

Role-based access control

RBAC: Motivations

- One challenging problem in managing large systems is the complexity of security administration
 - Whenever the number of subjects and objects is high, the number of authorizations can become extremely large
 - Moreover, if the user population is highly dynamic, the number of grant and revoke operations to be performed can become very difficult to manage

RBAC: Motivations

- End users often do not own the information for which they are allowed access. The corporation or agency is the actual **owner** of data objects
- Control is often based on employee functions rather than data ownership
- RBAC has been proposed as an *alternative* approach to DAC and MAC both to simplify the task of access control management and to directly support function-based access control

RBAC: Motivations

- RBAC assigns permissions to specific groups with meaning in the organization, rather than directly to users
 - Users can be easily reassigned from one role to another.
 - Roles can be granted new permissions as new applications and systems are incorporated, and permissions can be revoked from roles as needed.
 - Permissions assigned to roles tend to change relatively slowly
- Let administrators confer and revoke user membership in existing roles without authorizing them to create new roles or change role-permission
 - Assigning users to roles requires less technical skill than assigning permissions to roles.

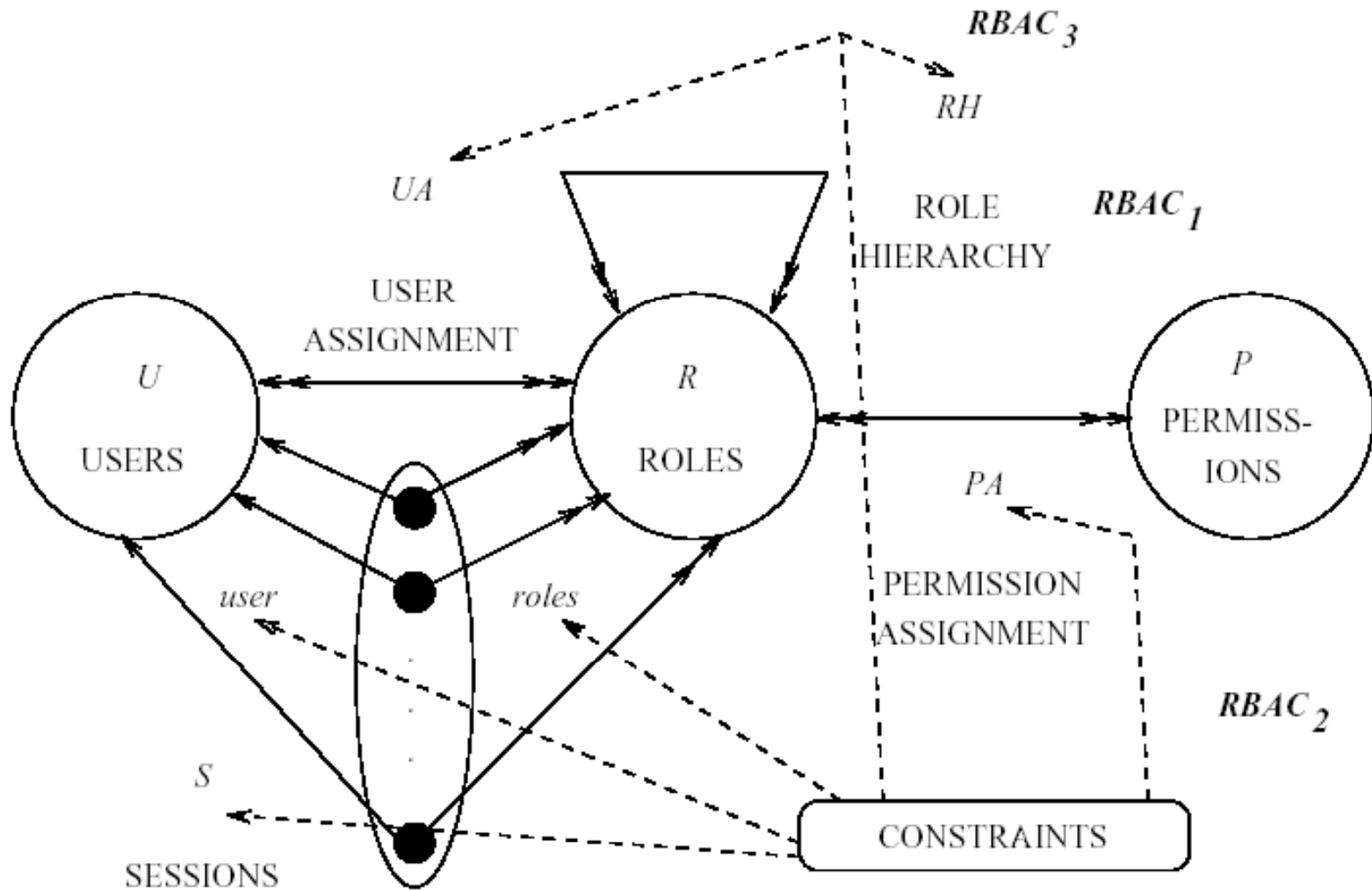
Role-Based Access Control - RBAC

- Simplify authorization management
 - **Subject-role-object** (role-object is persistent) rather than subject-object
 - Roles are created for various job functions
 - Users are assigned roles based on responsibility
- Express organizational policies
 - Separation of duties
 - Define conflicting roles that cannot be executed by the same user
 - Delegation of authority
- Supports
 - Least-privilege
 - SoD
 - Data abstraction

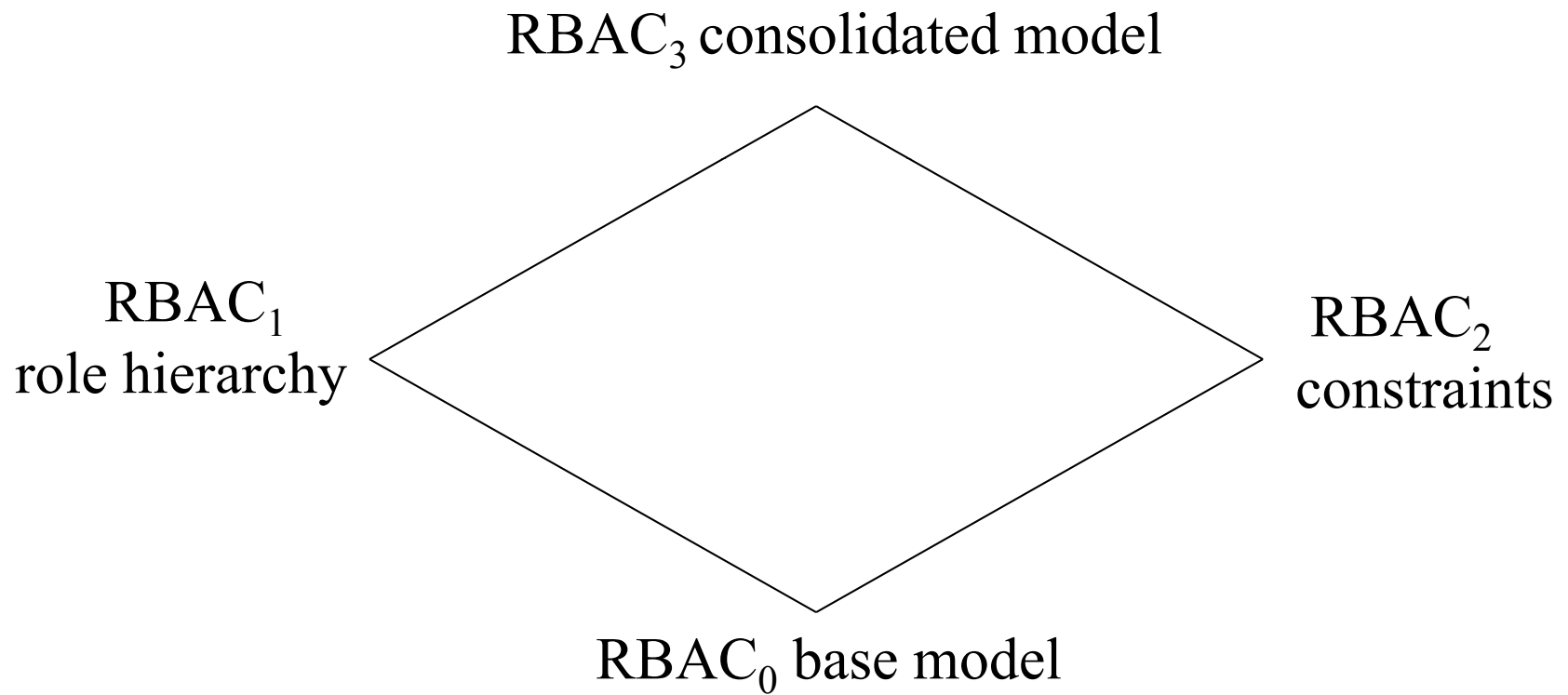
RBAC – Basic Concepts

- **User** – a human being, a machine, a process, or an intelligent autonomous agent, etc.
- **Permission**: Approval of particular mode of access to an object
 - Access modes and objects are domain dependent
 - OS objects: Files, directories, devices, ports; Access: Read, Write, Execute
 - DB objects: Relation, tuple, attribute, views; Access: Insert, Delete, Update
- **Role** – job function within the context of an organization with an associated semantics regarding its authority and responsibility
 - mediator between collection of users and collection of permissions
- **Permission assignment (PA)**: role-permission
- **User assignment (UA)**: user-role
- **Session**: Dynamically activate subset of roles that user is a member of

