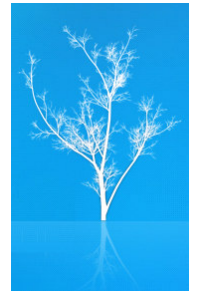


## LDAP Stored Procedures and Triggers arrive in ApacheDS

- Originally presented at *ApacheCon US 2006* in Austin
- Latest presentation materials are at <http://people.apache.org/~ersiner>
- Presented by *Ersin Er*, [ersiner@apache.org](mailto:ersiner@apache.org)



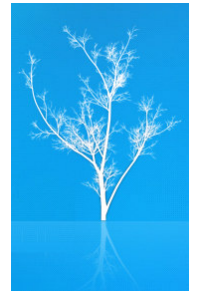
# Agenda



- Stored Procedures
  - Why do we need them in LDAP?
  - Representing Stored Procedures
  - Executing Stored Procedures
  - Demos
- Triggers
  - Why do we need them in LDAP?
  - Model of LDAP Triggers
  - Integration with LDAP Stored Procedures
  - Demos (including a complete case study)



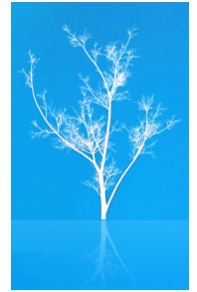
## Stored Procedures for LDAP (Why?)



- Bulk processing
- Controlled by user
- Extending server's capability *easily*
- LDAP Extended Operations?



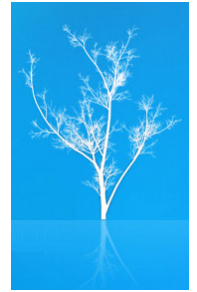
## Model of LDAP Stored Procedures



- Implementation technology
- Storage place
- Storage format
- Storage method
- Calling
  - Parameters
  - Return value
- Security

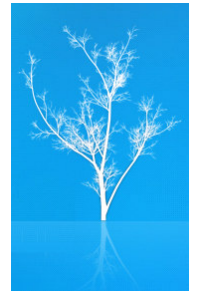


## What's an LDAP stored procedure?



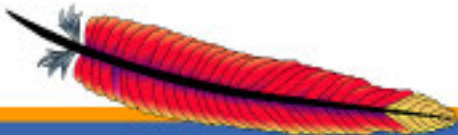
- A piece of code
- Implemented in any technology
- Stored in the Directory Information Tree
- Represented with schema elements
- Manipulated by standard LDAP operations (add, delete)



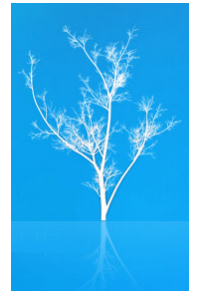


## Stored Procedures in ApacheDS

- “Java” *scheme* realization of the generic model
- A “Java” LDAP stored procedure is
  - A public static method of a Java class
  - Represented by two attributes and an object class
  - Stored with its class (as expected) in compiled form (byte-code)



## DEMO 1

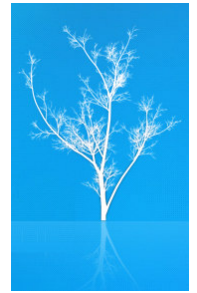


- Let's load the following SP on the DIT!

```
public class HelloWorld
{
    public static void helloWorld()
    {
        System.out.println( "Hello world!" );
    }
}
```

- Note: ApacheDS expects SPs under “ou=Stored Procedures,ou=system” by default





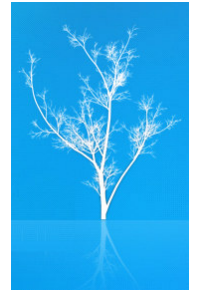
## So we want to *call* it?

- Call from where?
  - Client side
  - Server side
- No standard SP Call operation
- For calling *any* LDAP stored procedure from client side
  - Use *Stored Procedure Execution (Extended) Operation*





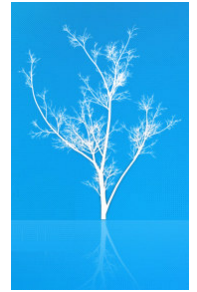
# Stored Procedure Execution (Extended) Operation



- Name of the stored procedure
- Where to find the stored procedure (optional)
  - A base search context (DistinguishedName)
  - Search scope: base, one, whole (Optional)
- SP impl. language (scheme) (optional)
- Parameters (optional)
  - type information (optional)
  - value



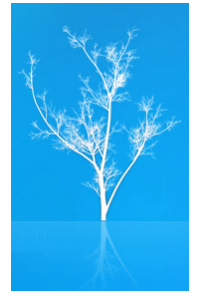
## DEMO 2



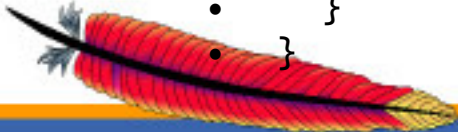
- Let's call the stored procedure
- SP name: "HelloWorld.helloWorld"
- Search context not given (it's under the default container)
- SP language scheme "Java" is not given, as it's default for ApacheDS
- No parameters (yet!)



