

Database Security

Access control

Notes_Set4

Role Based Access Control

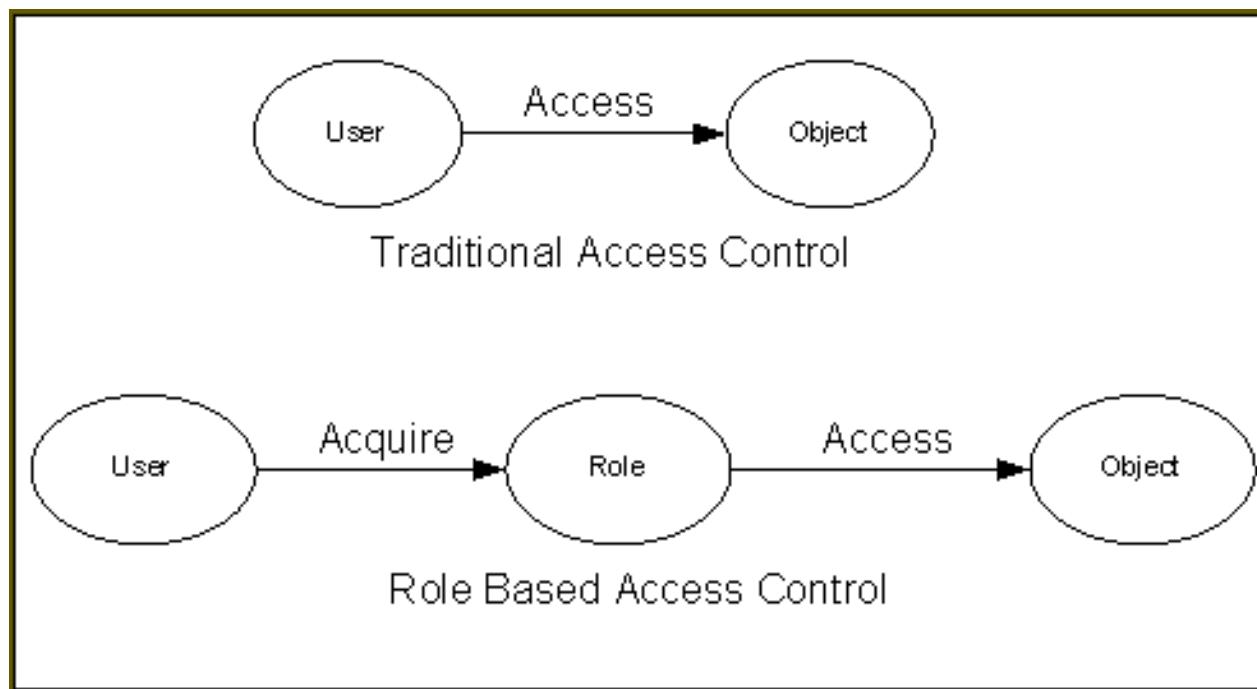
Role Based Access Control

- **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
 - 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
 - Role-based access control (RBAC) emerged rapidly in the 1990s as a proven technology for managing and enforcing security in **large-scale enterprise wide systems**.
 - Role-based access control is most commonly implemented **in small and medium-sized enterprises**. Such organizations typically have simple workflows, a limited number of roles, and a pretty simple hierarchy, making it possible to effectively determine and describe user roles.

RBAC Motivation

- Multi-user systems
- Multi-application systems
- Permissions are associated with roles
- Role-permission assignments are persistent v.s. user-permission assignments
- Intuitive: competency, authority and responsibility

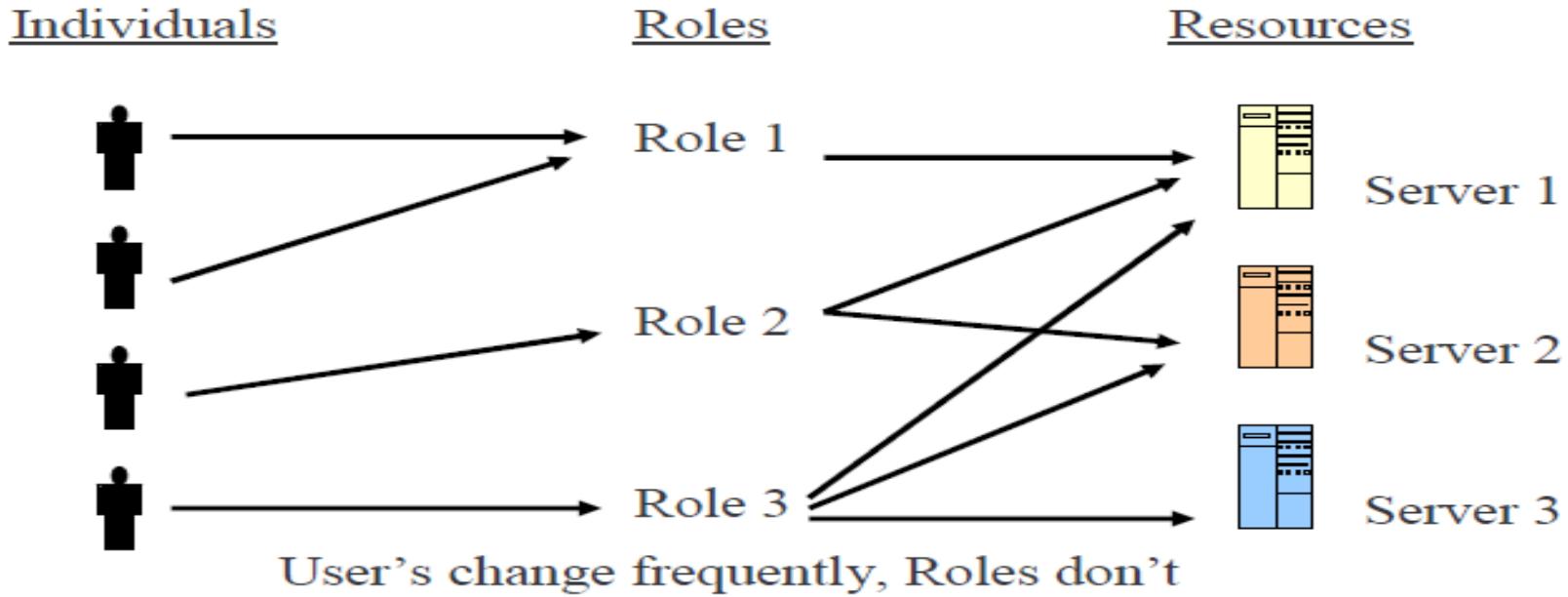
Role-Based Access Control



Role-Based Access Control

- Its basic notion is that **permissions are associated with roles, and users are assigned to appropriate roles.**
- A role is “[is a collection of permissions](#)”
- Roles can be created using the CREATE ROLE and DESTROY ROLE commands.
- The GRANT and REVOKE commands discussed under DAC can then be used to assign and revoke privileges from roles.
- Because roles represent organizational functions, an RBAC model can directly support security policies of the organization
- DBMS vendors have recognized the importance and the advantages of RBAC, and today most of the commercial DBMSs support RBAC features at some extents

Role Based Access Control



Access Control policy is embodied in various components of RBAC such as

- Role-Permission relationships
- User-Role relationships
- Role-Role relationships

The main components of RBAC

- The main components of the role-based approach to access control:
 - **User** – an individual (with UID) with access to a system
 - **Role** – a named job function (indicates the level of authority)
 - **Permission** – equivalent to access rights
 - **Session** – a mapping between a user and a set of roles to which the user is assigned in the context of a working time
 - **Object** – a system resource that requires permission to access
 - **Operation** – any action in the protected network
- The basic rules of RBAC are:
 - A user can execute an operation only if there is a role assigned to the subject.
 - Identification and authentication are not considered operations.
 - All user activities are carried out through operations.

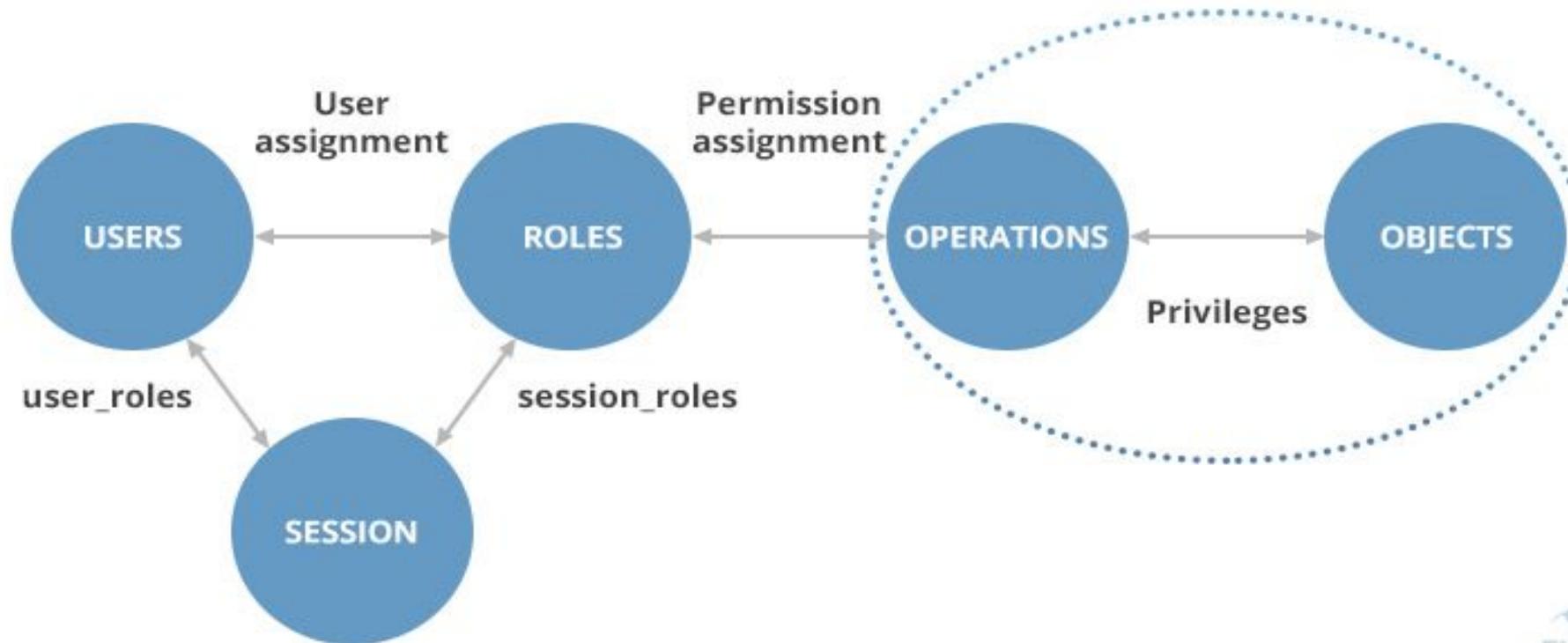
RBAC Reference Model

- The NIST RBAC model is defined in terms of four model components .
 - Core RBAC
 - Hierarchical RBAC
 - Static Separation of Duty Relations
 - Dynamic Separation of Duty Relations
- Each Component is defined by subcomponents:
 - Set of basic elements sets
 - A set of RBAC relations involving those elements sets.
 - A set of mapping functions that yield instances of members from one element set for a given instance from another element set.

