Sun microsystems

CIFS Access Control and Identity Mapping in OpenSolaris

Afshin Salek
Alan Wright
cifs-discuss@OpenSolaris.org

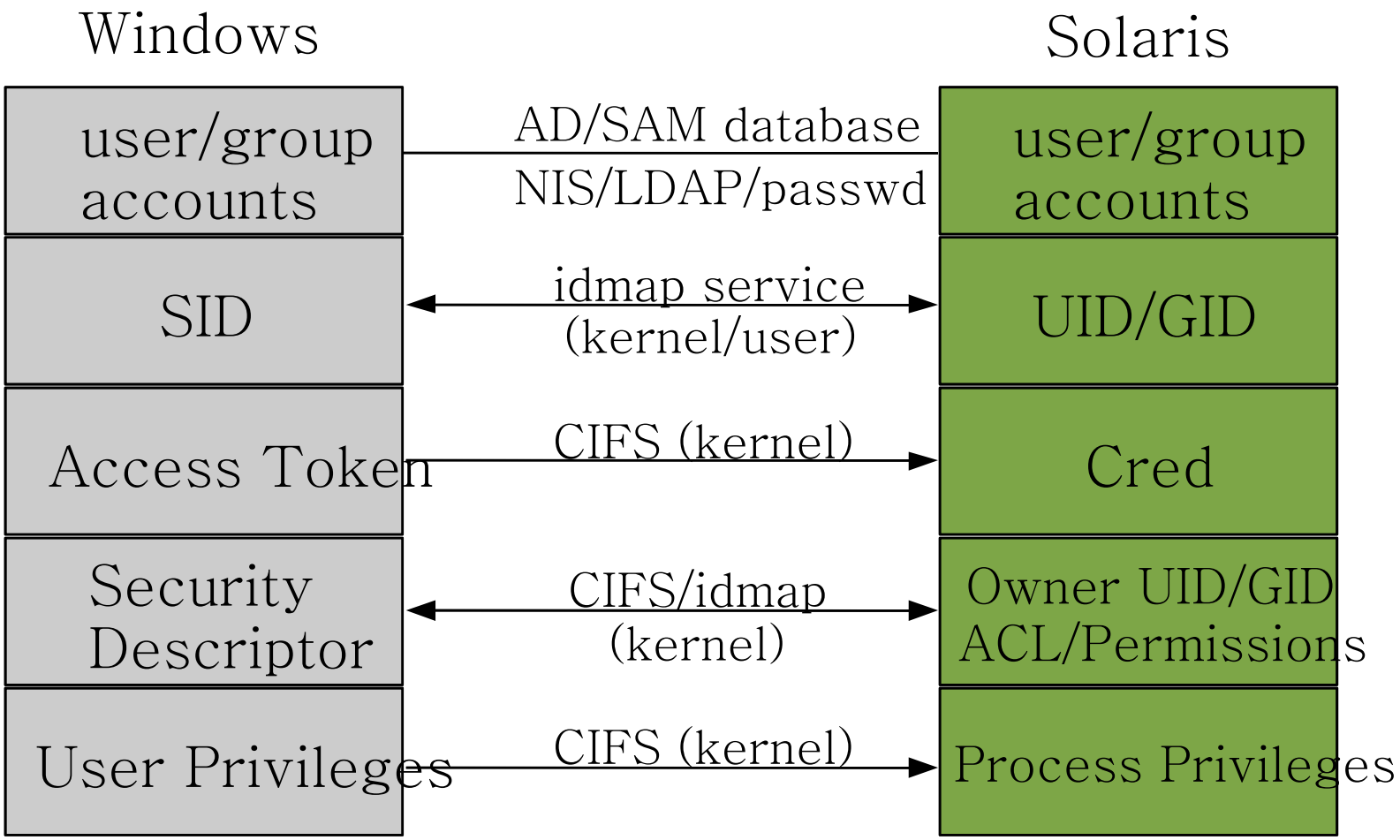
Agenda

- Access control components
- Identity mapping
- Authentication
  - Access token
  - Solaris credential
- Access control
  - Security descriptor and ZFS ACL

Access Control & Authentication

- Authentication
  - In a multiuser or network operating system, the process by which the system validates a user's logon information.
- Access Control
  - The mechanisms for limiting access to certain items of information or to certain controls based on users' identity and their membership in various groups.

Access Control Components



Identity Mapping

Solaris idmap Service

- Maps Windows accounts (names and SIDs) to POSIX accounts (names and UIDs/GIDs) and vice-versa
- An independent service i.e. not part of CIFS
- Current consumers
  - CIFS server
  - CIFS client
  - ZFS
  - NFSv4 (nfsmapid(1M))

Account Names

- Account names
  - Human understandable representation of accounts
- Namespace
  - *Windows:* same namespace for user/group names
    - Names are unique within a domain
  - *Solaris:* separate namespaces
    - A user and group account can have the same name
- Case sensitivity
  - *Windows:* case-**in**sensitive
  - *Solaris:* case-sensitive

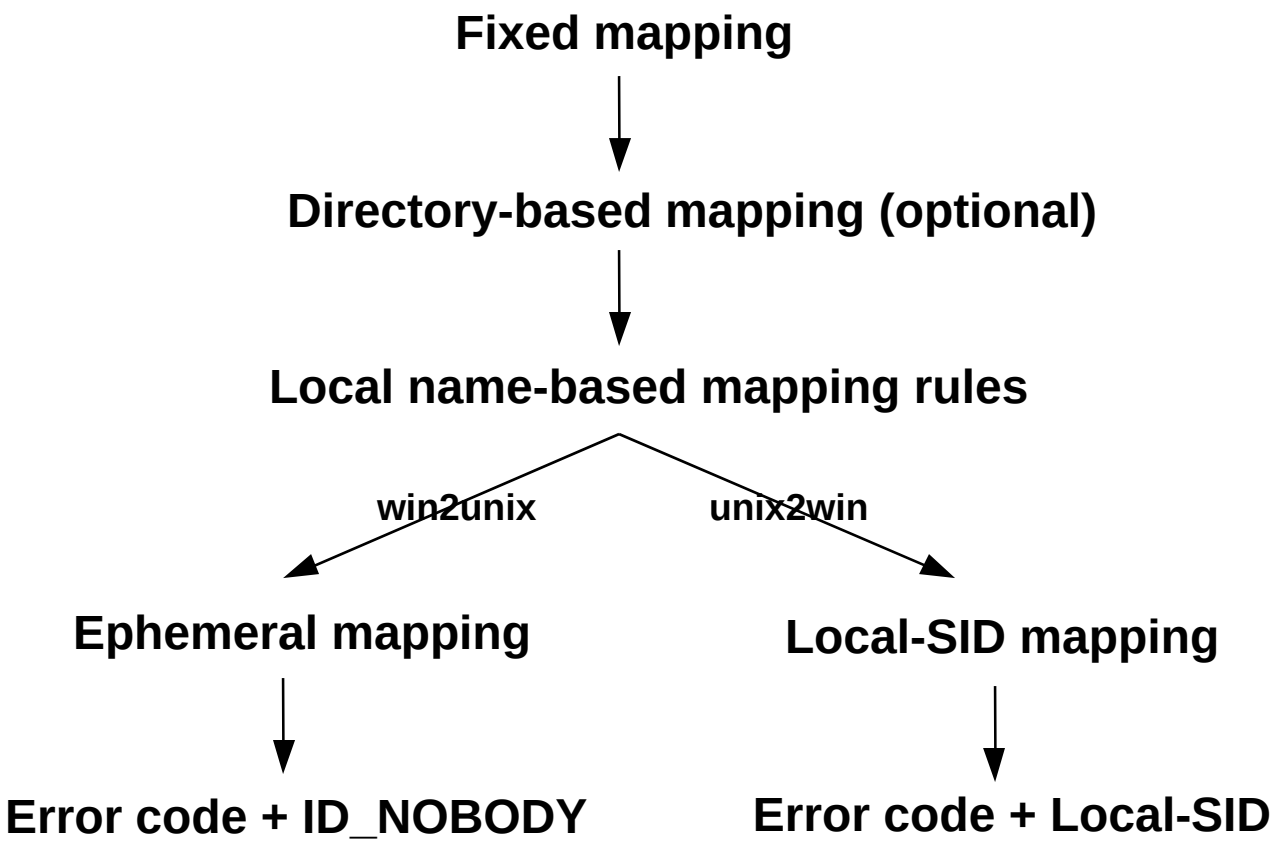
Account IDs

- Account IDs
  - OS internal representation of accounts
- Namespace
  - *Windows:* One huge hierarchical namespace
    - SIDs are universally unique
  - *Solaris:* two separate flat, fixed size namespaces
    - A user and group account can have the same ID even within the same domain
    - Users and groups can have the same IDs across domains

