# LDAP Theory and Management

Brad Marshall

brad.marshall@member.sage-au.org.au

# Contents

- What is LDAP?
- Directory services
- LDAP Models
- Namespaces
- Schema
- Replication
- LDIF and DSML
- Search Filters
- LDAP Protocol

# Contents cont

- Directory Life Cycle
- Directory Management
- Namespace Design
- Topology Design
- Replication Design
- Management Models
- Data Maintanence
- Metadirectories
- Virtual directories

# History of LDAP

- Originally started as a front end to X.500
- Provides much of X.500's functionality at a lower implementation cost
- Removed redundant and rarely used operations
- Uses TCP rather than OSI stack
- University of Michigan wrote first LDAP implementation
- Most early LDAP implementations were based on it
- U.Mich eventually realized didn't need X.500 and wrote lightweight server
- Meant it was easier to deploy, and more people started using it

# What is LDAP?

- LDAP = Lightweight Directory Access Protocol
- Stanard protocol that can be used to access information over a network
- Based on X.500
- Directory Service
- Described in RFC1777 and RFC2251
- Stores attribute based data

# What is LDAP cont

- Data generally read more than written to
  - No transactions
  - No rollback
- Client-server model
- Based on entries
  - Collection of attributes
  - Globally unique distinguished name (DN) - like domain name
- Entries arranged in hierarchical tree structure

# Directory Services

- Directories are a specialized database
- Used in everyday lives - phone book, TV guides, etc
- Everyday directories fairly static
- Online directories are dynamic, flexible, secure and can be personalized
- Examples of online directories are finger, DNS, NIS and unix password file

# Why use LDAP

- Centrally manage users, groups and other data
- Don’t have to manage separate directories for each application - stops the “N + 1 directory problem”
- Distribute management of data to appropriate people
- Allow users to find data that they need
- Not locked into a particular server
- Ability to distribute servers to where they are needed
- Standards available for sharing data

# What LDAP can do

- Find things - local office phone book, Internet directories such as Four11, BigFoot etc
- Manage things - users and groups. Removes “N + 1” directory problem
- Lightweight database applications - sort bookmarks, small bits of data
- Security - store username and passwords, digital certificates, PKI etc

# What LDAP can’t do

- LDAP isn’t a relational database
  - No rollback for failed operations
- LDAP isn’t a filesystem
  - No locking or seeking
  - Can’t search blobs of data
- LDAP isn’t good for dynamic objects
  - Optimized for read, not write
  - No locking to ensure transactions occur in a certain order - corruption could occur
- No generic SQL-like reporting language
- Not useful without applications

# LDAP vs Databases

- Read-write ratio - LDAP is read optimized
- Extensibility - LDAP schemas are more easily changed
- Distribution - with LDAP data can be near where it is needed, and highly distributed
- Replication - with LDAP data can be stored in multiple locations
- Different performance - directories used for many different applications, and queries are usually simpler, but many more of them
- Data sharing - LDAP is designed for sharing data, databases designed for one application

# LDAP vs Databases cont

- Database objects have complex relationships
- Transaction model - LDAP transactions are simple - usually changing one entry, databases can modify much more
- Size of information - LDAP is better at storing small bits of information
- Type of information - LDAP stores information in attributes
- Naming model - LDAP is hierachical
- Schemas - database schemas are entirely user defined, directories have core schemas to help interoperability
- Standards are more important for directories - LDAP clients can talk to any LDAP server, but database client can only talk to the database it was designed for

# LDAP vs NIS

- Uses arbitrary ports
- No data encryption
- No access-control mechanism
- Uses a flat (non scalable) namespace
- Uses a single-key database (providing only basic searching abilities)
- All changes had to be made by the superuser on the domain master
- Does not provide directory services for non name service applications
- No ability for clustering (slapd replication and round robin DNS for LDAP)

# X.500

- Designed by International Telecom Union (ITU) and International Organization for Standards (ISO)
- First designed in 1988
- First general purpose directory system
- Designed to be extensible - rich search operations
- Open standard
- Relies on large suite of protocols
- Early implementations buggy and slow
- Based on OSI network protocols, not TCP/IP

# LDAP Directory Services

Two general types of directory servers:

- Stand alone LDAP servers
  - LDAP is only access mechanism
  - Data stores tuned for LDAP
- LDAP gateway servers
  - Translate betwen LDAP and some other protocol
  - Original use of LDAP - gateway to X.500

Doesn’t really matter where data is stored, as long as it is accessible via LDAP

# Common LDAP Applications

- White pages
- Authentication and authorization
- Personalization
- Roaming profiles
- Public Key Infrastructure (PKI)
- Message delivery

# LDAP Models

- Information Model - defines the types of data and basic units of info stored in directory
- Naming Model - defines how to organise and refer to the data
- Functional Model - defines operations in LDAP protocol
- Security Model - defines framework for protecting information

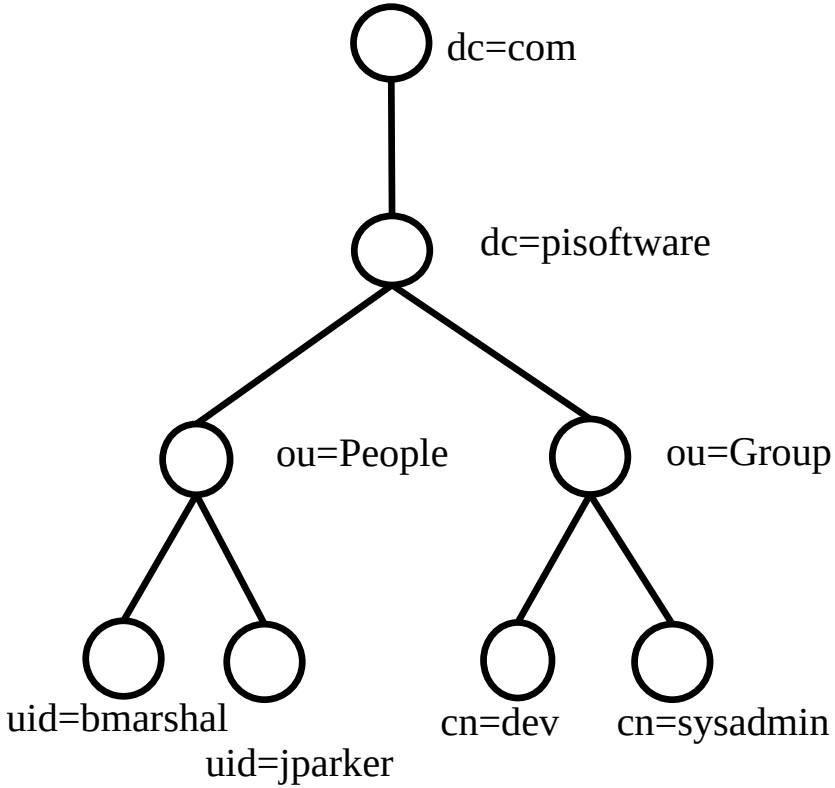
# Acronyms

| Acronym | Definition |
|---|---|
| LDAP | Lightweight Directory Access Protocol |
| DN | Distinguish Name |
| RDN | Relative Distinguished Name |
| DIT | Directory Information Tree |
| LDIF | LDAP Data Interchange Format |
| DSML | Directory Services Markup Language |
| OID | Object Identifier |

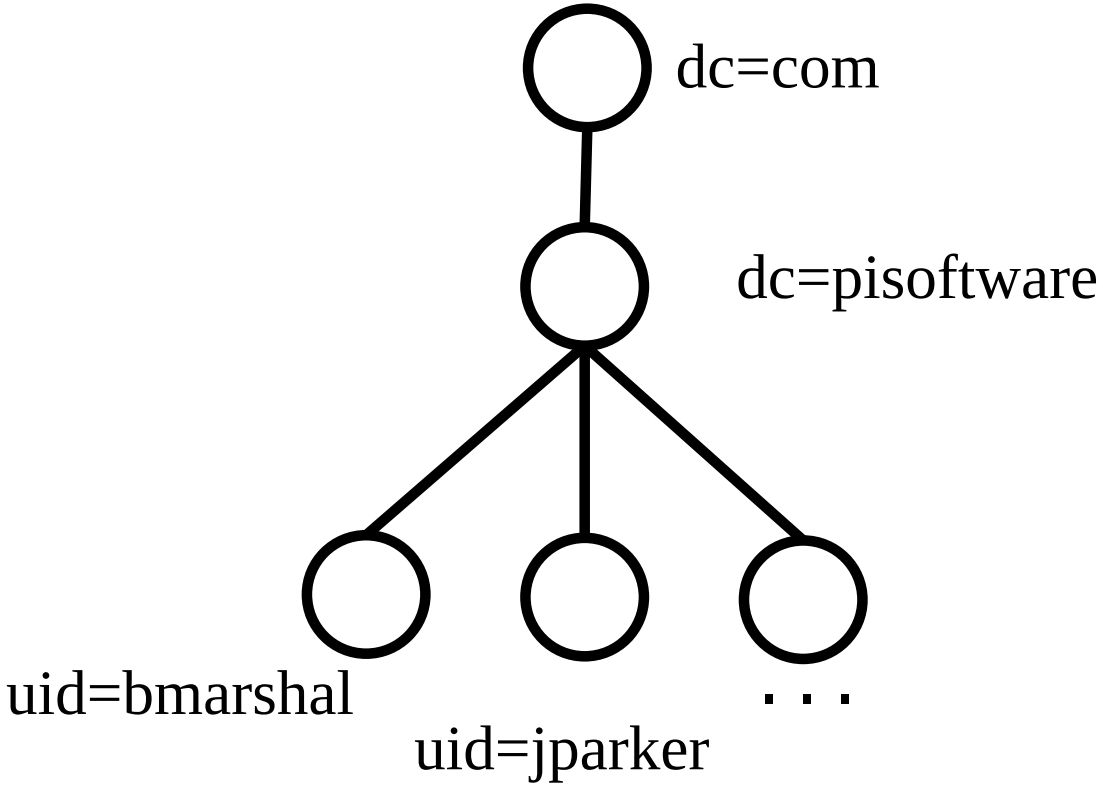
# Namespaces

- Hierarchical data structure
  - Entries are in a tree-like structure called Directory Information Tree (DIT)
- Consistent view of data - uniform namespace
  - Answers request
  - Refer to server with answer