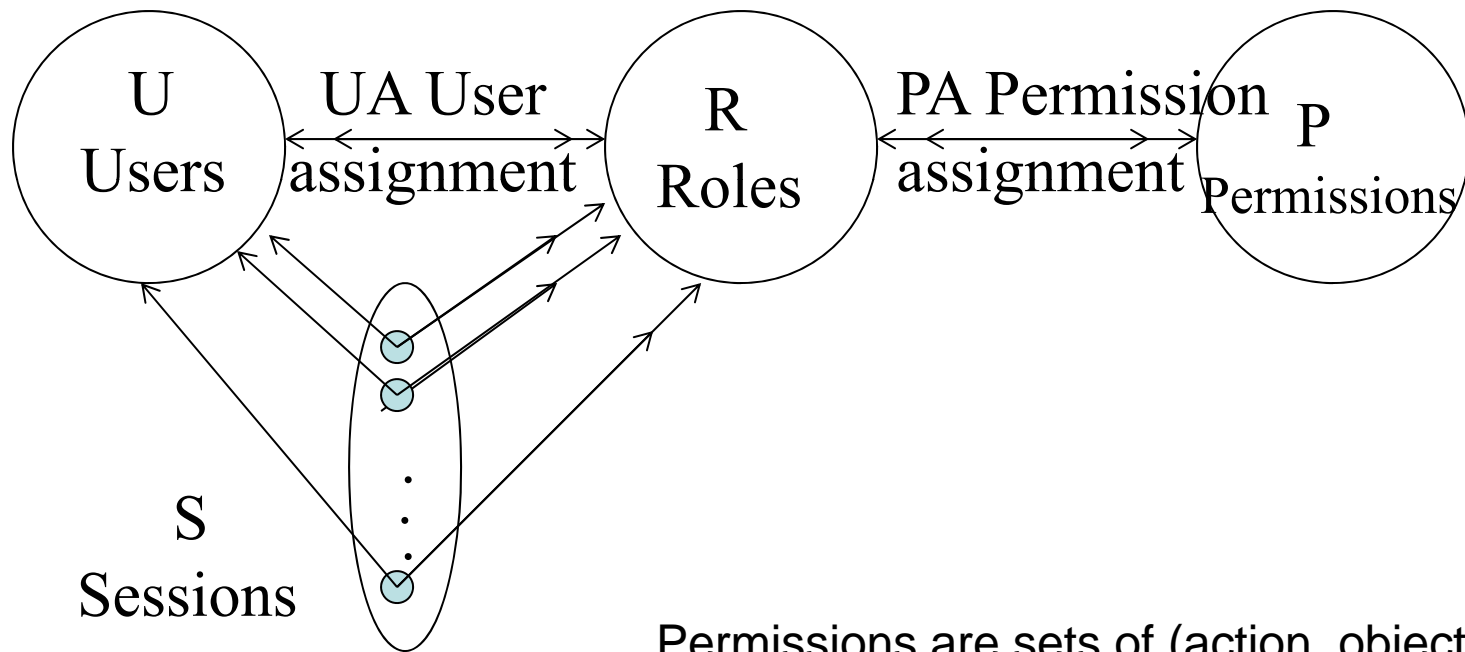
RBAC Models



RBAC



RBAC₀



Permissions are sets of (action, object) pairs, e.g., (read, Table1), (write, Table2), etc.

RBAC₀

- UA: user assignments
 - Many-to-many
- PA: Permission assignment
 - Many-to-many mapping
- Session: mapping of a user to possibly many roles
 - Multiple roles can be activated simultaneously
 - Permissions: union of permissions from all roles
 - Each session is associated with a single user
 - User may have multiple sessions at the same time

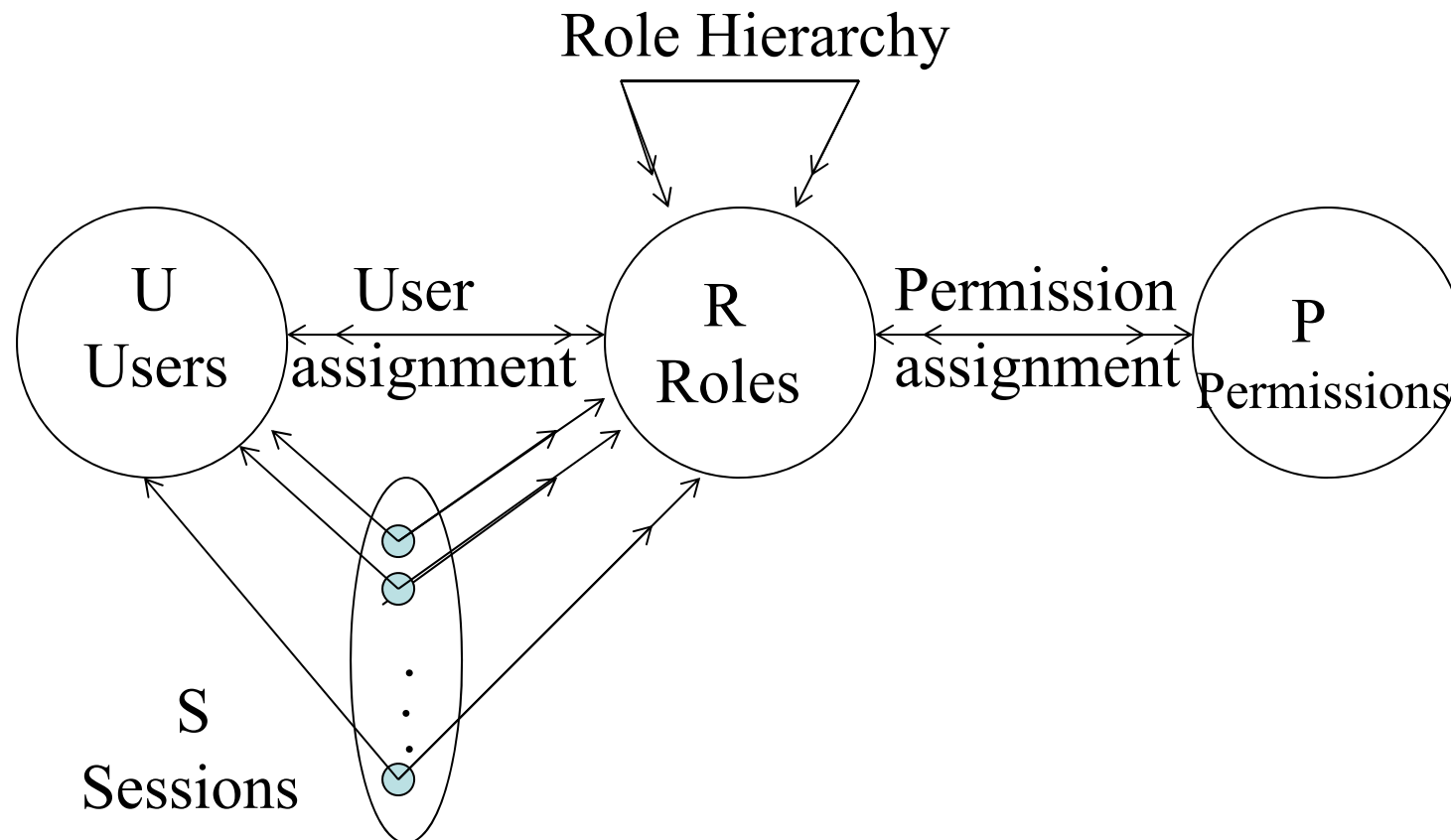
RBAC₀ Components

- **Users, Roles, Permissions, Sessions**
- $PA \subseteq P \times R$ (many-to-many)
- $UA \subseteq U \times R$ (many-to-many)
- $user: S \rightarrow U$, mapping each session s_i to a single user $user(s_i)$
- $roles: S \rightarrow 2^R$, mapping each session s_i to a set of roles $roles(s_i) \subseteq \{r \mid (user(s_i), r) \in UA\}$ and s_i has permissions $\cup_{r \in roles(s_i)} \{p \mid (p, r) \in PA\}$

RBAC₀

- Permissions apply to data and resource objects only
- Permissions do NOT apply to RBAC components
- Administrative permissions: modify U,R,S,P
- Session: under the control of user to
 - Activate any subset of permitted roles
 - Change roles within a session