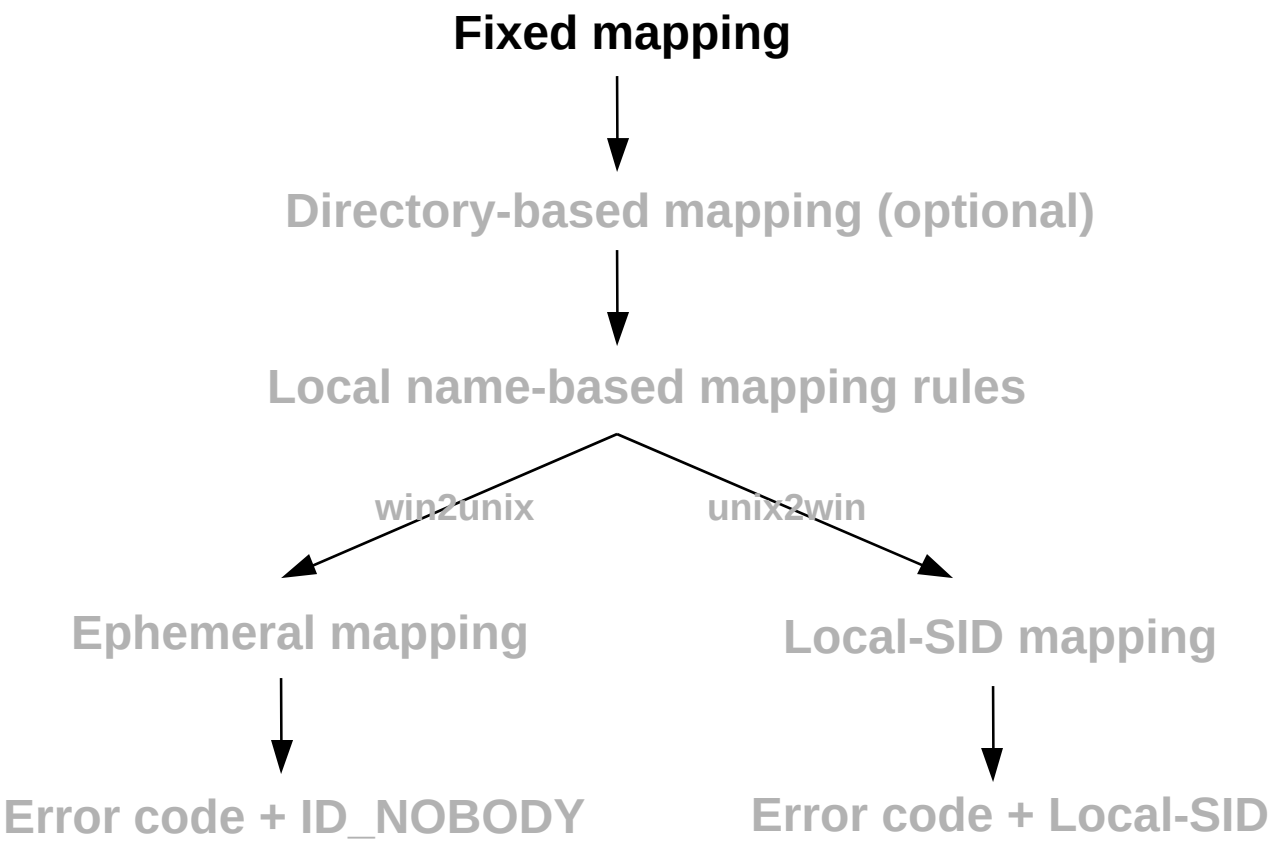
Solutions

- Name-based mapping
  - idmap rules
  - Directories (AD, native LDAP)
- ID-based mapping
  - MS Identity mapping for UNIX (IDMU)
- Ephemeral ID mapping
  - Steal previously unused "negative" UID/GID namespace
  - Allocate ephemeral IDs to SIDs on demand

Mapping Mechanisms



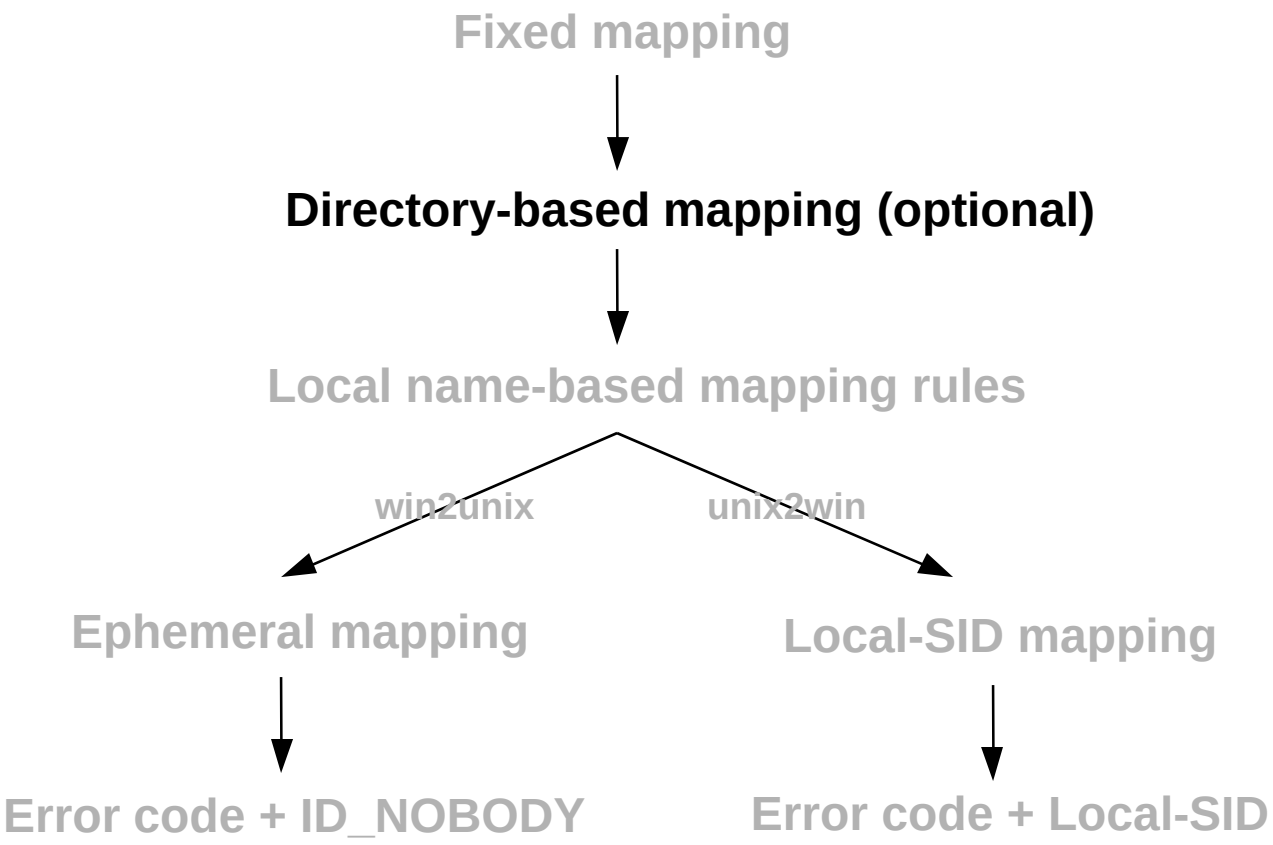
Mapping Mechanisms



Fixed Mappings

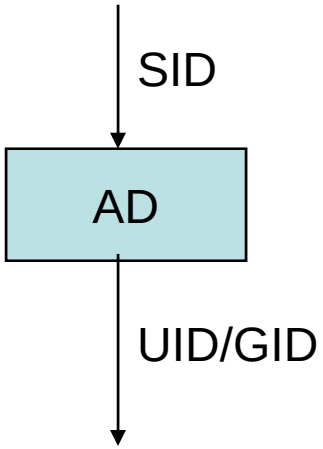
- Hard-coded mappings
  - wingroup Local System (S-1-5-18) = gid 2147483548
  - winuser Creator Owner (S-1-3-0) = uid 2147483548
  - wingroup Creator Group (S-1-3-1) = gid 2147483549
  - wingroup Anonymous Logon (S-1-5-7) -> gid: 60001

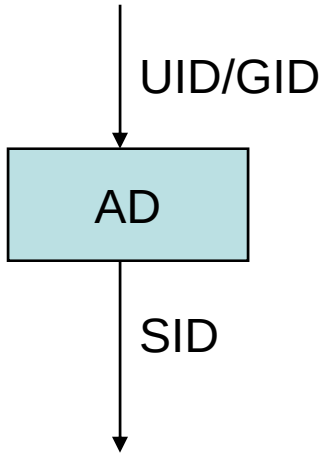
Mapping Mechanims



Directory-based ID Mapping

- Identity Management for UNIX (IDMU)
  - Microsoft optional AD component
  - Adds user interface for UNIX parameters – UID/GID, home directory, shell, etc
- Disabled by default