# Namespaces - Hierarchal



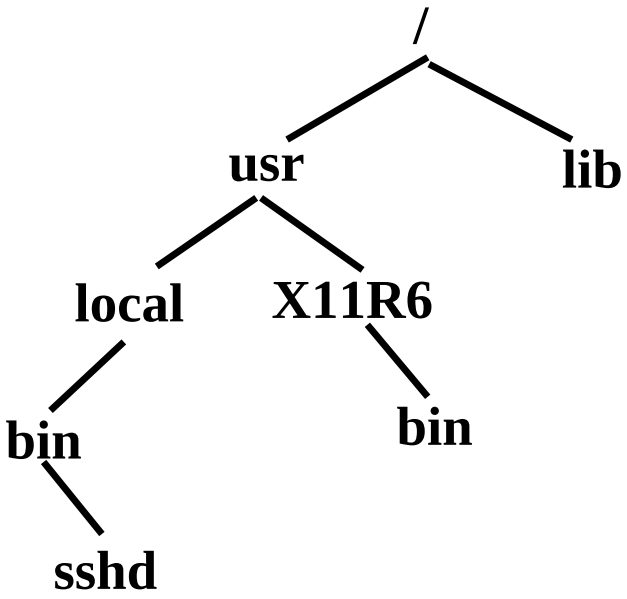
# Namespaces - Flat

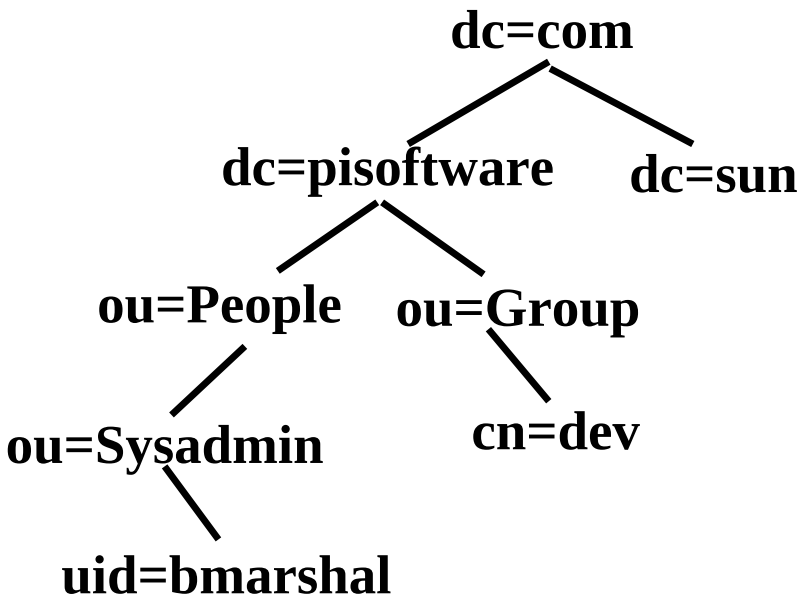


# Namespaces cont

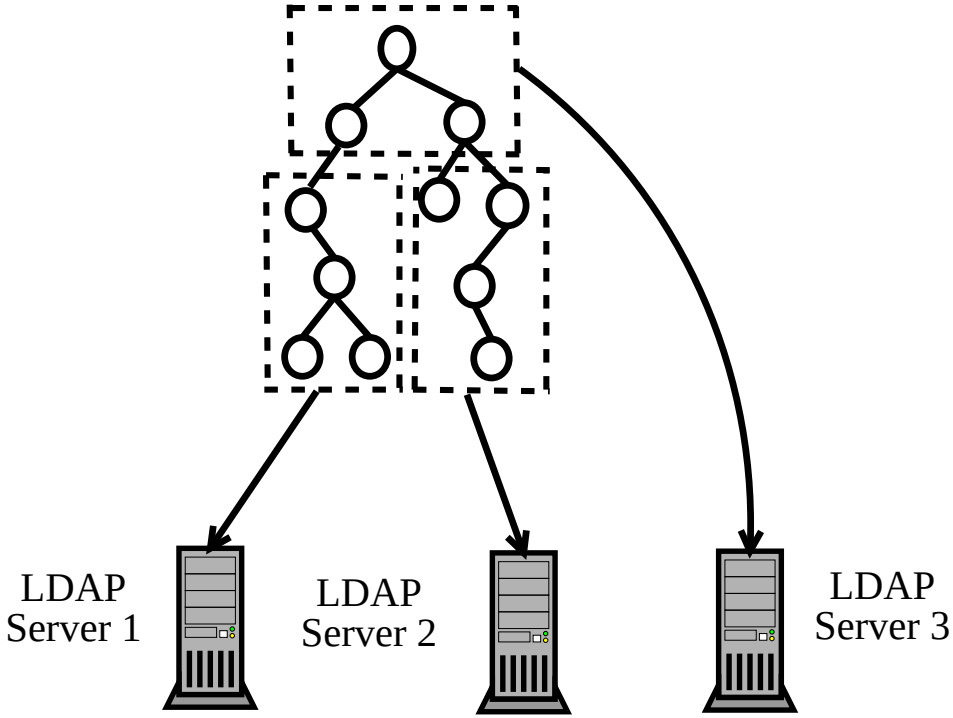
- Directory tree is similar to unix file system
  - No root entry in ldap
  - Each entry in ldap can both contain data and be a container
    - In unix, an entry is either a file or a directory - not both
  - LDAP distinguished names are read from bottom to top, unix file systems from top to bottom

# Namespaces cont



# Global View



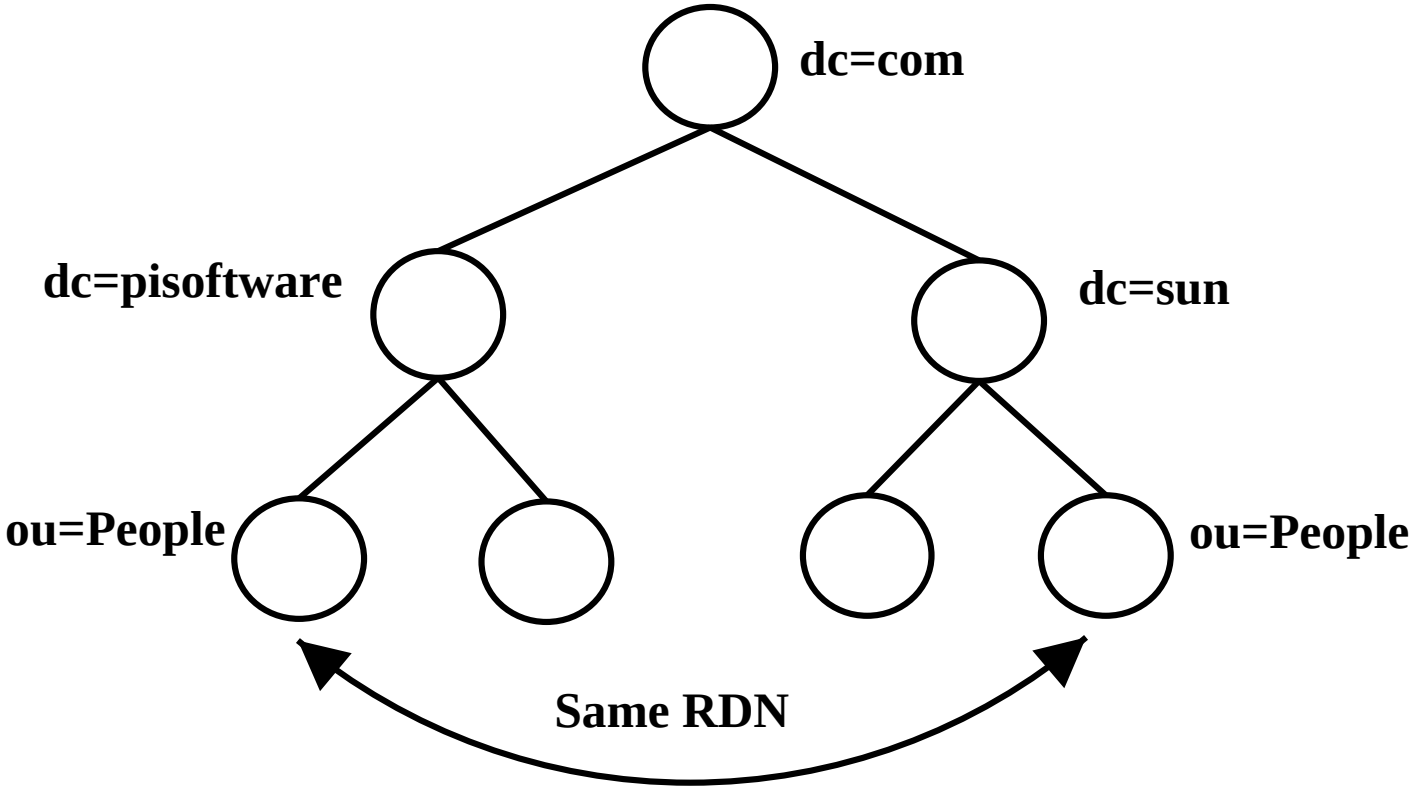

Note each server must contain a subtree

# Distinguished Names

- Defined in RFC2253 “Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names”
- Built up by starting at the bottom, and connecting each level together with commas
- Contain two parts
  - Left most part is called relative distinguished name
  - Remainder is base distinguished name
- Eg: uid=bmarshal,ou=People,dc=pisoftware,dc=com
  - RDN is uid=bmarshal
  - Base DN is ou=People,dc=pisoftware,dc=com

