each entry consist of a name prefix and the corresponding zone identifier
  - Remember that a name prefix corresponds to a zone in the zone-based context distribution approach
  - We have discussed this method in detail earlier while discussing distribution of contexts and name resolution mechanisms
  - This type of name cache is not suitable for use with the structure-free context distribution approach

Types of Name Caches (Cont'd)

- Full-name cache
  - Each entry consist of an object's full pathname and the identifier, and location of its authoritative name server
  - Hence, requests for accessing an object whose name is available in the local cache can be directly sent to the object's authoritative server
  - This type of name cache can be conveniently used with any naming mechanism, although it is mainly used by the naming systems that use structure-free context distribution approach
  - Notice that in a prefix cache an entry usually serves as a mapping information for several objects, but in full name cache each entry serves as mapping information for single object
  - Hence, full name caches are larger in size as compared to prefix caches

Name Cache Implementation

- The two commonly used approaches to name cache implementation are:
- A cache per process
  - A separate name cache is maintained for each process in process's own address space and is usually small in size
  - Accessing of cached information is fast
  - No memory area of the OS is occupied by name caches
  - Every new process has to create its cache from scratch as the process oriented name cache vanishes with the process
  - Caches will have short lifetime if processes are short lived
  - Hit ratio low due to start-up misses, which are initial misses that occur when a new empty cache is created

Name Cache Implementation (Cont'd)

- To alleviate problem of startup misses, use cache inheritance to give cached data a long life time by each process inheriting its cache contents from its parent process
- The use of process oriented cache is limited only to a single process due to which there is a possibility that the same naming information duplicated in several caches on same node

- A single cache for all processes of a node
  - Single cache maintained at each node for all the processes of that node
  - These caches are larger in size and located in memory area of the OS, thus slower as compared to process oriented caches
  - Cache information is long lived and is removed only by replacement policy finds it suitable for being removed

Name Cache Implementation (Cont'd)

- Hence the problem of startup cache miss problem is not present with this approach
- Higher average hit-ratio as compared to process-oriented caches, as there will be larger collection of names in the cache at all times
- There is no duplication of naming information on the same node

- Multicache Consistency
  - When a naming data update occurs, related name cache entries become stale and must be invalidated or updated properly
  - Two commonly used approaches for multicache consistency of name caches are immediate invalidate and on-use update

Multicache Consistency

- Immediate invalidate
  - All related name cache entries are immediately invalidated when a naming data update occurs
  - Two methods of doing this:
    - When ever a naming data update operation is performed, an invalidate message is broadcasted to all nodes in the system
    - Each node's name caches are then examined for the presence of the updated data, if present the cache entry is invalidated
    - Its use becomes prohibitive for big networks with large number of nodes
    - To avoid the use of broadcast protocol, in the second approach, the storage node of naming data (for example storage node of directory) keeps a list of nodes against each data that corresponds to nodes on which data is cached
    - When a storage node receives a request for a naming data update, only the nodes in the corresponding list are notified about a naming update

Multicache Consistency (Cont'd)

- This method is acceptable only if small number of nodes share a naming data when that data is modified and there is a low rate of update to naming data

- On-use update
  - This is more commonly used method for maintaining name cache consistency
  - No attempt is made to invalidate all related cache entries when a naming data update occurs
  - When a client uses stale data, it is informed by the naming system that the data being used is either incorrectly specified or stale
  - On receiving a negative reply, necessary steps are taken (either by broadcasting or multicasting a request or by using some other implementation dependent approach) to obtain updated data which is then used to refresh the cache entry

Naming and Security

- An important job of the naming system of several centralized and distributed operating systems is to control unauthorized access to both named objects & information in the naming database
- Many different security mechanisms have been proposed and used by operating systems to control unauthorized access to various resources of the system
- Naming related access control mechanisms are three types:
  - Object names as protection keys
    - An object's name acts as a protection key for the object
    - Only the user who knows name of an object can access the object by using its name
    - An object can have several keys in those systems that allow an object to have multiple names