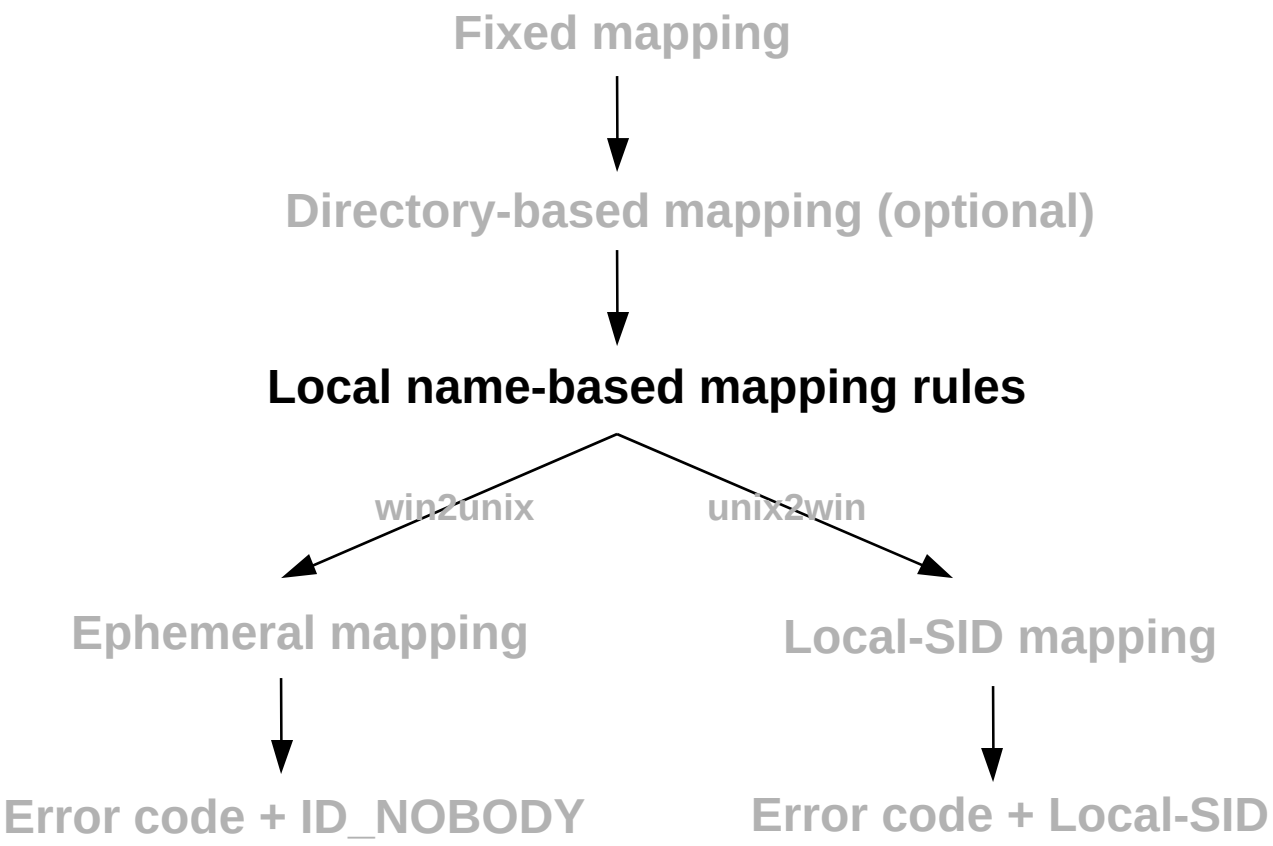




Directory-based NAME Mapping

- Uses mapping information stored in user/group objects in the directory server
  - Windows user/group objects in AD may contain corresponding Unix name.
  - Solaris user/group objects in native LDAP server may contain corresponding Windows name.
- Disabled by default
- Modes of operation
  - AD-only mode
  - Native-LDAP-only mode