# Distinguished Names cont

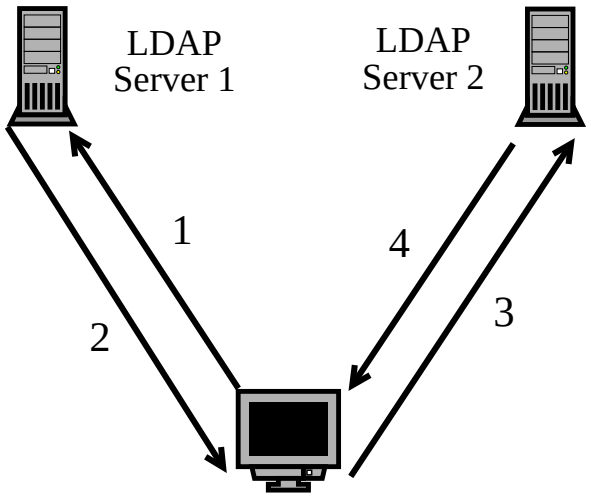
- In each base DN, each RDN is unique
  - This ensures no two entries have the same DN



# LDAP Entry

- Entries are composed of attributes
- Attributes consist of types with one or more values
- Type describes what the information is
- Value is the actual information in text format
- Attributes have a syntax which specifies what type of data - see Schema later on

# Referrals




1. Client requests information
2. Server 1 returns referral to server 2
3. Client resends request to server 2
4. Server 2 returns information to client

# Aliases

- Aliases are used to point one LDAP entry to another
- Allows you to have structures that aren’t hierarchal
- Similar in sense to using a symlink in unix
- Not all LDAP servers support aliases - big performance hit

# Aliases cont

- Created by:
  - Entry with object class of alias
  - Attribute named aliasedObjectName that points to DN of the alias
- Can use either referrals or putting a LDAP url in an entry

# Schema

- Set of rules that describes what kind of data is stored
- Helps maintain consistency and quality of data
- Reduces duplication of data
- Ensures applications have consistent interface to the data
- Object class attribute determines schema rules the entry must follow

# Schema cont

- Schema contains the following:
  - Required attributes
  - Allowed attributes
  - How to compare attributes
  - Limit what the attributes can store - ie, restrict to integer etc
  - Restrict what information is stored - ie, stops duplication etc

# Objectclass

- Used to group information
- Provides the following rules:
  - Required attributes
  - Allowed attributes
  - Easy way to retrieve groups of information
- Entries can have multiple object classes
  - Required and allowed attributes are the union of the attributes of each of the classes

# Objectclass inheritance

- Object classes can be derived from others
- Extends attributes of other objectclass
- No multiple inheritance
- Can’t override any of the rules
- Special class called top - all classes extend
  - Only required attribute is objectclass
  - Ensures all entries have a objectclass

# Attributes

Attributes have:

- Name - unique identifier, not case sensitive
- Object identifier (OID) - sequence of integers separated by dots
- Attribute syntax:
  - Data attributes can store - eg integer, string etc
  - How comparisons are made
- If multi valued or single valued

# Attribute Abbreviations

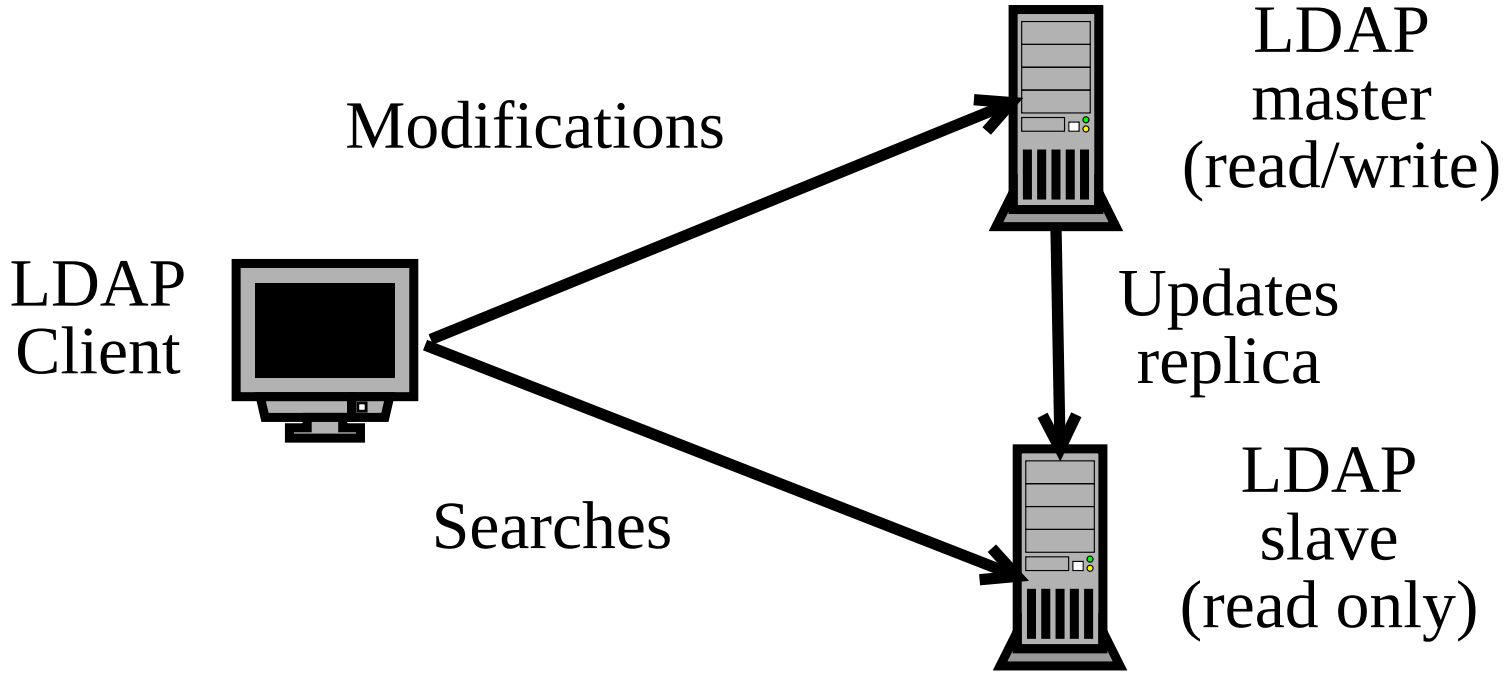
See RFC2256

| | |
|---|---|
| uid | User id |
| cn | Common Name |
| sn | Surname |
| l | Location |
| ou | Organizational Unit |
| o | Organization |
| dc | Domain Component |
| st | State |
| c | Country |

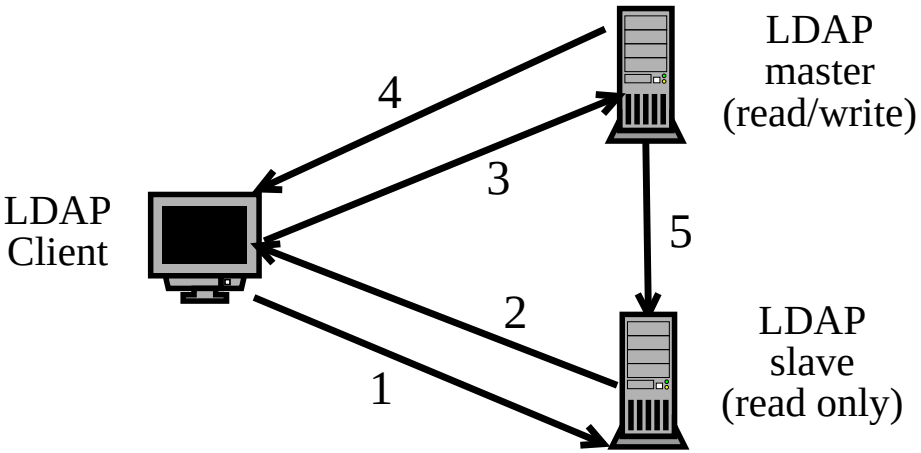
# Replication

- Replication is method used to keep copy of directory data on multiple servers
- Increases:
  - Reliability - if one copy of the directory is down
  - Availability - more likely to find an available server
  - Performance - can use a server closer to you
  - Speed - can take more queries as replicas are added
- Temporary inconsistencies are ok
- Having replicas close to clients is important - network going down is same as server going down
- Removes single point of failure

# Replication Options - Mods to Master

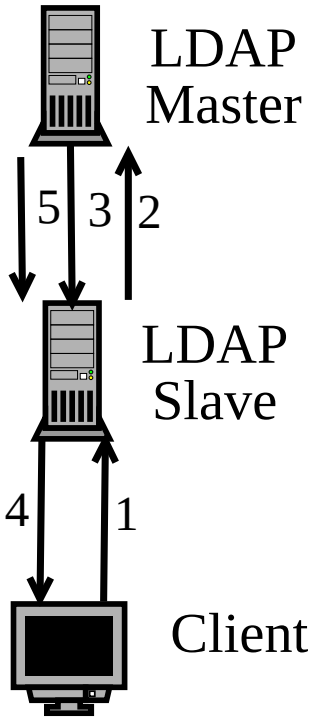


# Replication Options - Referrals




1. Client sends modification to replica
2. Replica returns referral of master to client
3. Client resubmits modification to master
4. Master returns results to client
5. Master updates replica with change