NIST RBAC Model

- RBAC can be implemented on four levels, according to the **NIST RBAC model**. Each subsequent level includes the properties of the previous.
- **1. Flat RBAC** is an implementation of the basic functionality of the RBAC model. All users and permissions are assigned roles. Users obtain the permissions they need by acquiring these roles. There may be as many roles and permissions as the company needs. A single user can be assigned to multiple roles, and one role can be assigned to multiple users.
- **2. Hierarchical RBAC**, as the name suggests, implements a hierarchy within the role structure. This hierarchy establishes the relationships between roles. Users with senior roles acquire permissions of all junior roles, which are assigned to their subordinates. The complexity of the hierarchy is defined by the needs of the company.
- **3. Constrained RBAC** adds a **separation of duties** (SOD) to a security system. SOD is a well-known security practice when a single duty is spread among several employees. It's quite important for medium-sized businesses and large enterprises. Separation of duties guarantees that no work can introduce fraudulent changes to your system that no one else can audit and/or fix
- **4. Symmetric RBAC** supports permission-role review as well as user-role review. It allows identification of the permissions assigned to existing roles (and vice versa). For example, by identifying permissions of a terminated employee, the administrator can revoke the employee's permissions and then reassign the role to another user with the same or a different set of permissions

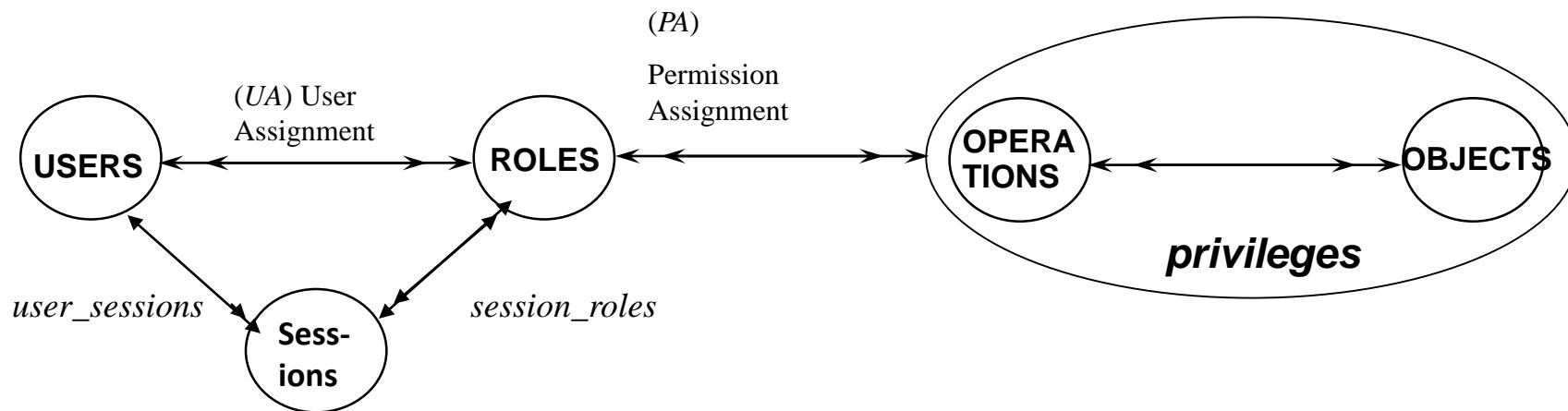
RBAC



Core RBAC

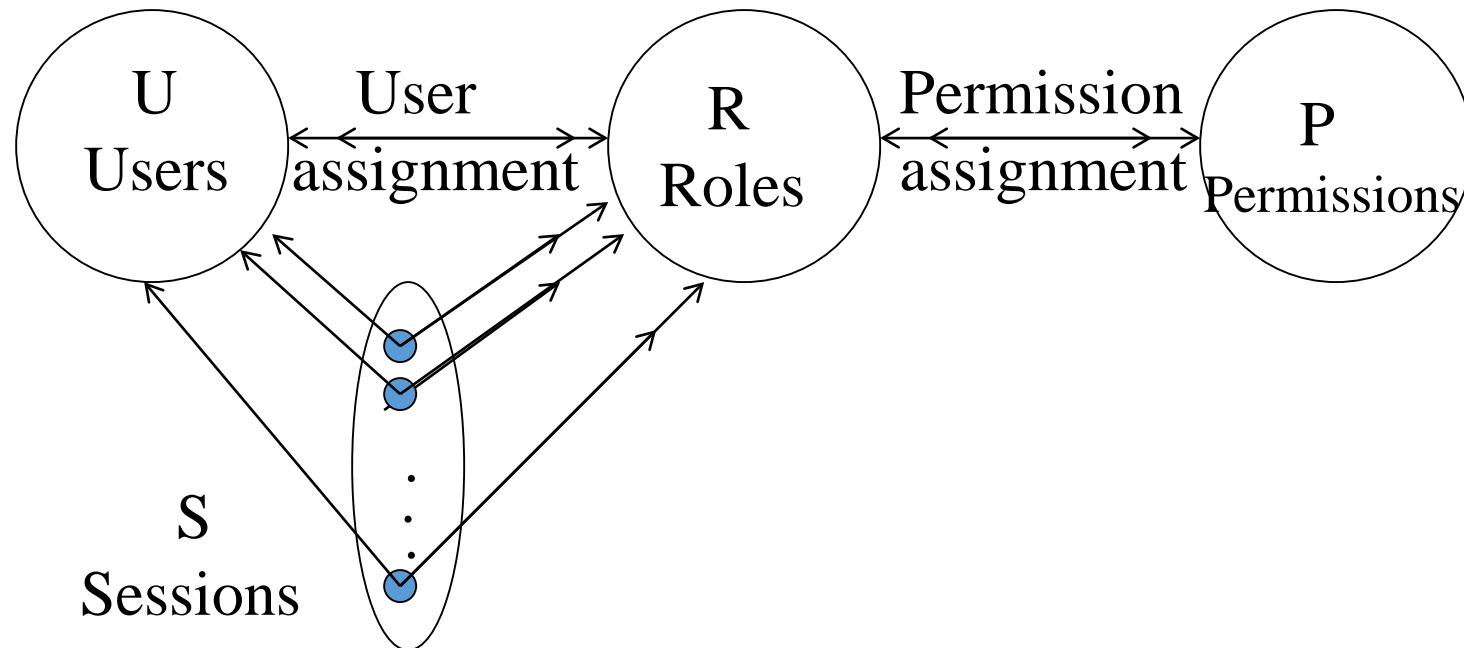
- It embodies the essential aspects of RBAC.
- The basic concept of RBAC is that users are assigned to roles, and users acquire permissions by being members of roles.
- Core RBAC includes requirements that user-role and permission-role assignment can be many-to-many.
- It includes requirements for user-role review whereby the roles assigned to a specific user can be determined as well as users assigned to specific role. A similar requirement for permission-role review is imposed as an advanced review feature.
- It allows includes the concept of user sessions, which allows selective activation and deactivation of roles.
- Finally it requires that users be able to simultaneously exercise permission of multiple roles. This precludes products that restrict users of activation of one role at a time.

Core (Flat) RBAC



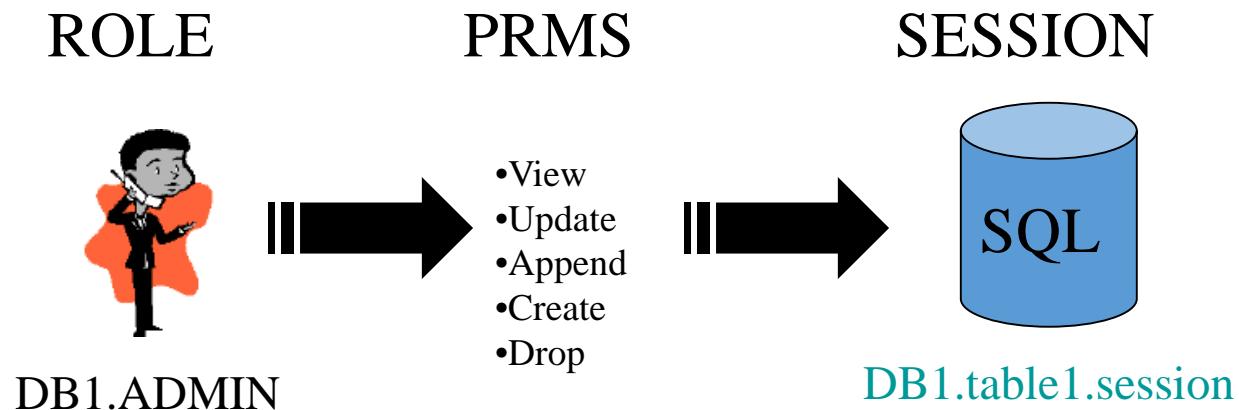
- Accommodates traditional but robust group-based access control
- Many-to-many relationship among individual users and privileges
- It ensures that only authorized users are given access to certain data or resources.
- Session is a mapping between a user and an activated subset of assigned roles
- Users create sessions during which they may activate a subset of roles to which they belong.
- Each session can be assigned to many roles, but it maps to only one user or a single subject.
- User/role relations can be defined independent of role/privilege relations
- Privileges are system/application dependent
- Many DBMSs have allowed the concept of roles, where privileges can be assigned to roles.