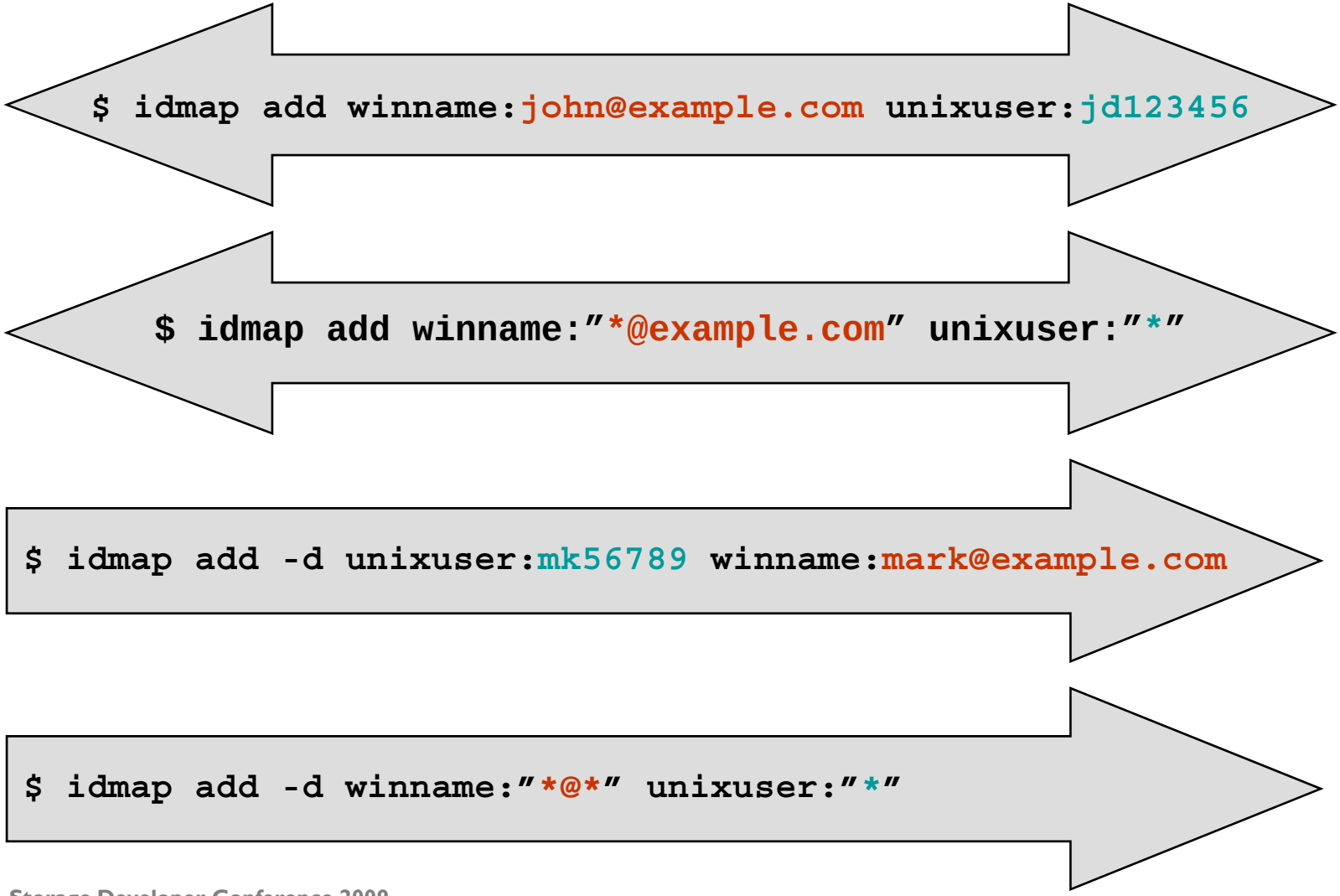
Mapping Mechanisms



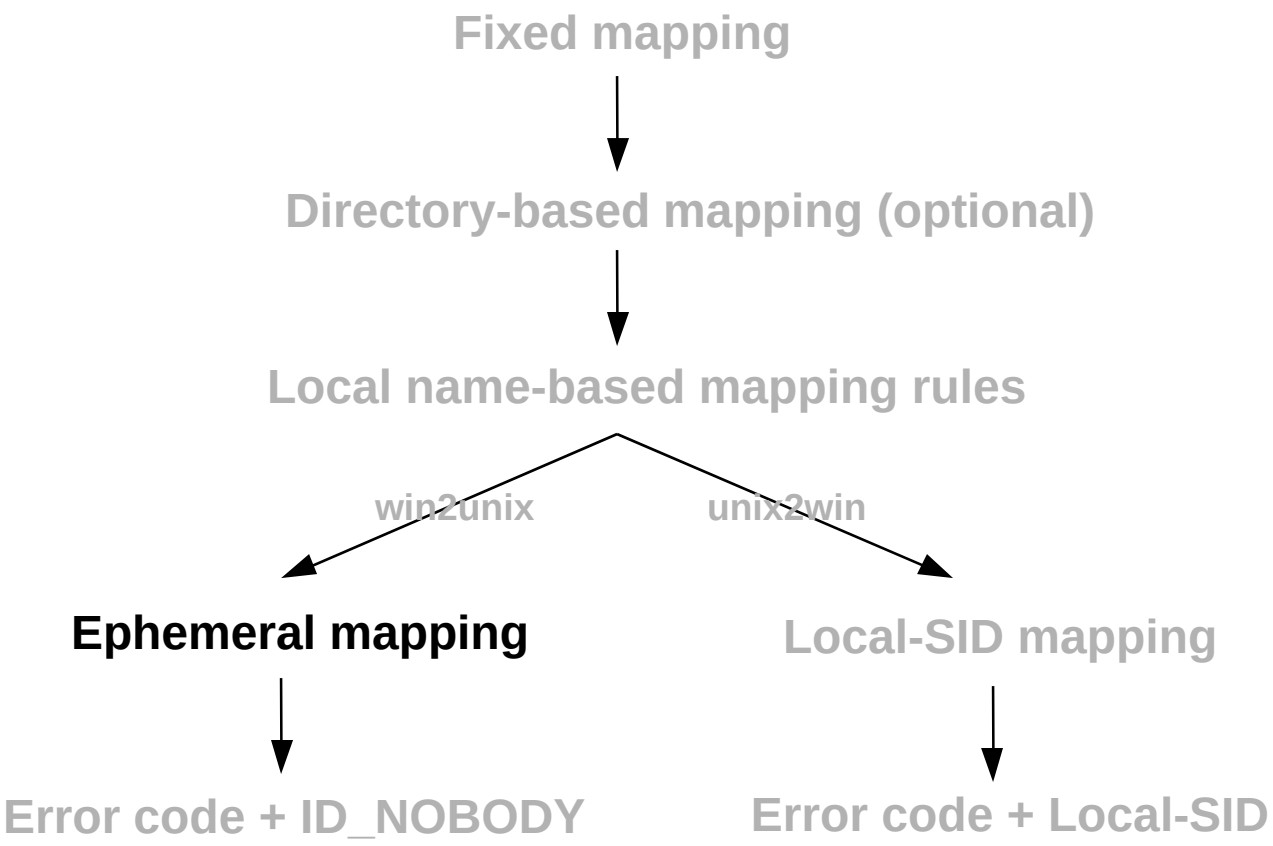
Local name-based Mapping Rules

- Maps using locally stored mapping rule
  - Add, list and remove rules using idmap(1M)
  - Export/import rules from Netapp's usermap.cfg file and Samba's smb.conf file using idmap(1M)
  - See idmap(1M) for ordering between rules

Local name-based Mapping Rules



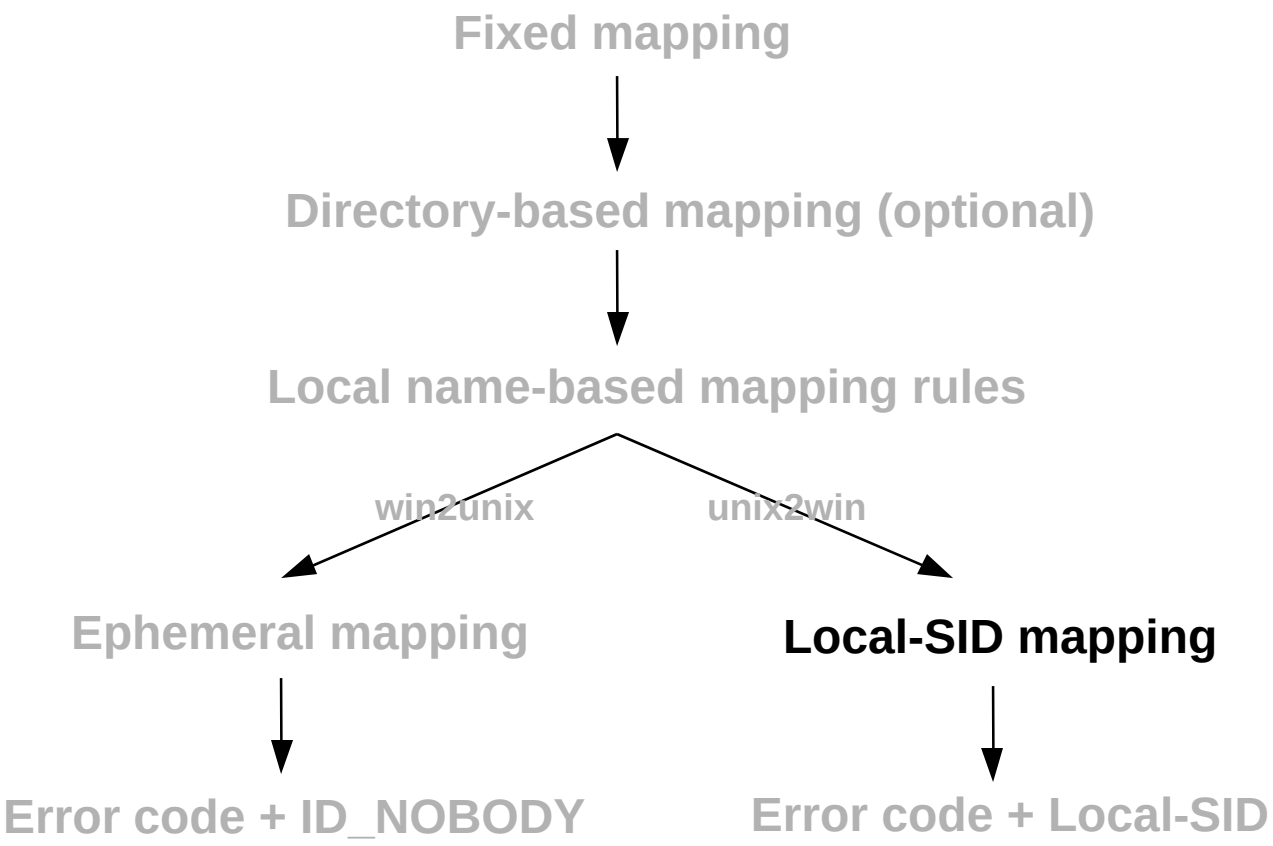
Mapping Mechanisms



Ephemeral Mapping (win2unix)

- If Windows identity cannot be mapped using any of previous methods then it is mapped to a dynamically allocated uid/gid
  - Uses next available uid or gid from 2^{31} to $2^{32} - 2$ (ephemeral uids/gids). See PSARC/2007/064
- Zero configuration
- Not stored in file system or any name service
- Not retained across reboots

Mapping Mechanisms



Local SID Mapping (unix2win)

- If a non-ephemeral Unix uid/gid cannot be mapped by any of previous methods then it is mapped to a algorithmically generated SID called local-SID
- The local-SID is generated as follows:
  - local-SID for UID = <machine SID> - <1000 + UID>
  - local-SID for GID = <machine SID> - <2^{31} + GID>
- <machine SID> is a unique SID generated by the idmap service for the host on which it runs
  - Generated the first time idmap runs

Local SID Mapping

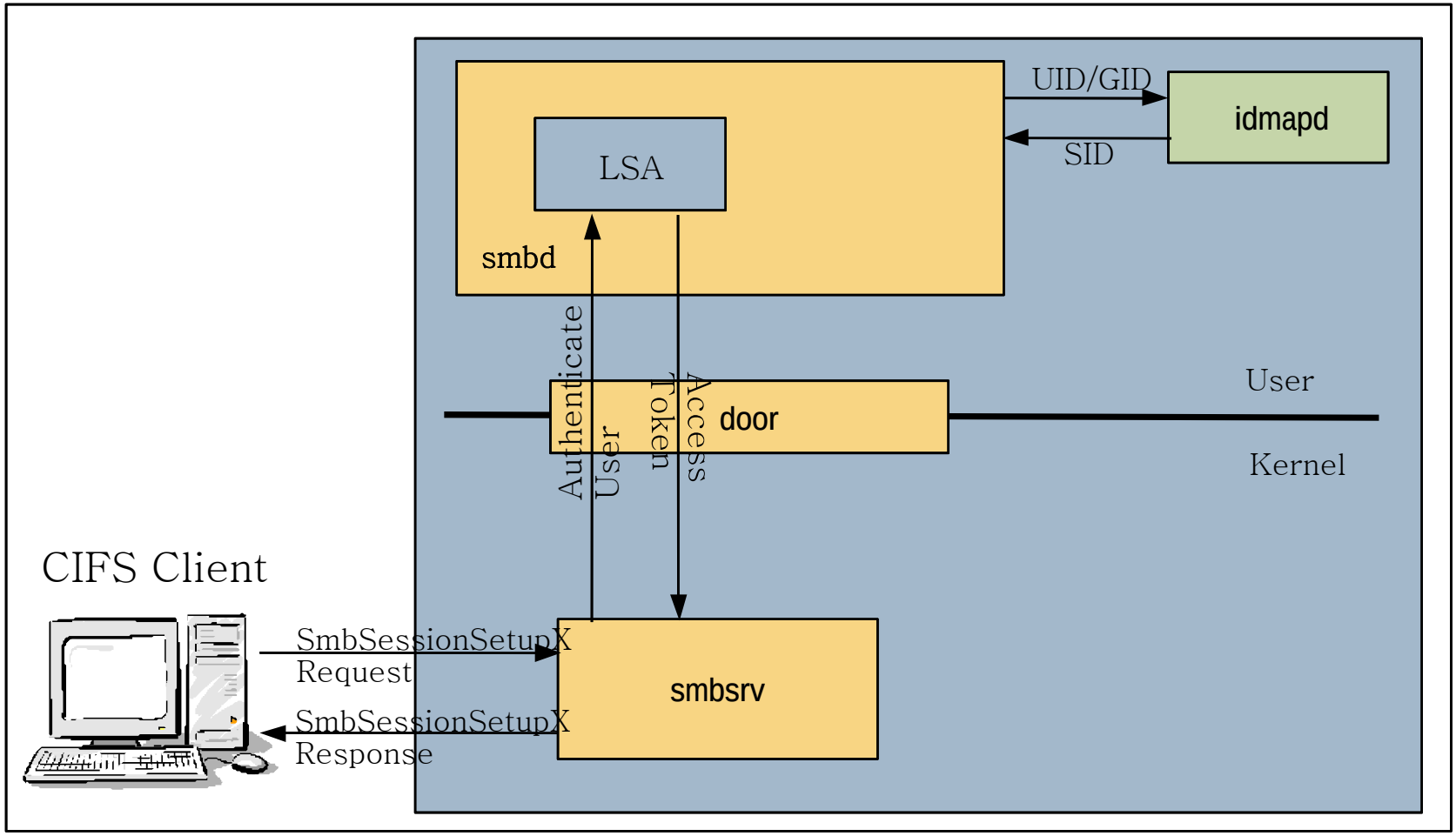

```
$ svcprop -p config/machine_sid system/idmap
S-1-5-21-735436889-4024298704-402121877

$ idmap show -c uid:70000

uid:70000 -> sid:S-1-5-21-735436889-4024298704-402121877-71000

$ idmap show -c gid:70000

gid:70000 -> sid:S-1-5-21-735436889-4024298704-402121877-2147553648
```

