ROLE BASED ACCESS CONTROL MODELS

Dr. Saeed Rajput

&

Reena Cherukuri

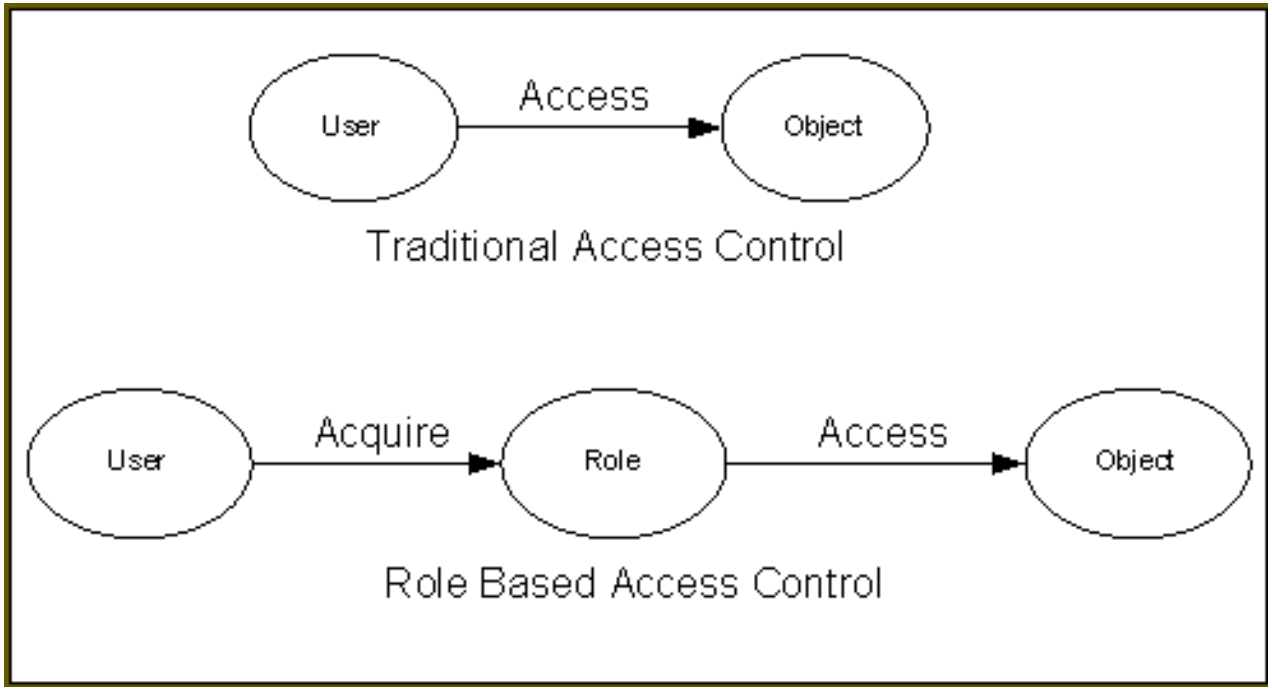
Introduction

- Access Control is a means by which the ability is explicitly enabled or restricted in some way (usually through physical and system-based controls)
- Computer-based access controls can prescribe not only who or what process may have access to a specific system resource, but also the type of access that is permitted.
- With Role-Based Access Control (RBAC), access decisions are based on the roles that individual users have as a part of an organization.
- Roles are closely related to the concept of user groups in access controls.
- Role brings together a set of users on one side and a set of permissions on the other whereas user groups are typically defined as a set of users.

Role-Based Access Control

- The concept of role-based access control began with multi-user and multi-application on-line systems pioneered in 1970's
- Introduces the concept of a **role** and a **permission**
  - A permission is an association between a transformation procedure and an object
    - A permission can be thought as an object-method pair or a class-method pair in an object-oriented environment
    - A permission can be thought as a table-query pair or a view-query pair in a database application
- Permissions are assigned to roles
- Users are assigned to roles

