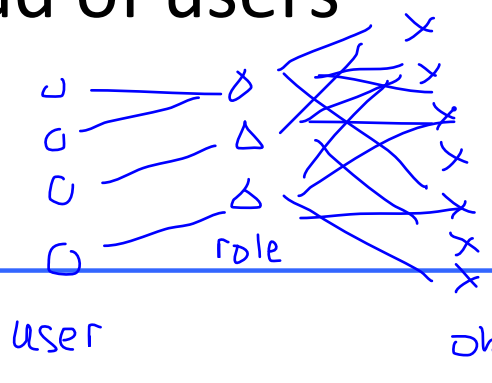
CS5322 Database Security

Role-Based Access Control: Motivation

- In discretionary access controls (DAC), each user could have access to a different set of objects
- Problems:
  - If there are large numbers of users and objects, the total number of authorizations can be extremely large
  - For a dynamic user population, the amount of grant and revoke operations can be difficult to manage

Role-Based Access Control: Motivation

- Observations:
  - Users can be categorized based on their *roles*
  - Users taking up the same role often have the same access rights
- Idea: Add a layer of *roles* between users and objects
  - Assign access rights to roles instead of users
  - Assign roles to users

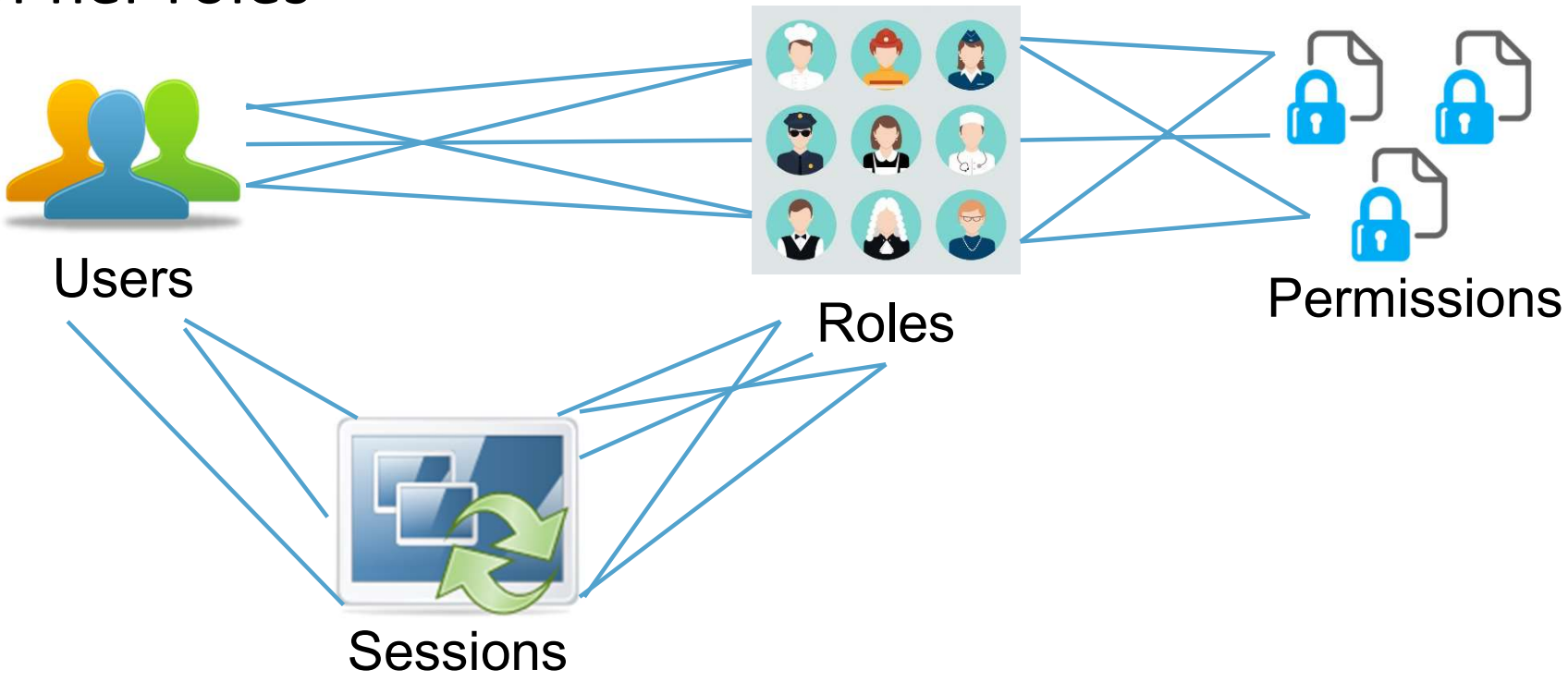


Role-Based Access Control: Motivation

- Idea: Add a layer of *roles* between users and objects
  - Assign access rights to roles instead of users
  - Assign roles to users
- Benefits
  - The amount of authorizations could be reduced since the number of roles is usually much smaller than the number of users
  - Access rights for roles are relatively stable, i.e., the system does not need to frequently change the authorizations for roles
  - Handling a dynamic set of users only requires assigning/re-assigning users to roles, without changing the access rights of roles