Authentication
Access Token and Solaris Credential

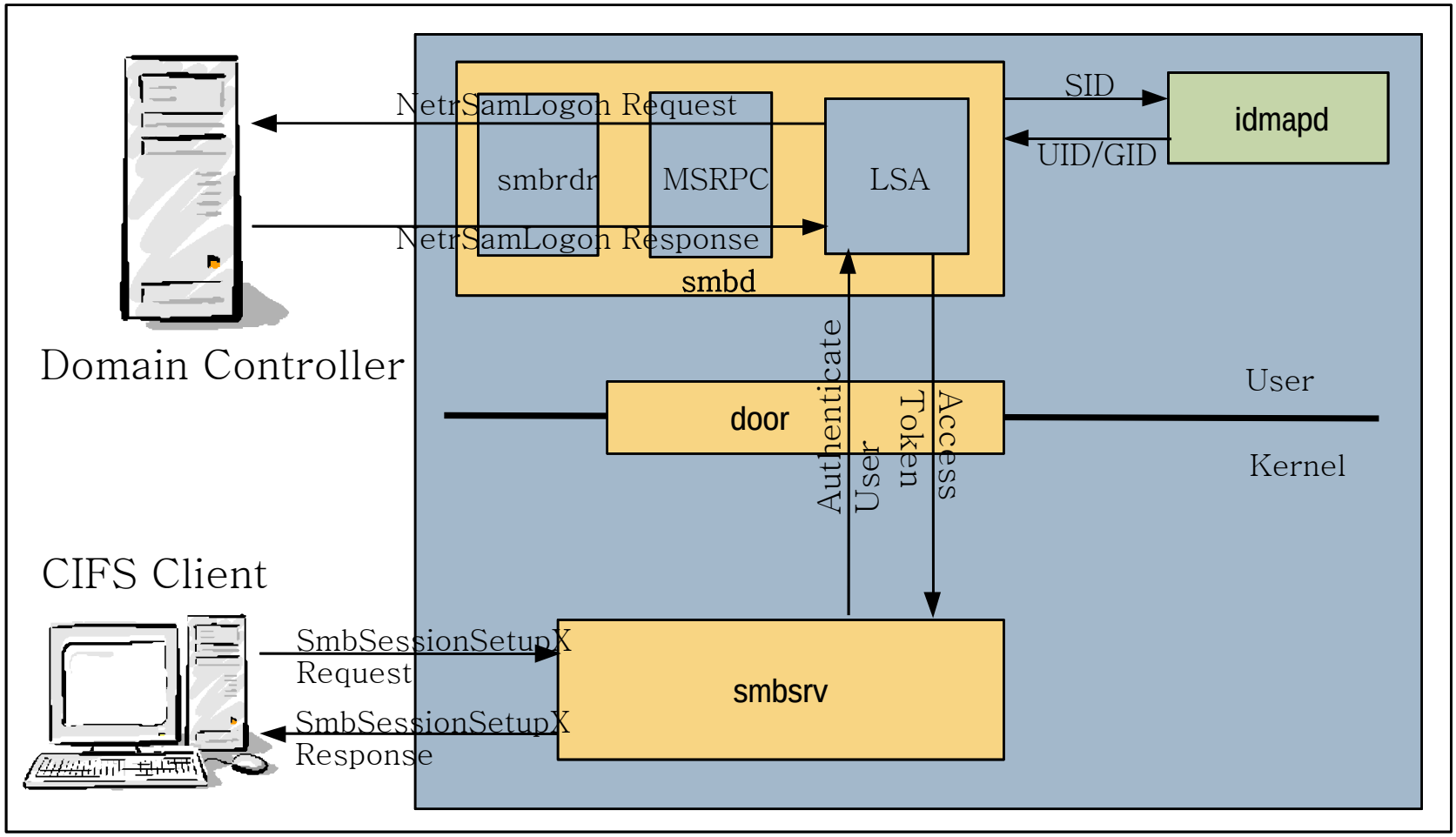
CIFS Server Operation Modes

- Solaris CIFS server operation mode determines where a connected user gets authenticated
  - Workgroup
    - CIFS server authenticates local users (users defined in /etc/passwd)
    - passwd(1) should be used to generate CIFS encrypted passwords (using smb PAM module)
  - Domain
    - CIFS server authenticates local users
    - Domain controller authenticates domain users

Local Authentication



Pass-Through Authentication



Typical Access Token

- Once a user is authenticated, an access token is created
  - Security identifier (SID) for the user
  - SID for all the groups to which the user belongs
  - User's privileges
  - …..

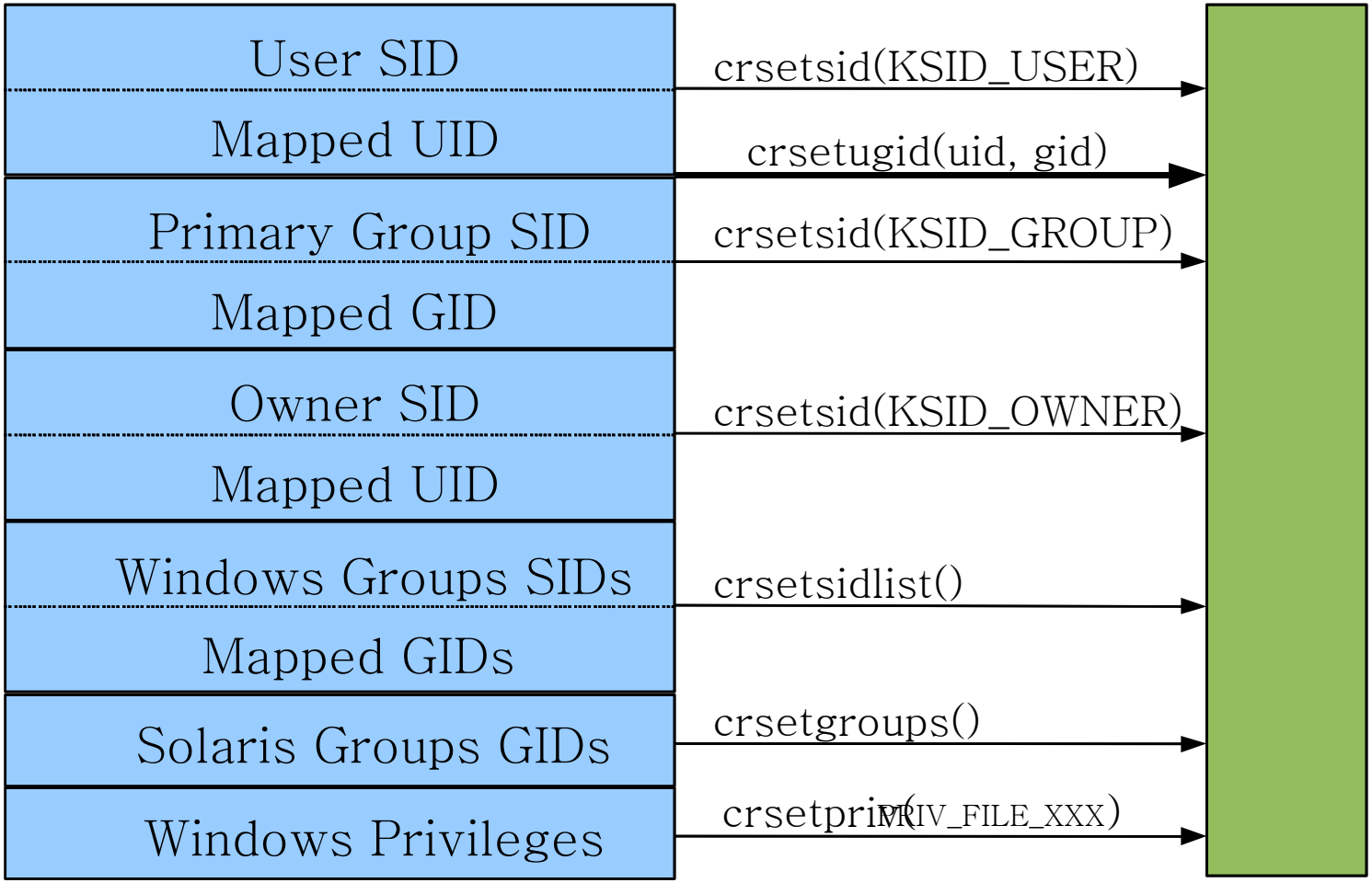
CIFS Server Access Token

- Authentication takes place is userspace (smbd) so Access Token is created in user-space
- All the SIDs are mapped to UIDs/GIDs using the idmap service
- Solaris groups for the mapped user are added to the token
- File systems enforce access control
  - Access token is transferred to kernel
  - smbsrv creates a Solaris credential based on the token