Role-Based Access Control



Role-Based Access Control

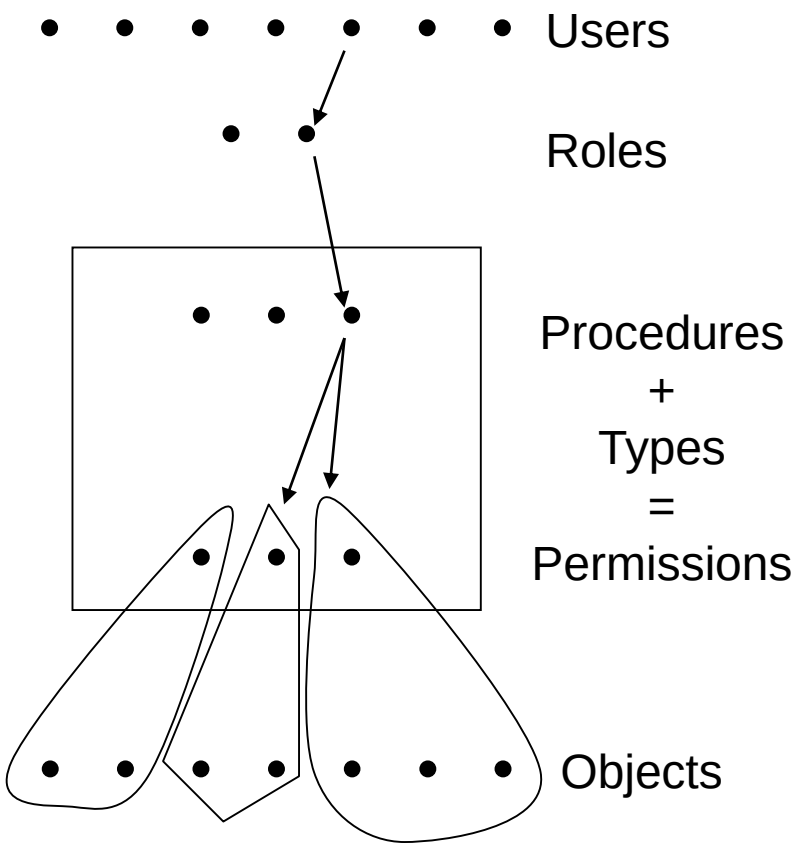
- RBAC is an access control mechanism which:
  - Describes complex access control policies.
  - Reduces errors in administration.
  - Reduces cost of administration.

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
- These components collectively determine whether a particular user will be allowed to access a particular piece of data in the system.
- RBAC components may be configured directly by the system owner or indirectly by appropriate roles as delegated by the system owner.
- The policy enforced in a particular system is the net result of the precise configuration of various RBAC components as directed by the system owner
- The ability to modify policy to meet the changing needs of an organization is an important benefit of RBAC

Role-Based Access Control

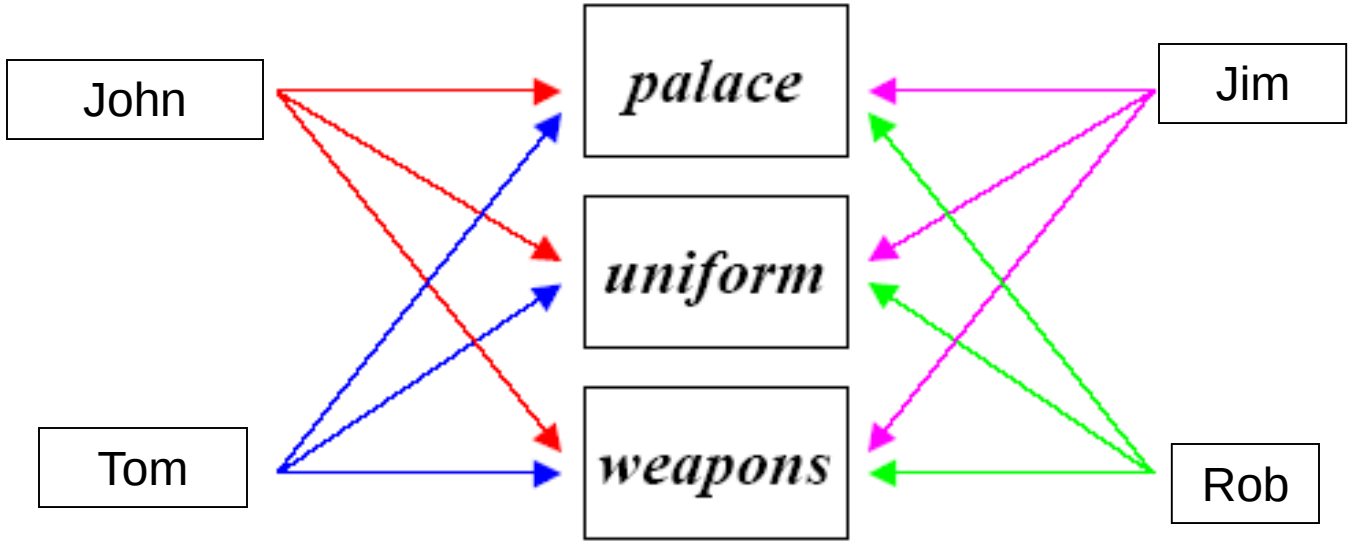
- The complexity of administration is reduced through
  - Assigning users to roles
  - Assigning permissions to roles
  - Organising roles into a hierarchy