# Replication Options - Chaining




1. Client sends modification to replica
2. Replica forwards request to master
3. Master returns result to replica
4. Replica forwards result to client
5. Master updates replica

# Replication - Single Master

- Single master
  - Single server with read-write copy of data
  - Use read-only replicas to handle load of reading and searching
  - Uses referrals or chaining to deal with mods being sent to replicas
  - Obvious single point of failure
  - Simplest to implement

# Replication - Floating Master

- Floating master
  - Only one server with writeable copy of data at any time
  - If master becomes unavailable, new master is chosen from remaining servers
  - Actual alogorithm used depends on server - usually voting

# Replication - Cascading

- Cascading replication
  - Master replicates to hub which replays updates to futher servers
  - Helps balance very heavy loads - master can be used for all writes, and pass off all replication to hub
  - Can replicate out to local hubs in geographically disperse topologies - reduces cost
  - Increase performance - put all reads on “bottom” servers

# Replication - Fractional

- Fractional replication
  - Replicates subsets of attributes
  - Useful for synchronizing from intranet to extranet and removing data for security reasons
  - Can help reduce replication costs by removing unneeded data

# Replication - Multi-master

- Multi-master
  - More than one server with writable copy of data
  - Servers are responsible for ensuring data propogates
  - Need a conflict resolution policy - most current servers use a last writer wins policy
  - Can have automatic failover when one is unavailable
  - Most complex to implement

# LDIF

- LDAP Data Interchange Format
  - Represents LDAP entries in text
  - Human readable format
  - Allows easy modification of data
  - Useful for doing bulk changes
    - dump db, run a script over, import back
  - Can use templates for additions
  - Good for backups and transferring data to another system

# LDIF Cont

- If attribute contains non-ascii chars, begins with a space, colon or less than, the value is stored in base64 and type separator is ::
- Can mix plain text and encoded values
- Can specify URLs, type separator is :<
- Entries in LDIF are separated with a blank line
- Continued lines start with spaces
- Utilities to convert from database to ldif and back
  - slapcat: ldbm database to ldif
  - slapadd: ldif to ldbm database
- Few non-LDAP related servers know how to deal with it

# LDIF Example

```
dn: uid=bmarshal,ou=People,
          dc=pisoftware,dc=com
uid: bmarshal
cn: Brad Marshall
objectclass: account
objectclass: posixAccount
objectclass: top
loginshell: /bin/bash
uidnumber: 500
gidnumber: 120
homedirectory: /mnt/home/bmarshal
gecos: Brad Marshall,,,,
userpassword: {crypt}lDIOnUp17x2jc
```

# DSML

- XML-based file format
- Can store both schema and data
- Can use as generic import/export format for directory data
- Many existing servers can deal with XML
- More complex than LDIF
- Not all LDAP servers support it

# DSML Example

```
<dsml:dsml xmlns:dsml="http://www.dsml.org/DSML">
  <dsml:directory-entries>
    <dsml:entry dn="uid=bmarshal,dc=example,dc=com"
      <dsml:objectclass>
        <dsml:oc-value>top</dsml:oc-value>
        <dsml:oc-value>account</dsml:oc-value>
      </dsml:objectclass>
      <dsml:attr name="cn">
        <dsml:value>Brad Marshall</dsml:value>
      </dsml:attr>
      <dsml:attr name=loginShell>
        <dsml:value>/bin/bash</dsml:value>
      </dsml:attr>
    </dsml:entry>
  </dsml:directory-entries>
</dsml:dsml>
```

# Search Filters

- Consists of:
  - Base and scope - what part to search
  - Filter - what criteria attributes must fulfill to be returned
  - Attributes - what attributes to return
- Prefix notation
- Standards
  - RFC 1960: LDAP String Representation of Search Filters
  - RFC 2254: LDAPv3 Search Filters

# Search Filters Operators

- Basic filter types
  - Presence
  - Equality
  - Substring
  - Greater-than or equal
  - Less-than or equal
  - Approximate
  - Extensible
- Joining filters
  - & and
  - | or
  - ! not

# Search Filter Details

- Presence
  - Simple checks for presence of attribute
  - Can return everything by using (objectClass=*)
- Equality
  - Checks for entries containing given attribute and value
  - Most commonly used search
  - Fastest and easiest to index
  - Case sensitivity depends on attribute syntax

# Search Filter Details cont

- Substring matching
  - Returns entries with values matching given substrings
  - Useful for finding information, eg in white pages etc
  - Not wildcard or regular expression matching
  - Restricted to matching start, middle or end of strings
  - Slower than presence or equality
  - Middle substring matches slower than start or end

# Search Filter Details cont

- Ordered matching (greater-than, less-than)
  - Order determined by attributes syntax and matching rules
- Approximate filter
  - Algorithm used determined by server
  - Common to have soundex
- Extensible
  - Allows applying an explicit matching rule to a search filter
  - Not very common, but powerful

# Search Filters Examples

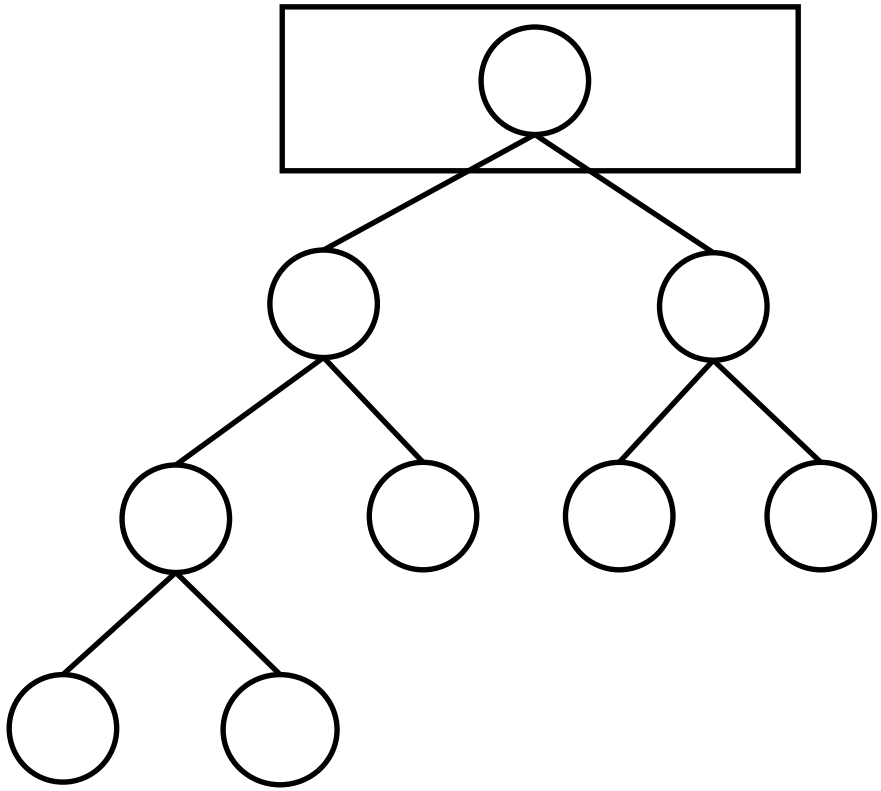
- (objectclass=posixAccount)
- (cn=Mickey M*)
- (|(uid=fred)(uid=bill))
- (&(|(uid=jack)(uid=jill))(objectclass=posixAccount))

# Search Scope

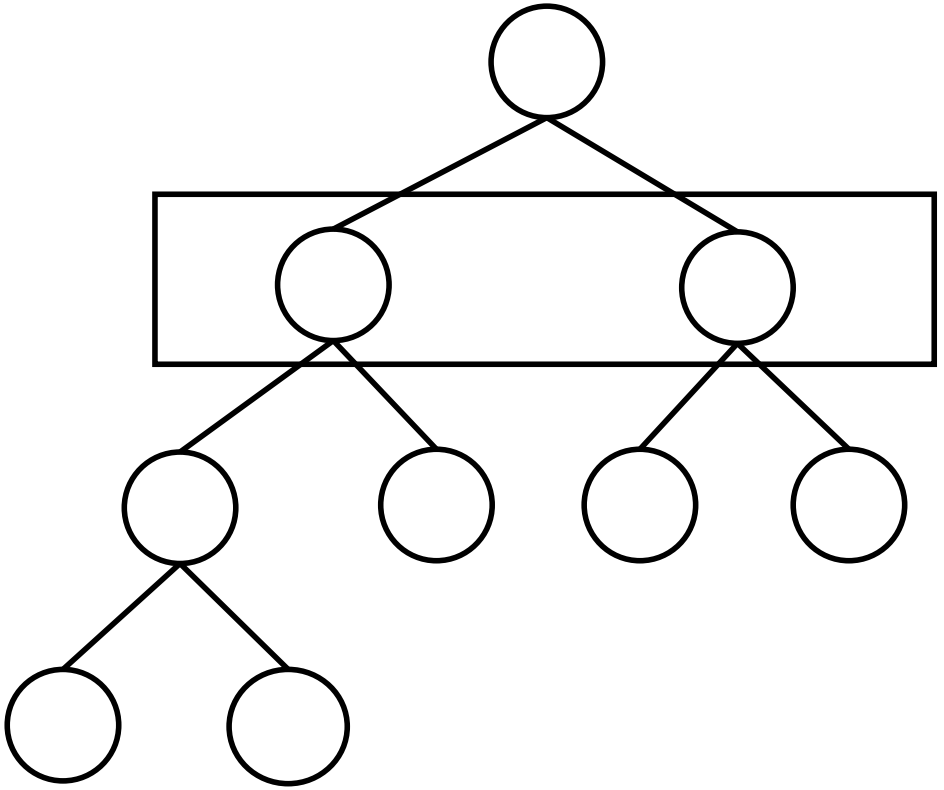
3 types of scope:

| | |
|---|---|
| base | limits to just the base object |
| onelevel | limits to just the immediate children |
| sub | search the entire subtree from base down |