Token to Cred Mapping

- Solaris cred structure had to be enhanced to make this mapping possible
- A new field (cr_ksid) is added to Solaris credential structure for storing SIDs
- cred is an opaque structure so new functions have been introduced to access this new field
  - crsetsid
  - crsetsidlist
  - crgetsid
  - crgetsidlist

Token to Cred Mapping (Diagram)



Access Control
Security Descriptor and Solaris ACLs

Access Control Levels

- Host-based access control lists
  - Share level
  - None, read-only, read-write lists of hosts
  - Enforced by CIFS server
  - Wide open by default
- Share ACL
  - Enforced by CIFS server
  - Everyone has full control by default
- File/folder ACL
  - Enforced by exported file system

Share ACL

- Only supported on ZFS
  - Special directory (.zfs/shares) per ZFS dataset
  - Each share is represented by a file
  - Share ACL is the file's ACL
- Enforced by CIFS server
  - Effective permission for connected user is determined at TreeConnect time in kernel by calling VFS VOP_ACCESS and cached in smb_tree structure
  - Each smb_fsop operation checks the requested access against the tree granted access mask
- Can be managed on Solaris or Windows
  - ls/chmod on Solaris
  - Windows share management GUI

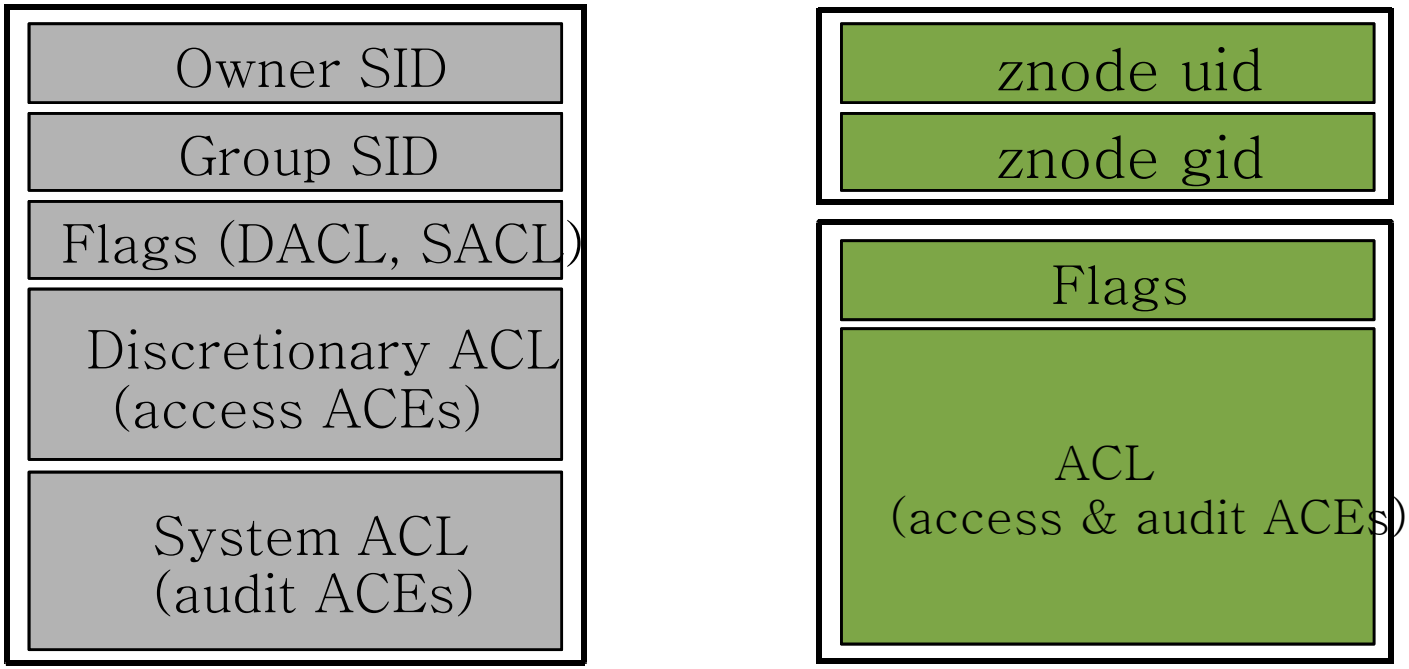
ZFS ACL

- Based on NFSv4 ACLs
- All files have ACL with at least one ACE
- Trivial ACL
  - represents POSIX permissions
  - always six ACEs
- Special ACEs
  - owner@: represents owner permissions
  - group@: represents group permissions
  - everyone@: represents other permissions
- ZFS is POSIX compliant

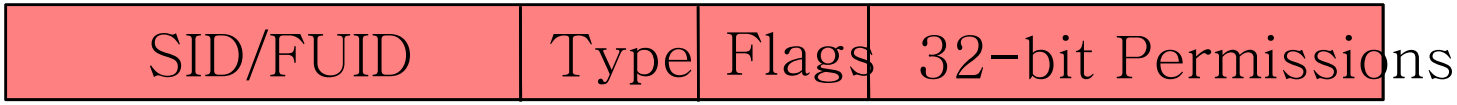
ZFS ACL examples


```
$ ls -V trivial.acl
-rw-r--r--   1 root     root  0 Sep 11 16:06   trivial.acl
                  owner@:--x-----------:-------:deny
                  owner@:rw-p---A-W-Co-:-------:allow
                  group@:-wxp----------:-------:deny
                  group@:r-------------:-------:allow
                 everyone@:-wxp---A-W-Co-:-------:deny
                 everyone@:r-----a-R-c--s:-------:allow

$ ls -V non-trivial.acl
----------+  1 afshin   other 0 Sep 11 16:28 non-trivial.acl
             user:afshin:-w-p---A-W----:-------:allow
             group:other:r-----a-R-c---:-------:allow
```

Security Descriptor vs. ZFS ACL



Windows/ZFS ACE format



Filesystem Unique Identifier (FUID)

- An unsigned 64-bit integer
- Upper 32-bit is an index into an auxiliary table of domain SIDs
- Lower 32-bit is a relative identifier within the domain above
- Domain index of 0 means FUID represents a standard POSIX UID/GID
- All ephemeral UIDs/GIDs will be stored as FUIDs with a non-zero domain index

SD vs. ZFS ACL: Differences

- Different account identifier (SID vs. UID/GID)
  - Unified by introducing FUID (PSARC 2007/064)
- ZFS ACE has user/group differentiator
- 1 entity vs. 2 entities (znode, zacl)
  - 2 VFS calls needed to get/set information
    - VOP_[GS]ETATTR, VOP_[GS]ETSECATTR
- DACL/SACL vs. ACL

SD vs. ZFS ACL: Differences (cont)

- ZFS ACL must have at least one ACE
  - NULL DACL ->
    ZFS: everyone@ ACE with full permissions
  - Empty DACL ->
    ZFS: user ACE with owner UID and owner implicit permissions
- CREATOR_OWNER/GROUP
  - only used in inheritance not access check
- owner@, group@
  - used to represent traditional owner/group permission groups

SD vs. ZFS ACL: Similarities

- Same ACE types (allow, deny, audit, etc)
- Same ACE permission bits
- Same ACE inheritance flags
  - ZFS `aclinherit` dataset property affects inheritance
- Same access check algorithm

SD to ZFS ACL Mapping

- Map SD flags to ZFS ACL flags
- If owner SID exists; map to UID
- If group SID exists; map to GID
- For each ACE:
  - Map SID to UID/GID
    - If everyone SID: set ACE_EVERYONE flag
    - If mapped to a GID: set ACE_IDENTIFIER_GROUP flag
  - Map flags
    - Some flags don't have the same values

SD to ZFS ACL (notes)

- SACL flags are ignored because ZFS only has one list
- No owner@ or group@ ACEs in the mapped ZFS ACL
  - POSIX owner/group permission groups will be empty

```
$ ls -V non-trivial.acl
----------+  1 afshin   other 0 Sep 11 16:28 non-trivial.acl
          user:afshin:-w-p---A-W----:-------:allow
          group:other:r-----a-R-c---:-------:allow
```

ZFS ACL to SD Mapping

- Map znode's uid to owner SID
- Map znode's gid to group SID
- For each ACE
  - Map UID/GID to SID via idmap service
  - Map flags
- Step above is done separately for access and audit ACEs to generate DACL and SACL
- Sort the result DACL before sending it to client

DACL Sort Issue

- Windows GUI needs DACL to be sorted
- Simply: access denied ACEs should appear before access allowed ACEs
- ZFS trivial ACL which represents traditional Unix permission bits is not sorted
- If a file's ACL is viewed and saved by a Windows client, the ACL will be sorted which will change the file's effective permissions