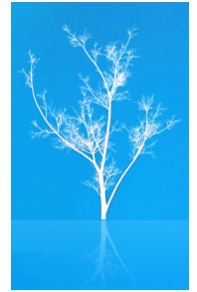
# DEMO 3



- Let's load the following stored procedure
- ```
public class Greeter
```
- ```
{
```
- ```
  public static String sayHello( String who, Integer times )
```
- ```
  {
```
- ```
    StringBuffer buffer = new StringBuffer();
```
- ```
    for ( int i = 0; i < times.intValue(); i++ )
```
- ```
    {
```
- ```
      buffer.append( "Hello " );
```
- ```
    }
```
- ```
    buffer.append( who );
```
- ```
    buffer.append( '!' );
```
- ```
    return buffer.toString();
```
- ```
  }
```
- ```
}
```



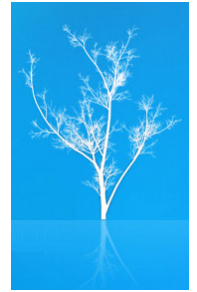
## DEMO 3 (continued)



- Let's call the stored procedure
- Parameters
  - who:String: "ApacheCon"
  - times:Integer: 3
- And the return value
  - *An Object!*

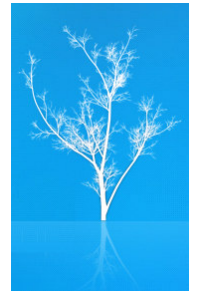


# “Java” SP execution progress (A reflection story)



- Find the SP entry
  - Use the SP name (what) and search context (where)
- Extract class name from SP name
- Load the class
- Extract method name from SP name
- Find the method in the class
  - Use method name and check parameters for assignment compatibility
- Call the method supplying parameters
- Return back the result Object



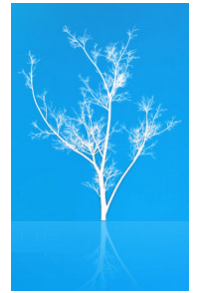


## A special SP parameter

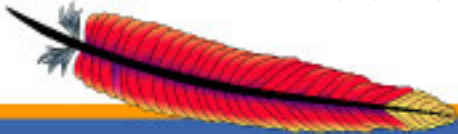
- type: “ldapContext”
- value: A distinguished name (as a String object)
- ApacheDS supplies a JNDI context at the specified DN with the user’s credentials
- Why do we need it?



## DEMO 4

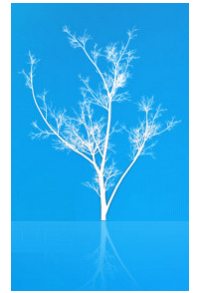


- Let's do a real world example
- With *delete* operation a single entry can be deleted at once
- It's a common requirement to delete a subtree at once
- There is an *delete operation control* for this
- But let's write our own *DelSubtree*, load and call it





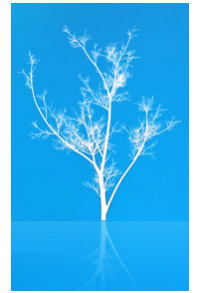
## Security Issues



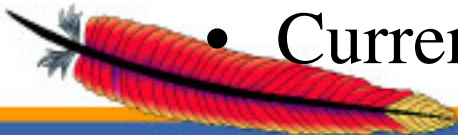
- Directory operations on stored procedures
  - Who can do what on stored procedures
- Permissions used during execution
  - Executor's verses owner's
- Authorization for executing stored procedures
- Stored procedures' capabilities within the server



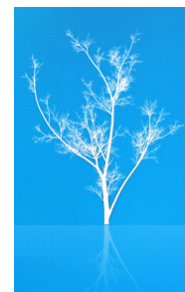
## Security Issues and ApacheDS



- Stored procedures
  - are standard user objects
  - any operation on them is possible
  - and subject to access control
- Stored procedures are executed with executor's permissions
- Currently, who is authorized to read an SP is also authorized to execute it
- Currently, execution is not sandboxed



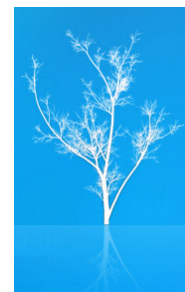
## Stored Procedures - Briefly



- LDAP stored procedures allow users to effectively define their own *extended operations* without requiring any server software extensions