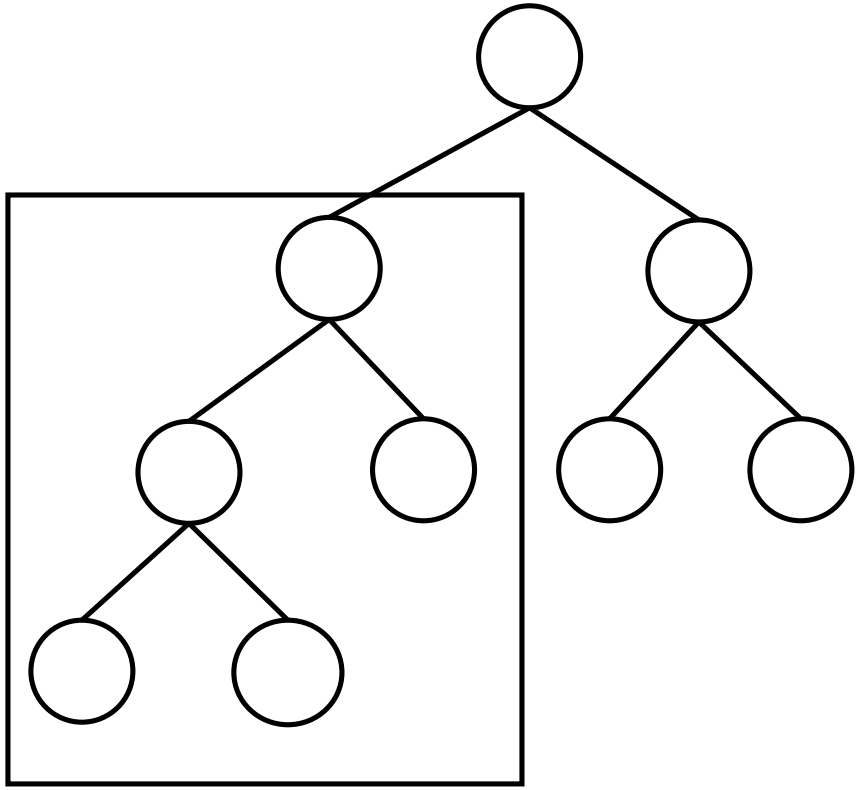
# Base Scope



# One Level Scope



# Subtree Scope



# Optimizing Searches

- Index commonly accessed attributes and appropriate filters
- Reduce opening and closing connections
- Only request attributes you need
- Limit the search scope if possible
- Minimize number of searches - combine searches into one, if possible
- Minimize updates - writes are slow
- Use replicas / high availability techniques on the server

# Operational Attributes

- Attributes used by server that usually aren't displayed
- Vary by directory vendor
- Usually time stamps, server access controls, admin information, last updated dates etc

# Extending Schema

To extend schemas, the process is as follows:

- Obtain Object Identifier (OID)
- Choose name prefix
- Create local schema file
- Define custom attribute types (if necessary)
- Define custom object classes

(Note these examples are from OpenLDAP, others should be similar)

# Obtaining OID

- OIDs found in protocols described by ASN.1, eg SNMP
- OIDs are hierachical
- Do not just arbitrarily pick an OID
- OIDs are obtained from IANA, under Private Enterprise arc at no cost
- OID from IANA will be something like 1.3.6.1.4.1.X
- For experiments, OID 1.1 is available (regarded as dead namespace)

# Choose name prefix

- Schema elements all have names
- Names should be descriptive and not likely to clash
- Convention is to prefix names with few letters to localize the changes
- The smaller the organization, the longer the prefix should be
- Examples are auFirm (Aust company) or comPisoftware (elements for pisoftware.com)

# Attribute Types and Object Classes

- attributetype directive used to define new attribute types
- Uses Attribute Type Description, as per RFC2252
- objectclasses directive used to define new object class
- Uses Object Class Description, as per RFC2252

# Attribute Type Description

```
AttributeTypeDescription = "(" whsp
      numericoid whsp   ; AttributeType identifier
    [ "NAME" qdescrs ]  ; name used in
                        ; AttributeType
    [ "DESC" qdstring ] ; description
    [ "OBSOLETE" whsp ]
    [ "SUP" woid ]      ; derived from this
                        ; other AttributeType
    [ "EQUALITY" woid   ; Matching Rule name
    [ "ORDERING" woid   ; Matching Rule name
    [ "SUBSTR" woid ]   ; Matching Rule name
```

# Attribute Type Description cont

```
    [ "SYNTAX" whsp noidlen whsp ]
    [ "SINGLE-VALUE" whsp ]
                ; default multi-valued
    [ "COLLECTIVE" whsp ]
                ; default not collective
    [ "NO-USER-MODIFICATION" whsp ]
                ; default user modifiable
    [ "USAGE" whsp AttributeUsage ]
                ; default userApplications
    whsp ")"

AttributeUsage =
            "userApplications"     /
            "directoryOperation"   /
            "distributedOperation" /
            "dSAOperation"
```

# Attribute Type Example

```
attributetype ( 2.5.4.41 NAME 'name'
    DESC 'names associated with the object'
    EQUALITY caseIgnoreMatch
    SUBSTR caseIgnoreSubstringsMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{32768} )
attributetype ( 2.5.4.4 NAME ( 'sn' 'surname' )
    DESC 'surnames associated with the object'
    SUP name )
```

# Object Class Description

```
ObjectClassDescription = "(" whsp
      numericoid whsp  ; ObjectClass identifier
    [ "NAME" qdescrs ]
    [ "DESC" qdstring ]
    [ "OBSOLETE" whsp ]
    [ "SUP" oids ]      ; Superior ObjectClasses
    [ ( "ABSTRACT" / "STRUCTURAL" / "AUXILIARY" )
                whsp ]   ; default structural
    [ "MUST" oids ]     ; AttributeTypes
    [ "MAY" oids ]      ; AttributeTypes
    whsp ")"
```

# Object Class Example

```
objectclass ( 1.3.6.1.1.1.2.0
    NAME 'posixAccount'
    SUP top
    AUXILIARY
    DESC 'Abstraction of an account
        with POSIX attributes'
    MUST ( cn $ uid $ uidNumber $
           gidNumber $ homeDirectory )
    MAY ( userPassword $ loginShell $
          gecos $ description ) )
```

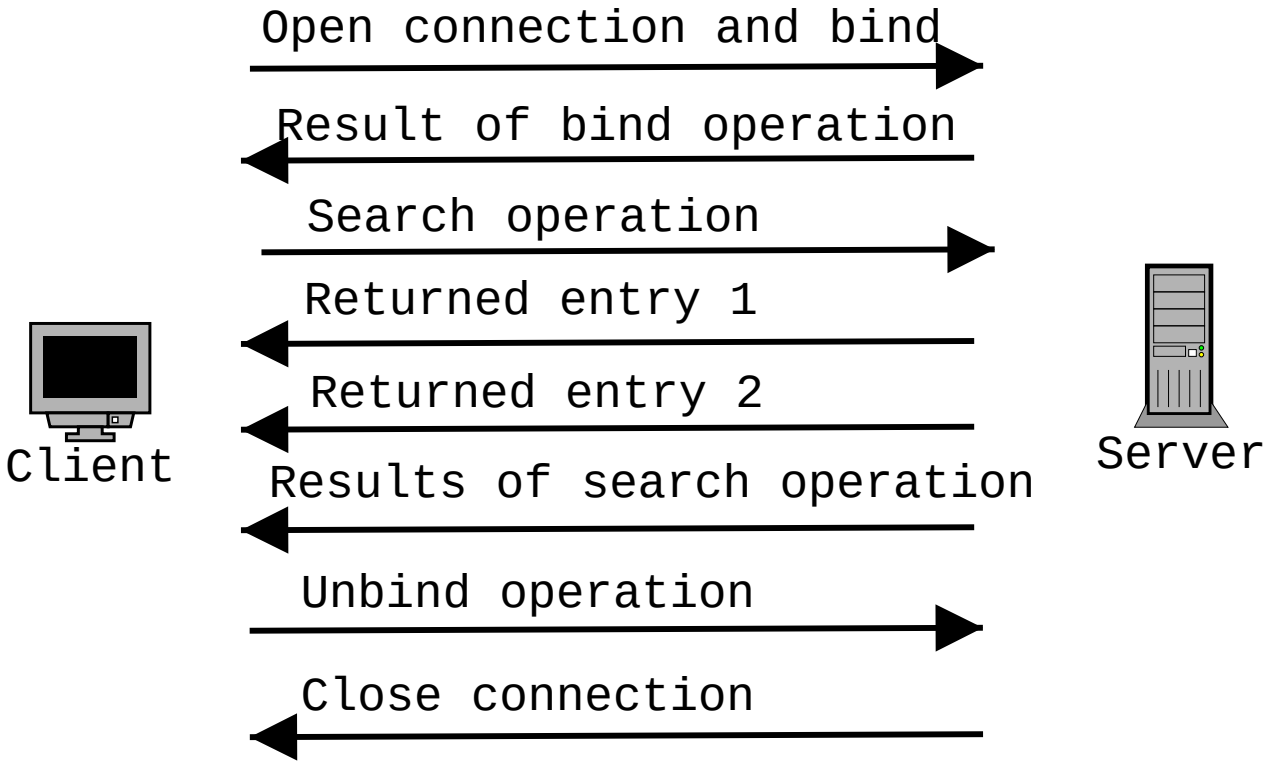
# LDAP Protocol

- Uses client server model
- Message oriented protocol - client sends messages to server and gets replies
- Can issue multiple requests at once - each response has message id to identify
- 9 basic protocol operations - interrogation, update and authentication
- LDAPv3 provides extended operations and controls
- Uses simplified version of Basic Encoding Rules (BER) - not plain text