Role-Based Access Control: Concepts

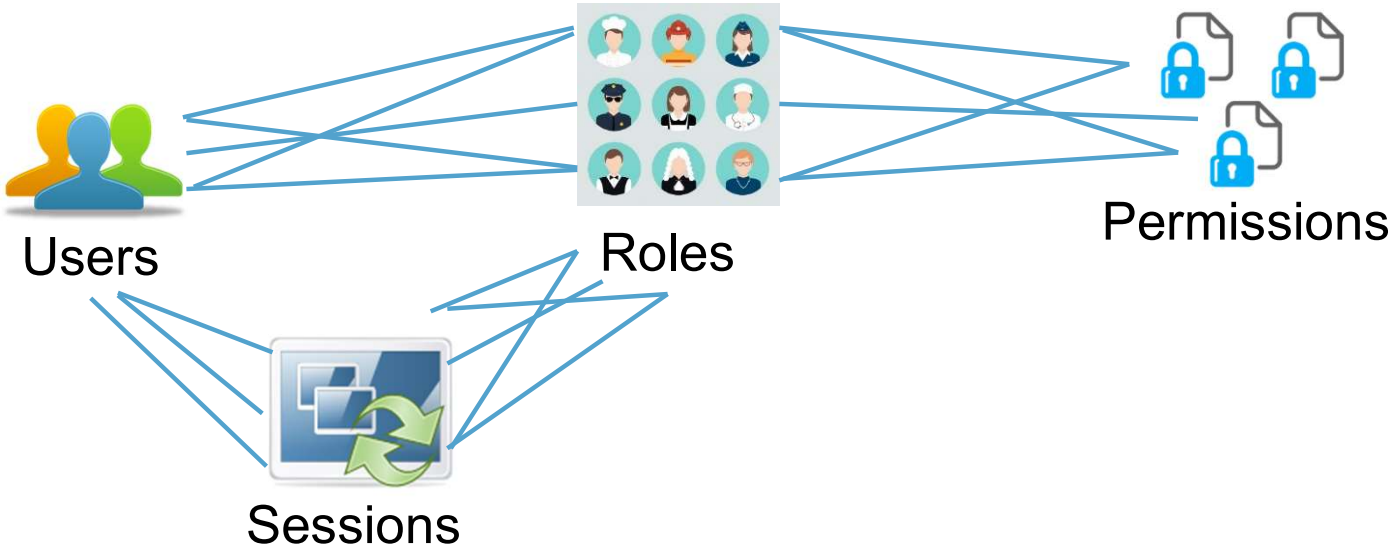
- Users
- Roles
- Permissions: Access rights to objects
- Sessions: A period in which a user activates some or all of her roles



Role-Based Access Control (RBAC) Models

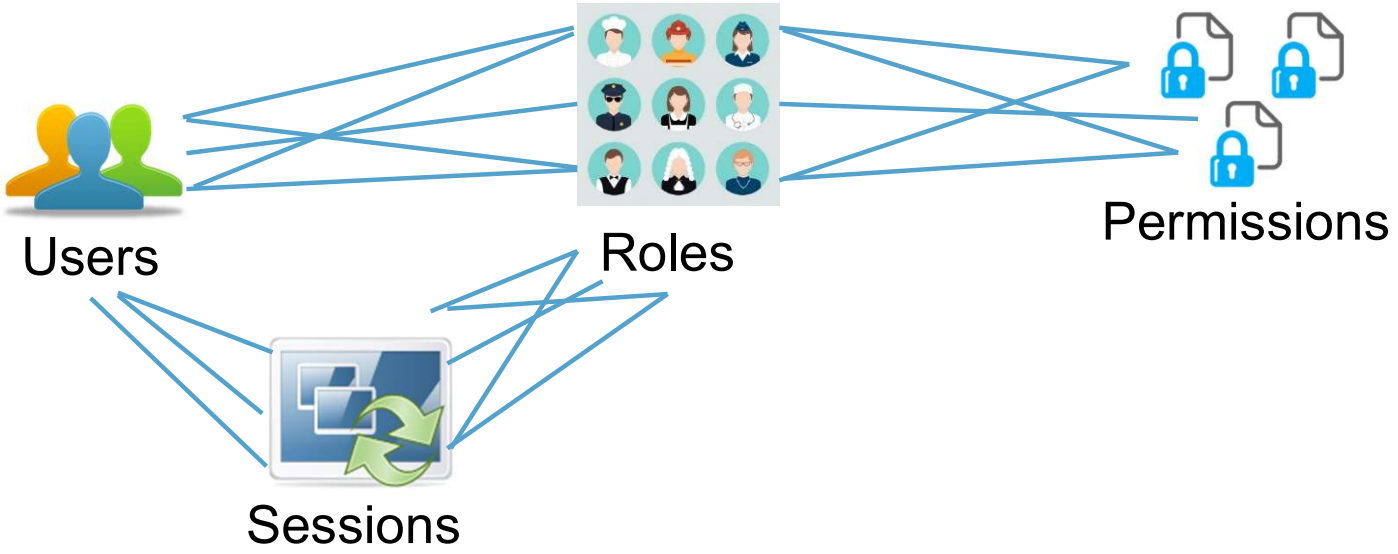
- $RBAC_0$, $RBAC_1$, $RBAC_2$, $RBAC_3$
  - Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, Charles E. Youman: Role-Based Access Control Models. IEEE Computer 29(2): 38-47 (1996)
- NIST RBAC Model
  - David F. Ferraiolo, Ravi S. Sandhu, Serban I. Gavrila, D. Richard Kuhn, Ramaswamy Chandramouli: Proposed NIST standard for role-based access control. ACM Trans. Inf. Syst. Secur. 4(3): 224-274 (2001)