RBAC₁ – RBAC₀ + Role Hierarchy



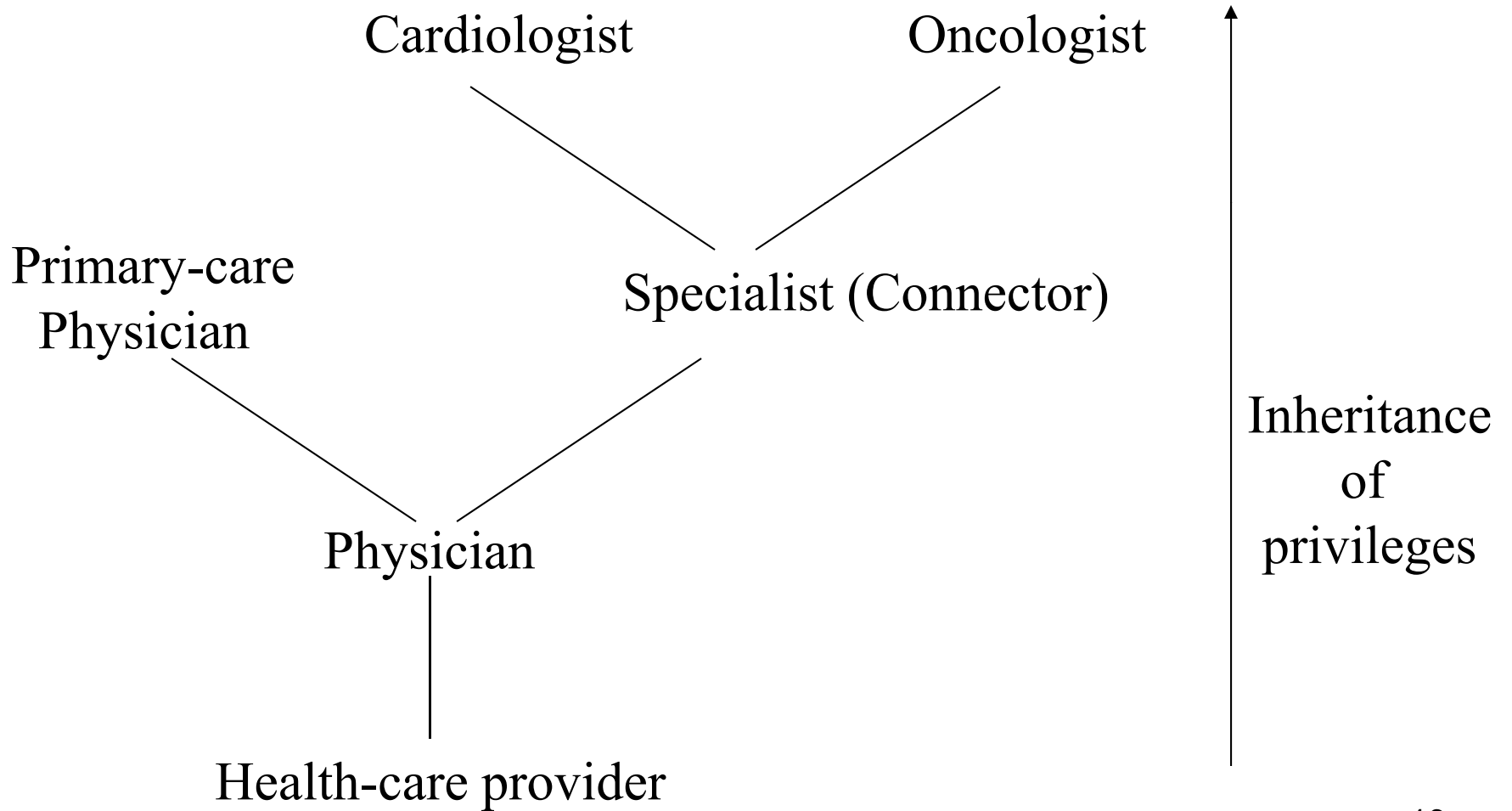
RBAC₁

- Role hierarchies for structuring roles to reflect an organization's line of authority and responsibility
- Inheritance of permission from junior role (bottom) to senior role (top)
- Partial order
 - Reflexive
 - Transitive
 - Anti-symmetric

RBAC₁ Components

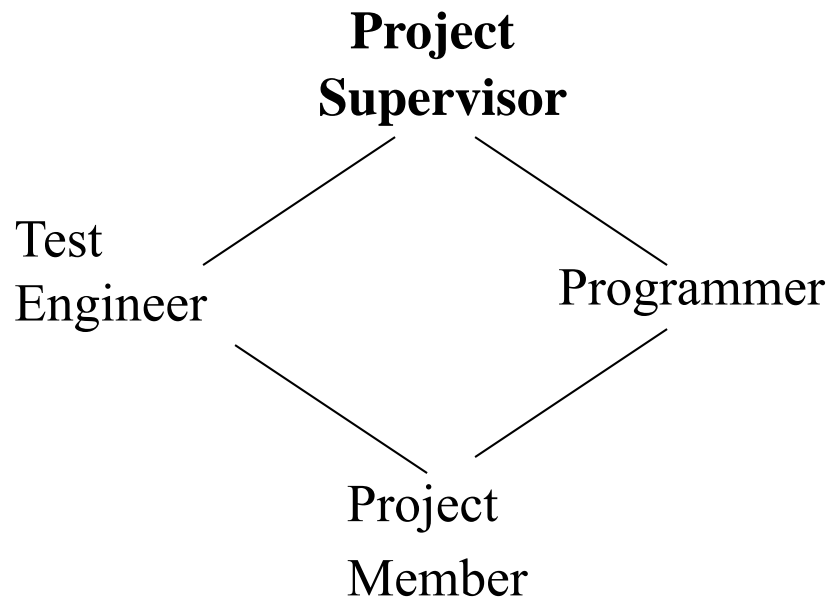
- Same as RBAC₀: **Users, Roles, Permissions, Sessions**, $PA \subseteq P \times R$, $UA \subseteq U \times R$, $\text{user}: S \rightarrow U$, mapping each session s_i to a single user $\text{user}(s_i)$
- $RH \subseteq R \times R$, partial order (\geq dominance)
- $\text{roles}: S \rightarrow 2^R$, mapping each session s_i to a set of roles $\text{roles}(s_i) \subseteq \{r \mid (\exists r' \geq r) [(\text{user}(s_i), r') \in UA]\}$ and s_i has permissions $\cup_{r \in \text{roles}(s_i)} \{p \mid (\exists r'' \leq r) [(p, r'') \in PA]\}$

RBAC₁: Role Hierarchy

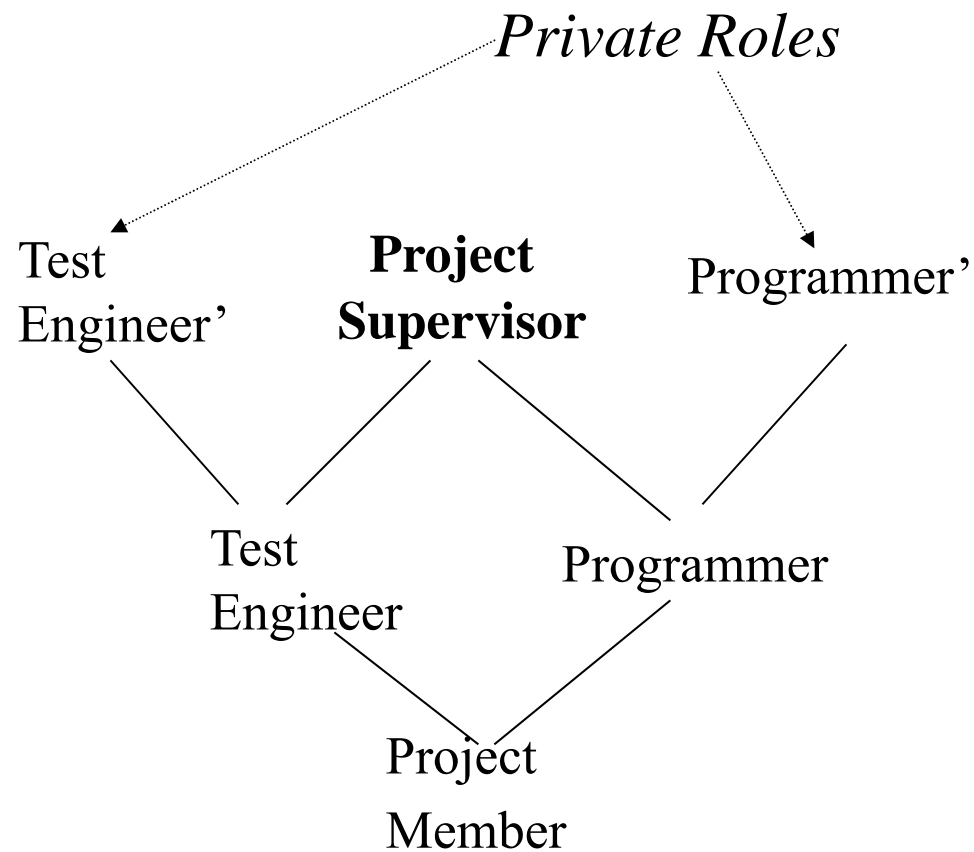


How to limit the scope of inheritance?