Naming and Security (Cont'd)

- In this case any of the keys can be used to access the object
- In this mechanism, user is not allowed by the system to define a name for an object that they are not authorized to access
- Obviously, if an user can not name an object, he or she can not operate on it
- This scheme is based on the assumption that object names can not be forged or stolen
- i.e., there is no way for a user to obtain the name of other user's objects and the names can not be guessed easily
- However, in practice object names are picked to be mnemonic, they can often be easily guessed
- Hence the scheme does not guarantee a reliable access control mechanism
- Another limitation is that it does not provide flexibility of specifying modes of access control like read only to some and read & write to somebody else

Naming and Security (Cont'd)

- Capabilities
  - A simple extension to the above scheme that overcomes its limitations
  - A Capability (object identifier, Rights information) is a special type of object identifier that contains additional information redundancy for protection
  - It can be considered as an unforgeable ticket that allows its holder o access an object (identified by its object identifier part) in one or more permission modes (specified by its access control information part)
  - Hence the capabilities are object names having the following properties
    1. A capability is a system oriented name that uniquely identifies an object
    2. In addition to identifying an object, it is also used to protect the object it references by defining operations that may be performed on the object it identifies

Naming and Security (Cont'd)

3. A client that possesses a capability of an object can access it in the modes allowed by it
4. There are usually several capabilities for same object and each one confers different access rights to its holders
    - The same capability held by different users provides the same access right to all of them
5. All clients that have capabilities to a given object can share the object
    - The exact mode of sharing depends on the capability possessed by each client of the same object
6. Capabilities are unforgeable protected objects that are maintained by operating system and only indirectly used by the users
    - Capability based protection relies on the fact that the capabilities are never allowed to migrate into any address space directly accessible by a user process (where they can be modified)

Naming and Security (Cont'd)

- Thus they can not be modified by user process
- If all capabilities are secure, the objects they protect are also secure against unauthorized access

- When a process want to perform an operation on an object, it sends to the name server a message containing the object's capability
- It verifies the capability and allows the requested operation else permission denied message is returned to the client process
- If allowed, the clients request is forwarded to the manager of the object
- The capability-based approach has no checking for the user identity and if it is required, separate user authentication mechanism must be used

- Associating protection with name resolution path
  - Protection can be associated either with an object or with the name resolution path used to identify the object

Naming and Security (Cont'd)

- The most common scheme provides protection on the name resolution path
- Systems using this approach employ an Access Control List (ACL) based protection, which controls access dependent on the identity of the user
- The mechanism based on ACL requires, in addition to the object identifier, another trusted identifier representing the accessing principal, the entity with which access rights are associated
- This trusted identifier might be a password, address, or any other identifier form that cannot be forged or stolen
- An ACL is associated with an object & specifies user name & types of access allowed for each user of that object
- Whe a user requests access to an object, the operating system checks the ACL of that object for every access made by any user and type of access
- If the user is listed for the requested access, the access granted

Naming and Security (Cont'd)

- Otherwise, a protection violation occurs and the user job is denied access to the object
- By associating an ACL with each context (directory) of the name space, access can be controlled to both the named objects and the information in the naming database
- To access an object, user must be having access permission to both directories of object's pathname & object itself
- Hence associating protection with the name resolution path of an object name provides additional layer of protection to the named object
- Also, in systems where objects have multiple pathnames (such as acyclic or general graphs), a given user may have different access rights to an object depending on the pathname used