$RBAC_0$



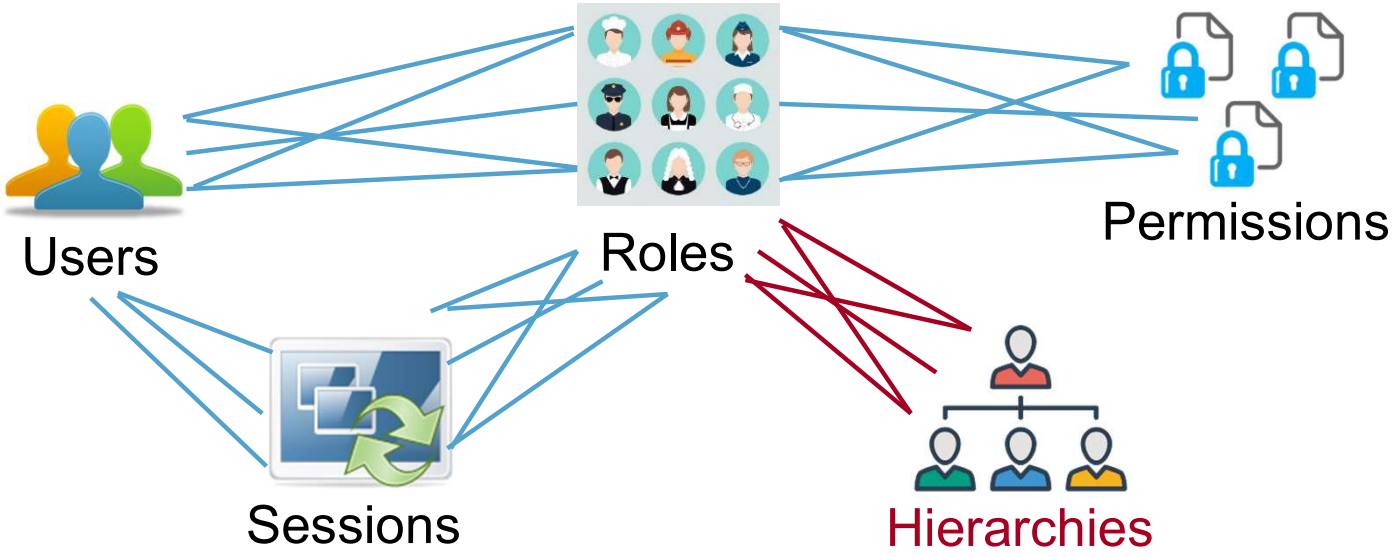
- A basic RBAC model that specifies the relationships among users, roles, permissions, and sessions
- Four relationships: PA, UA, SU, SR
  - PA: Permission assignment
    - Many-to-many
  - UA: User assignment
    - Many-to-many
  - SU: Session-User
    - Many-to-one from sessions to users
  - SR: Session-Role
    - Many-to-many; a session's permissions are the union of all of its corresponding roles' permissions

$RBAC_0$

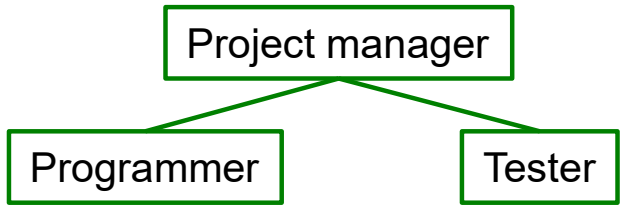


- Permissions are defined as access rights to objects
  - Permissions are set by system administrators
- Rights to modify users, roles, permissions, PA, and UA are referred to as *administrative permissions*
  - These rights are for administrators and not for any roles
- Sessions are set by users
  - A user can activate any subset of her roles in a session
  - A user can change roles within a session

$RBAC_1$

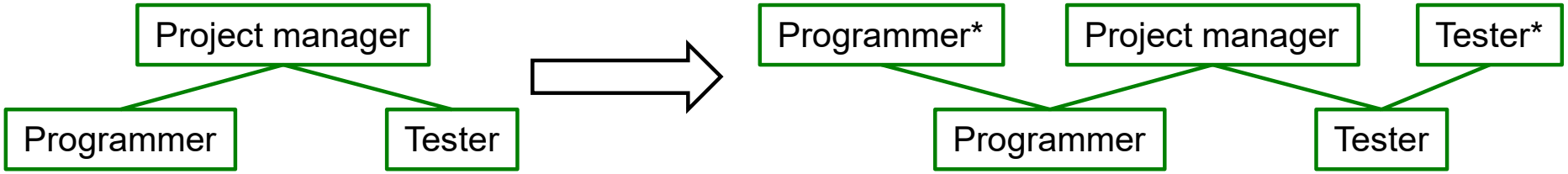


- Improves $RBAC_0$ by incorporating *role hierarchies*
- A role hierarchy RH is a partial order relationship defined over the roles
  - Partial order: reflexive, transitive, anti-symmetric
- Example:



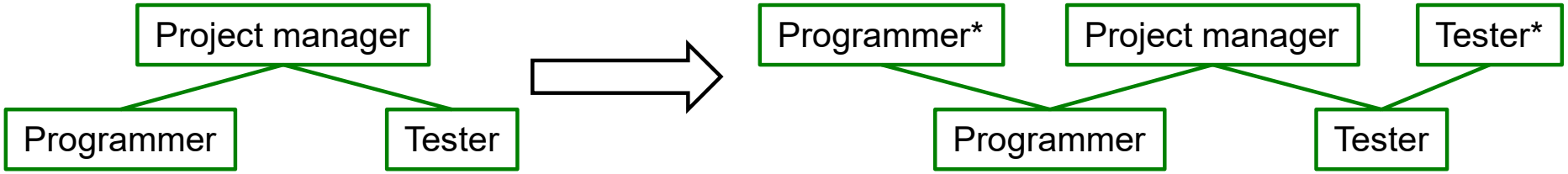
- Rights given to a lower level role will be automatically given to its ancestor roles in the hierarchy, but not vice versa
  - E.g., programmer and tester's rights are automatically given to project manager