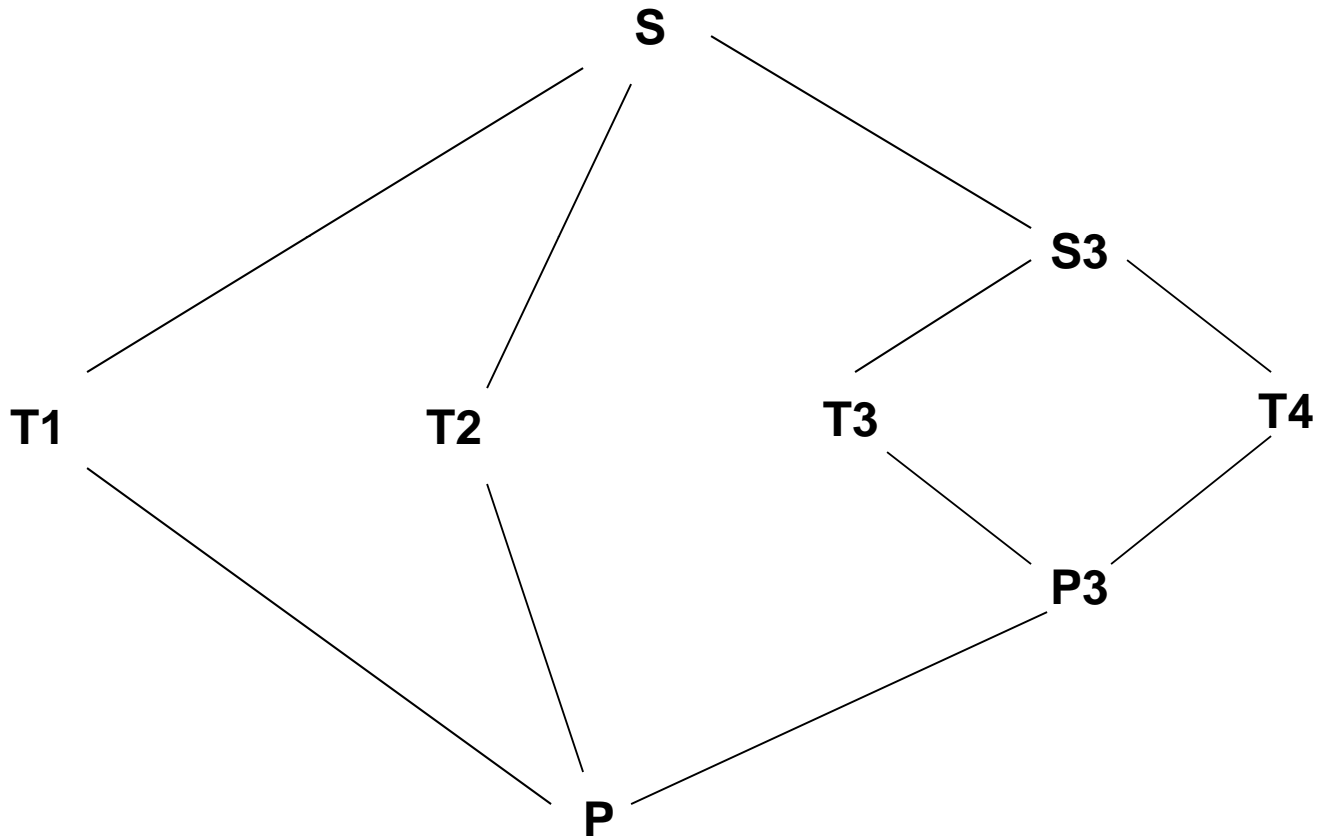
- E.g. do not let boss see incomplete work in progress?