Distributed Computing

Lecture 48

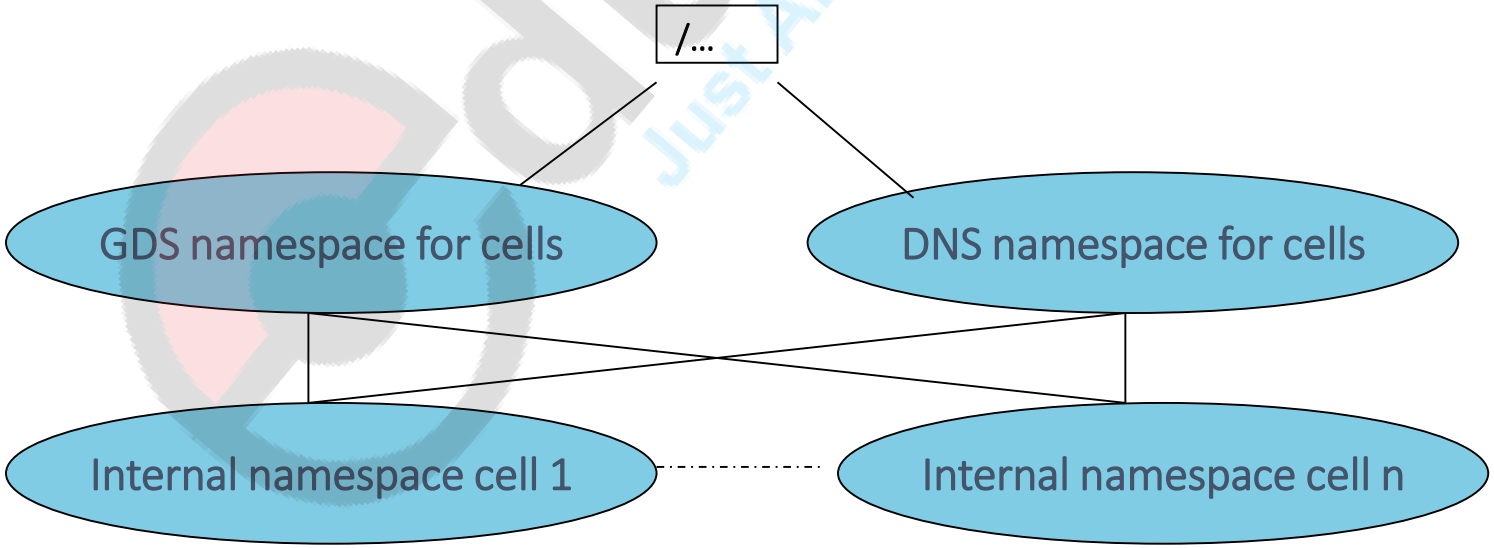
DCE Directory Structure

DCE Directory Service

- As a case study of how the naming concepts and mechanisms described till now can be used to build a naming system for a DCS, the DCE directory service is discussed in brief
- In DCE system all users, machines, and resources etc that have a common purpose and share common services are grouped into cells
- Components of DCE Directory service for intracell and intercell naming:
  - Cell Directory Service (CDS)
    - Controls intracell naming environment and every cell has at least one CDS server
  - Global Directory Service (GDS)
    - It Controls global naming environment between cells
    - It links cells together so that any cell can be located from any other cell

DCE Name Space

- GDS implementation is based on international standard X.500 directory service
- Since many DCE users use the Internet, DCE also supports standard Internet Domain Name System (DNS) for cell naming
- Hence DCE names can also be specified in DNS notation
- DCE uses Single global namespace approach for object naming



DCE Name Space (Cont'd)

- DCE name space is a single worldwide structure, with global root (/…), below which appears GDS name space, used to name each cell
- Each name is unique and has different parts and separated by slash(/)
  - Prefix (Indicates whether the name is local to current cell or global to the entire DCE name space the prefix (/.:) is used for local name and /… is used to denote for global name
  - Cell Name: This is an optional part specified only when the prefix /… is used for the first part of the name
    - This part of the name can be specified either in X.500 notation or in DNS notation
    - A global name should contain this part, whereas a local name must not contain this part
  - Local name: uniquely Defines an object within a cell
    - Unix like hierarchical naming scheme is used for local names

X.500 Notation

- It uses hierarchical, attribute based naming scheme e.g.