## Triggers for LDAP (Why?)

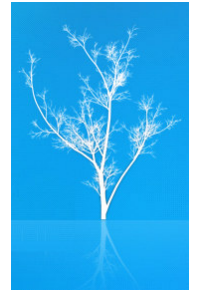


- Tracking DN references (referential integrity)
- Custom action needs upon some operations on some entries (logging, firing an external process)
- Existing solutions lacks some capabilities or are hard to use (e.g. requires server side plugins)
- It's better to keep it simple and powerful ;-)





## A Trigger

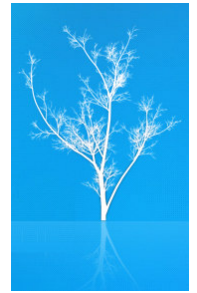


**<Trigger Specification> :**

**<Action Time>**

**<Trigger Event>**

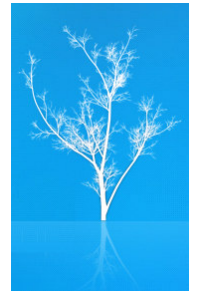
**<Triggered Action>**



# An LDAP Trigger

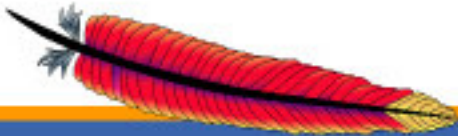
- **Action Time:** AFTER
- **Trigger Event:** Change inducing LDAP operations
- **Triggered Action:** LDAP Stored Procedures!
- Which entries is a trigger defined *on*?
  - A specific entry
  - *Trigger Execution Domains*
- All these information are stored as regular schema objects (so can be browsed, replicated, etc.)



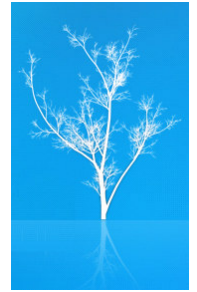


## Trigger Specification Examples

- **AFTER Delete**  
`CALL "BackupUtilities.backupDeletedEntry"  
($ldapContext(""), $name, $deletedEntry)`
- **AFTER Add**  
`CALL "Logger.logAddOperation"  
($entry, $attributes, $operationPrincipal)`



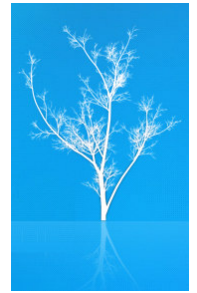
# Stored Procedures - Triggers Integration



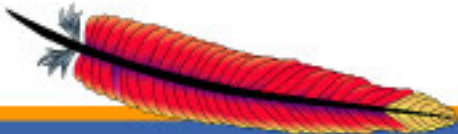
- SPs can be supplied parameters like:
  - operation specific standard request parameters (\$entry for Add, \$name for Delete, ...)
  - operation specific useful parameters (\$deletedEntry for Delete, ...)
  - generic parameters (\$ldapContext, \$operationPrincipal, ...)
- All available parameters have predefined corresponding Java types
- SP call options are supported as specified in the SP Execution Operation



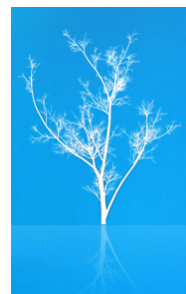
# DEMO 1



- Let's backup an entry when it's deleted
- Write a Java stored procedure and load it
- Put an entryTriggerSpecification attribute in an entry
  - **AFTER Delete**
  - **CALL "BackupUtilities.backupDeletedEntry"**
  - **(\$ldapContext(""), \$name, \$deletedEntry)**



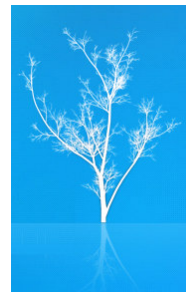




## Was it impressive?

- Not very much!
- The trigger was effective only on a single entry
- And even our trigger specification has been deleted!
- Well, the trigger specification might be effective in the new location of the entry too
  - What if the entry is deleted from the backup context?
  - *Has anyone said infinite loop?*