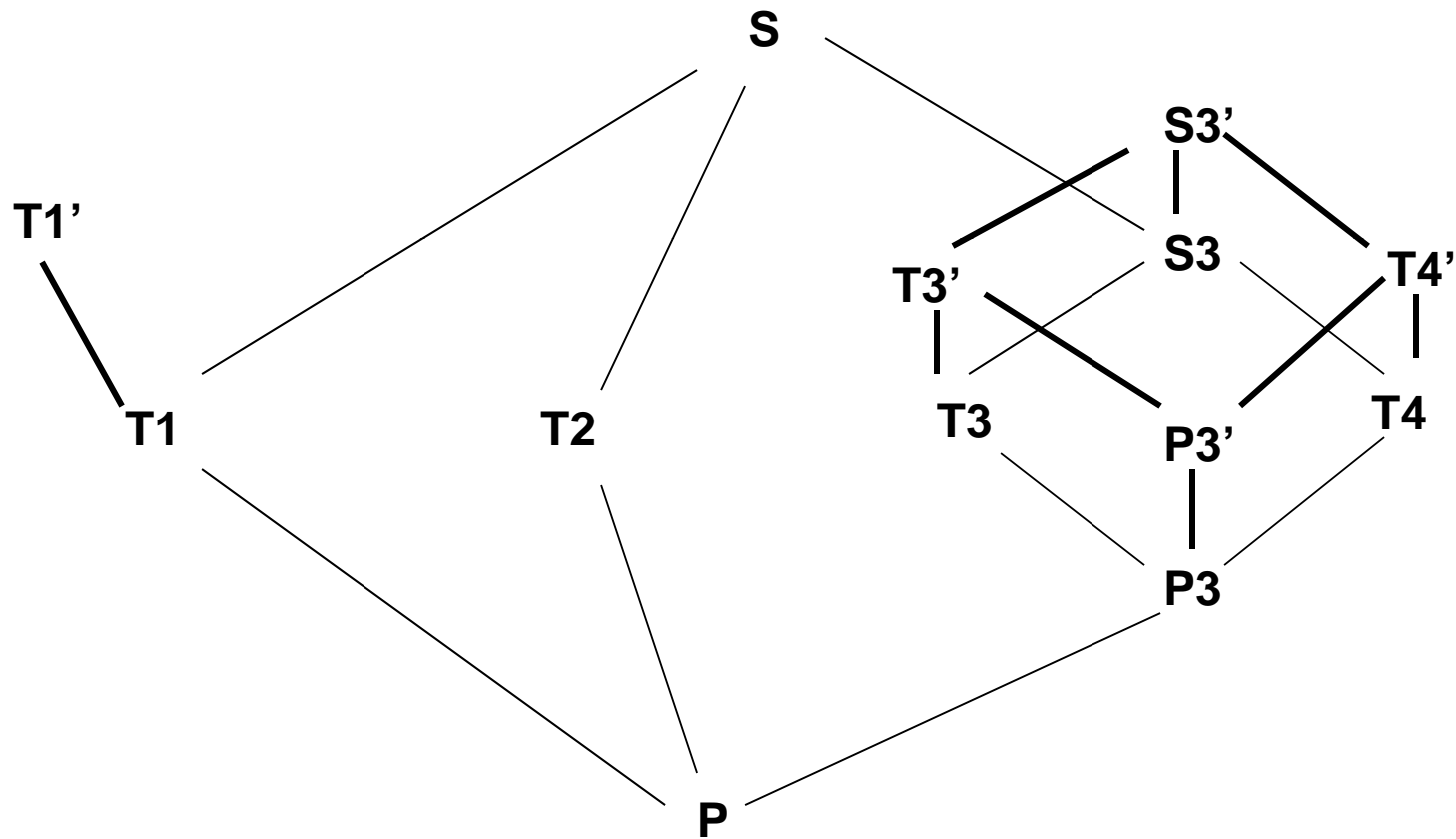
RBAC₁ – Limit Scope of Inheritance



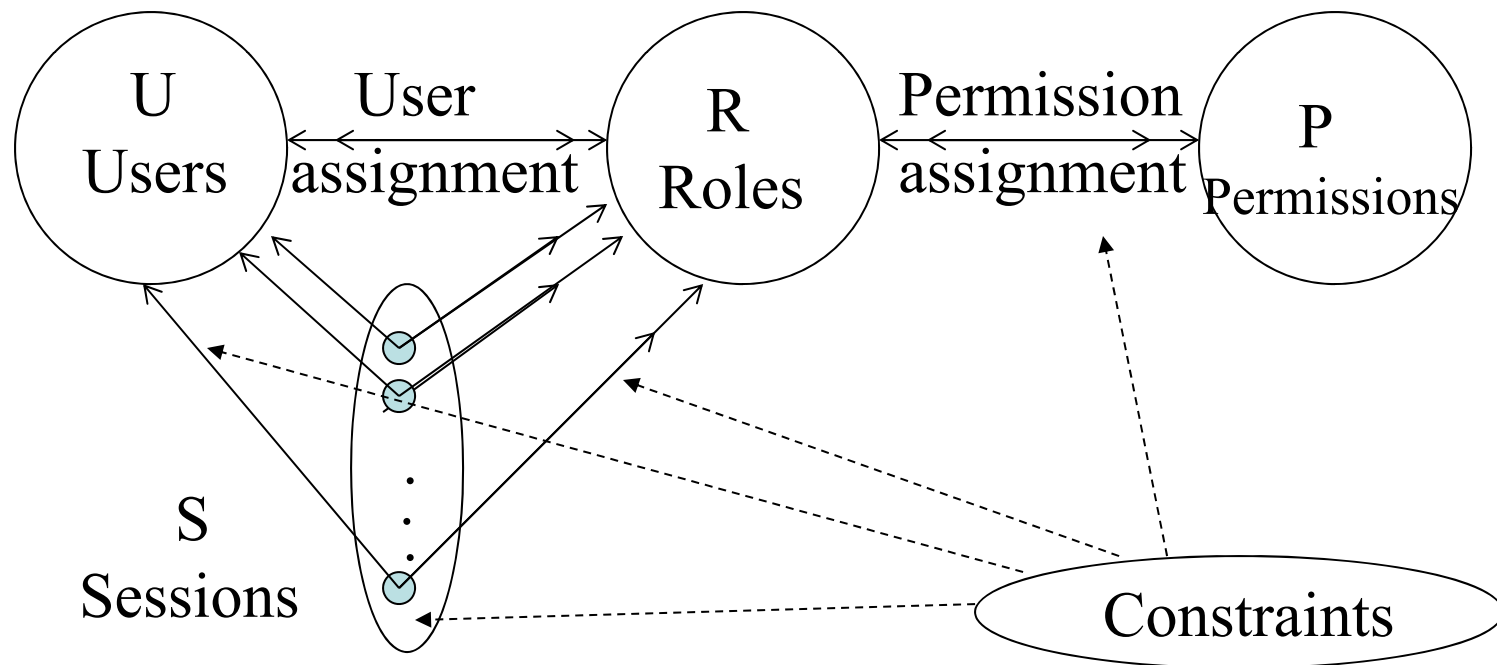
Role Hierarchies with Private Roles



Role Hierarchies with Private Roles



RBAC₂ – RBAC₀ + Constraints



RBAC₂ – RBAC₀ + Constraints

- Enforce high-level organizational policies
 - Mutually disjoint roles: Separation of duties
 - UA: Same user cannot be both accounts manager and purchasing manager
 - Violation is caused only as a result of collusion
 - Dual constraint of permission assignment
 - PA: Permission to issue checks cannot be assigned to both accounts & purchasing managers (**limit distribution of powerful permissions**)
 - Cardinality:
 - A role can have maximum number of members
 - Maximum number of roles to each user
 - Any problem in enforcing minimum number?
 - Can also apply to PA
 - Others: Limit number of roles at runtime (per session) or based on history or pre-requisite (e.g., user can only be assigned to the testing role if assigned to project role already; permission to read a file is assigned to a role if permission has been granted to read the directory)
- Any problem if one user has multiple user ids?

RBAC – Static SoD Constraints

- SSoD places restrictions on the set of roles.
- No user is assigned to t or more roles in a set of m roles
- Prevents a person being authorized to use too many roles
- These constraints can be enforced based on the users assigned to each role

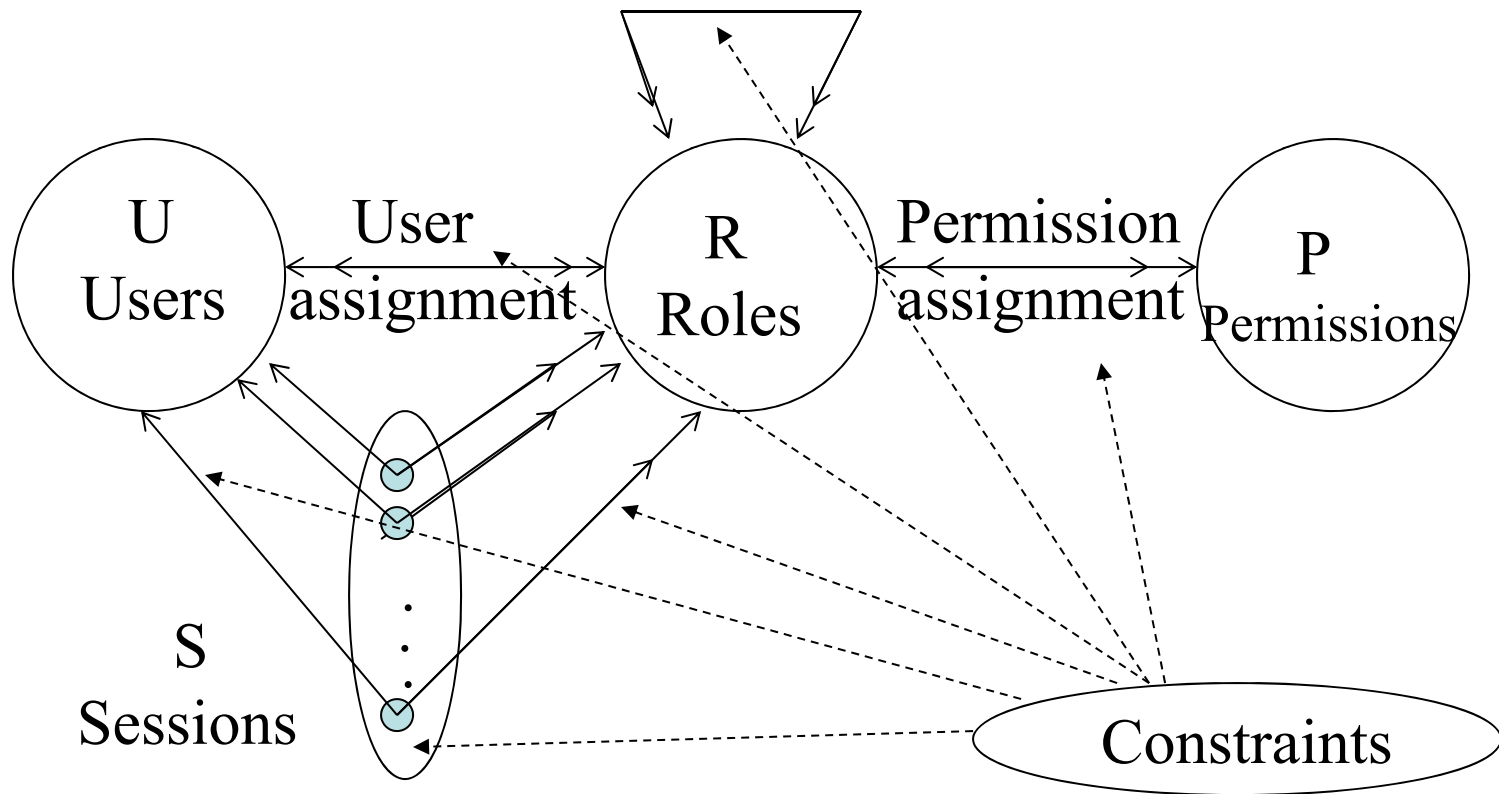
RBAC – Dynamic SoD Constraints

- These constraints limit the number of roles a user can activate in a single session
- Examples of constraints:
 - No user may activate t or more roles from the roles set in each user session.
 - If a user has used role $r1$ in a session, he/she cannot use role $r2$ in the same session
 - What if user terminates one session in one role and logs in with another role?
- Enforcement of these roles requires keeping the history of the user access to roles within a session