RBAC₀



SESSIONS

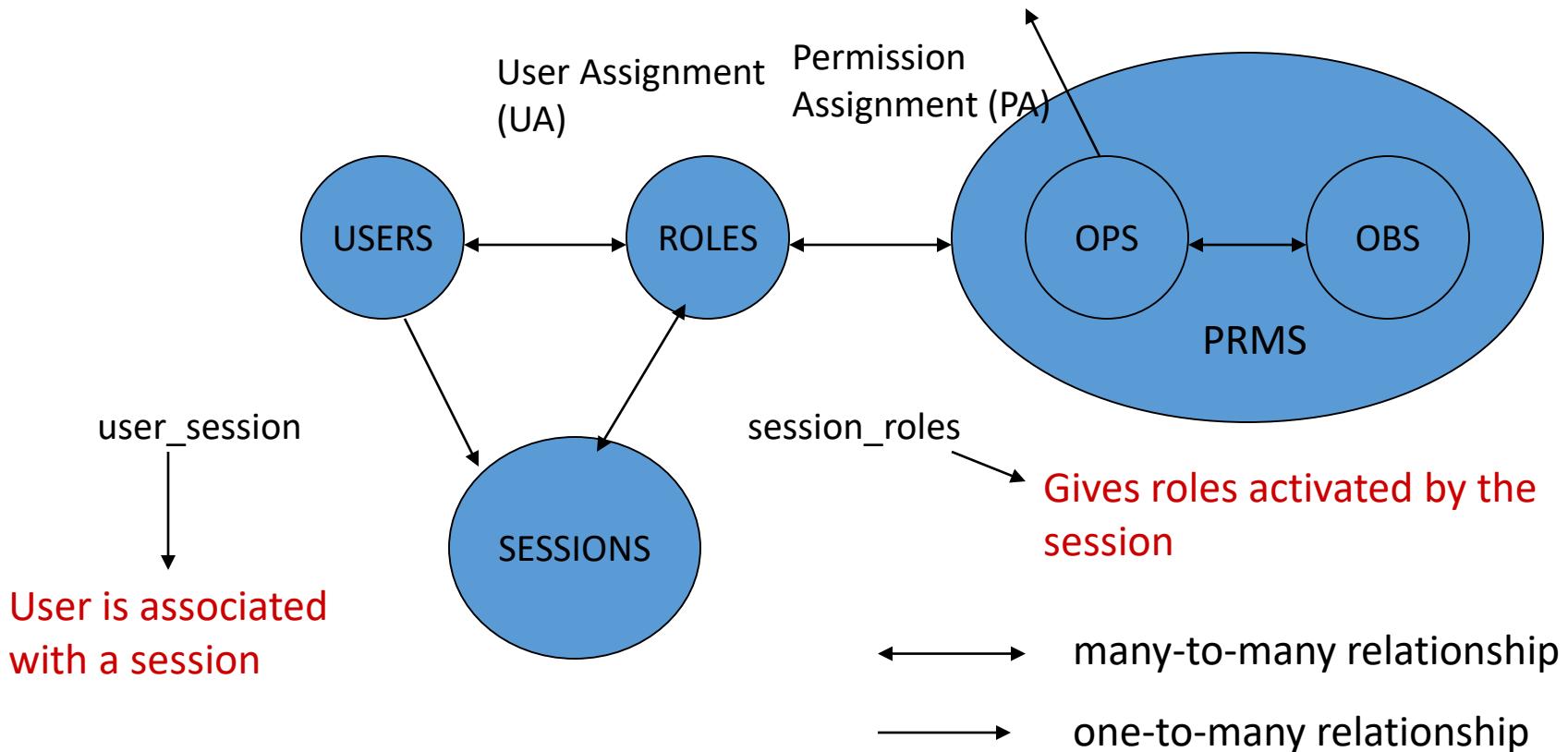
Permissions available to a user in a session.



Role-Based Access Control

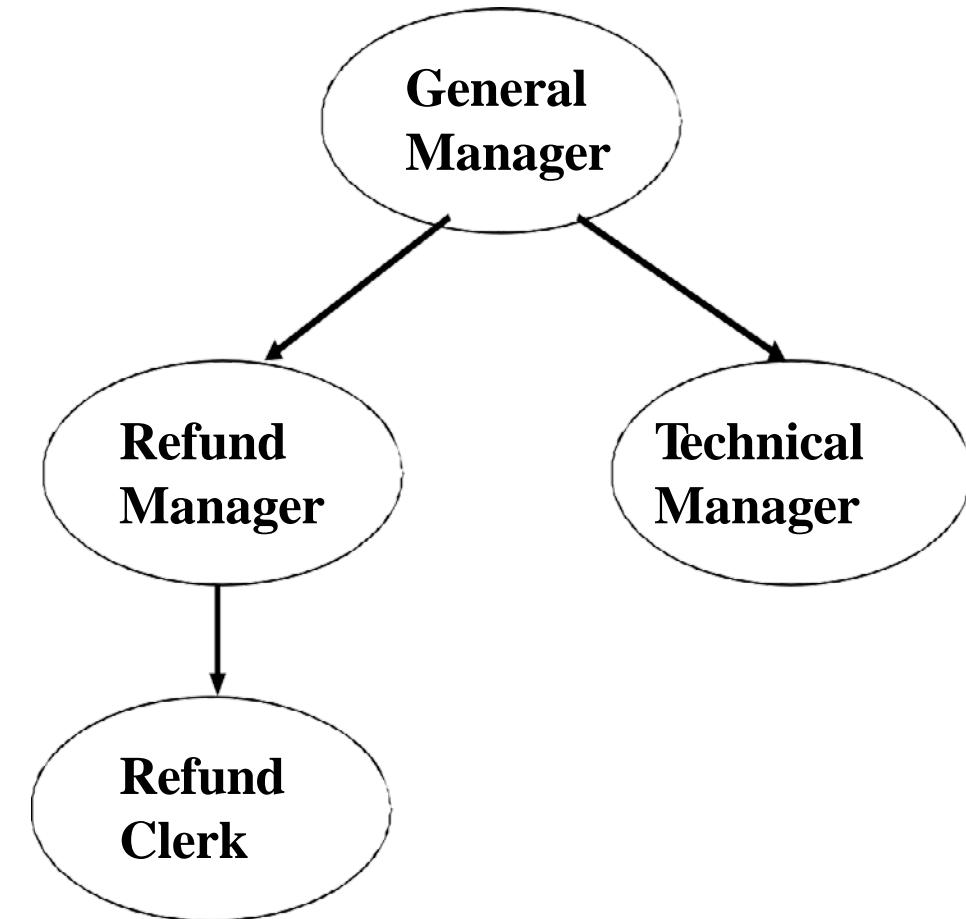
file system operations: read, write and execute

DBMS operations: Insert, delete, append and update

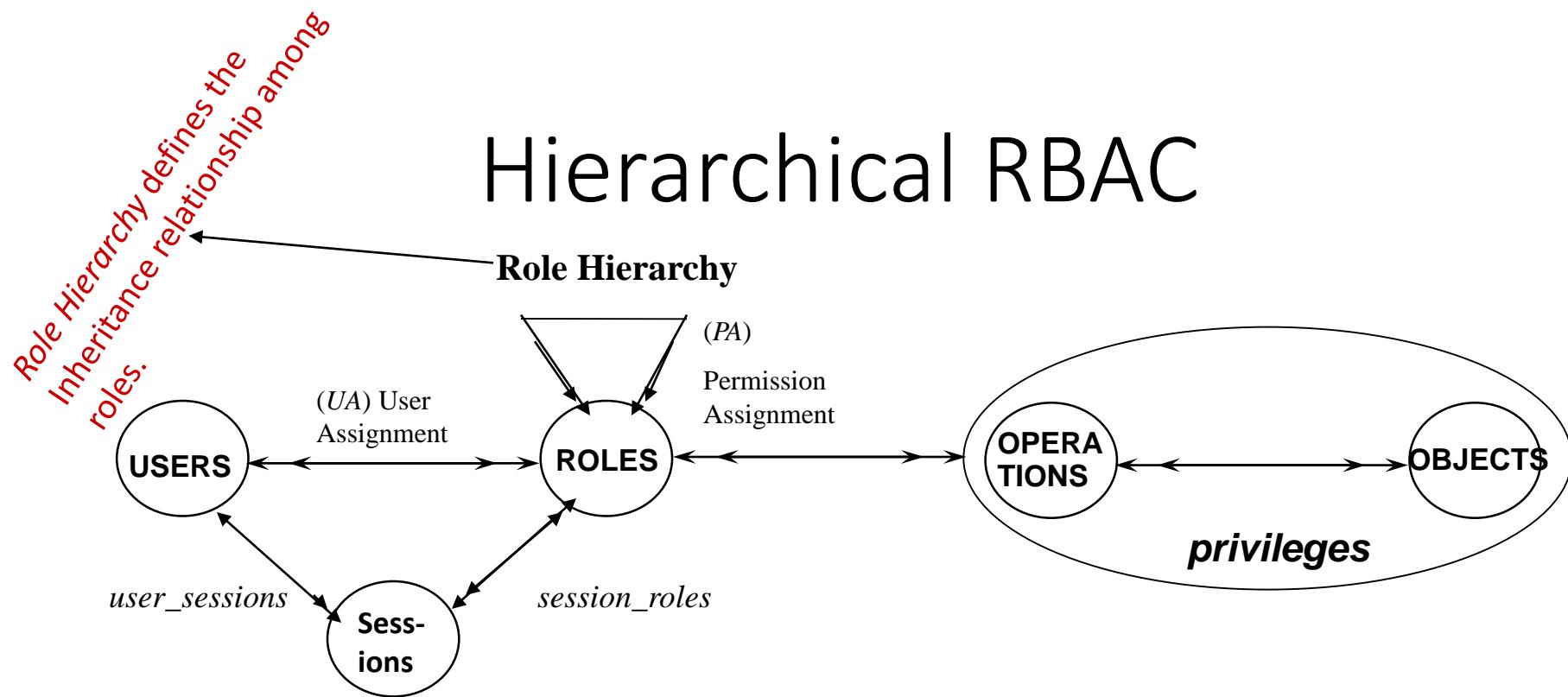


Hierarchical RBAC

- Role hierarchies are a natural means for structuring roles to reflect an organization's line of authority and responsibility
- By convention, junior roles at the bottom are connected to progressively senior roles as one moves up the hierarchy.
- The hierarchic diagrams are partial orders, so they are reflexive, transitive, and antisymmetric