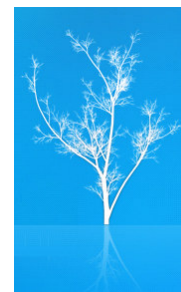




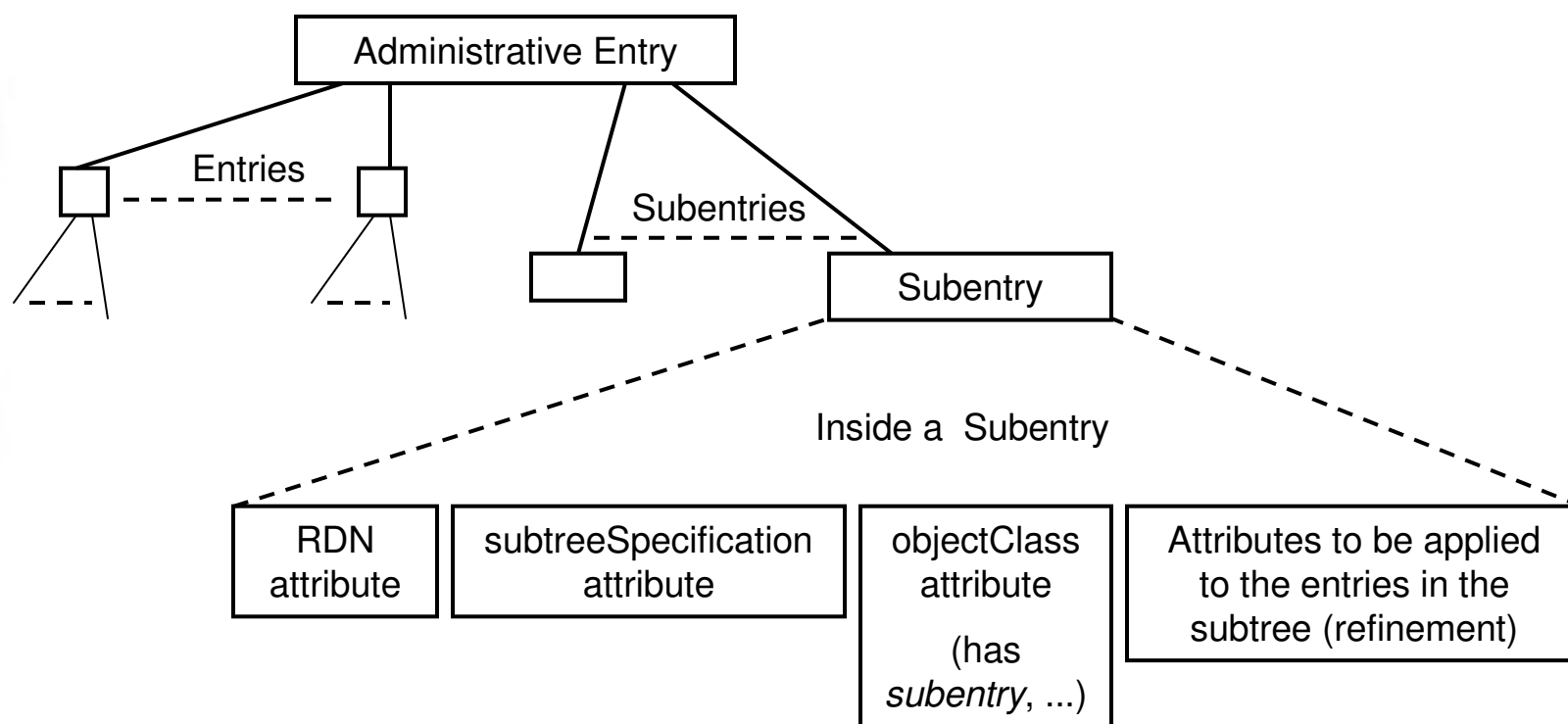
## Trigger Execution Domains (DACD)

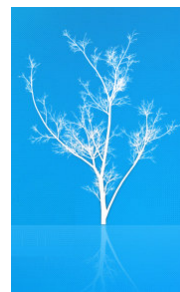
- X.500 Subentries and subtreeSpecification
  - A Subentry holds a subtreeSpecification attribute
  - subtreeSpecification allows specifying a *subtree of entries with chop specifications and refinements*
  - Other attributes in the Subentry are *applied* to the selection of entries
  - A building block of X.500 Administrative Model
  - RFC 3672 - Subentries in the Lightweight Directory Access Protocol
- Trigger Execution Domains
  - Instead of entryTriggerSpecification,
  - use *prescriptiveTriggerSpecification* in *triggerExecutionSubentry*
  - to define triggers on a set of entries