RBAC₂

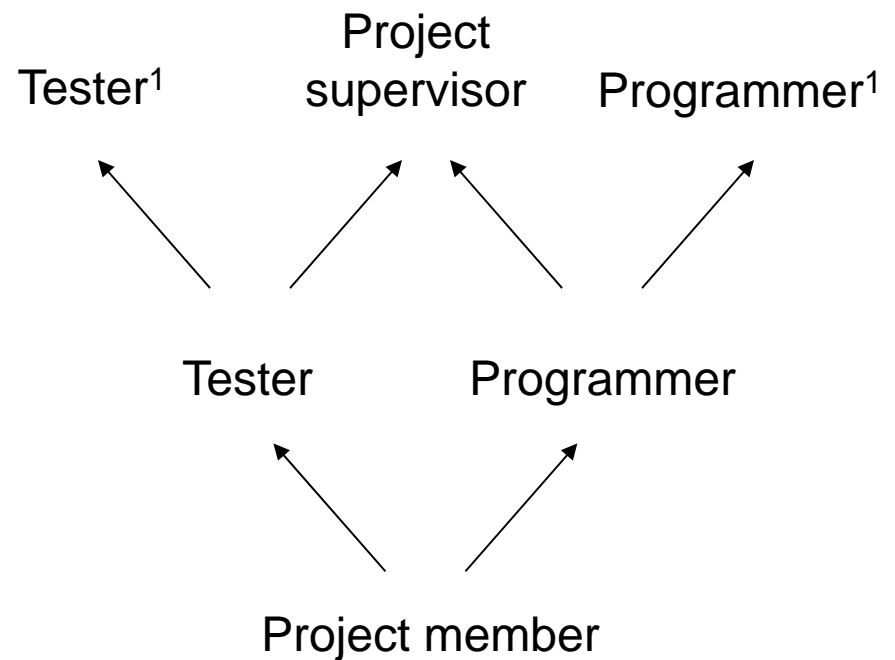
- How to implement role hierarchy with constraints?
 - Specify a constraint that a permission assigned to a (junior) role must also be assigned to an inherited (senior) role
 - Specify a constraint that a user assigned to a (senior) role must also be assigned to any parent (junior) role
- RBAC₁ is redundant (?)

RBAC₃ – RBAC₁ + RBAC₂

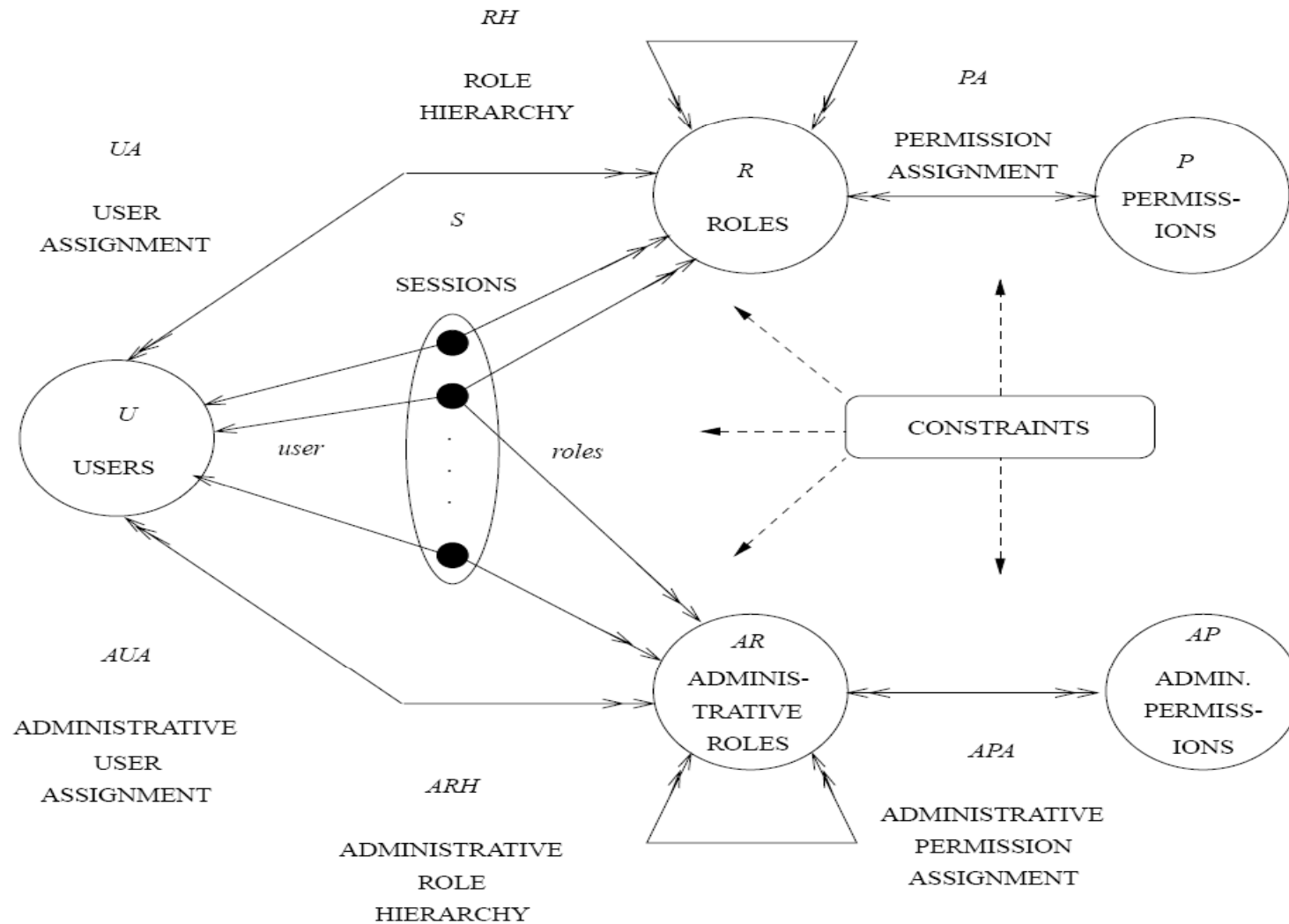


RBAC₃ – RBAC₁ + RBAC₂

- Constraints can apply to role hierarchy
 - E.g. 2 or more roles cannot have common senior/junior role
 - E.g. limit the number of senior/junior roles that a given role may have
- Interactions between RH and constraints
 - E.g. Programmer & tester are mutually exclusive. Project supervisor inherits both sets of permissions. How?
 - E.g., Cardinality constraint – a user can be assigned to at most one role. How about Tester? Do cardinality constraint applies to only direct membership or they also carry on to inherited membership?



RBAC Models (+ Administrative Roles)



RBAC System and Administrative Functional Specification

- Administrative Operations
 - Create, Delete, Maintain elements and relations
- Administrative Reviews
 - Query operations
- System Level Functions
 - Creation of user sessions
 - Role activation/deactivation
 - Constraint enforcement
 - Access Decision Calculation

Case Study: Oracle Enterprise Server

- Create password-protected role for update
 - Create role update_role identified by passwd;
- Grant update privileges to protected role
 - Grant insert, update on app.table1 to update_role;
- Create non-password protected role for query
 - Create role query_role;
- Grant select privileges to unprotected role
 - Grant select on app.table1 to query_role;
- Grant both roles to users
 - Grant update_role, query_role to user1;

Case Study: Oracle Enterprise Server

- User1 activates the roles
 - Set role update_role identified by passwd, query_role;
- Set default active role for User1
 - Alter user user1 default role query_role;
- Assignable privileges
 - System: create session, create table, select any table
 - Object:
 - Table: select, update, insert, delete, alter, create index
 - View: select, update, insert, delete
 - Procedures & functions: execute

Comparison of DBMSs

Item	Feature	Informix	Sybase	Oracle
1	Ability for a role grantee to grant that role to other users	Yes	No	Yes
2	Multiple active roles for a user session	No	Yes	Yes
3	Specify a default active role set for a user session	No	Yes	Yes
4	Build a role hierarchy	Yes	Yes	Yes
5	Specify static separation of duty constraints on roles	No	Yes	No
6	Specify dynamic separation of duty constraints on roles	(Yes)	Yes	No
7	Specify maximum or minimum cardinality for role memberships	No	No	No
8	Grant DBMS system privileges to a role	No	Yes	Yes
9	Grant DBMS object privileges to a role	Yes	Yes	Yes

Source: Role-Based Access Control Features in Commercial Database Management Systems, C. Ramaswamy, R. Sandhu

Configuring RBAC to Enforce MAC and DAC

Configuring RBAC for MAC

- *Construction (Liberal *-Property) (write-up)*
 $R = \{L_1R. \dots L_nR, L_1W. \dots L_nW\}$ where L_i denote label i

RH which consists of two disjoint role hierarchies. The first role hierarchy consists of the “read“ roles $\{L_1R. \dots L_nR\}$ and has the same partial order as \geq_{MAC} ; the second partial consists of the “write” roles $\{L_1W. \dots L_nW\}$ and has a partial order which is the inverse of \geq_{MAC} .