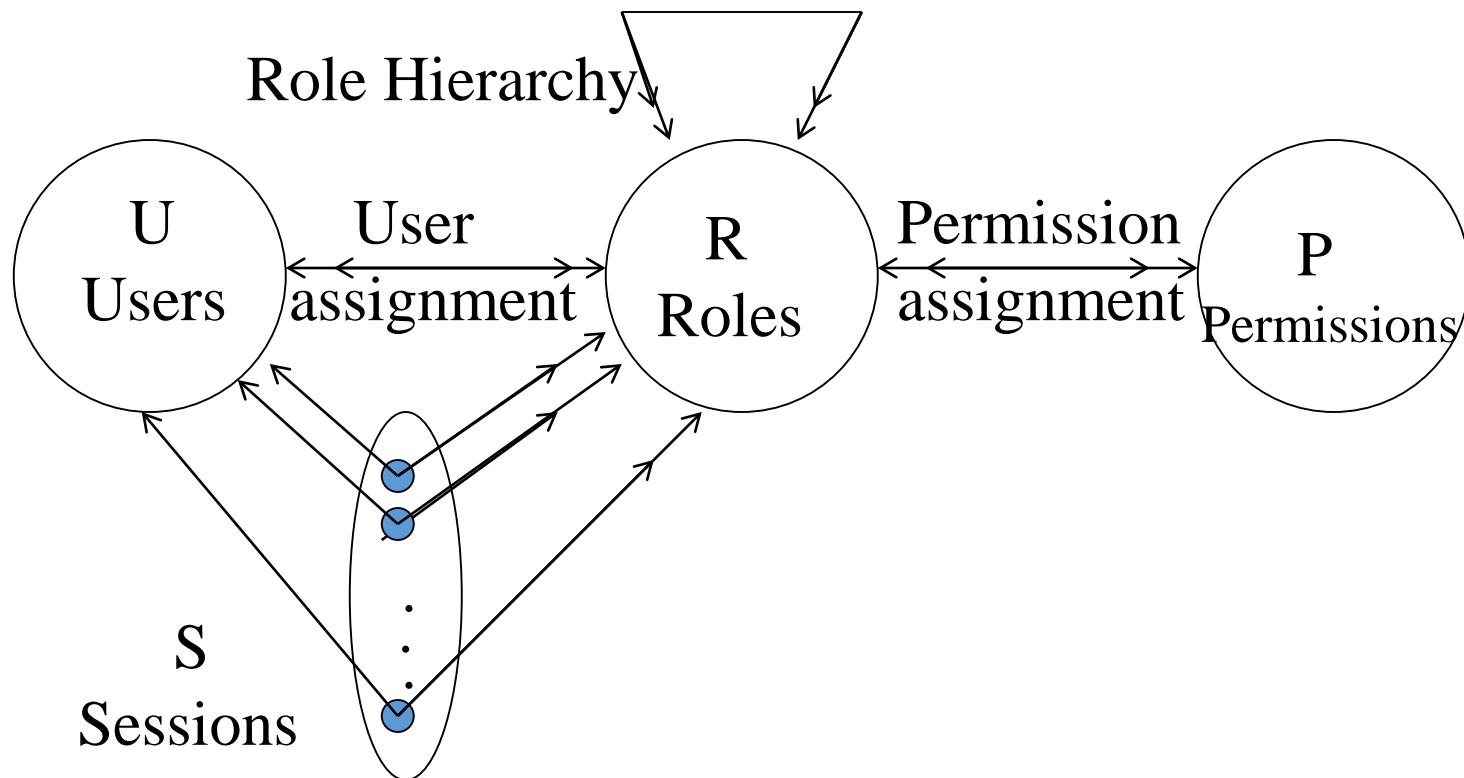


Hierarchical RBAC

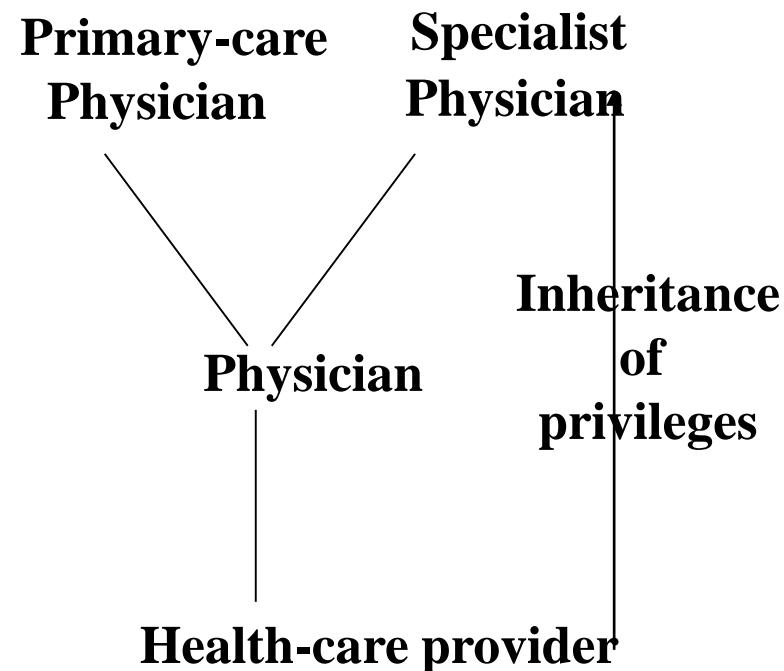


- Role/role relation defining user membership and privilege inheritance
- Reflects organizational structures and functional delineations
- Two types of hierarchies:
 - - Limited hierarchies
 - - General hierarchies

Hierarchical RBAC



Role Hierarchy



Example: Role of **Specialist** could contain the roles of **Doctor** and **Intern**. members of the role **Specialist** are implicitly associated with the operations associated with the roles **Doctor** and **Intern** without the administrator having to explicitly list the **Doctor** and **Intern** operations. Moreover, the roles **Cardiologist** and **Rheumatologist** could each contain the **Specialist** role.

Authorized Users

Mapping of a role onto a set of users in
the presence of a role hierarchy

ROLES set

Admin.DB1

User.DB2

User.DB3

User.DB1

- View
- Update
- Append

permissions object

First Tier USERS set



User.DB1



User.DB1

Authorized Permissions

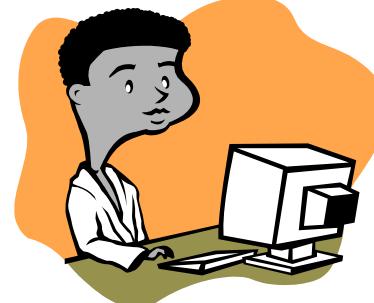
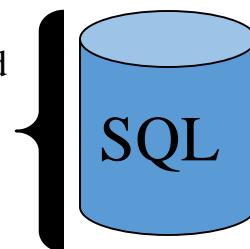
Mapping of a role onto a set of permissions
in the presence of a role hierarchy