$RBAC_1$: Role Hierarchies



- In the above role hierarchy, Project manager can see everything that Programmer and Tester see
- What if we want to
  - allow programmers and testers to hold their "work in progress" in their private space, and
  - let project manager only see the completed work
- Solution: Add two additional roles, Programmer* and Tester*

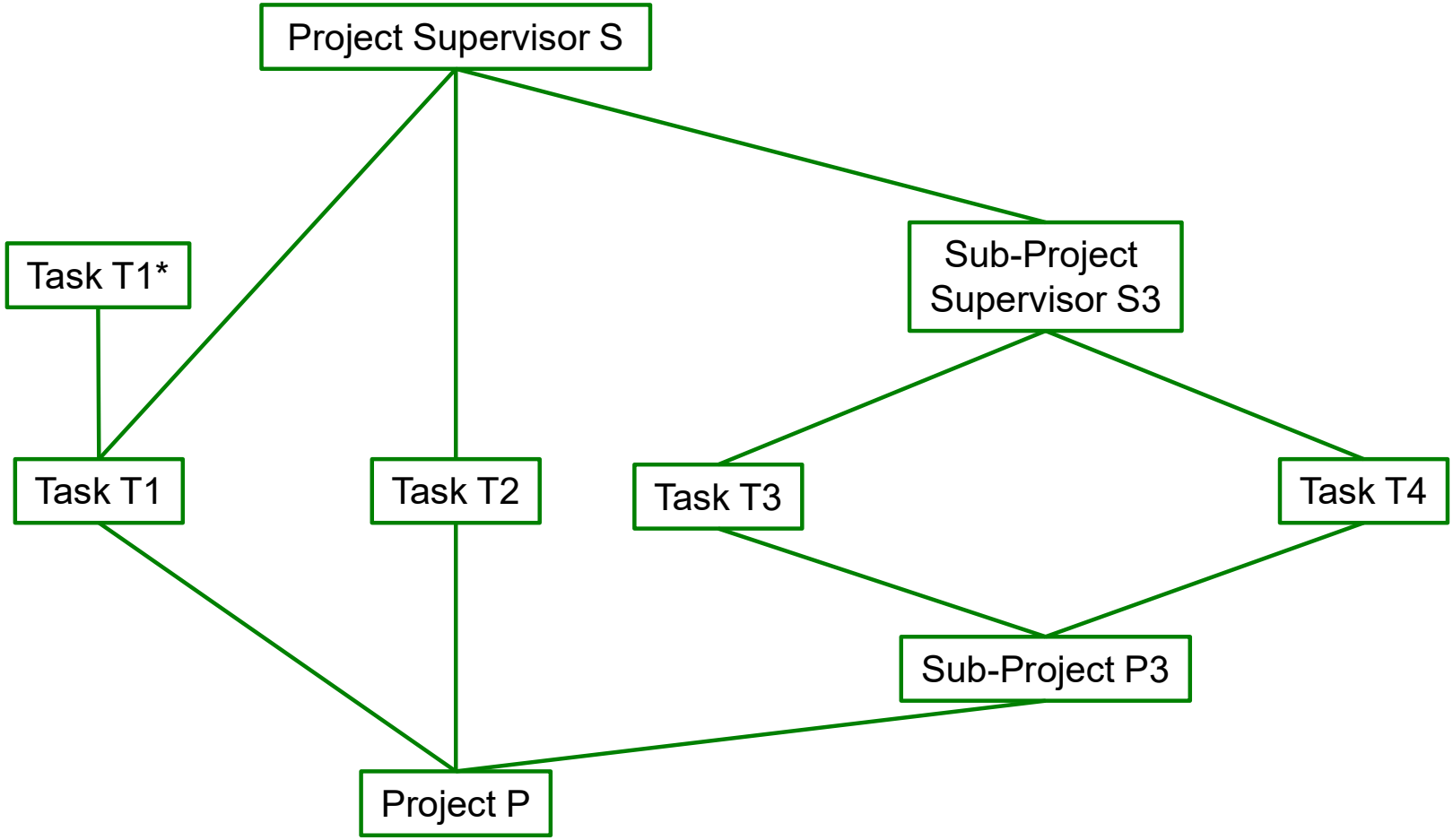
$RBAC_1$: Role Hierarchies



- Solution: Add two additional roles, Programmer* and Tester*
  - Programmer* has access rights to some private objects that are not given to Programmer
  - Hence, Project manager would not obtain those access rights from Programmer*
  - The case for Tester* is similar
- Programmer* and Tester* are referred to as *private roles*