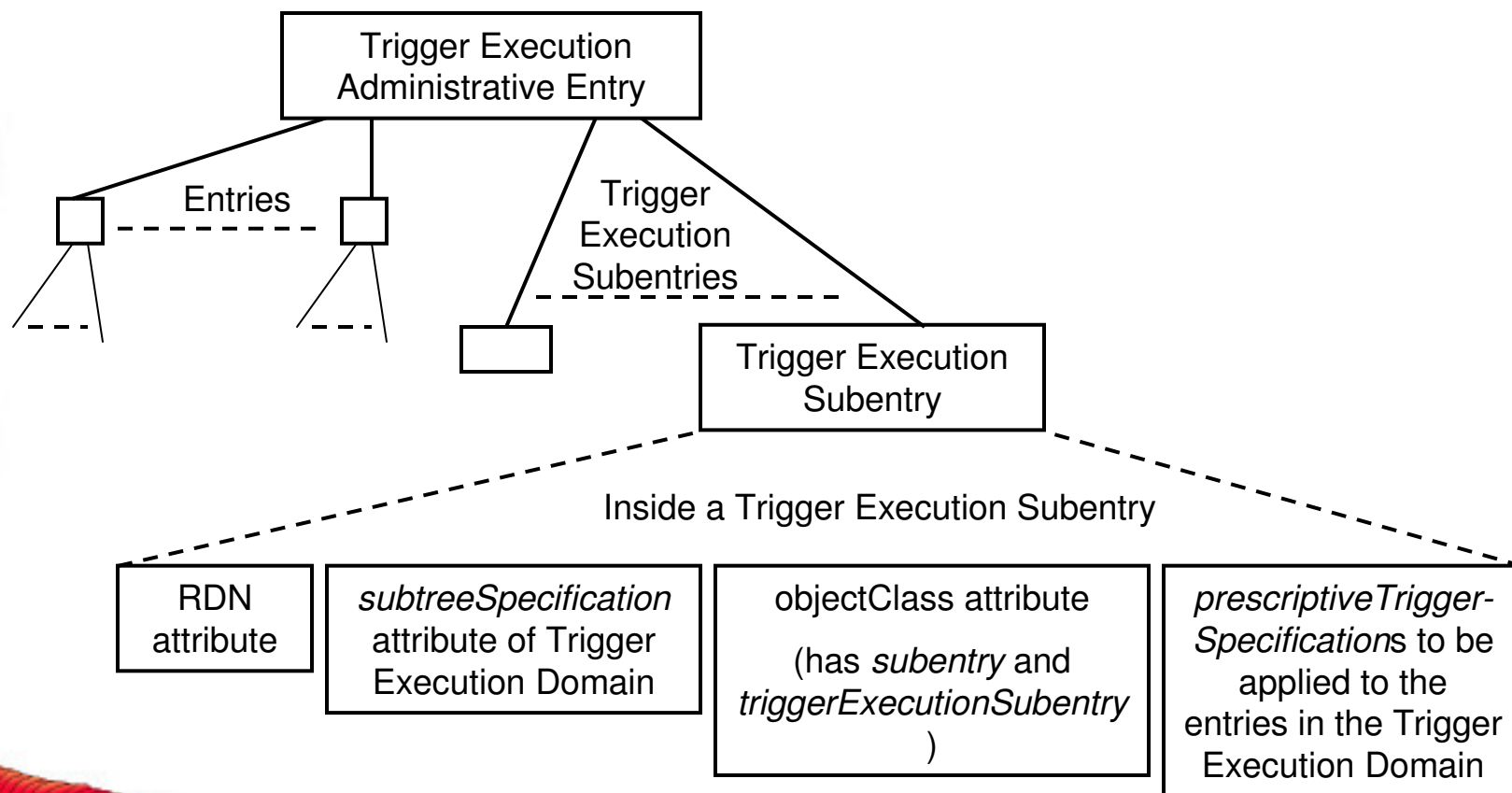


## X.500 Administrative Model

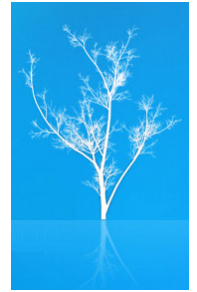




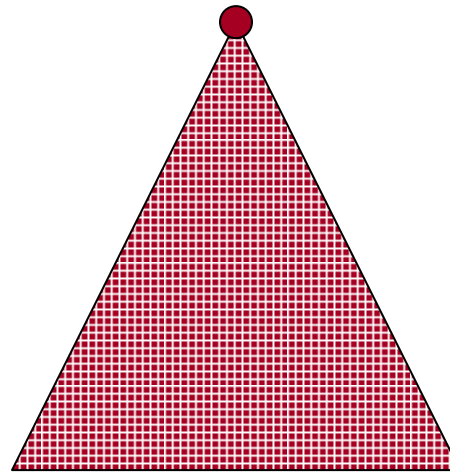
## X.500 Administrative Model - Trigger Execution Aspect



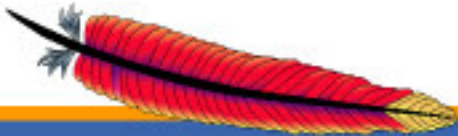
## What can be specified (How a TED can be specified) with a subtreeSpecification ? (1)