ROLES set

User.DB1
User.DB2
User.DB3
Admin.DB1

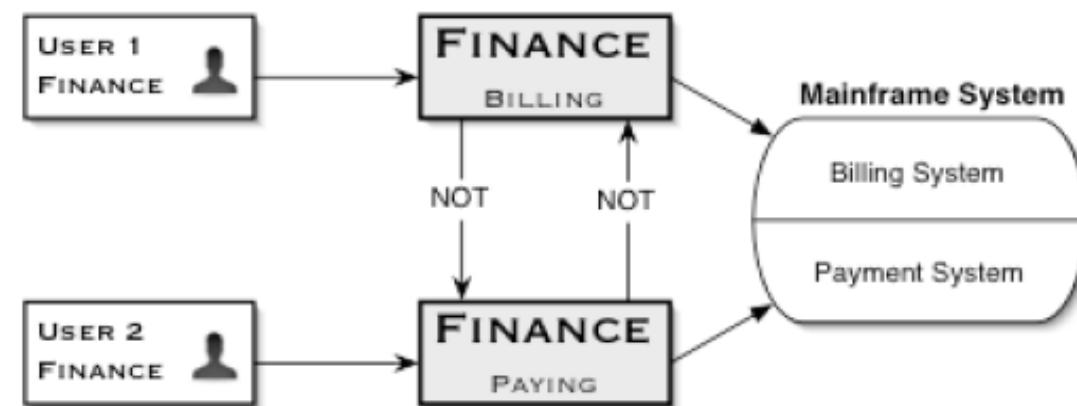
PRMS set

- View
- Update
- Append
- Create
- Drop

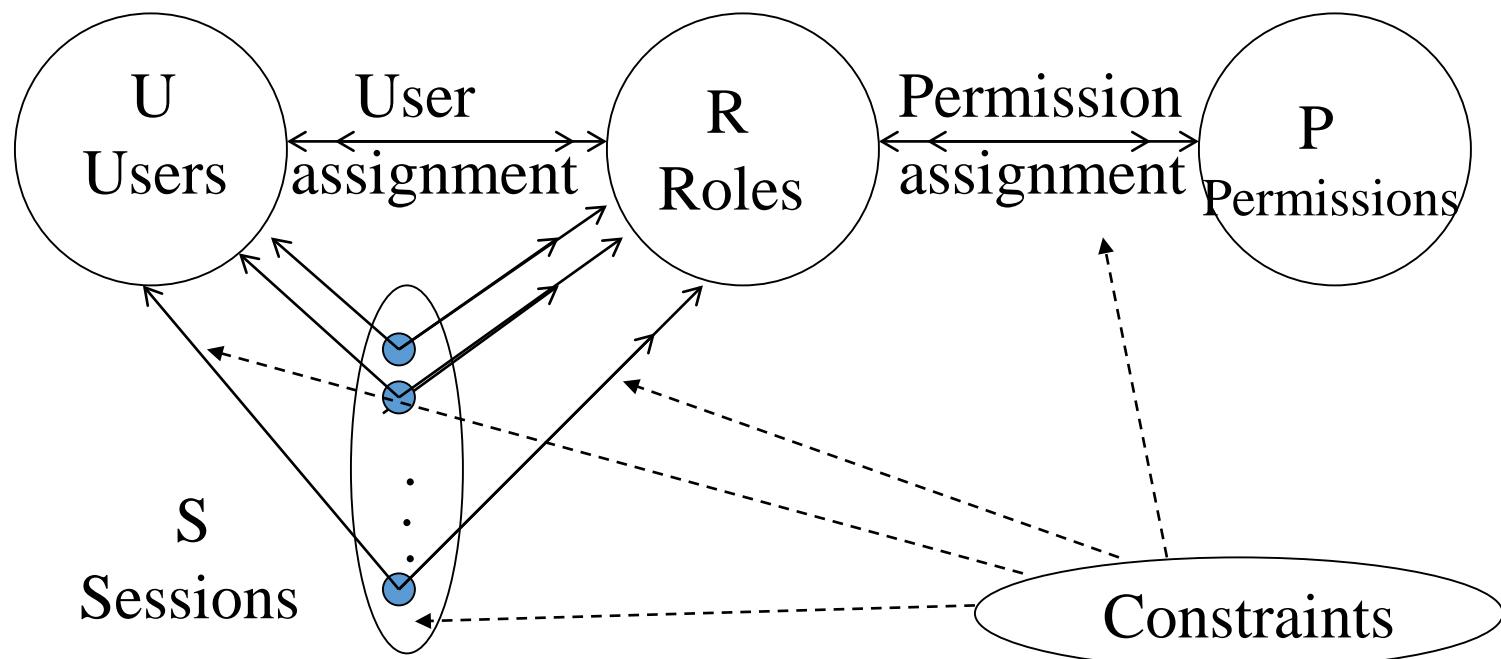


Constrained RBAC

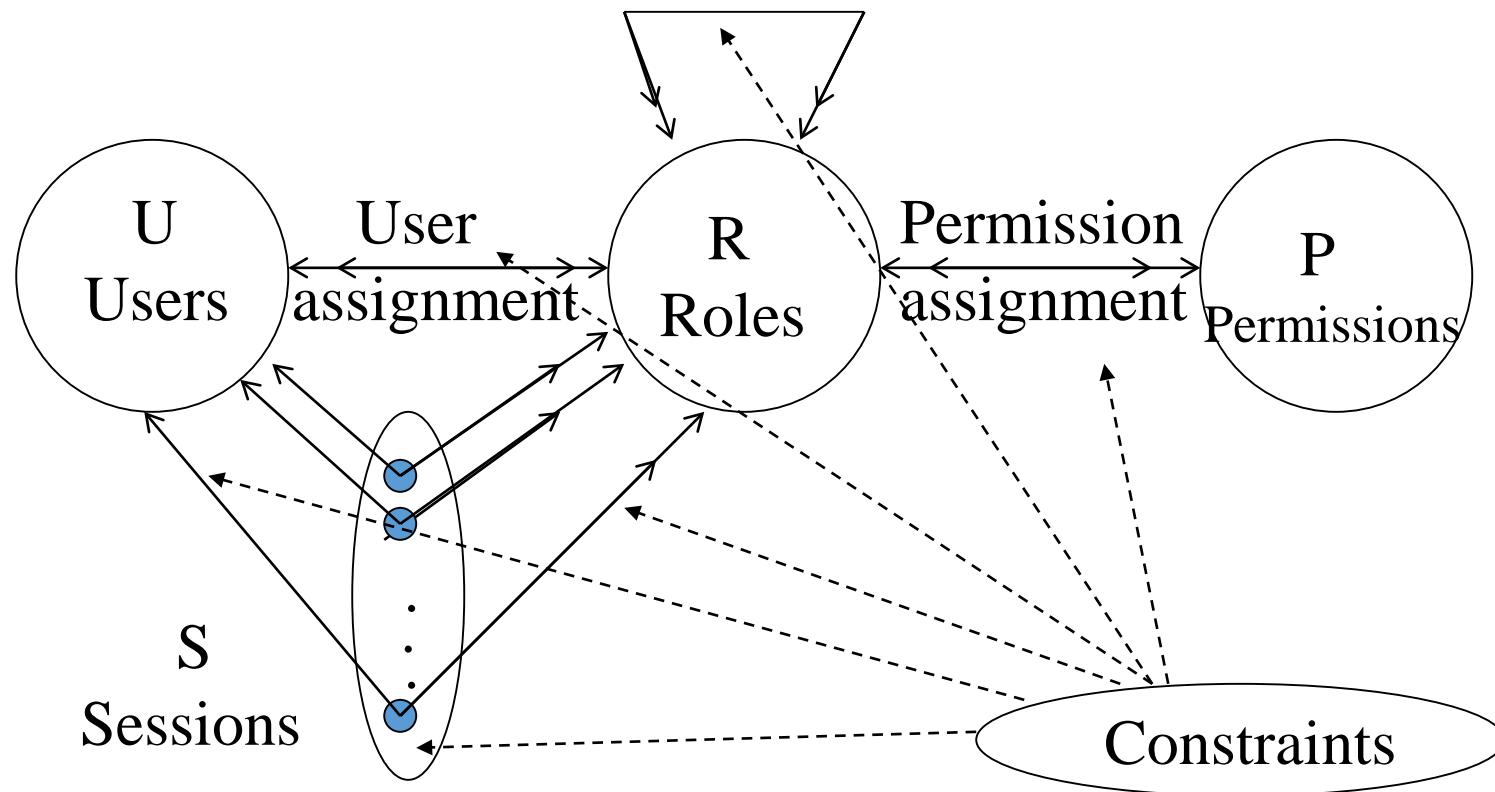
- Another important consideration in RBAC systems is the possible temporal constraints that may exist on roles, such as the time and duration of role activations, and timed triggering of a role by an activation of another role.
- Using an RBAC model is a highly desirable goal for addressing the key security requirements of Web-based applications.
- Roles can be assigned to workflow tasks so that a user with any of the roles related to a task may be authorized to execute it and may play a certain role for a certain duration only.
- Constrained RBAC : The capability of expressing *Separation of Duties* constraints
 - SoD constraints are used to represent conflicts of interest
 - Static SoD : a user cannot be authorized for two roles (defining the conflicting roles)
 - Dynamic SoD : a user can be authorized for both roles but cannot act simultaneously in both roles (Enforcing the control at access time)



Constrained RBAC



Constrained RBAC



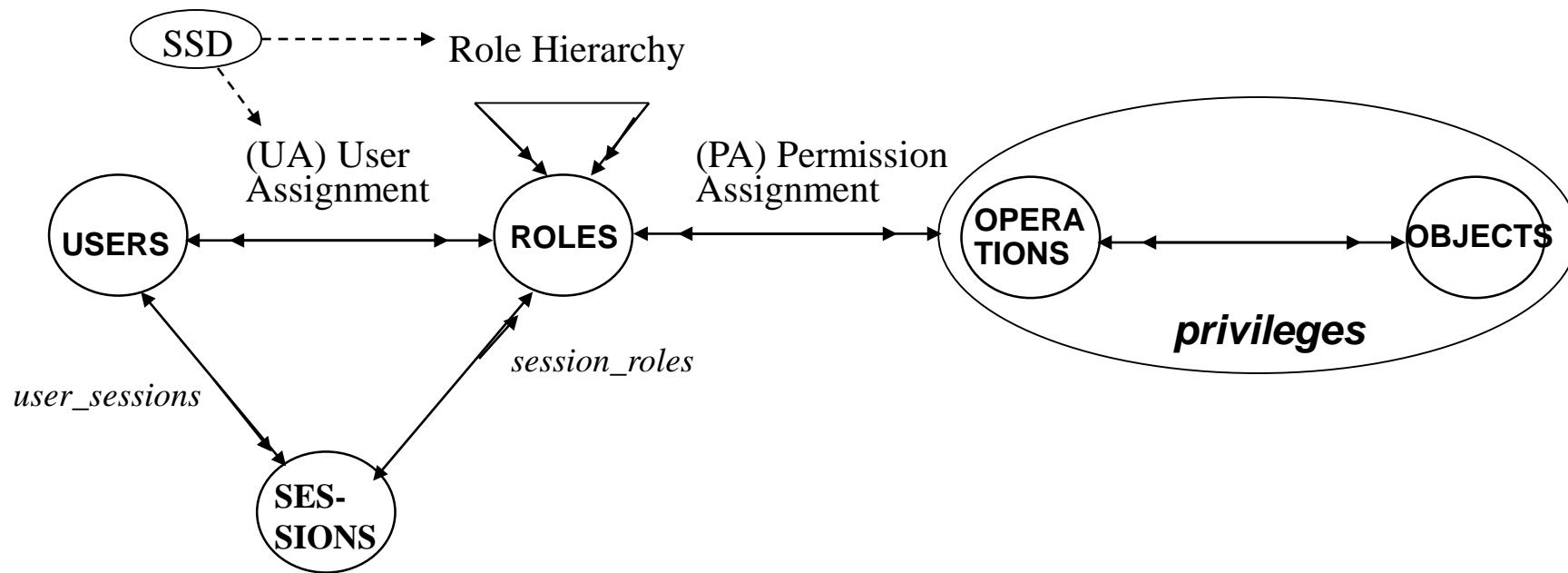
Conflict of Interest

- Static Separation of Duty: user cannot be authorized for both roles, e.g., teller and auditor
 - SSoD policies deter fraud by placing constraints on administrative actions and thereby restricting combinations of privileges that are made available to users
- Dynamic Separation of Duty: user cannot act simultaneously in both roles, e.g., teller and account holder
 - DSoD policies deter fraud by placing constraints on the roles that can be activated in any given session thereby restricting combinations of privileges that are made available to user

Static Separation of Duty

- SSD relations
 - prevent conflict of interests that arise when a user gains permissions associated with conflicting roles
 - SSD relations are specified for any pair of roles that conflict.
- A bank defines
 - teller role as being able to perform a savings deposit operation.
 - requires read and write access to specific fields within a savings file.
 - accounting supervisor role is allowed to perform correction operations. operations require read and write access to the same fields of a savings file as the teller.
 - The accounting supervisor may not be allowed to initiate deposits or withdrawals but only perform corrections.
 - The teller is not allowed to perform any corrections once the transaction has been completed.