  /country=US/OrgType=COM/OrgName=IBM/Dept=ENG/Person=Nancy/

- Which uniquely identifies a person named Nancy who belongs to the engineering department of the company named IBM in U.S.A
- In X.500 terminology each component of a name is called the *relative distinguished name* (RDN) and the full name is called *distinguished name* (DN)
- It also provides aliasing similar to the symbolic link in file system
- The X.500 name tree is called *Directory Information Tree* (DIT) and the entire directory structure is including the data associated with the nodes is called *Directory Information Base* (DIB)

X.500 Notation (Cont'd)

- X. 500 uses an object oriented information model for grouping DIB entries into classes
- Each DIB entry has an *ObjectClass* attribute that determines the class of the object to which the entry refers.
- e.g., in the example *Country*, *OrgType*, *OrgName*, *Dept* and *Person* are all examples of values of *ObjectClass* attribute
- The definition of a class determines which attributes are mandatory, and which are optional for entries in the given class
- The *ObjectClass* attributes are always mandatory, whose value must be name of one or more classes
- If the *ObjectClass* attribute of an object has two or more values, that object inherits the mandatory and optional attributes of each of the corresponding classes

The DNS Notation

- The DNS is the standard scheme for naming hosts and other resources on the internet
- It uses a hierarchical, tree structured name space partitioned into domains
- In this scheme, a name consists of one or more strings called labels separated by the delimiter "." and there is no delimiter in the beginning or end of a name
- Names are written with the highest level domain on the right
- The internet DNS name space is partitioned both organizationally and according to geography
- It divides the world into top-level domains consisting of country names

The DNS Notation (Cont'd)

- e.g. *us* (United states) which is the default and hence omitted, *uk* (United Kingdom), *fr* (France), *in* (India) and so on
- Individual countries have their own next level of domains of the name space tree
- e.g. *edu* stands for education, *com* (commercial), *gov* (government) , *mil* (military) etc
- The organizational domains further have subdomains such irctc.co.in, google.co.in etc.
- There are registration authorities responsible for the registration of names in a domain at a particular are different
- e.g. google.co.in has be agreed upon by the authority who manage the domain co.in

Intracell Naming

- All names within a cell managed by Cell Directory Service (CDS)
- CDS Directories
  - The CDS of a cell basically manages the CDS directories of a cell that are organized in hierarchical tree structure
  - Manages names & attributes of all objects within the cell
  - Each directory has a number of directory entries
  - Each entry has a name, a set of attributes & protection information
  - The server managing the object also manages its protection information
  - Permission to access a directory does not imply permission to access the named object
  - The server knows which user have what type of access rights for an object

Intracell Naming (Cont'd)

- Replication of naming information
  - For better performance and reliability CDS supports with replication of its information with the unit of replication being a directory
  - A collection of directories forms a *clearinghouse*
  - A clearinghouse is a physical database managed by a CDS server
  - Every DCE cell must run at least one CDS server, most will run two or more with critical information replicated among them
  - Each CDS server maintains one or more clearinghouse
  - Each replica of a directory resides in different clearinghouse
  - The root directory is replicated in all clearinghouses to allow a search for any name to be begun by any CDS server

Intracell Naming (Cont'd)

- When a new directory is created, it automatically creates an entry for this directory in its parent directory
- This entry is used to track the location of a child directory even when the parent and child directories are located in different clearinghouses
- The root directory contains entries for all its children directories
- They in turn contain entries for their own children directories and so on
- Hence given the root directory, this connectivity enables CDS server to find every directory and thus every entry in the name space

- Consistency of Replicated Naming Information
  - To maintain consistency of the naming information, DCE uses primary-copy protocol for directory update operations
  - That is for each replicated directory one copy is designated as primary copy and the remaining as secondary copies

Intracell Naming (Cont'd)