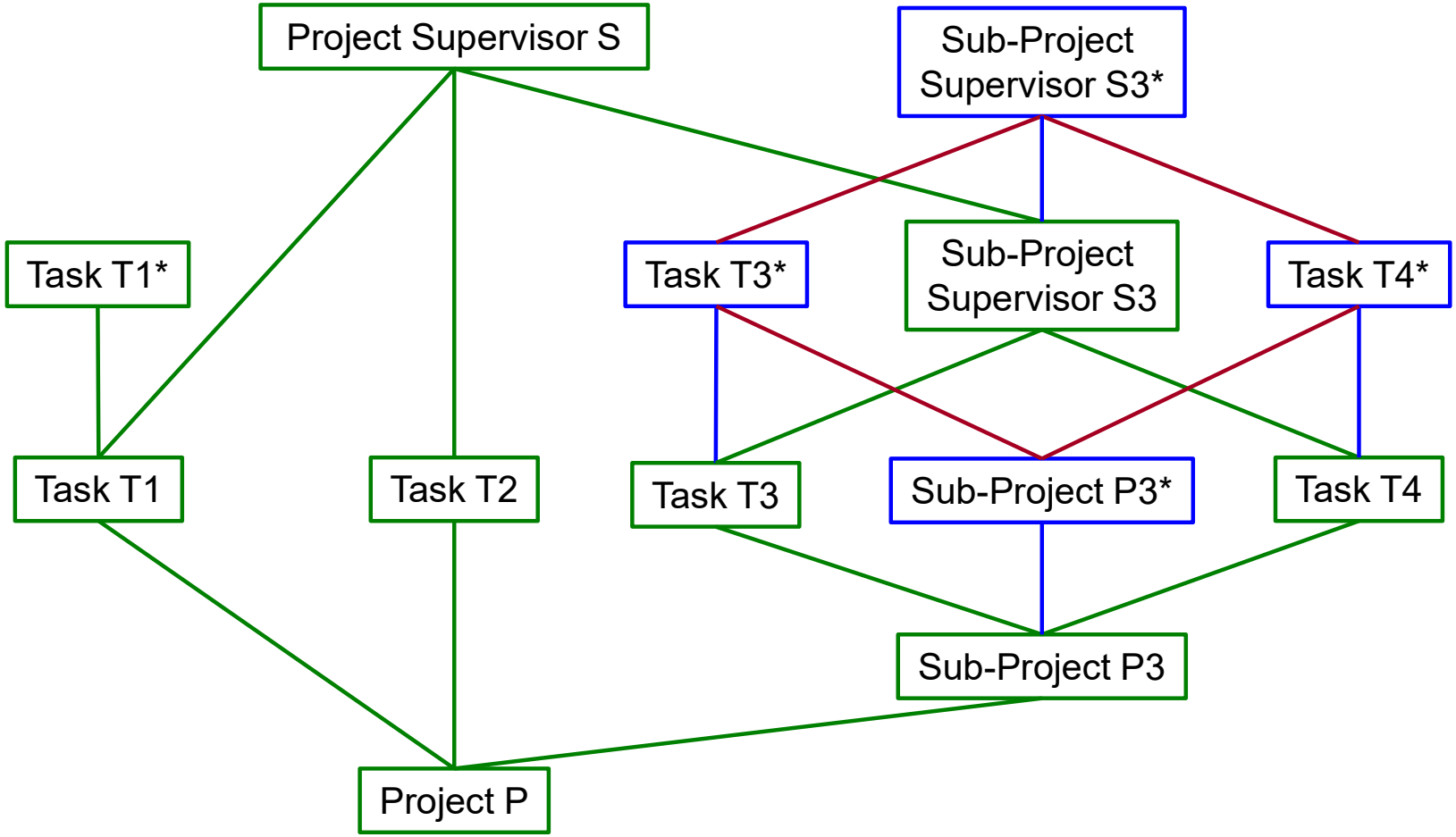
$RBAC_1$: Role Hierarchies

- Private roles can also form a sub-hierarchy among themselves

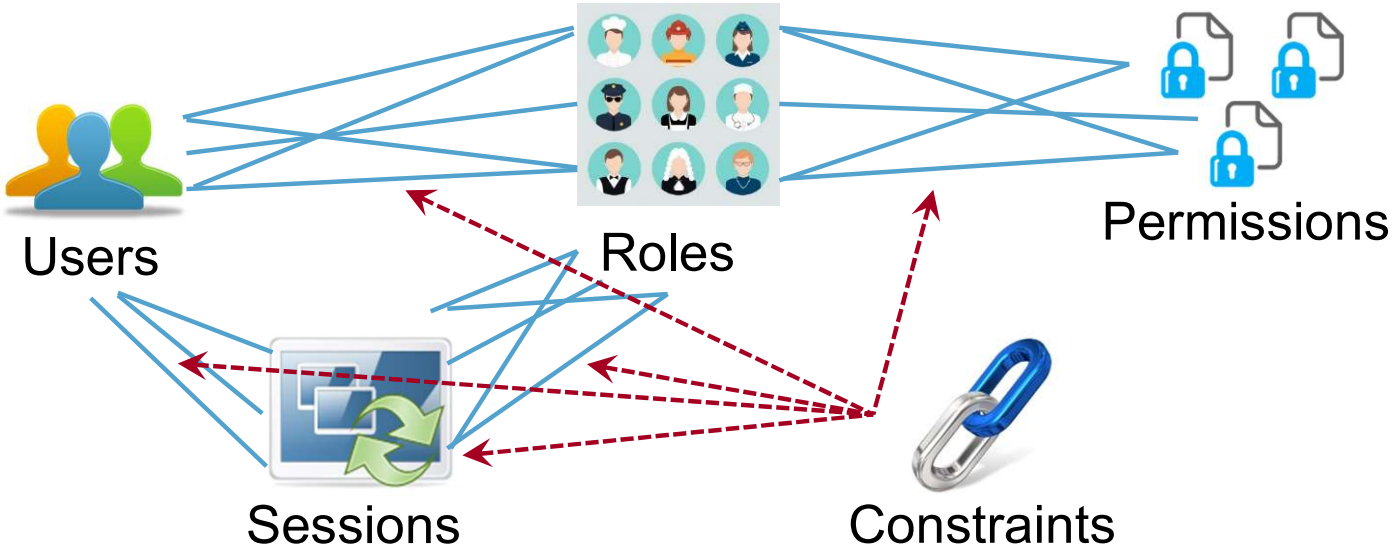


$RBAC_1$: Role Hierarchies

- Private roles can also form a sub-hierarchy among themselves