Administrative Point

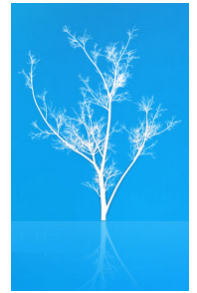


subtreeSpecification= { }

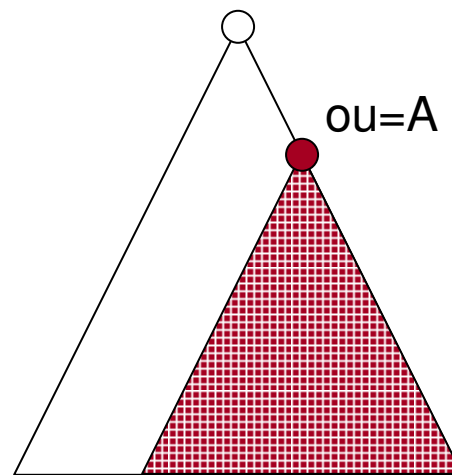




## What can be specified (How a TED can be specified) with a subtreeSpecification ? (2)



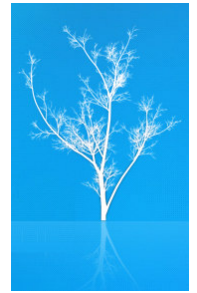
Administrative Point



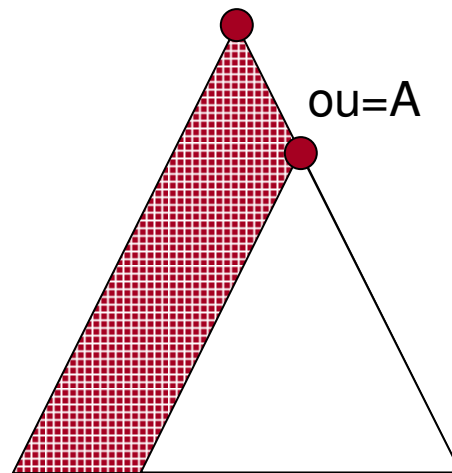
subtreeSpecification=  
{ base "ou=A" }



## What can be specified (How a TED can be specified) with a subtreeSpecification ? (3)



Administrative Point

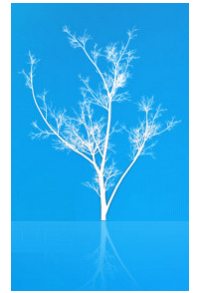


subtreeSpecification=

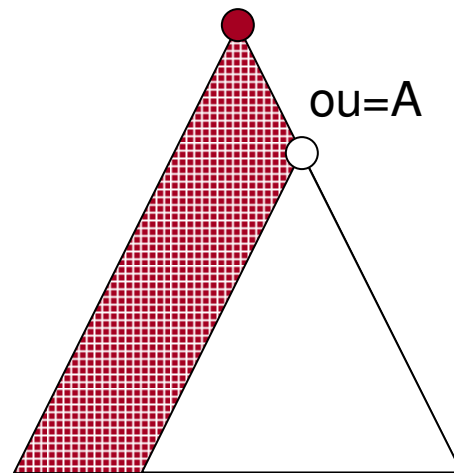
```
{ specificExclusions { chopAfter: "ou=A" } }
```



## What can be specified (How a TED can be specified) with a subtreeSpecification ? (4)



Administrative Point



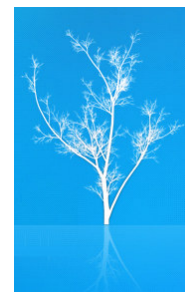
subtreeSpecification=

```
{ specificExclusions { chopBefore: "ou=A" } }
```