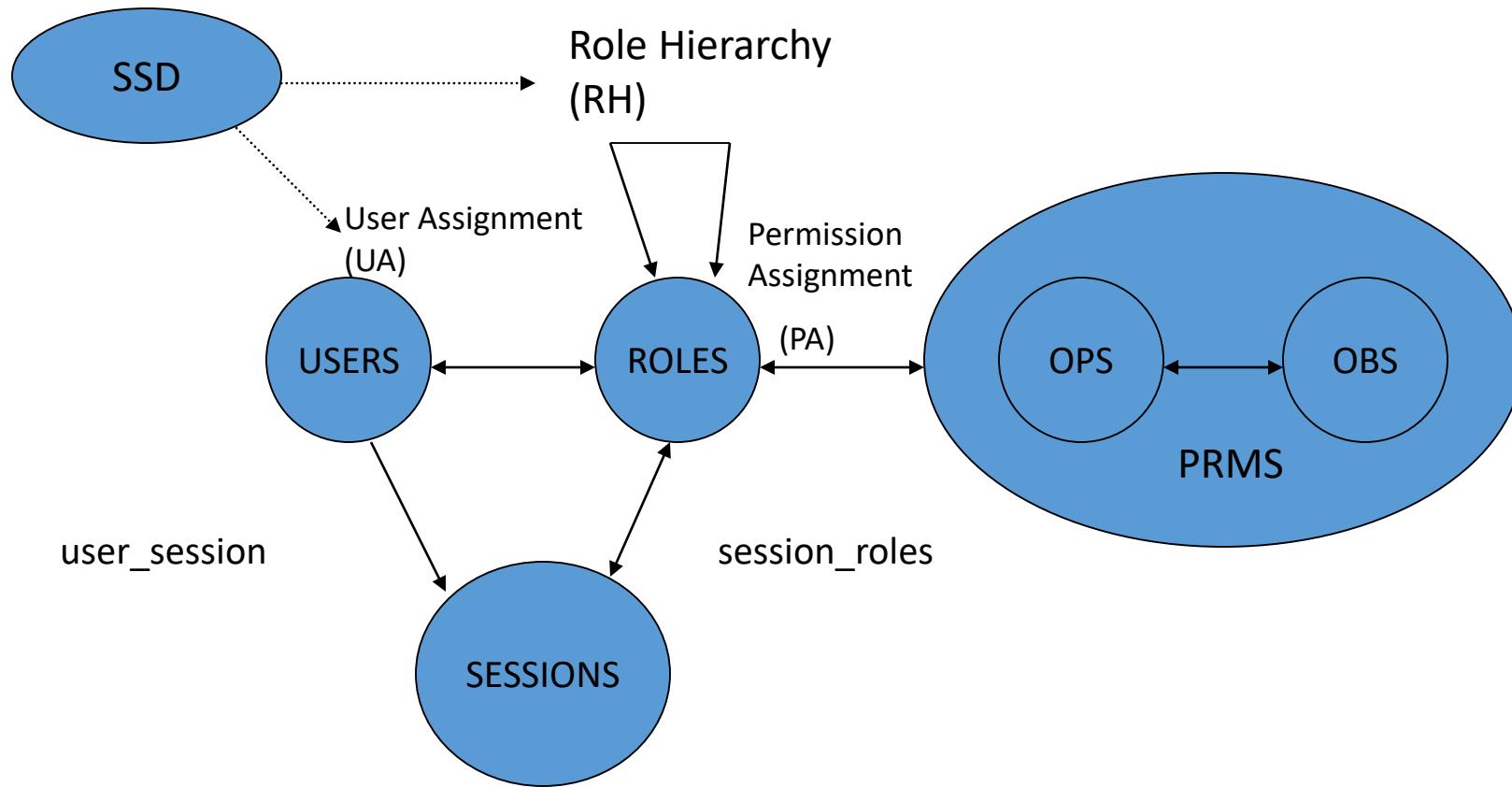
Static Separation of Duty Relations



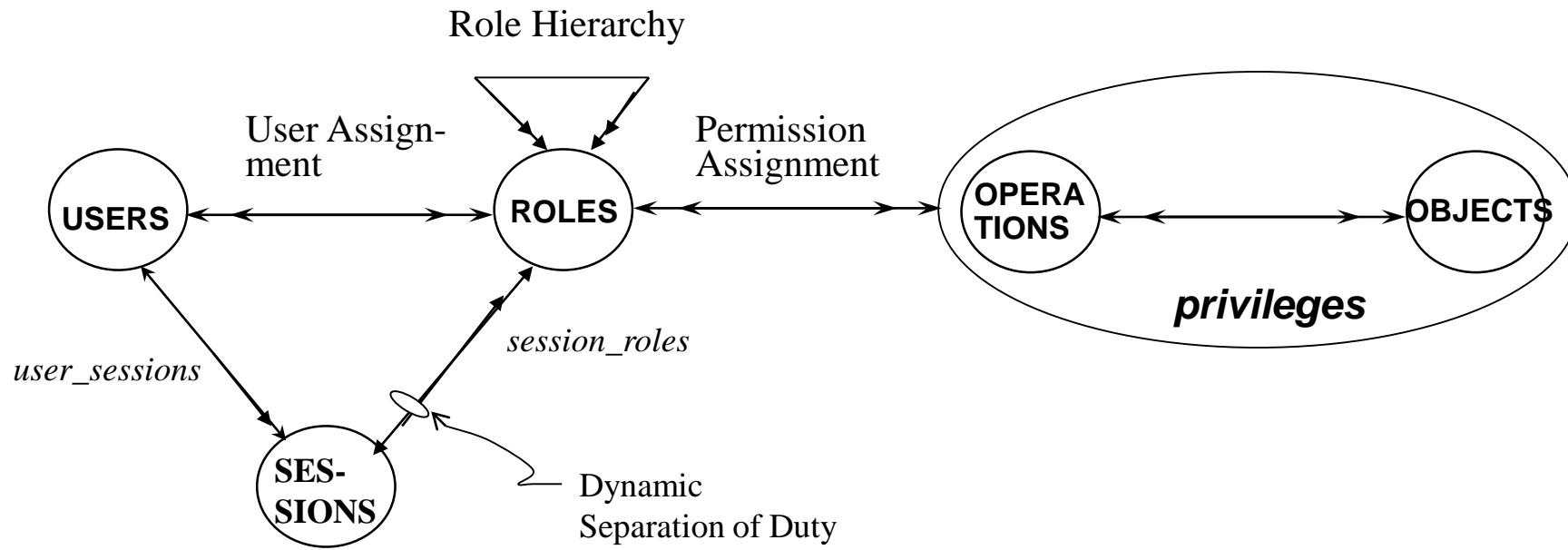
SoD policies deter fraud by placing constraints on administrative actions and thereby restricting combinations of privileges that are available to users.

E.g., no user can be a member of both Cashier and AR Clerk roles in Accounts Receivable Department

SSD with Hierarchical RBAC

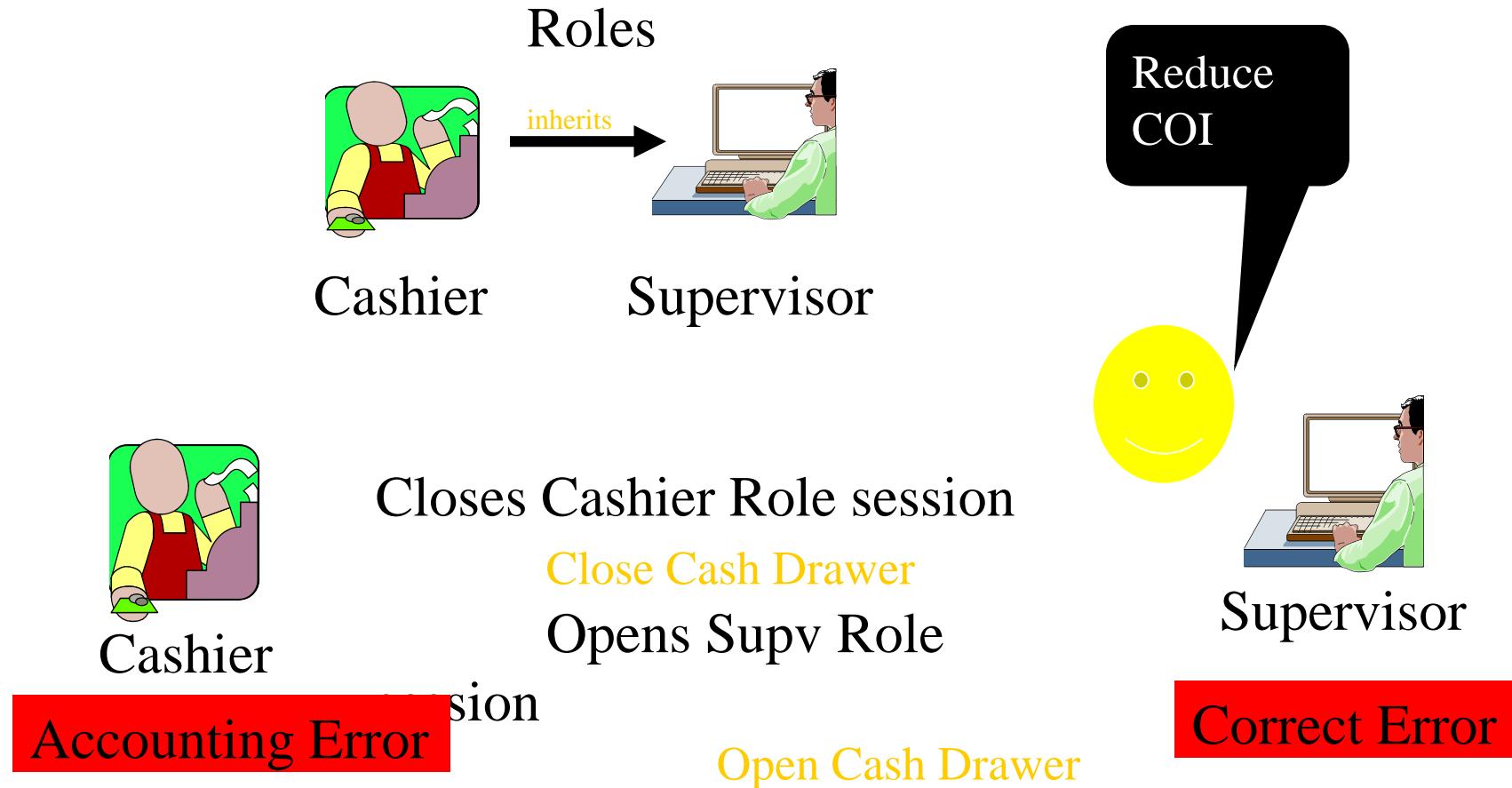


Dynamic Separation of Duty Relations



DSoD policies deter fraud by placing constraints on the roles that can be activated in any given session there by restricting combinations of privileges that are available to users