- Read operations can be performed using any copy of a replicated directory, primary or secondary
- All update operations are directly performed only on the primary copy
- One of the following two approaches are used to update the secondary copies
- Update propagation: In this method, when the primary copy of a directory is updated, changes are immediately sent to all the secondary copies, like for example naming information which need to be kept consistent at all times
- Skulking: In this method, less critical updates are accumulated and sent together as a single message to secondary copies periodically

CDS Implementation

- CDS implementation uses client – server model with CDS server daemon and client daemon processes
- A CDS server running on server machine stores and manages one or more clearing houses and handles requests to create, modify or look up names in its local clearinghouse
- The client or CDS clerk runs on every client machine that uses CDS
- The CDS clerk receives requests from client application and interacts with one or more CDS servers to carry out the request and return the result to the client application
- A CDS clerk also maintains a name cache in which it saves the results of name resolution requests for future use
- The cache is written periodically to the disk, so that the information can survive a system reboot or application restart

How does the CDS Clerks learn about CDS Servers

1. *By broadcasting*: CDS servers periodically broadcast their existence
   - CDS clerks learn about all about those CDS servers resident on the same LAN as that of CDS clerk
2. *During a name resolution*: If local CDS server can not resolve a name with the information in its local clearinghouse, it returns location of another CDS server that has more information about resolving the given name
3. *By management command*: A DCE administrator can use a CDS control program to create information about a CDS server in CDS clerk's cache
   - This is normally used when CDS clerk and CDS server reside in different LANs so that broadcast messages sent by the CDS server on its own LAN cannot be received by the CDS clerk on different LAN

Name Resolution

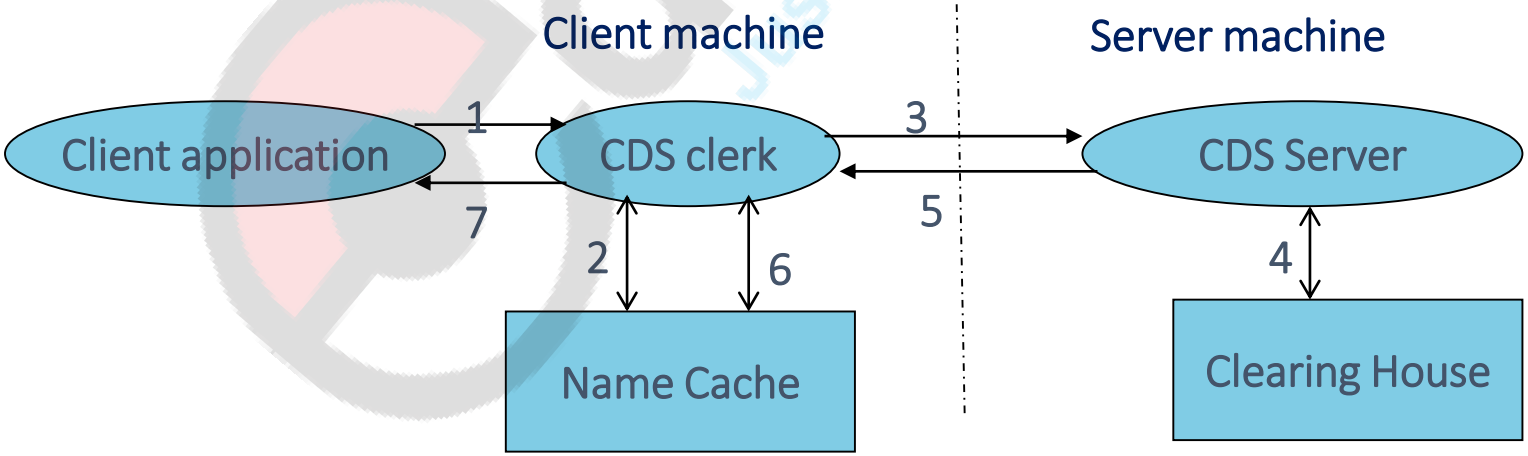
- A name resolution operation called lookup in CDS is performed as shown in the figure below