DACL Sort Issue: Illustration


```
$ ls -v file.3
-rw-r--r--   1 marks    staff          0 Oct  9 15:49 file.3

0:owner@:execute:deny
1:owner@:read_data/write_data/append_data/write_xattr/write_attributes/write_acl/write_owner:allow
2:group@:write_data/append_data/execute:deny
3:group@:read_data:allow
4:everyone@:write_data/append_data/write_xattr/execute/write_attributes/write_acl/write_owner:deny
5:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
```

After viewed and saved by Windows client (note how Unix permissions have changed):

```
$ ls -v file.3
-r--r--r--+  1 marks    staff          0 Oct  9 15:49 file.3

0:owner@:execute:deny
1:group@:write_data/append_data/execute:deny
2:everyone@:write_data/append_data/write_xattr/execute/write_attributes/write_acl/write_owner:deny
3:owner@:read_data/write_data/append_data/write_xattr/write_attributes/write_acl/write_owner:allow
4:group@:read_data:allow
5:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
```

ACL Inheritance

- ZFS ACL inheritance is affected by:
  - POSIX inheritance rules (umask, creation mode, etc)
  - aclinherit ZFS property setting
  - ACL inheritance flags
- Default ZFS behavior is primarily accommodating POSIX so it is not similar to Windows behavior and will be confusing for CIFS users
- CIFS server will apply Windows inheritance rules for CIFS operations regardless of ZFS settings

Questions?

Appendix

Dir-based Idmap AD-only mode

Used when AD user/group objects contain corresponding Unix name

AD object

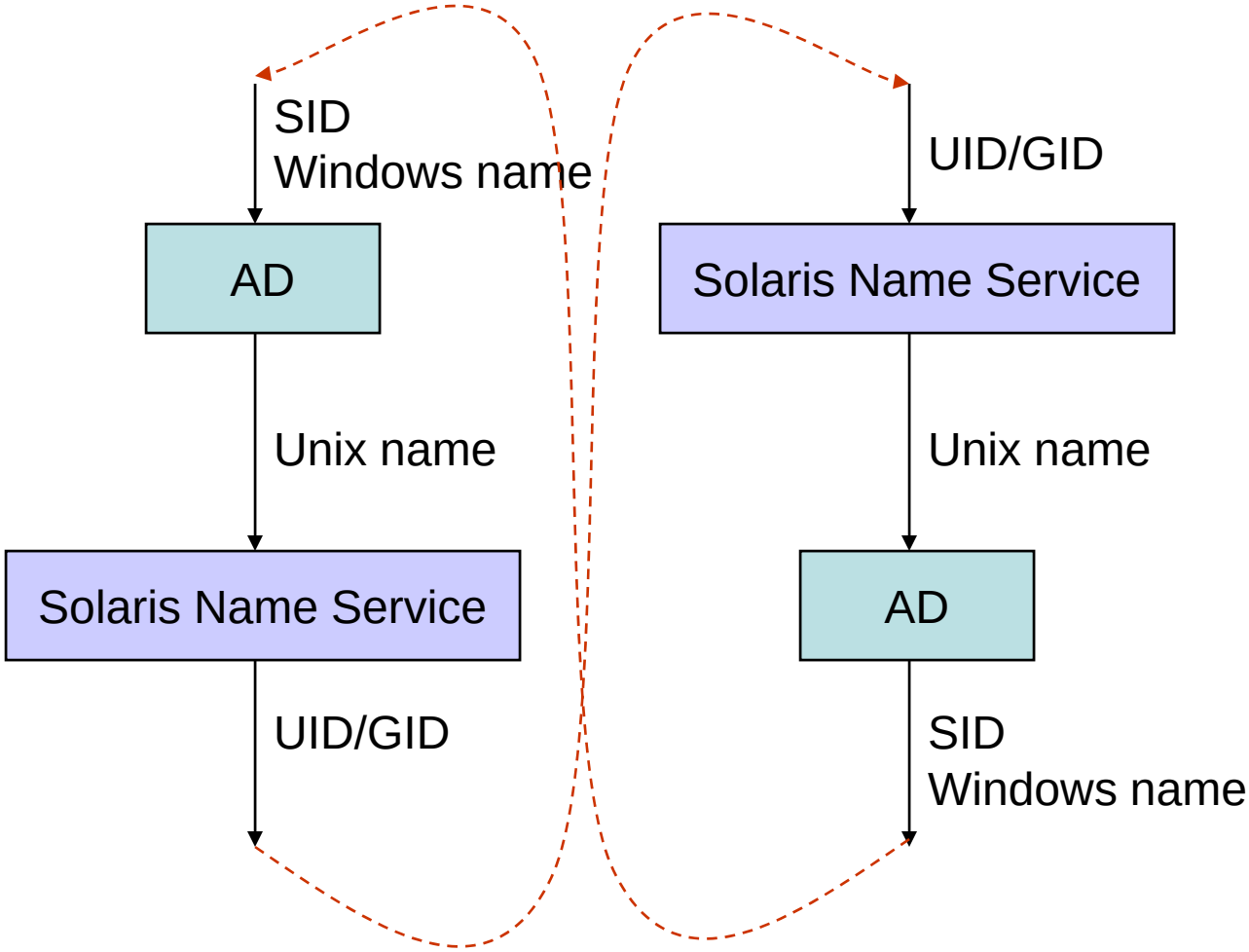
dn: cn=john doe,ou=users,dc=example
samAccountName: john
objectSID: S-1-5-21-11111-22222-33333
unixusername: jd123456

Unix entry

jd123456:x:123456:10:John Doe:/home/jd123456:/bin/ksh

```
svccfg -s idmap setprop config/ds_name_mapping_enabled = boolean: true
svccfg -s idmap setprop config/ad_unixuser_attr = astring: unixusername
svccfg -s idmap setprop config/ad_unixgroup_attr = astring: unixgroupname
```

AD-only Mode (symmetrical)



Dir-based Idmap Native-LDAP-only

Used when Solaris user/group objects in native LDAP server contains corresponding Windows name

native LDAP object

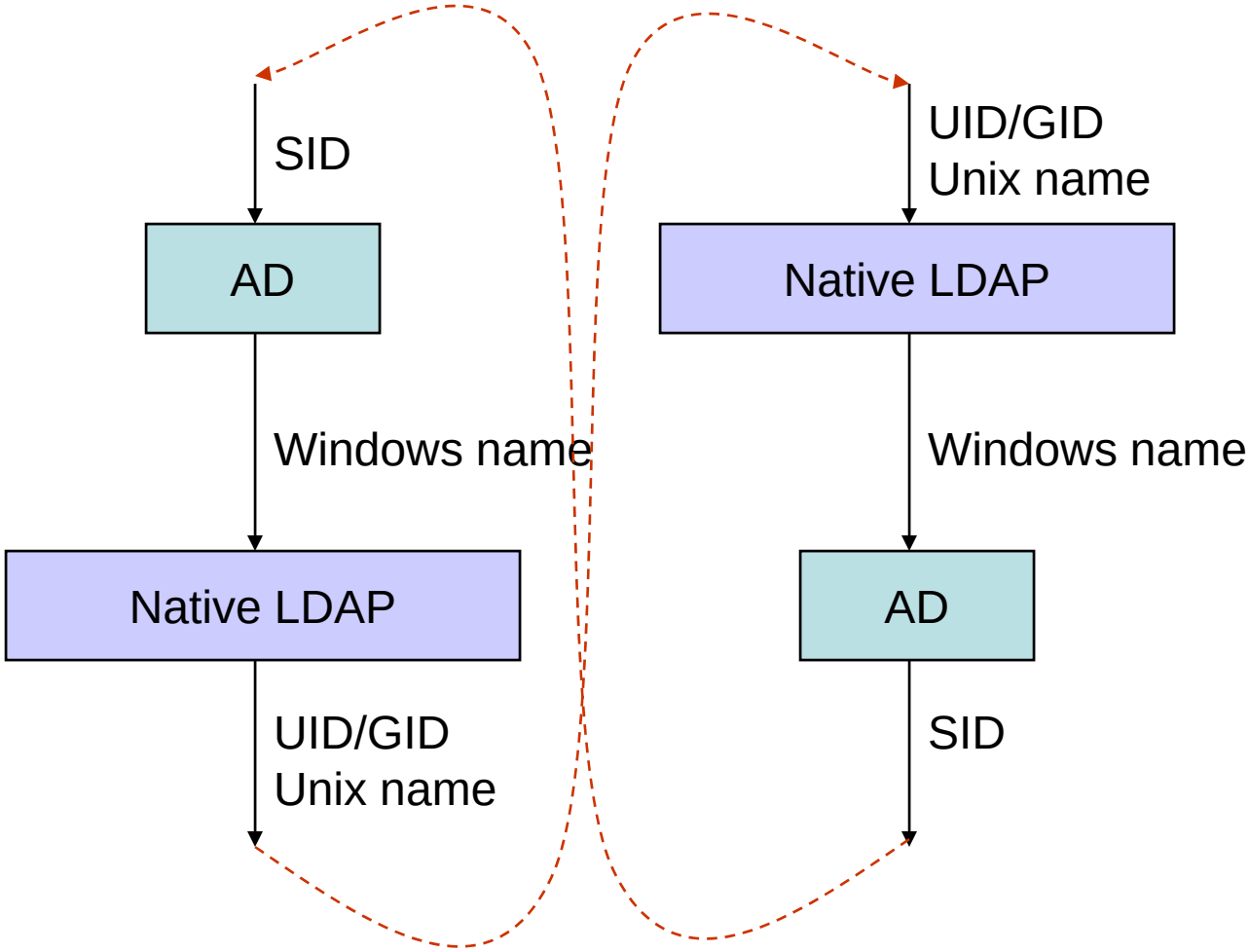
dn: uid=jd123456,ou=passwd,dc=example
uid: jd123456
uidNumber: 123456
winname: john@example