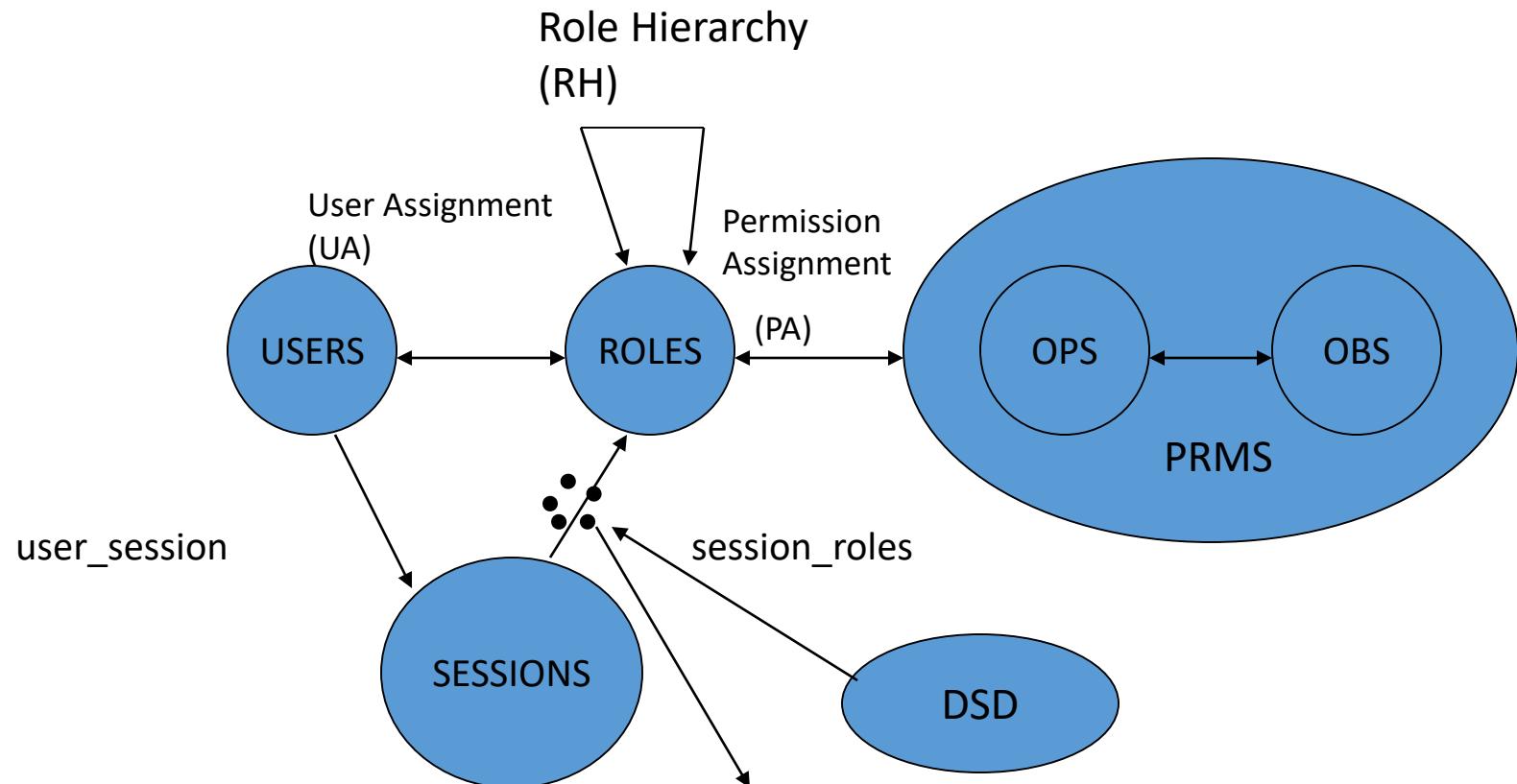
Dynamic Separation of Duty Relations



DSD example

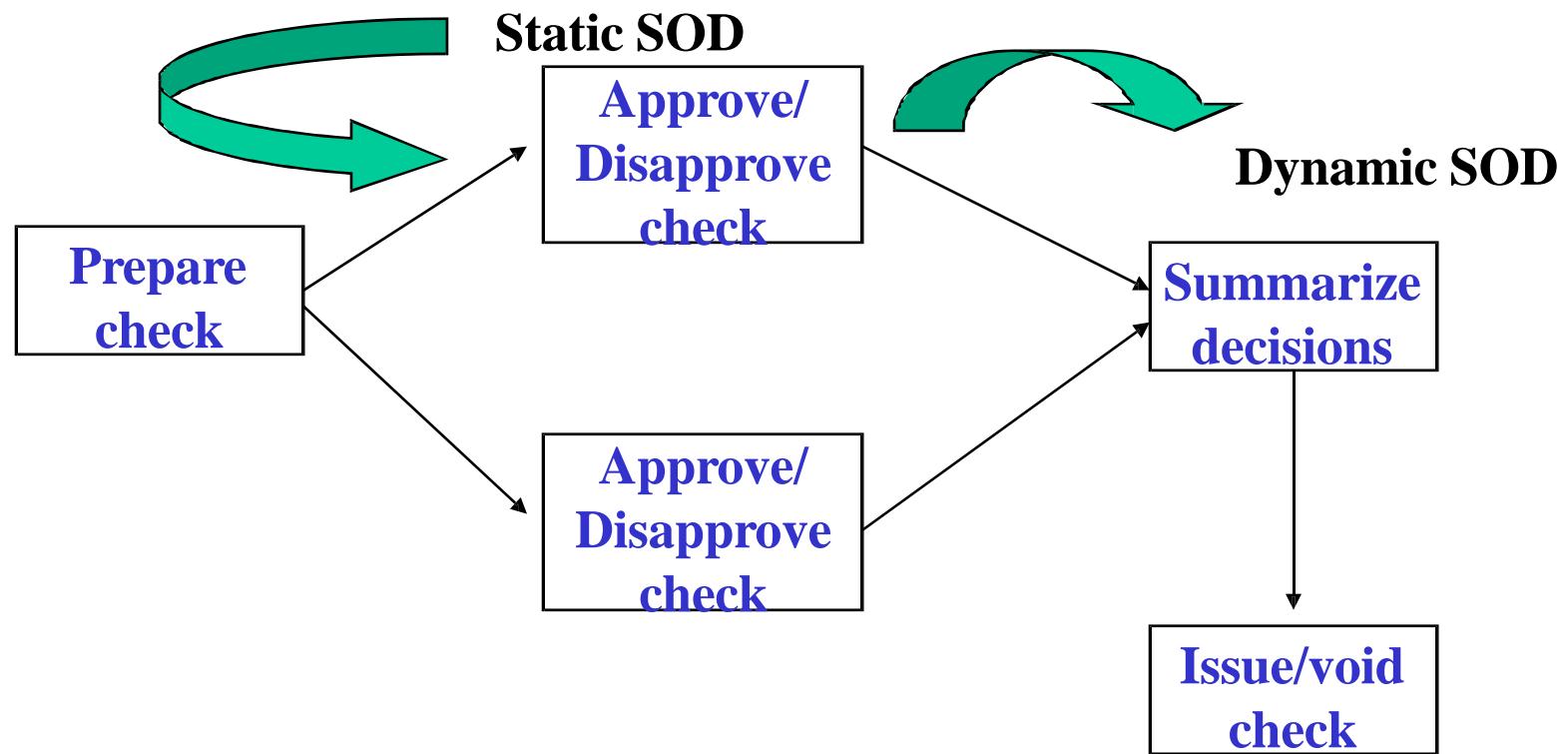
- A user may be authorized for both the roles of Cashier and Cashier Supervisor,
- where the supervisor is allowed to acknowledge corrections to a Cashier's open cash drawer.
- If the individual acting in the role Cashier attempted to switch to the role Cashier Supervisor,
- RBAC would require the user to drop the Cashier role, and thereby force the closure of the cash drawer before assuming the role Cashier Supervisor.
- As long as the same user is not allowed to assume both of these roles at the same time, a conflict of interest situation will not arise.
- this can be achieved through the establishment of a static separation of duty relationship, DSD relationships generally provide the enterprise with greater operational flexibility.

Dynamic Separation of Duty



DSD relations place constraints on the roles that can be activated in a user's session. If one role that takes part in a DSD relation is activated, the user cannot activate the related (conflicting) role in the same session

Constrained RBAC



Access Control in Commercial DBMSs

- Most of the commercial DBMSs also support RBAC features(Informix, Sybase, Oracle)
- However, in most cases they only supports flat RBAC
- MongoDB employs Role-Based Access Control (RBAC) to govern access to a MongoDB system. A user is granted one or more [roles](#) that determine the user's access to database resources and operations. Outside of role assignments, the user has no access to the system.
- MongoDB does not enable access control by default. You can enable authorization using the [--auth](#) or the [security.authorization](#) setting.
- Enabling [internal authentication](#) also enables client authorization. Once access control is enabled, users must [authenticate](#) themselves
- A role grants privileges to perform the specified [actions](#) on [resource](#). Each privilege is either specified explicitly in the role or inherited from another role or both.
- A privilege consists of a specified resource and the actions permitted on the resource.
- A [resource](#) is a database, collection, set of collections, or the cluster. If the resource is the cluster, the affiliated actions affect the state of the system rather than a specific database or collection. For information on the resource documents, see [Resource Document](#).
- An [action](#) specifies the operation allowed on the resource. For available actions see [Privilege Actions](#).
- **Inherited Privileges :** A role can include one or more existing roles in its definition, in which case the role inherits all the privileges of the included roles. A role can inherit privileges from other roles in its database. A role created on the admin database can inherit privileges from roles in any database.
- **View Role's Privileges:** You can view the privileges for a role by issuing the [rolesInfo](#) command with the showPrivileges and showBuiltInRoles fields both set to true.
- You can assign roles to users during the user creation. You can also update existing users to grant or revoke roles. For a full list of user management methods, see [User Management](#)
- A user assigned a role receives all the privileges of that role. A user can have multiple roles. By assigning to the user roles in various databases, a user created in one database can have permissions to act on other databases.
- MongoDB provides [built-in roles](#) that provide set of privileges commonly needed in a database system.
- If these built-in-roles cannot provide the desired set of privileges, MongoDB provides methods to create and modify [user-defined roles](#).