1. The client application sends a lookup request to its local CDS clerk in an RPC message
2. If the name is found in the name cache it returns a reply to client and the name resolution is complete
3. If the name not in the cache, CDS clerk does an RPC with a CDS server it knows about
4. With the directories available in its local clearinghouse, the CDS server tries to resolve as many components of the name as possible
5. If the name can be completely resolved, the CDS server returns the result of the name resolution to the CDS clerk
6. The CDS clerk caches this information in the name cache for future use

Name Resolution (Cont'd)

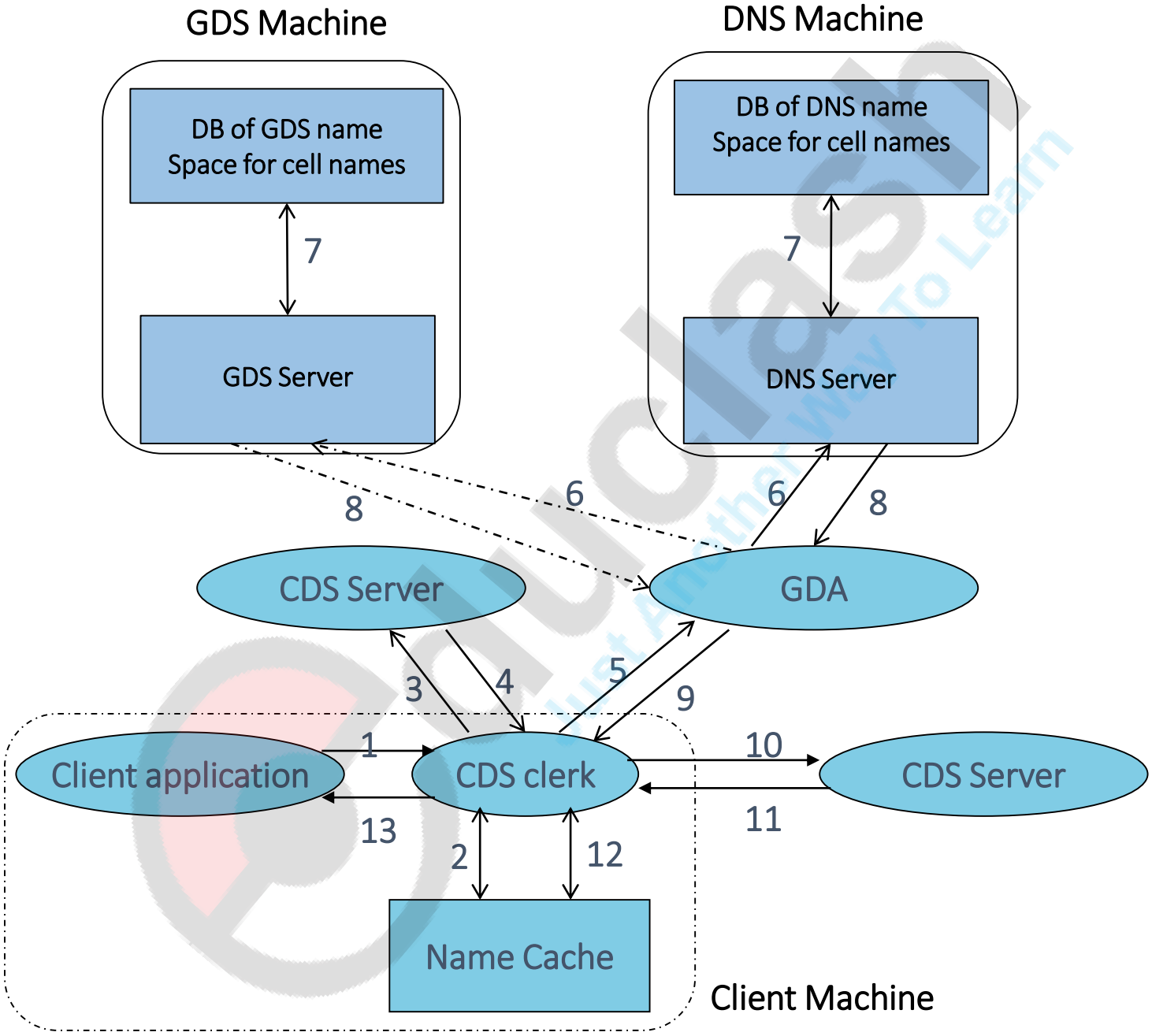
7. The CDS clerk returns a reply to client and resolution is complete