$P = \{ (o,r),(o,w) \mid o \text{ is an object in the system} \}$

Constraint on UA : Each user is assigned to exactly two roles xR and LW where x is the label assigned to the user and LW is the write role corresponding to the lowermost security level according to \geq_{MAC}

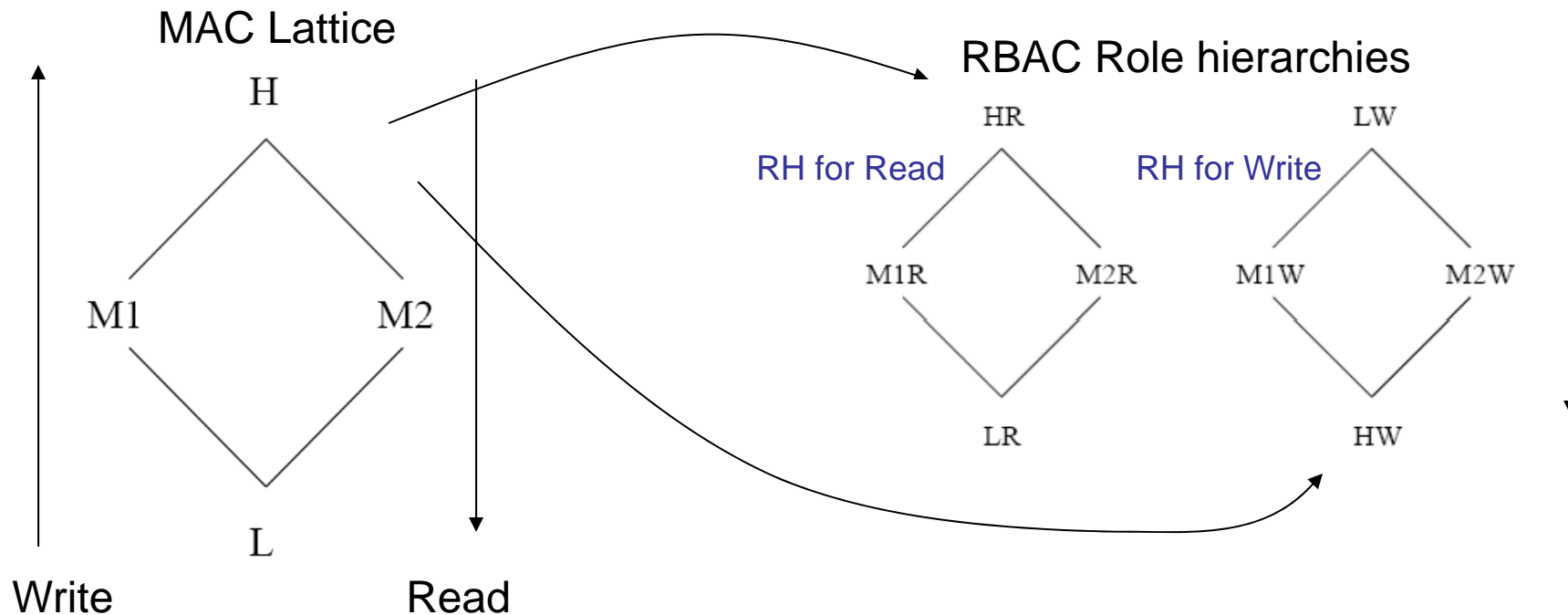
Constraint on sessions: Each session has exactly two roles yR and yW ($x \geq y$)

Constraints on PA :

(o,r) is assigned to xR iff (o,w) is assigned to xW

(o,r) is assigned to exactly one role xR such that x is the label of o

Configuring RBAC for MAC

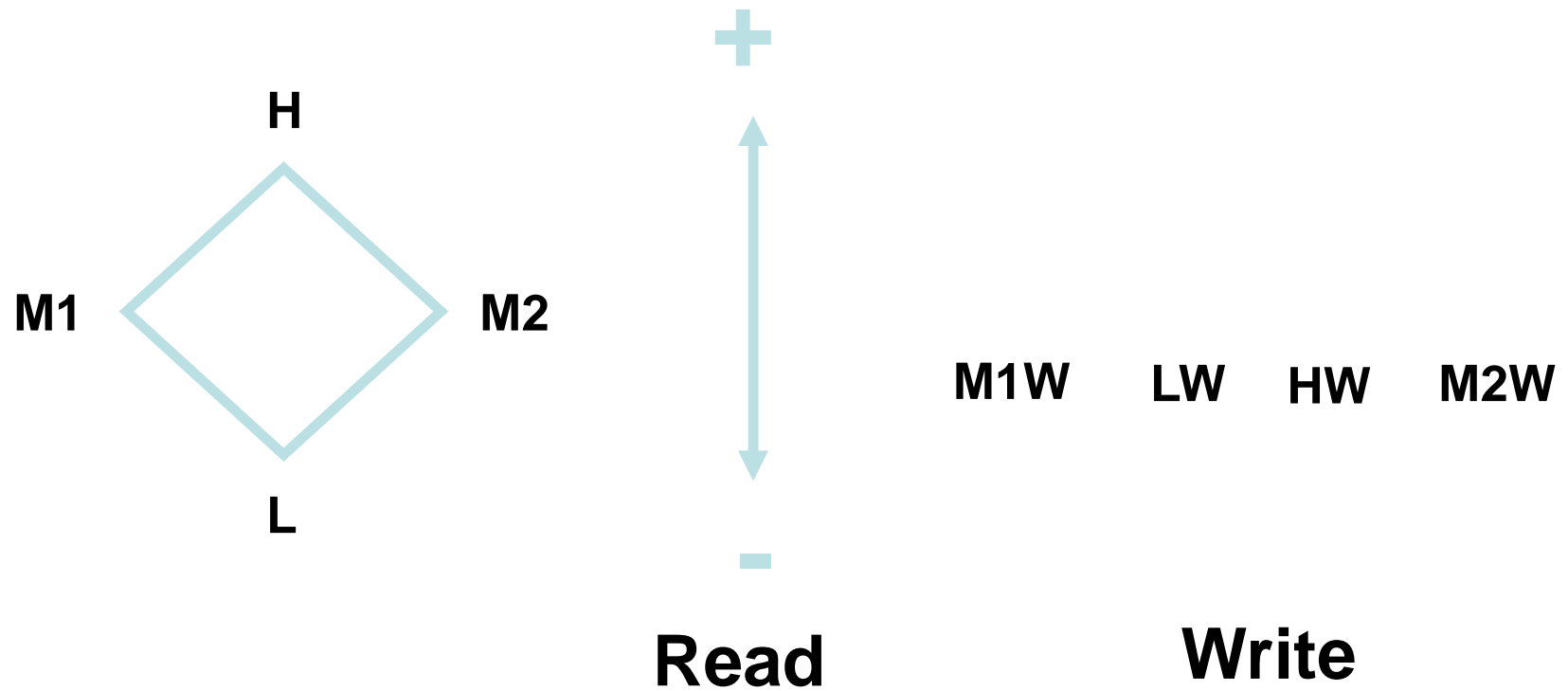


Each user with label x is assigned roles xR & LW (why?)

Additional Constraints:

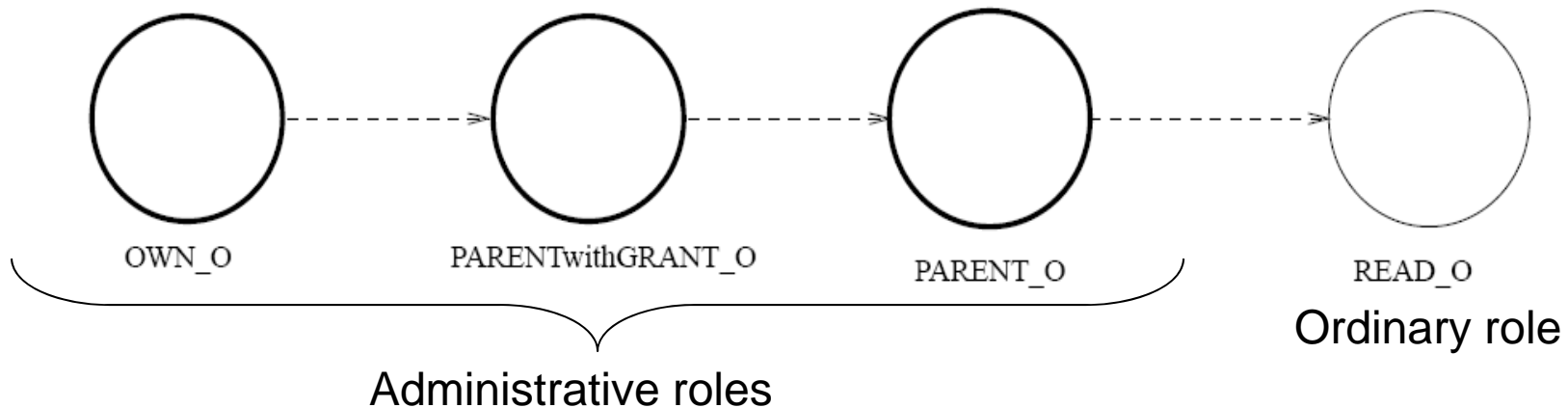
- Each session has exactly two matching roles yR and yW ($x \geq y$)
- For each object with label x , a pair of permissions (o,r) & (o,w) is assigned to exactly one matching pair of xR and xW roles

What about STRICT *-PROPERTY?