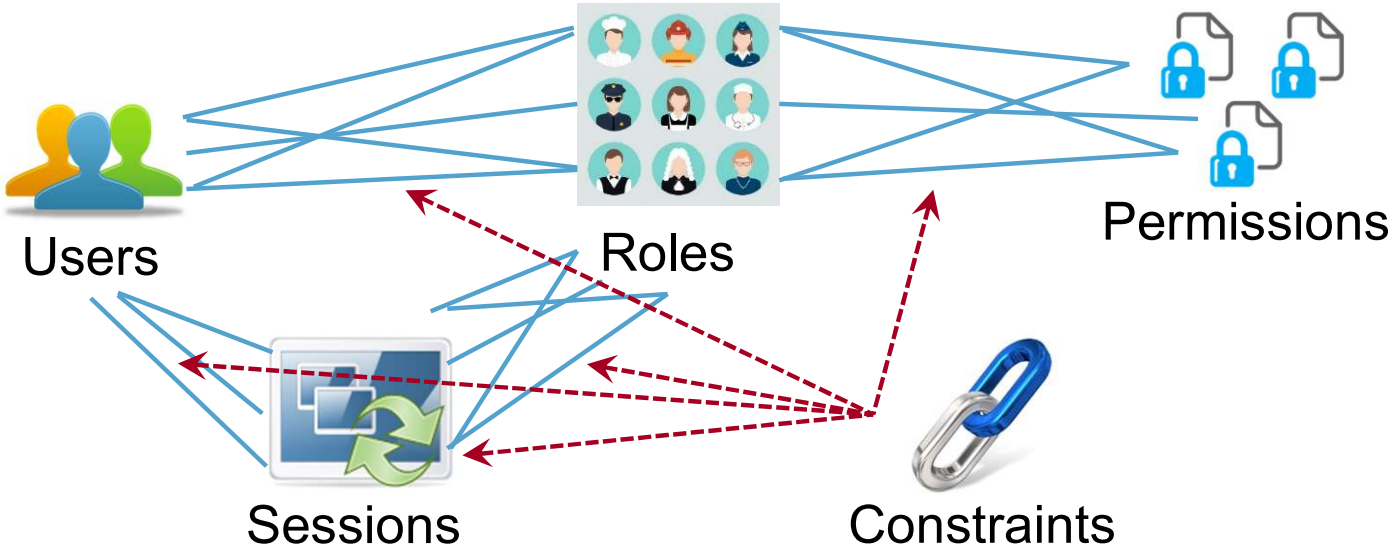


$RBAC_2$



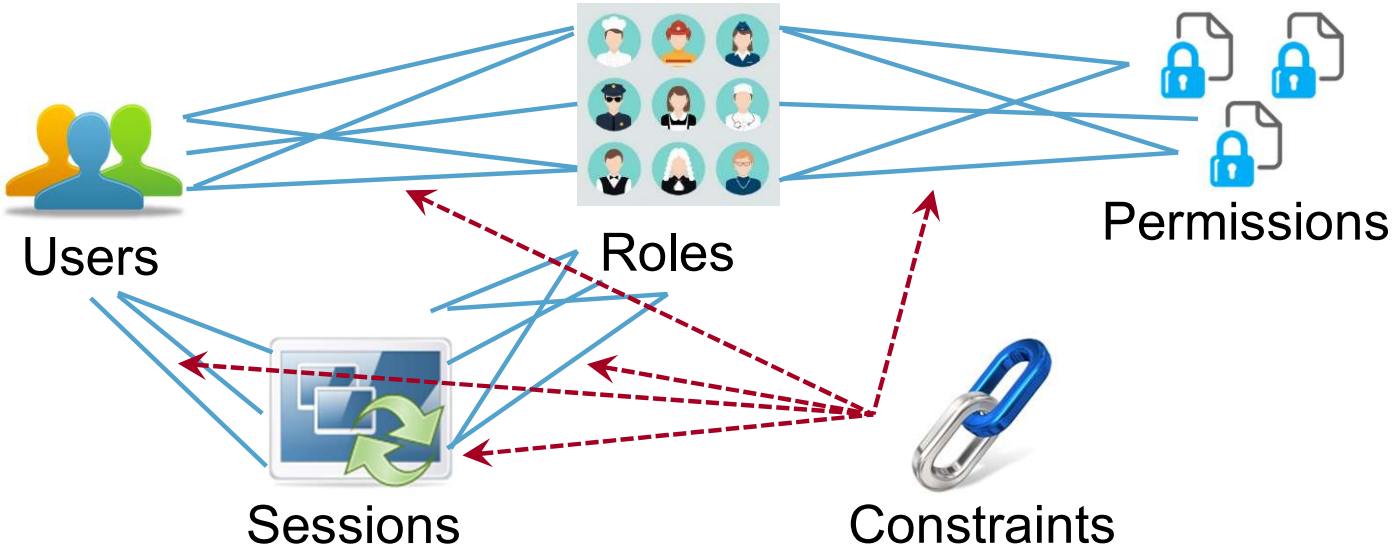
- Improves $RBAC_0$ by incorporating *constraints*
- Constraints may apply on sessions, S-U, S-R, UA, and PA
  - To enforce high-level organizational policies