RBAC –SQL Commands

- CREATE ROLE role-name IDENTIFIED BY passw | NOT IDENTIFIED;
 - Example: CREATE ROLE teller IDENTIFIED BY cashflow;
- DROP ROLE role-name;
- GRANT role TO user | role | PUBLIC [WITH ADMIN OPTION];
 - to perform the grant of a role, a user must have the privilege for the role with the ADMINoption, or the system privilege GRANT ANY ROLE
 - The ADMIN option allows the receiver to modify or drop the role
 - Example: GRANT teller TO Bob;
- The grant command for authorization granting can have roles as subjects
 - Example: GRANT select ON Employee TO teller;

RBAC –SQL Commands

- SET ROLE role-name IDENTIFIED BY passwd;
 - The set command is used enable and disable roles during sessions
 - Example: SET ROLE teller IDENTIFIED by cashflow;
- SET ROLE ALL [EXCEPT role-name]
 - it can only be used for roles not requiring passwords
 - SET ROLE ALL;
 - SET ROLE ALL EXCEPT banker;
- SET ROLE NONE;
 - It disables roles for the current session

Implementing Role-Based Access Control

- Role-based access control allows organizations to improve their security posture and comply with security regulations. However, implementing role-based access control across an entire organization can be complex and may result in pushback from stakeholders. To succeed in your move to RBAC, you should treat the implementation process as a series of steps:
- Understanding your business needs
 - Before you move to RBAC, you should run a comprehensive needs analysis to examine job functions, supporting business processes and technologies. You should also consider any regulatory or audit requirements and assess the current security posture of your organization. You may also benefit from other types of access control.
- Planning the scope of implementation
 - Identify the scope of your RBAC requirements and plan the implementation to align with the organization's needs. Narrow your scope to focus on systems or applications that store sensitive data. This will also help your organization manage the transition.
- Defining roles
 - It will be easier to define your roles once you have performed the needs analysis and understand how individuals perform their tasks. Watch out for common role design pitfalls like excessive or insufficient granularity, role overlap, and granting too many exceptions for RBAC permissions.
- Implementation
 - The final phase involves rolling out the RBAC. Do this in stages, to avoid an overwhelming workload and reduce disruption to the business. First, address a core group of users. Start with coarse-grained access control before increasing granularity. Collect feedback from users and monitor your environment to plan the next stages of implementation.

RBAC Implementation

- **1. Inventory your systems**
 - Figure out what resources you have for which you need to control access, if you don't already have them listed. Examples would include an email system, customer database, contact management system, major folders on a file server, etc.
- **2. Analyze your workforce and create roles**
 - You need to group your workforce members into roles with common access needs. Avoid the temptation to have too many roles defined. Keep them as simple and stratified as possible.
 - For example, you might have a basic user role, which includes the access any employee would need, such as email and the intranet site. Another role might be a customer service rep, that would have read/write access to the customer database, and a customer database administrator, that would have full control of the customer database.
- **3. Assign people to roles**
 - Now that you have a list of roles and their access rights, figure out which role(s) each employee belongs in, and set their access accordingly.

RBAC Implementation

- **4. Never make one-off changes**
 - Resist any temptation to make a one-off change for an employee with unusual needs. If you begin doing this, your RBAC system will quickly begin to unravel. Change the roles as required or add new ones when really necessary.
- **5. Audit**
 - Periodically review your roles, the employees assigned to them, and the access permitted for each. If you discover, for example, that a role has unnecessary access to a particular system, change the role and adjust the access level for all employees in that role.
 - As an example, many healthcare organizations, given the need for regulatory compliance in controlling access to medical records, use RBAC to define exactly what access to medical records each type of clinician may need. While a doctor might have almost unlimited access to the records of patients he/she manages, a receptionist might be limited to basic contact information needed to manage appointments. Given the large number of staff members in well stratified roles, RBAC is an efficient way to control record access in compliance with HIPAA, and other regulations.
 - There are tools that can help with setting up RBAC. Many systems, such as Microsoft Active Directory, have built in roles that you can use as a starting point, which you can extend to fit your unique situation. You can also use an identity management system to automate the assignment of privileges based on role.

RBAC Pros and Cons

- RBAC is the most popular approach to restricting access. The main advantage of this model is that companies no longer need to authorize or revoke access on an individual basis, bringing users together based on their roles instead. Establishing a set of roles in a small or medium-sized company isn't challenging. On the other hand, setting up such a system at a large enterprise is no easy task.
- There are several **limitations to the RBAC** model. You can't set up a rule using parameters that are unknown to the system before a user starts working. Permissions can be assigned only to user roles, not to objects and operations. Also, using RBAC, you can restrict access to certain actions in your system but not to certain data.

Advantages of RBAC

- In some cases, role-based access control has been accepted as the **best-practice model**. If the model for the roles and permissions has been defined and has been implemented and enforced throughout the company, **RBAC can offer numerous advantages**:
 - **Flexibility**: The company only assigns roles to an employee as required. Any modifications to the organizational structure or permissions are quickly applied to all employees when the company modifies the corresponding role.
 - **Reduced administration work**: RBAC has rendered the time-consuming process of individually assigning permissions obsolete.
 - **Less error prone**: Assigning permissions individually is a more complex process and is thus more error prone than using role-based access control for assigning permissions.
 - **Increased efficiency**: Reducing the amount of work and error rate increases the efficiency of IT and other employees. There is no longer any need for manual modifications, error handling, wait times or individual permission requests.
 - **Security**: Access permissions are defined exclusively via the role model which prevents you from giving more permissions than needed to individual employees. This is in line with the Principle of Least Privilege (PoLP).
 - **Transparency**: The naming of roles is usually straightforward and thus increases transparency and comprehensibility for users.

disadvantages of RBAC

- Role-based access control also has some **disadvantages**:
 - **Labor-intensive setup**: Translating organizational structures into the RBAC model requires a lot of work.
 - **Temporary assignments**: If a user only needs extended access permissions temporarily, it is easier to forget about them when using RBAC than when assigning permissions individually.
 - **Application**: In small companies, creating and maintaining roles would be more labor intensive than assigning permissions individually. Therefore, the RBAC model is only used when a certain number of roles and employees has been reached. However, even in large companies, role-based access control suffers from the drawback that it is easy to end up creating a large number of roles. If a company has ten departments and ten roles, this will already result in 100 different groups.

Disadvantage of RBAC

- The main disadvantage of RBAC is what is most often called the '**'role explosion'**
 - due to the increasing number of different (real world) roles (sometimes differences are only very minor) you need an increasing number of (RBAC) roles to properly encapsulate the permissions (a permission in RBAC is an action/operation on an object/entity). Managing all those roles can become a complex affair.
- Because of the abstraction choices that form the foundation of RBAC, it is also not very well suited to manage individual rights, but this is typically deemed less of a problem.
- Following are the disadvantages of RBAC (Role based access model):
 - If you want to create a complex role system for big enterprise then it will be challenging as there will be thousands of employees with very few roles which can cause role explosion.
 - Using RBAC, some restrictions can be made to access certain actions of system but you cannot restrict access of certain data.
 - The permissions and privileges can be assigned to user roles but not to operations and objects.

What's not so good with RBAC?

- It is coarse-grained. If you have a role called doctor, then you would give the doctor role a permission to "view medical record". That would give the doctor the right to view all medical records including their own. This is what leads to role explosion
- It is static. RBAC cannot use contextual information e.g. time, user location, device type...
- It ignores resource meta-data e.g. medical record owner.
- It is hard to manage and maintain. Very often, administrators will keep adding roles to users but never remove them. You end up with users that dozens if not hundreds of roles and permissions
- It cannot cater to dynamic segregation-of-duty.
- It relies on custom code within application layers (API, apps, DB...) to implement finer-grained controls.
- Access reviews are painful, error-prone and lengthy

The main challenges of RBAC

- **Problem 1: Role Explosion**
 - If VP of Marketing access request scenario sounds familiar, it is because it happens all too often. Role Explosion happens when the level of granularity needed for your access control is too detailed. Role Explosion is difficult and costly to manage and makes access control confusing and complicated, reducing the access control effectiveness. Additionally, there are several other issues created that need to be monitored carefully when adding more roles to your access control deployment. One of these problems occurs when a user has too many roles assigned to them and then changes jobs or responsibilities within the company. IT system administrators either forget, or even make a conscious decision to leave old roles in place. The quantity of roles can lead to security holes that are often too difficult to find and close.
- **Problem 2: Security Risk Tolerance**
 - As a system administrator, it is important to understand the risks to your system. Conducting a security risk analysis with a proactive risk prevention plan is essential for RBAC deployment. RBAC is data focused; data is categorized relevant to the organizational structure and that leads to access control role definition. If your organization is reactive to security risks, RBAC may not be the optimal way of securing access to your network data. RBAC requires that you have intimate knowledge of the security layout of your company and of how permissions are being granted before deployment. Once deployed, it is hard to react to changing security threats and risks. So be careful and “measure twice, and cut once” with your RBAC policies. In an era of increased scrutiny of security effectiveness due to changing data privacy and protection regulations, this dilution of the security model significantly increases the residual risk of data breach, with significant consequences both financial and reputational.

The main challenges of RBAC

- **Problem 3: Scalability & Dynamism**
 - Yes, at the start of your RBAC deployment, you knew exactly what roles you needed to define, and who they needed to be assigned to. But, it's now a year later and the organization has grown. More people have joined the company and in the rush of onboarding all the new people, the organization charts and job definitions have not been updated or clearly defined.
 - This is where RBAC will become difficult to maintain and manage. These "dead ends" limit your deployment's scalability and may require a redesign to get back on track. Even worse, with potential time pressures, you may need to implement a "work around" solution that in the long run can contribute to the problem rather than rectify it. Almost like a game of IAM whac-a-mole, you're constantly addressing new problems.
 - This more often than not is cycle of major rework every 2-3 years, if at all, to remediate a lack of incremental management of the role taxonomy in-line with the changing needs of the organization's structure, which remains dynamic and reactive, as it must, to react to customers' needs and more agile business models in a digital world.
- **Problem 4: Expensive and Difficult Implementation**
 - Your company has been using computers and collecting data for a long time but has never really needed any kind of access control as part of the organization security policy. If you need to plug the holes and decide that RBAC is the way to go, you may find the need for the duplication of servers and other infrastructures which support RBAC cost prohibitive and adding complexity. You will also need to consider cost and risk with migrating users to the new systems while phasing out the older ones. Most of the time migrations have a variety of difficulties and unforeseen challenges, and resulting in security holes in both systems along with other costly defects, such as unplanned downtime and data loss.

NIST Model: Open Issues

- Role engineering
- Standard for constraint specification
- RBAC administration
- Role revocation

References

- Ravi S. Sandhu “**Role-Based Access Control**”
- Gail - Joon Ahn and Ravi Sandhu “**Role-Based Authorization Constraints Specification**”
- Sandhu R. “**Issues in RBAC**”, 1st Workshop on Role-based Access Control, pp. 21-24, 1995.
- Sandhu R. et. al. “**Role-based Access Control Models**”. IEEE Computer, 29(2):38-47 February 1996
- Sandhu R., Ferraiolo D. and Kuhn R. “**The NIST Model for Role-Based Access Control**”.