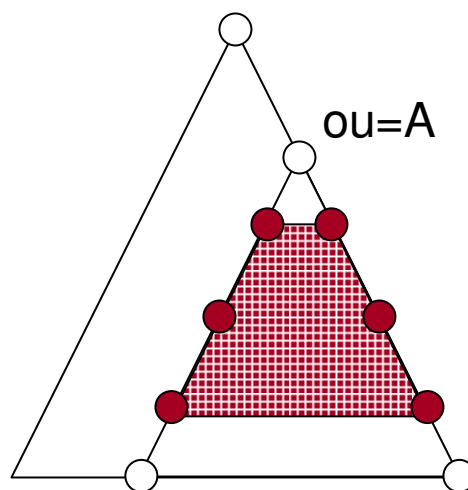




## What can be specified (How a TED can be specified) with a subtreeSpecification ? (5)



Administrative Point

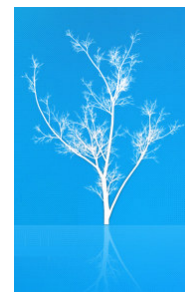


subtreeSpecification=

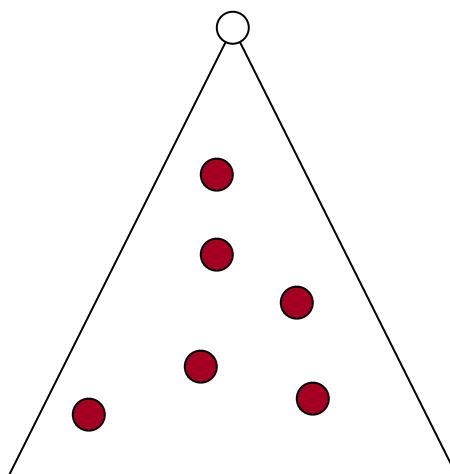
{ base "ou=A", minimum 1, maximum 3 }



## What can be specified (How a TED can be specified) with a subtreeSpecification ? (6)



Administrative Point

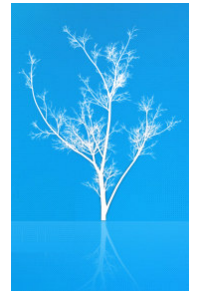


subtreeSpecification=

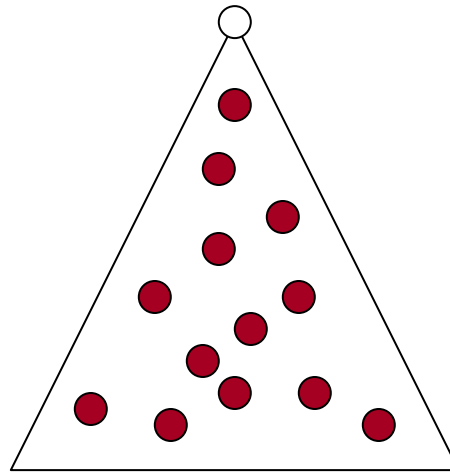
```
{ specificationFilter item:student }
```



## What can be specified *(How a TED can be specified)* with a subtreeSpecification ? (7)



Administrative Point

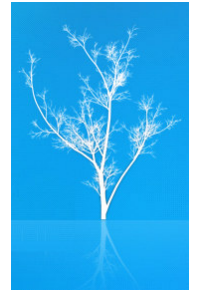


subtreeSpecification=

```
{ specificationFilter or: { item:student, item:faculty } }
```



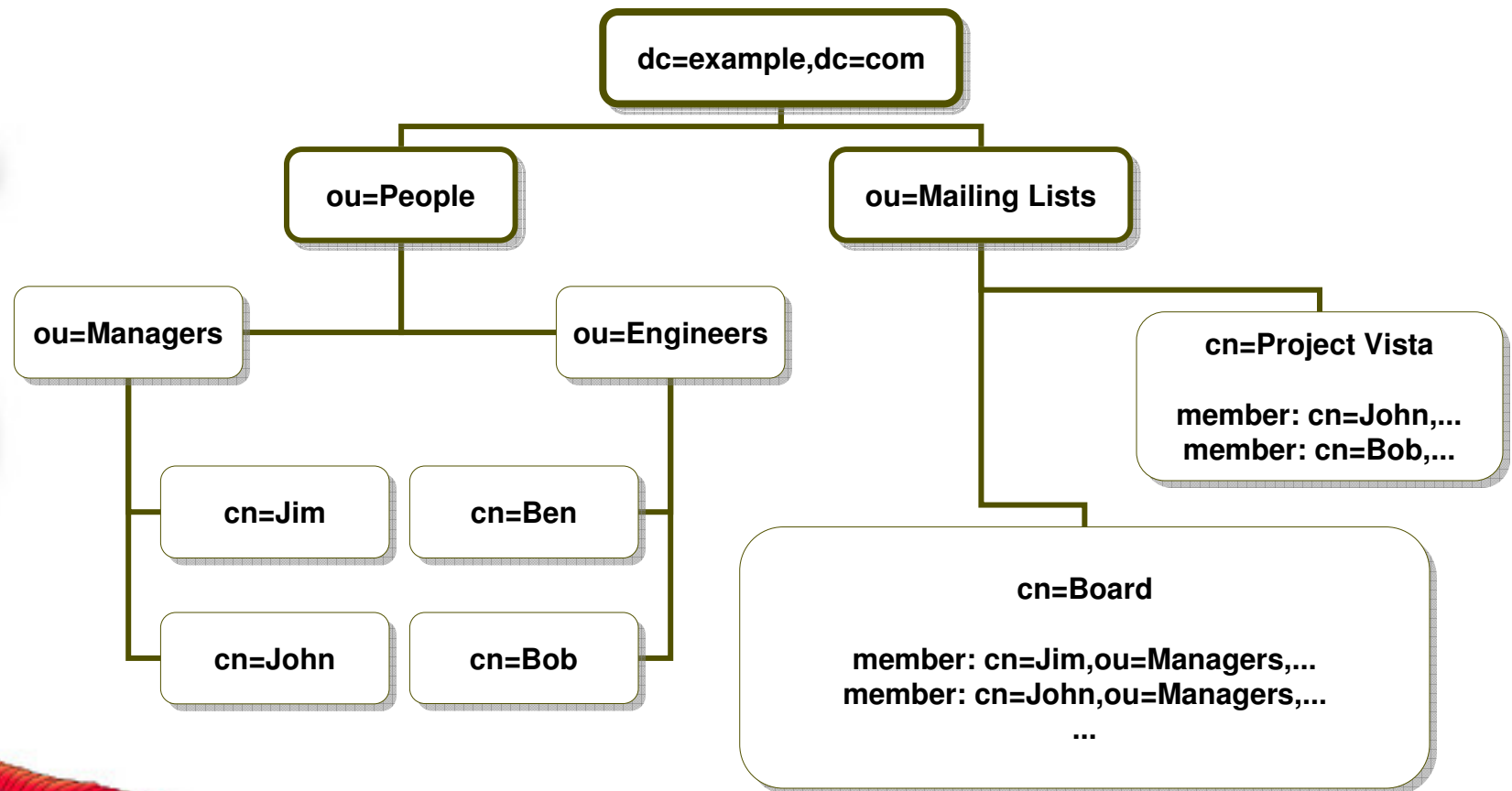
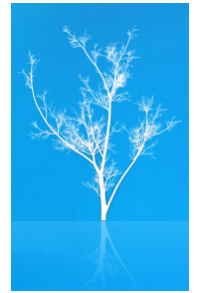
## DEMO 2 (Extensive)