$RBAC_2$



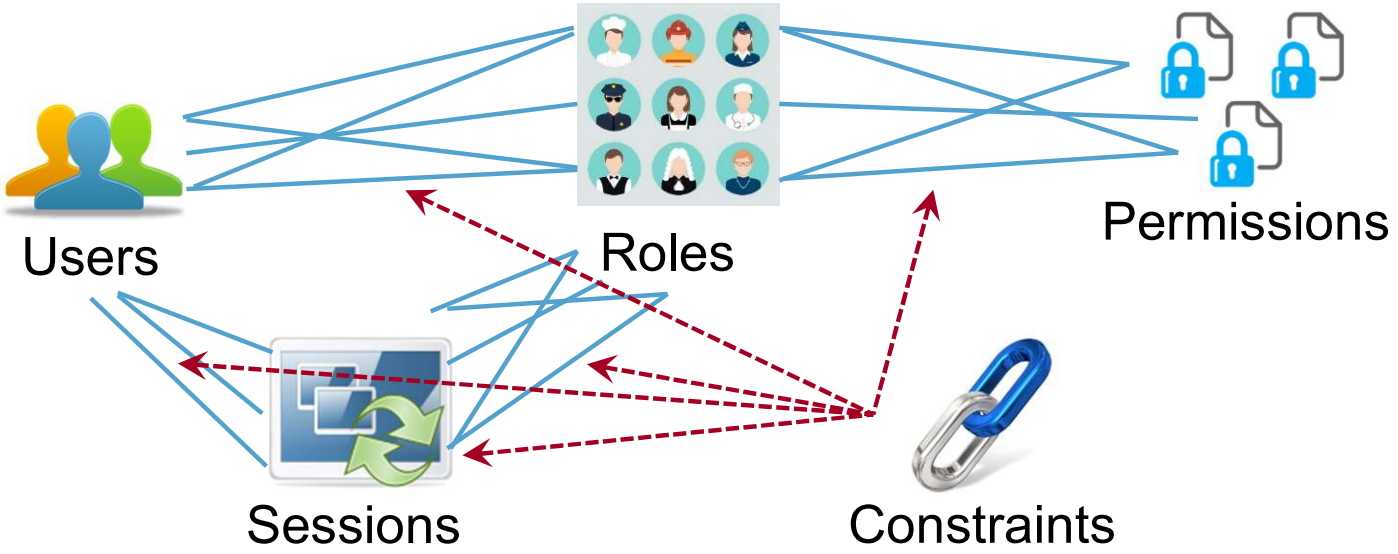
- Examples of constraints:
  - Mutually disjoint roles: separation of duties
    - To avoid collusion, no user can be assign the roles of a purchasing manager and an auditor simultaneously
  - Cardinality:
    - Restrict the maximum number of roles that a user may take
  - Role dependencies:
    - A user can be assigned role A only if she is previously assigned role B

$RBAC_2$



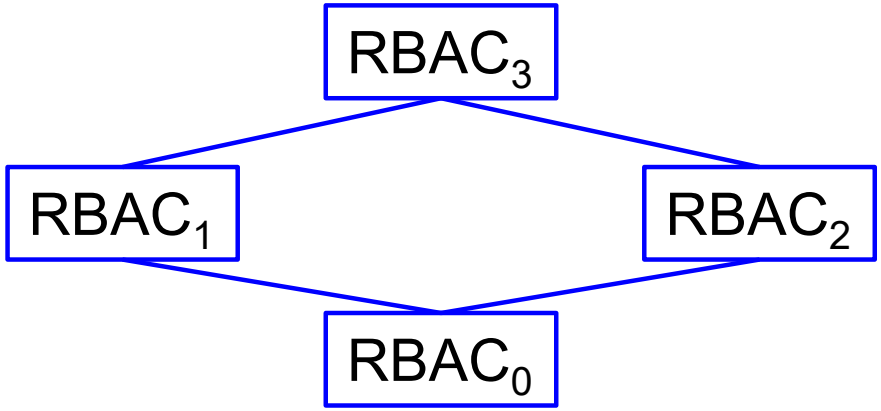
- Observation: Many constraints are based on user ids
- Question: What if the system gives the same user multiple ids?
- Answer: It could defeat the purpose of constraints
- Therefore, for constraints to be effective, the system needs to carefully manage user ids

$RBAC_2$



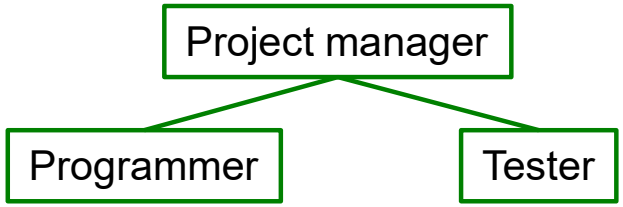
- Question: Can role hierarchies be regarded as constraints?
  - Yes
- If so, why do we need $RBAC_1$?
  - Because role hierarchies are so important that they deserve separate treatment