Configuring RBAC for DAC

- The basic idea is to simulate the owner-centric policies of DAC using roles that are associated with each object.
 - Strict DAC – only owner can grant access
 - Liberal DAC – owner can delegate discretionary authority for granting access to an object to other users
- *Create an Object.* For every object O that is created, three administrative roles and one regular role are also created (we show only Read operation)



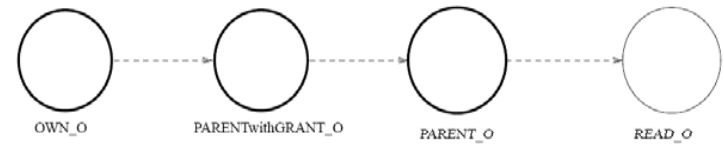
Eight Permissions

- The following eight permissions are also created along with creation of each object O.
 - canRead_O: assigned to the role READ_O (authorizes read operation on object O)
 - destroyObject_O: assigned to the role OWN_O (authorizes deletion of the object)
 - addReadUser_O, deleteReadUser_O: assigned to the role PARENT_O (add/remove users to/from role READ_O)
 - addParent_O, deleteParent_O: assigned to the role PARENTwithGRANT_O (add/remove users to/from role PARENT_O)
 - addParentWithGrant_O, deleteParentWithGrant_O: assigned to the role OWN_O (add/remove users to/from PARENTwithGRANT_O)
- Object deletion removes the roles OWN_O, PARENT_O, PARENTwithGRANT_O and READ_O along with the 8 permissions

Roles and associated Permissions

- **OWN_O**
 - destroyObject_O, addParentWithGrant_O, deleteParentWithgrant_O
- **PARENTwithGRANT_O**
 - addParent_O, deleteParent_O
- **PARENT_O**
 - addReadUser_O, deleteReadUser_O
- **READ_O**
 - canRead_O

Strict DAC

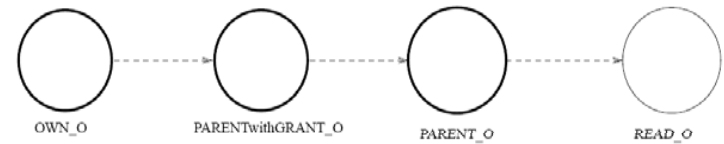


- Only owner has discretionary authority to grant access to an object.
- Example:
 - Alice has created an object (she is owner) and grants access to Bob. Now Bob cannot propagate the access to another user.
- Cardinality constraints on roles:
 - $OWN_O = 1$
 - $PARENT_O = 0$
 - $PARENTwithGRANT_O = 0$
- By virtue of the role hierarchy, owner can change assignments of the role $READ_O$

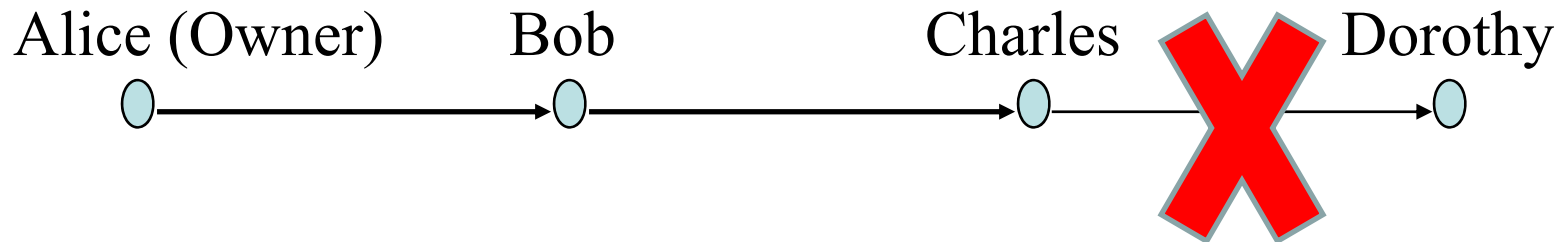
Liberal DAC

- Owner can delegate discretionary authority for granting access to other users.
 - One Level grant
 - Two Level Grant
 - Multilevel Grant

One Level Grant

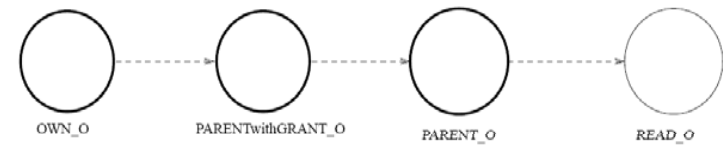


- Owner can delegate authority to another user but they *cannot* further delegate this power.



- Cardinality constraints as:
 - Role OWN_O = 1
 - Role PARENTwithGRANT_O = 0

Two Level Grant

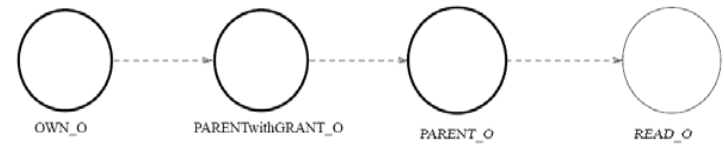


- In addition to a one level grant *the owner* can allow some users to delegate grant authority to other users.



- Cardinality constraints as:
 - Role OWN_O = 1

Multi-Level Grant

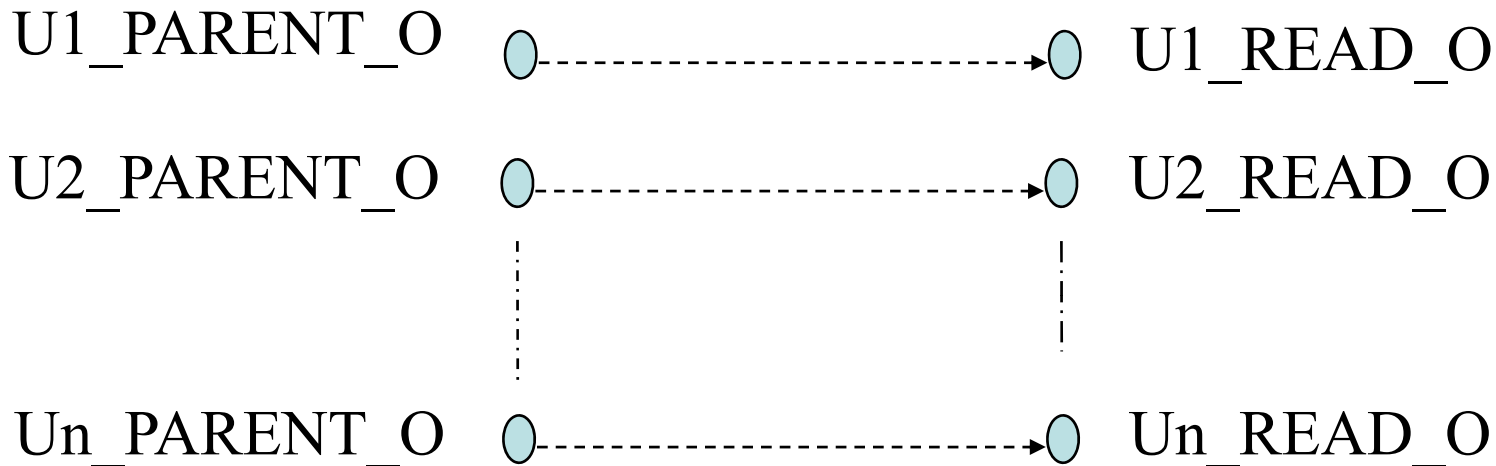


- In addition to a one level grant the owner can allow *some users* to delegate grant authority to other users.
- Cardinality constraints as:
 - Role OWN_O = 1
- Additional permission
 - PARENTwithGRANT_O
 - AddParentWithGrant_O
 - DeleteParentWithGrant_O
 - Grant independent revocation
 - Alternatively, leave delete with OWN_O

Revocation

- Grant-Independent Revocation
 - Grant may be revoked by anyone (not necessarily the granter)
 - Alice grants Bob access, but Bob's access may be revoked by Charles
- Grant-Dependent Revocation
 - Revocation is tied to the granter
 - Alice grants Bob access, and only Alice can revoke Bob's access

Grant-Dependent Revocation (One-level grant)



READ_O role associated with members of PARENT_O

We need a different administrative role U_PARENT_O and a regular role U_READ_O for each user U authorized to do a one-level grant by owner.

We also need two new administrative permissions

- addU_ReadUser_O, deleteU_ReadUser_O: assigned to U_PARENT_O
- authorize the operations to add users to role U_Read_O and delete users from U_Read_O
- cardinality of U_PARENT_O = 1

Summary

- Group is NOT the same as Role
- Role hierarchy is NOT the same as company (report-to) hierarchy
- RBAC can be used to configure DAC and MAC