# LDAP Protocol Operations

- Interrogation operations
  - search
  - compare
- Update operations
  - add
  - delete
  - modify
  - rename
- Authentication and control operations
  - bind
  - unbind
  - abandon

# Typical LDAP conversation



# Bind Operations

- How client authenticates to directory
- Opens TCP connection, gives distinguished name and credentials
- Server checks credentials are correct, returns result to client
- If credentials fail, returns an anonymous bind
- Authentication lasts while connection is open, or until client reauthenticates
- Credentials can be password, digital certificate or other

# Authentication Levels

There are 3 basic levels of authentication:

- Anonymous - no username and password
  - Enabled by default, disable using “disallow bind_anon” in slapd.conf
- Unauthenticated - only username (sometimes called reference bind)
  - Disabled by default, enable using “allow bind_anon_cred” in slapd.conf
- Authenticated - username and correct password
  - Enabled by default, disable using “disallow bind_anon_simple” in slapd.conf

# LDAP Security Model

- Authentication
  - How to verify identities
- Password policies
  - Criteria passwords must satisfy to be valid - ie, length, age etc
- Encryption
  - Ensuring data is private
- Access control
  - Who can access what data
- Auditing
  - Determine if security model is being followed - ie, logs

# Security Model - Authentication

- Anonymous access
- Simple password
- Proxy authentication
- Simple password over encrypted connection
- Certificate based authentication
- SASL based authentication

# Password policies

- Minimum length
- Maximum age
- Syntax checking
- History keeping
- Expiration policy
- Account lockout policy

# Access Control

Set permissions on:

- Entire directory
- Subtree
- Specific entries
- Specific attributes
- Any entry that matches a search filter

# LDAP URLs

Definition taken from RFC2255

```
<ldapurl> ::= "ldap://" [ <hostport> ]
          "/" <dn> [ "?" <attributes>
          [ "?" <scope> "?" <filter> ] ]
<hostport> ::= <hostname>
           [ ":" <portnumber> ]
<dn> ::= a string as defined in RFC 1485
<attributes> ::= NULL | <attributelist>
<attributelist> ::= <attributetype>
                  | <attributetype>
                  [ "," <attributelist> ]
<attributetype> ::= a string as defined
                        in RFC 1777
<scope> ::= "base" | "one" | "sub"
<filter> ::= a string as defined in RFC 1558
```

# LDAP URLs

**DN** Distinguished name

**Attribute list** List of attributes you want returned

**Scope**

| | |
|---|---|
| base | base object search |
| one | one level search |
| sub | subtree search |

**Filter** Standard LDAP search filter

# LDAP URL examples

- ldap://foo.bar.com/dc=bar,dc=com
- ldap://argle.bargle.com/dc=bar,
  dc=com??sub?uid=barney
- ldap://ldap.bedrock.com/dc=bar,
  dc=com?cn?sub?uid=barney

# LDAP Related Standards Bodies

- Consortiums and Standard Bodies
  - OASIS
    - Directory Services Markup Language
  - Distributed Management Task Force (DMTF)
    - Common Informational Model (CIM)
  - Network Applications Consortium (NAC)
    - Users Group
  - Open Group Directory Interoperability Forum (DIF)
    - LDAP2000 Interoperability
  - Internet Engineering Task Force (IETF)
    - LDAP Standards

# IETF Working Groups

- IETF Working Groups
  - LDAPExt
    - Access Controls
    - APIs
  - LDUP
    - Replication
  - LDAPbis
    - LDAPv3 Protocol revisions

# LDAPv3

- Internationalization - using UTF-8
- Referrals and Continuations
- Security
  - Authentication via SASL
  - Integrity and Confidentiality Protection via TLS or SSL
- Extensibility
- Feature and schema discovery
  - LDAPv3 servers have a directory entry called root DSE (Directory Server Entry)
  - Contains: protocol supported, schemas, other useful info

# Directory Uses

3 main uses of directories

- Network directory
  - Provides user account management
- Enterprise directory
  - Global repository for shared information
  - Details about people, organisations and devices
- Business directory
  - Details about trading partners, customers, suppliers etc

# Directory Life Cycle

3 main parts to life cycle:

- Design
  - Understand directory requirements and gather data
- Deployment
  - Actually try out directory
- Maintenance
  - Update data, keep service running, and improve

# Design Cycle

- Directory needs
  - Understand what applications require
- Data
  - Find existing data sources
  - Determine relationship to directory
- Schema
  - Develop schemas for applications requirements
- Namespace
  - Organise data layout

# Design Cycle cont

- Topology
  - Determine where servers are located
  - Determine how data is divided
- Replication
  - Ensure data is available where it is needed
- Security
  - Essential to meeting security and privacy requirements of users

# Deployment Cycle

- Choose directory software
  - Evaluate and choose software
- Piloting
  - Inital testing to check design
- Performance and scaling testing
  - Testing to see if system can handle the task
- Tuning
  - Check process as well as technical
- User feedback
  - Expose users to test system
- Going into production
  - Move from pilot to production

# Maintenance Cycle

- Data maintenance
  - Keeping data up to date
- Data backups
  - Ensure backups don’t interfer with operations
  - Replication can help here
- Schema additions
  - Extend schemas when adding new applications
- Monitoring
  - Integrate with existing system if possible
- Expansion
  - Try to plan for expansion - growth may surprise you
- Changing requirements
  - Don’t get caught out

# Directory Management

Directory management consists of:

- Namespace and directory tree design
- Determine root naming context
- Integration with existing data
- Synchronization
- Maintaining data
- Choosing mangement model

# Basic Design Steps

- Define applications for directory
- Find data sources and relationships
- Schema design
- Namespace design and topology
- Replication
- Extraction and population
- Testing and deployment

# Schema Design

- Be aware of existing schemas and use them if possible
- Use schemas provided with applications
- Define and document any new schemas required
- Be prepared for schema evolution as things change

# Namespace Design

- Designing a namespace is Hard
- Requires indepth knowledge of what the directory will be used for
- Hard to reorganise once data is put in - requires downtime, etc
- Needs to support applications that want to use it - be aware of existing standards

# Namespace Design cont

- Try not to break out into different departments - what happens when person moves?
- Avoid designs that will have to change - eg, don’t base on org chart
- Don’t go overboard - too much hierachy can get confusing
- Probably won’t get it right first - will take some iterations

# Namespace Design cont

Good namespace design helps with:

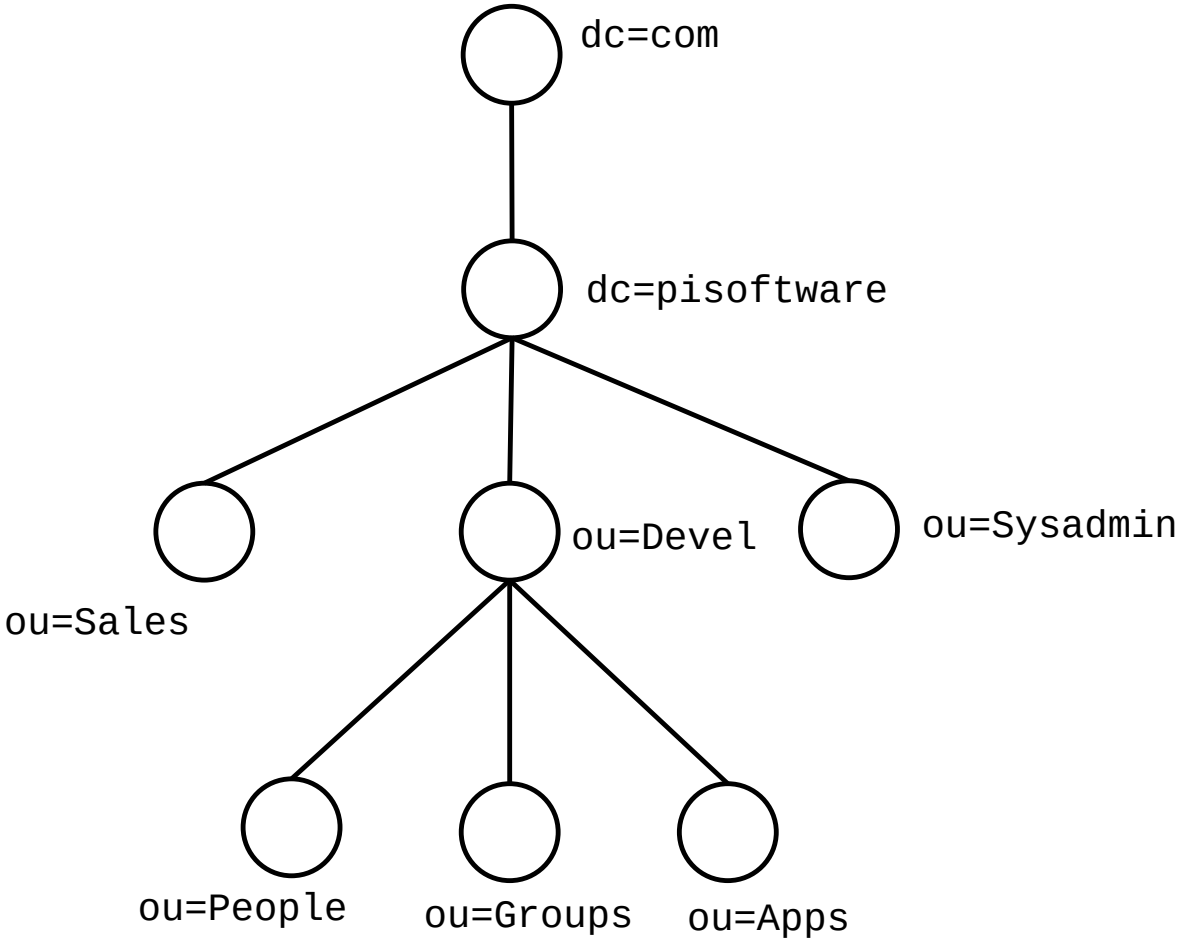
- Delegation of authority
- Replication of data
- Security and access control
- Scalability
- Physical placement of servers