$RBAC_3$



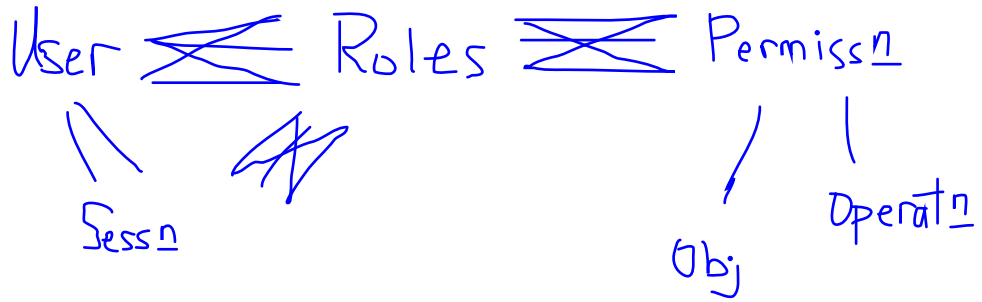
- A combination of $RBAC_1$ and $RBAC_2$
  - I.e., has both role hierarchies and constraints
- Could incorporate constraints on role hierarchies
  - E.g., a user cannot be both a project manager and a programmer for a project
- This could lead to some tricky issues

$RBAC_3$: Tricky Issue



- Suppose that we have the following separation-of-duty (SoD) constraint:
  - A user cannot be a programmer and tester simultaneously for the same project
- But the project manager for the project would have access rights of a programmer and tester simultaneously
- Does this violates the SoD constraint?
- It depends
  - The system should explicitly specify whether or not a role like this is allowed given the SoD constraint

NIST RBAC Model



- Similar to $RBAC_3$, but explicitly defines
  - The mappings from permissions to objects and operations
  - Two types of separation-of-duties (SoD) constraints
    - Static SoD
    - Dynamic SoD

Separation of Duties (SoD) constraints

- Static separation of duties (SSoD)
  - An SSoD constraint has two components, RS and n
    - RS: a set of roles
    - n: a natural number >= 2
  - It requires that no user can be assigned n or more roles in RS
  - Example:
    - RS = {Programmer, Tester, Project manager}
    - n = 2
    - Meaning: those three roles must be assigned to different users

Separation of Duties (SoD) constraints

- Dynamic separation of duties (DSoD)
  - A DSoD constraint has two components, RS and n
    - RS: a set of roles
    - n: a natural number >= 2
  - It requires that no user can activate n or more roles in RS in the same session
  - Example:
    - RS = {Programmer, Tester, Project manager}
    - n = 2
    - Meaning: no user can activate more than one of those roles in the same session
    - But the user can be assigned more than one roles in RS

Exercise

- Consider an SSoD constraint with
  - RS = {r1, r2, r3, r4}
  - n = 3
- Is the following user assignment (UA) relation valid?
- {(u1, r1), (u2, r1), (u3, r1), (u1, r2), (u4, r2), (u5, r2), (u1, r3), (u2, r3), (u3, r3), (u4, r4)}

no u1 has 3 roles

Exercise

- Consider an SSoD constraint with
  - RS = {r1, r2, r3, r4}
  - n = 3
- Is the following user assignment (UA) relation valid?
- {(u1, r1), (u3, r1), (u5, r1), (u1, r2), (u2, r2), (u3, r2), (u5, r2), (u2, r3), (u4, r3)}

Exercise

- Is there any issue if we have the following static separation of duties (SSoD) and dynamic separation of duties (DSoD) constraints at the same time?
- SSoD: ({r1, r2, r3, r4}, 3)
- DSoD: ({r1, r2, r3, r4}, 3)

2nd constraint here will not be violated, a user can at max only have 2 roles, then of course they can only activate their max 2 roles in a session

Exercise

- Is there any issue if we have the following static separation of duties (SSoD) and dynamic separation of duties (DSoD) constraints at the same time?
- SSoD: ({r1, r2, r3, r4}, 3)
- DSoD: ({r1, r2, r3, r4}, 2)

makes sense, 2 roles but only 1 per session

Exercise

- Is there any issue if we have the following static separation of duties (SSoD) and dynamic separation of duties (DSoD) constraints at the same time?
- SSoD: ({r1, r2, r3, r4}, 2)
- DSoD: ({r1, r2, r3, r4}, 3)

2nd one redundant

Coming Next

- How RBAC is used in Oracle

RBAC in Oracle: Examples

- CREATE ROLE BruceWayne IDENTIFIED BY IamBatman
  - Create a role BruceWayne
  - Users who want to use this role in a session must provide the password “IamBatman”
- GRANT insert, update on T TO BruceWayne
  - Allow BruceWayne to perform insert and update on table T
- REVOKE insert on T FROM BruceWayne
  - Revoke BruceWayne’s insertion rights on table T

RBAC in Oracle: Examples

- CREATE ROLE BruceWayne IDENTIFIED BY IamBatman
  - Create a role BruceWayne
  - Users who want to use this role in a session must provide the password “IamBatman”
- GRANT BruceWayne to Cedric
  - Grant the role BruceWayne to user Cedric
- User Cedric: SET ROLE BruceWayne IDENTIFIED BY IamBatman
  - User Cedric activates BruceWayne as his role in a session

RBAC in Oracle: Examples

- CREATE ROLE BruceWayne IDENTIFIED BY IamBatman
  - Create a role BruceWayne
  - Users who want to use this role in a session must provide the password “IamBatman”
- GRANT BruceWayne to Cedric
  - Grant the role BruceWayne to user Cedric
- REVOKE BruceWayne FROM Cedric
  - Revoke the role BruceWayne from Cedric
- DROP ROLE BruceWayne
  - Remove the role BruceWayne

RBAC in Oracle: Examples

- What about separation of duties?
- Not implemented with PL/SQL
- Need to use a separate method: Oracle Database Vault
  - Somewhat similar to VPD

Access Control: Summary

- We have talked about
  - Discretionary access control
  - Mandatory access control
  - Role-based access control
  - How these methods are applied in Oracle Database