- Think about mail lists whose configurations are stored in a directory
- Mail list members can be added/removed manually, or according to specific conditions like being in a specific subtree (or not)
- If a mail list member (likely a person's entry) is deleted from DIT, it should also be unsubscribed from the lists it was member of

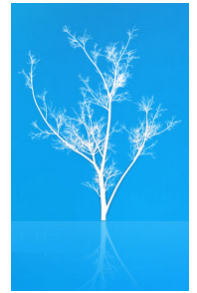


## Sample LDAP Tree





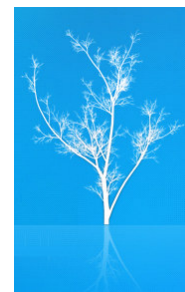
# Requirements



1. If any person entry under `ou=Managers,ou=People` is created (Add), add it to the Board list
2. If any person entry under `ou=People` is deleted, remove it from all lists
3. If any person entry is renamed under `ou=People`, correct membership registries in all lists
4. If any person entry is moved to `ou=Managers,ou=People`, add it to the Board list
5. If any person entry is moved from `ou=People` (to say `ou=Fired subtree`), remove it from all lists

# Implementation of Requirement 1

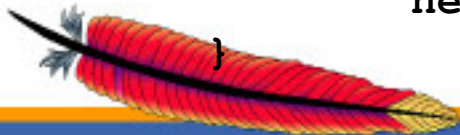
## Stored procedure



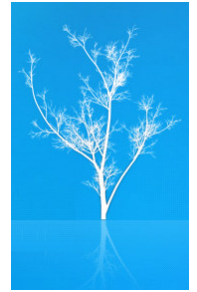
```
public static void subscribeAddedManagerToBoardList (
    LdapContext ctx,
    Name addedEntryName ) throws NamingException
{
    String boardMailListCtxName =
        "cn=Board," + mailListsCtxName;

    Attributes newMember = new BasicAttributes (
        "member",
        addedEntryName.toString(),
        true );

    ctx.modifyAttributes (
        boardMailListCtxName,
        DirContext.ADD_ATTRIBUTE,
        newMember );
}
```



## Implementation of Requirement 1 subtreeSpecification

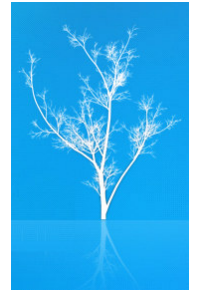


- A subtreeSpecification added in a triggerExecutionSubentry subordinate to the domain root
- { base “ou=Managers,ou=People”,  
specificationFilter item:person }





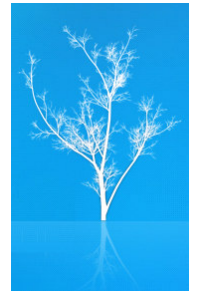
## Implementation of Requirement 1 prescriptiveTriggerSpecification



**AFTER Add**

```
CALL "MailListManager.  
    subscribeAddedManagerToBoardList"  
($ldapContext(""), $entry)
```



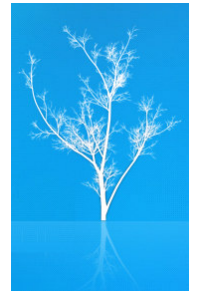


## To ease the remaining tasks...

- subtreeSpecifications can be reused in the same subentry as the prescriptiveTriggerSpecification attribute is multi-valued
- Stored procedures can be collected in the same class and can use each other

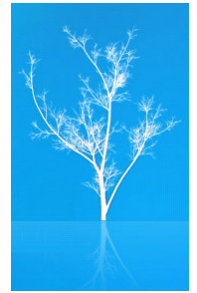


## What's coming next?



- BEFORE, INSTEADOF Triggers
- Mutable parameters for stored procedure called from Triggers





## LDAP Stored Procedures and Triggers arrive in ApacheDS

- Originally presented at *ApacheCon US 2006* in Austin
- Latest presentation materials are at <http://people.apache.org/~ersiner>
- Presented by *Ersin Er*, [ersiner@apache.org](mailto:ersiner@apache.org)