# Tree Design

3 most commonly used tree designs for a single company

- Organizational
- Flat
- Geographical

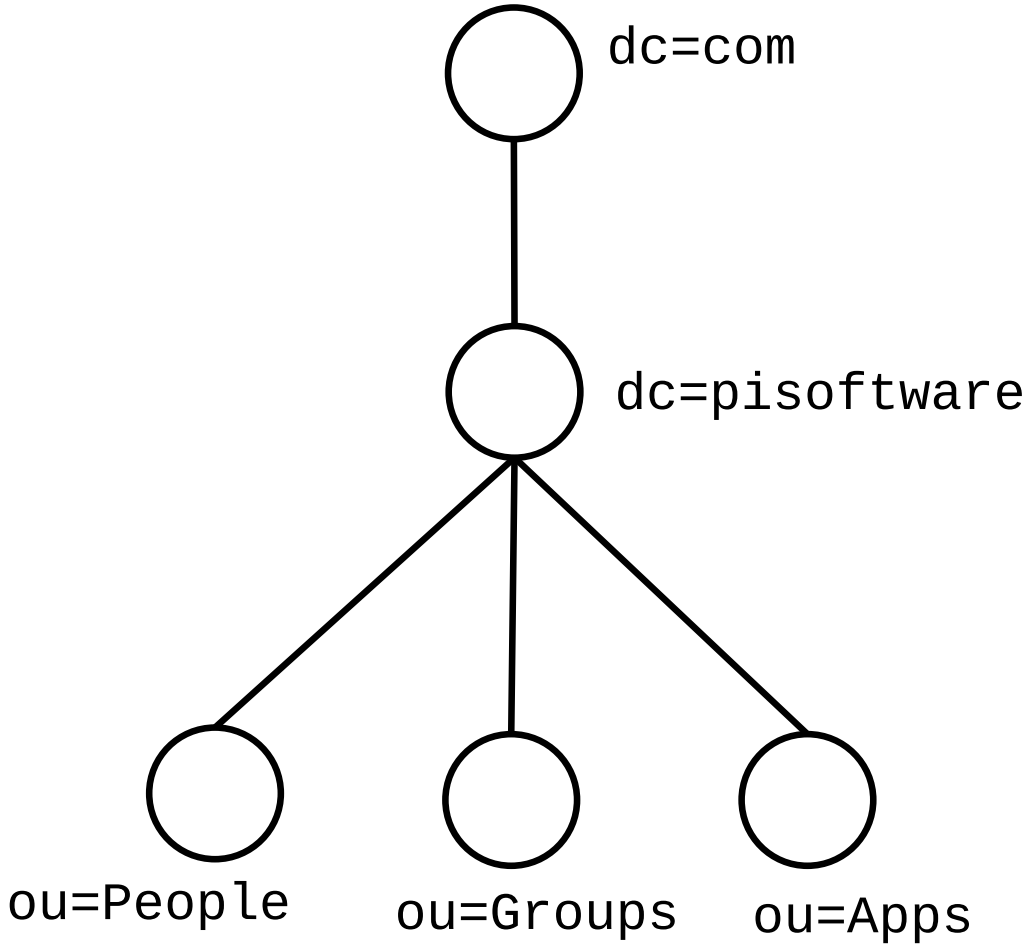
# Organizational Based Tree



# Organizational Based Details

- Mirrors organizational structure
- Good for using delegated administration management model
- People changing jobs mean a change to distinguished name
- Groups refer to distinguished names - no referential integrity checking

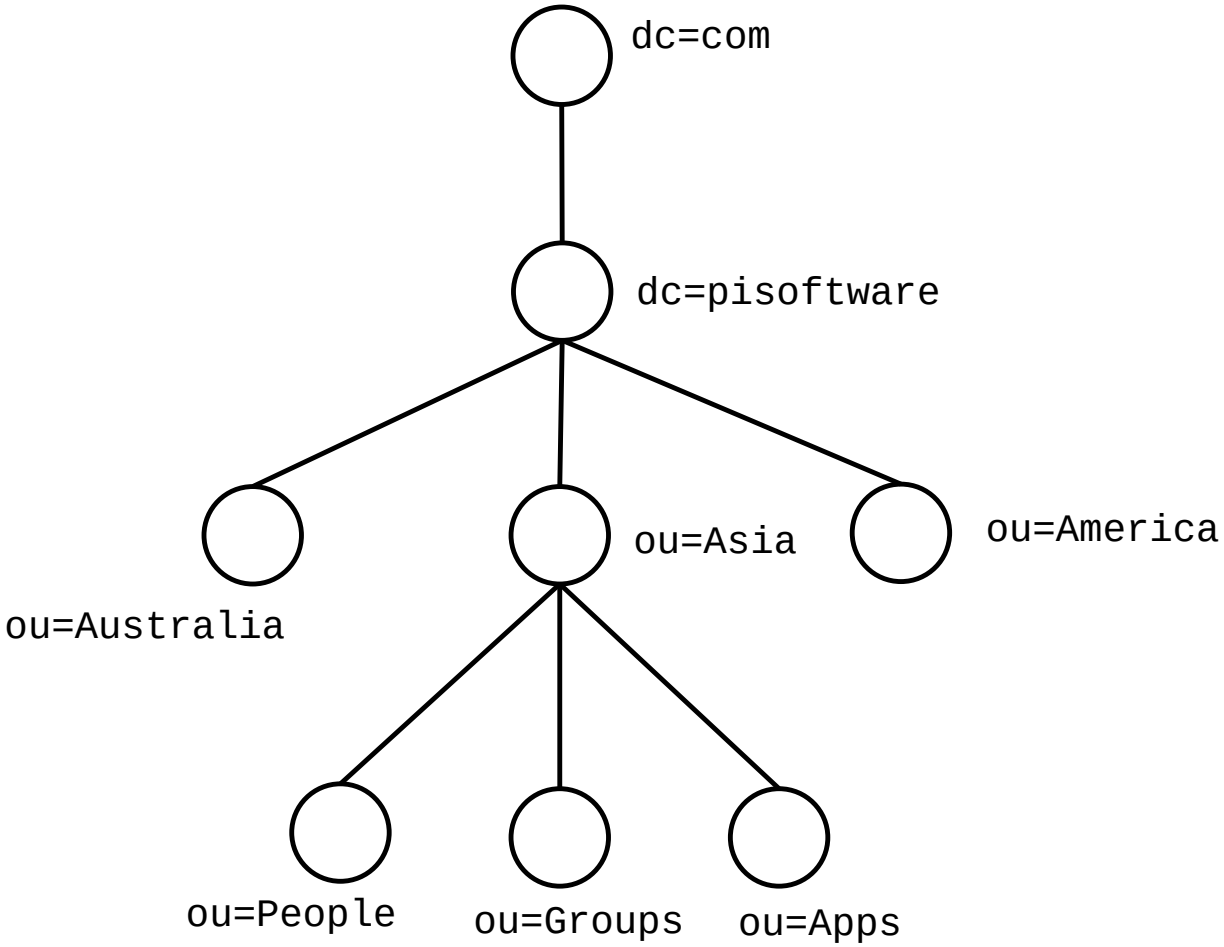
# Flat Tree



# Flat Details

- Simplest layout - all people and groups are in one level down from root
- Entries only added or removed on hiring and firing - not internal job changes
- Most directory servers can’t split single branch across multiple servers
- Harder to delegate admin control

# Geographical Tree



# Geographical Details

- Branch for each geographical location of the company
- Each geographical location has a master, also replicates rest of tree
- Rarer for people to move between geographical locations
- Geographical dispersement of company might not match political

# Choosing Root Naming Context

- Root naming context is servers top level entry
- Some servers can have multiple root naming contexts
- Two main choices:
  - Traditional
  - Domain component
- Active Directory doesn't give a choice for root naming context, most others do

# Traditional Style

- Traditionally has been ‘o=company,c=country’
  - Company is free-form string of company name
  - Country is ISO 2 letter country code
- Inherited from X.500
- Can cause conflicts between similarly named companies

# Domain Component Style

- Use DNS name to generate base DN
- See RFC2377 for more details - "Naming Plan for Internet Directory-Enabled Applications"
- example.com gives dc=example,dc=com
- Already globally unique
- Already registered
- Can trace back to who owns it easily
- Some CAs only allow traditional X.500 attributes, which doesn’t include dc

# Topology Design

- Directory tree can be spread over multiple physical servers
- Design is important to:
  - Increase performance
  - Increase availability
  - Better manage directory
  - Hold more entries than a single server could
- Users see one large directory
- Closely related to replication design

# Topology Design

- Each server is delegated a partition of the directory
- Similar to DNS
- One server is the root and is common ancestor to all subpartitions
- Partitions must be a complete subtree, and not exclude any entries
- Servers can hold more than one partition, eg read-only copies