- If the result can only be partially resolved by the contacted CDS server in step 4; it returns partially resolved result and another CDS server location
- The process will iteratively continue until the name is resolved fully
- CDS server does only name resolution and not object accessing operation to be done separately by the client using RPC with server that manages the named object



Intercell Naming in DCE

- CDS clerks must have a way to locate CDS servers in another cell to access an object belonging to it
- Either GDS(Global Directory Service) or DNS(Domain name system) notation might be used for naming and these two name spaces map a cell name to CDS server in that cell
- Global Directory Agent (GDA) exists in any cell that needs to communicate with other cells, which may exist on the same machine as CDS server or on an independent machine
- CDS servers of a cell have information about the GDA location
- The fig. in the next slide shows the steps involved in how the intercell name resolution is done
- A cell may have more than one GDA for increased availability and reliability


GDS Machine
DNS Machine
DB of GDS name
Space for cell names
7
GDS Server
DB of DNS name
Space for cell names
7
DNS Server
8
6
6
8
CDS Server
GDA
3
4
5
9
1
Client application
CDS clerk
10
CDS Server
13
11
2
12
Name Cache
Client Machine

Intercell Naming in DCE (Cont'd)

1. A client Application sends a lookup RPC request to CDS clerk
2. CDS clerk checks it cache for the name; if found returns the reply to the client and operation is complete
3. If not, the CDS clerk does and RPC with a CDS server asking for the location of GDA
4. The CDS server returns the location of the GDA to the CDS clerk
5. The CDS clerk then does an RPC with the GDA, sending the cell name embedded in the name to be resolved
6. The GDA checks to see which notation has been used in the cell name
   - If X.500 notation, GDA does an RPC with GDS server; else if it is DNS notation, the GDA does an RPC with DNS server
7. The GDS server or DNS server lookup the cell name in their database

Intercell Naming in DCE (Cont'd)

8. It returns to the GDA, the address of CDS server in the named cell
9. The GDS forwards the information to CDS clerk
10. The CDS clerk now uses this information to send the name lookup request to the CDS server of the cell to which the named object belongs
    - CDS server resolves name using the directories in its clearinghouse
    - If it cannot resolve the name completely, then iterative approach is used
11. CDS server returns the result of the name resolution to the CDS clerk
12. The CDS clerk caches the information in its cache for future use
13. The CDS clerk finally returns a reply to the client and the name resolution operation is complete

User Interface to the DCE Directory Service

- Browsing interface:
  - For users with least privilege for the naming information
  - It allows the users to only view the content and structure of cell directories with only those directories to which he has read permission
- XDS application interface:
  - Users can create, modify and delete directory entries using XDS (X/Open Directory Server) Application program interface to write an application that makes direct calls to DCE Directory Service
- Administrative Interface:
  - For users with maximum privilege to naming information
  - It allows administrators to configure or reconfigure the naming information within the system including the distribution and replication of various directories in the clearinghouses of CDS servers

Assignment - 3

- What are desirable features of a DFS? How are they implemented in AFS?
- Describe architecture of NFS in detail. What are different protocols used by NFS & how are they implemented?
- Short note - Block Caching, Stable storage, File model, NFS vs AFS
- Enumerate various approaches for generating system oriented names
- Discuss the concept of partitioning namespace by clustering conditions and describe the different clustering condition methods with appropriate examples
- Short note on: Namespace, Name cache DCE Directory service