AD object

dn: cn=john doe,ou=users,dc=example
samAccountName: john
objectSID: S-1-5-21-11111-22222-33333

```
svccfg -s idmap setprop config/ds_name_mapping_enabled = boolean: true
svccfg -s idmap setprop config/nldap_winname_attr = astring: winname
```

Native-LDAP-only Mode (symmetrical)



Mixed Mode

Used when AD objects contain Unix name and native LDAP objects contain Windows name

native LDAP object

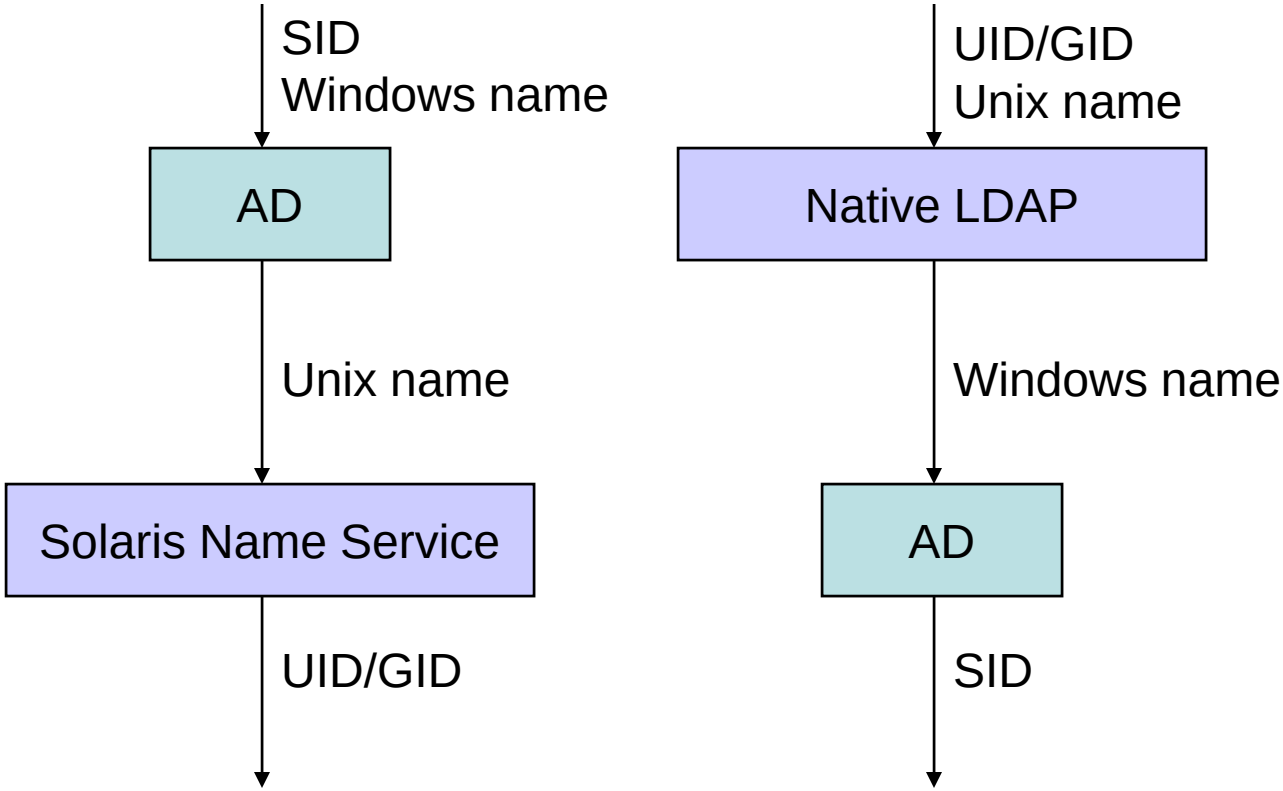
dn: uid=jd123456,ou=passwd,dc=example
uid: jd123456
uidNumber: 123456
winname: john

AD object

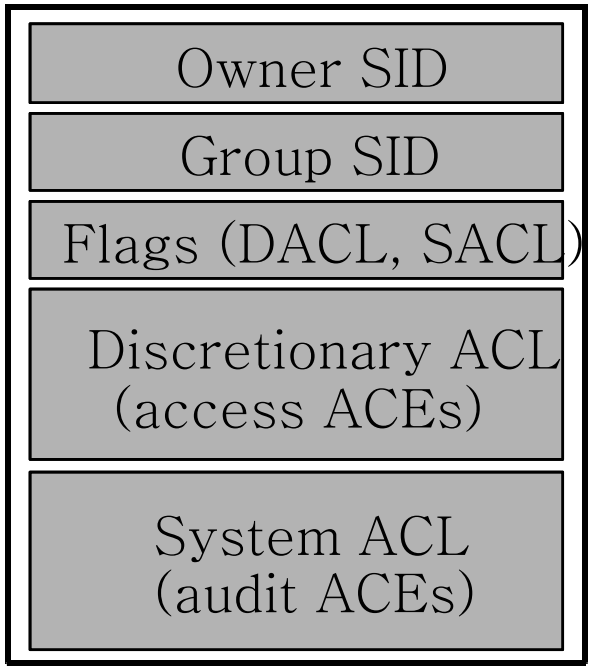
dn: cn=john doe,ou=users,dc=example
samAccountName: john
objectSID: S-1-5-21-11111-22222-33333
unixusername: jd123456

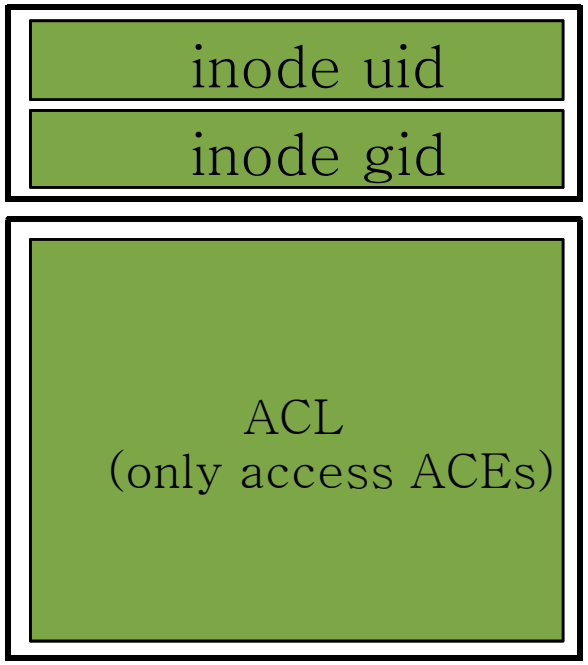
```
svccfg -s idmap setprop config/ds_name_mapping_enabled = boolean: true
svccfg -s idmap setprop config/nldap_winname_attr = astring: winname
svccfg -s idmap setprop config/ad_unixuser_attr = astring: unixusername
svccfg -s idmap setprop config/ad_unixgroup_attr = astring: unixgroupname
```

Mixed Mode (asymmetrical)

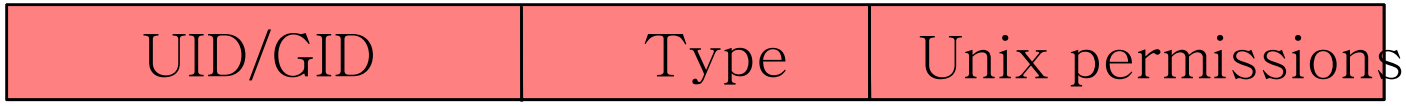


Security Descriptor vs. POSIX ACL





POSIX ACE format



SD vs. POSIX ACL: Differences

- SID vs. UID/GID
- 1 entity vs. 2 entities
- No explicit 'deny' ACEs
- ~15 vs. 3 permission bits
- Inheritance rules are different
- Different access check algorithm
- No audit ACEs

SD mapping (non-ZFS)

- SD to POSIX ACL
  - CIFS service maps SD to ZFS ACL
  - ZFS ACL is mapped to POSIX ACL using Solaris acl_translate() which implements IETF draft "Mapping Between NFSv4 and Posix Draft ACLs"
  - This translation is not always possible and could fail
  - Ephemeral IDs cannot be used in POSIX ACLs
- SD from POSIX ACL
  - POSIX ACL is mapped to ZFS ACL using Solaris acl_translate()
  - CIFS service maps the ZFS ACL to SD