# Topology Design - References

- Directory glued together using knowledge references
- Immediate superior knowledge references
  - Points up the tree
  - Hooks back up to ancestor
- Subordinate knowledge references
  - Points down the tree to other partitions
  - Hook to hang other partitions off

# Topology Design - Why Partition

- Too many entries for a single server
- Increase performance and scalability
- Applications mostly read and modify entries in local section of tree
- Replication traffic to a single partition is too large
- If partitioning allows traffic to remain local, won't have to cross slow WAN links
- If directory use will rapidly expand

# Topology Design - Why Not Partition

- All entries can fit on a single server
- Applications modify entries across whole tree
- Not much replication traffic
- Organisation is centrally located, no slow WAN links
- Directory use is static and won't exceed capabilities of single server

# Topology Design - Considerations

- Consider directory enabled applications and performance requirements
- Understand server software limitations and capabilities
- Consider physical layout of network and location of applications
- Ensure namespace design fits within constraints - ie, flat namespace isn’t good for partitioning

# Replication Design

- Increases reliability and performance
- Allows locating data close to users and applications that need them
- Can manage data where it is being used
- Gives redundancy - applications can fail over to replicas
- Can distribute load across multiple servers, increasing performance
- Closely related to topology design

# Replication Design Strategy

- High availability
  - Multiple directories at one site
  - Use multi-master to ensure write-failover
- Local availability
  - Replicas at local offices
  - Useful if network links are unreliable or intermittant
  - Helps reduce load on master server
  - Masters can be at:
    - Central office
    - Local server is master for local part of directory tree

# Replication Design Strategy cont

- Disaster recovery
  - Two masters at geograhically disperse locations
  - Provides failover and data protection
- Load balancing
  - Spread user connections over servers
  - Dedicate servers to read-only activities
  - Dedicate servers to particular activities - ie, server for mail

# Replication Issues to Consider

- Quality of links between remote sites
- Amount of bandwidth between sites
  - Is replication traffic too much for link
- Physical location of users
  - How many users
  - What type of work they do
- Number of applications
- Type and proportion of operations done
- Size of directory

# Management Models

3 main management models

- Centralized administration
- Distributed administration
- User self-administration

# Centralized Administration

- Good for smaller organisations
- All changes are made in the one location or group of people
- Minimal admin infrastruture
- Low potential for duplication of data
- Can cause bottlenecks in changes being made - long delays
- Often seen as most secure - not always the case
  - Possibly too much trust in central group
  - Within sub-departments, possibility that changes may not happen quickly enough - eg, employee is fired and notification takes time to make it to central accounts department

# Distributed Administration

- Centralised department co-ordinates sub-groups, who do actual admin work
- Reduces admin bottlenecks
- Very scalable
- Similar to how DNS admin is done - root nameservers handle top level, then delegated out
- Namespace design is important - hard to distribute across flat namespace

# User Self-Administrated

- User creates and manages own account - eg Internet shopping
- Web forms often used to let end users admin own accounts
- Common to seperate end user admined accounts from other part of directory
- Lower level of trust of data done this way
- Usually use other form of data / contact - eg frequent flyers database, using email address

# Data Maintenance

- Maintenance is most important part
- Usually involved with maintance the longest, often least though about aspect
- Configuration is comparitively simple, managing contents is hard
- The best designed directory is useless with out of date or incomplete data

# Managing Directory Contents

- People can have multiple accounts, or a single account (or group) can have multiple people associated
- Need to manage association between accounts and people, as well as managing the people
- This is essential to be able to terminate all accounts associated with a terminated employee
- Important as ldap doesn’t ensure referential integrity - can be left with dangling references

# Managing Data Flow

Managing data integration between directories has a number of approaches:

- Replication
- Data export/import
- Scripting
- Metadirectories
- Virtual Directories

# Replication

- Exactly duplicating data between directories
- This creates a copy of data for distribution, performance, scalability and redundancy
- No LDAP standard for replication, so hard to replicate between different vendors
- IETF is developing LDUP, an independant replication protocol

# Data export/import

- Export data out from one directory, then import into next
- Can manipulate data after exporting
- Servers must support either of 2 main standards
  - LDIF
  - DSML
- Updates only happen periodically, data is out of sync between these times

# Scripting

- Writing scripts to manipulate data between servers
- Allows choosing subset of data to transfer
- Easy to change / manipulate data
- Common languages are Perl, Java and C

# Metadirectories

- Consolidates subsets of data from various locations
- Presents it as LDAP, XML, HTTP etc
- Consolidates and aggregates details about objects
- Can provide multiple views of data
- Helps manage relationships between directories
- Manages data flow between directories
- Can reflect information back to original or new data sources
- Not just a consolidated directory of all information

# Metadirectories cont

- Consists of:
  - Metadirectory namespace
  - Join engine
  - Connectors
- Comes in two different forms
  - Add on to existing directory
  - Stand alone directory

# Metadirectories Namespace

- Also called meta view
- Unified view of underlying directories / data sources
- Usually stored in a LDAP directory
- Uses join engine to synchronize underlying sources

# Metadirectories Join Engine

- Managed data flow between external sources and metadirectory views
- Defines authoritive source for each bit of data
- Controls which data sources can be changed via meta directory
- Manipulates and unifies connector views to provide meta view
- Links items in connector views with items in meta view
- Can construct attributes from other values

# Metadirectories Connectors

- Provides data from external sources to join engine
- Connects between directory and data sources
- Can provide name mapping to translate from original data source to LDAP
- Defines which attributes to use or discard
- Provides a connector view
- Two types:
  - Direct connector: talks directly to source, usually either LDAP or SQL
  - Indirect connector: converts from source data into LDAP

# Virtual Directories

- LDAP frontend to databases / directories
- Provides schema adaptors
- Failover and load balancing
- Namespace conversions
- Value manipulations
  - Add/modify/delete attributes
  - Remove or construct attributes based on criteria

# Data Flow Analysis

- Schema mapping
  - Mapping schemas between directories
- Determining authoritative source
  - Find which directory has most up to date for each part of data
- Data tranformation
  - Not all systems store data in the same format, eg may use extra lookup tables
  - Change data representation to be the same format in each directory
- Namespace translation
  - Sometimes can’t translate directly - can get conflicts
  - Sometimes can just change root naming context

# Data Sources

- Existing directory services
- Operating systems
- Existing databases
- Flat files or spreadsheets
- Applications
- Sysadmins
- End users

# Common Data Problems

Things to be careful of:

- Too much information - can’t find what is needed
- Not enough information
- Poor quality information
- Out of date information

# Tips for Success

- Carefully pick and stick to scope of project
- Start small, show incremental successes
- Pick low hanging fruit, and stay visible
- Deploy in parallel with existing systems
- Design for end result - doesn’t have to get there straight away
- Identify who’s responsible early
- Use info with greatest value first
- Allow time for political and process issues
- Be careful what you promise and manage expectations

# Important RFCs

| RFC | Title |
|---|---|
| RFC2247 | Using Domains in LDAP DNs |
| RFC2251 | LDAPv3 Protocol |
| RFC2252 | LDAPv3 Attribute Types |
| RFC2253 | LDAPv3 Disinguished Name |
| RFC2254 | LDAPv3 Search Filters |
| RFC2255 | LDAPv3 URL Format |
| RFC2256 | A Summary of the X.500(96) User Schema |
| RFC2307 | LDAP Network Information Services Schema |
| RFC2377 | LDAP Naming Plan |

# Important RFCs cont

| RFC | Title |
|---|---|
| RFC2798 | LDAP inetOrgPerson schema |
| RFC2829 | LDAPv3: Authentication Methods |
| RFC2830 | LDAPv3: Start TLS |
| RFC2849 | LDIFv1 |
| RFC2891 | LDAPv3: Server Side Sorting of Search Results |
| RFC3112 | LDAP Authentication Password Schema |
| RFC3377 | LDAP(v3): Technical Specification |

# New RFCs

| RFC | Title |
|---|---|
| RFC 4510 | LDAP: Technical Specification Road Map |
| RFC 4511 | LDAP: The Protocol |
| RFC 4512 | LDAP: Directory Information Models |
| RFC 4513 | LDAP: Authentication Methods and Security Mechanis |
| RFC 4514 | LDAP: String Representation of Distinguished Names |
| RFC 4515 | LDAP: String Representation of Search Filters |
| RFC 4516 | LDAP: Uniform Resource Locator |
| RFC 4517 | LDAP: Syntaxes and Matching Rules |

# New Extension RFCs

| RFC | Title |
|---|---|
| RFC 4525 | LDAP Modify-Increment Extension |
| RFC 4527 | LDAP Read Entry Controls |
| RFC 4528 | LDAP Assertion Control |
| RFC 4529 | Requesting Attributes by Object Class in LDAP |
| RFC 4530 | LDAP entryUUID Operational Attribute |
| RFC 4532 | LDAP "Who am I?" Operation |

# References

- Understanding and Deploying LDAP Directory Services
  Timothy A. Howes, Mark C. Smith and Gordon S. Good
  Macmillan Network Architecture and Development Series

- Implementing LDAP
  Mark Wilcox
  Wrox Press Ltd

- LDAP Programming, Management and Integration
  Clayton Donley
  Manning Publications

# References cont


- Summary Report from Catalyst 99 Directory Namespace and Best Practices Workshop, The Burton Group and The Network Applications Consortium
- OpenLDAP Administrators Guide, http://www.openldap.org/doc/
- LDAP Linux HOWTO, Luiz Ernesto Pinheiro Malere, Linux Documentation Project
- LDAP Implementation HOWTO, Roel van Meer, Linux Documentation Project
- SunONE Directory Server 5.2 Deployment Guide
- SunONE Meta Directory Server 5.1 Deployment Guide
- The Slapd and Slurpd Administrator's Guide, University of Michigan, Release 3.3