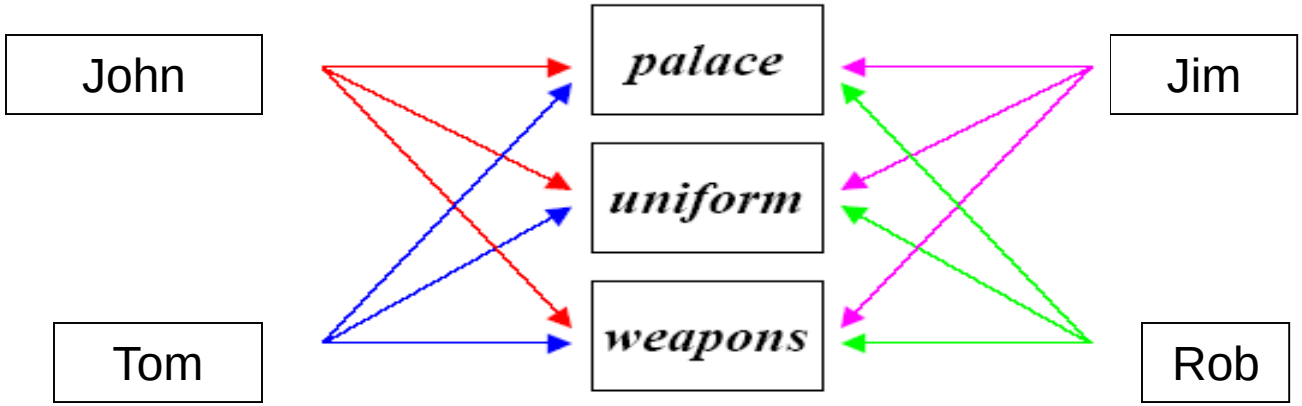
Role-Based Access Control

- RBAC supports three well-known security principles:
  - Least Privilege
  - Separation of duties
  - Data Abstraction
- Least Privilege is supported because RBAC can be configured so only those permissions required for tasks conducted by members of the role are assigned to role.
- Separation of duties is achieved by ensuring that mutually exclusive roles must be invoked to complete a sensitive task.
- Data abstraction is supported by means of abstract permissions such as credit and debit for an account.
- The degree to which data abstraction is supported will be determined by the implementation details

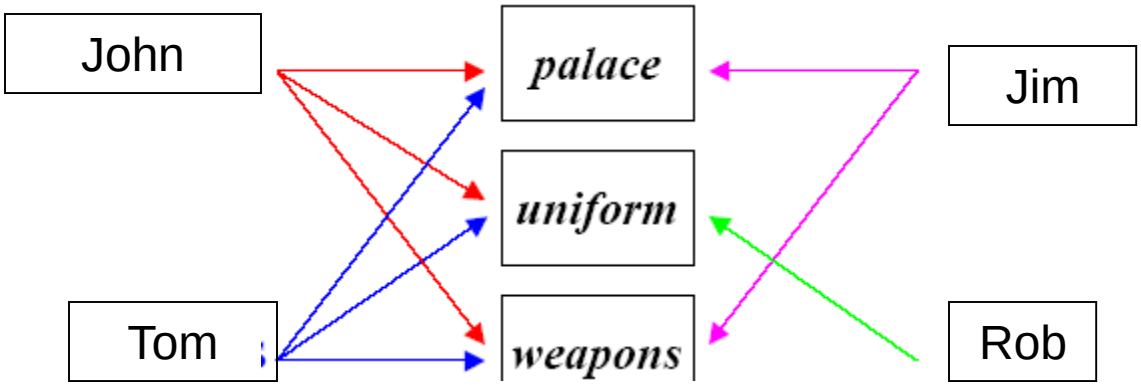
Example: The Three Musketeers (User/Permission Association)



Example: The Three Musketeers (RBAC)



Example: The Three Musketeers (RBAC)



Role-Based Access Control

- RBAC is a rich and open-ended technology, which ranges from very simple at one extreme to fairly complex and sophisticated at the other.
- Treating RBAC as a single model is therefore unrealistic.
- A single model would either include or exclude too much, and would only represent one point along a spectrum of technology and choices.
- To address the issues of scope and terminology (many models use different terminology to describe the same concepts) RBAC standardization was proposed by NIST(National Institute of Standards and Technology).

RBAC Standard

Two Main Parts
- -- RBAC Reference Models
- -- Requirement Specification

Four Components
- -- Core RBAC
- -- Hierarchical RBAC
  - --- Limited Hierarchies
  - --- General Hierarchies
- -- Static Separation of Duty Relations
  - --- Without Hierarchies
  - --- With Hierarchies
- -- Dynamic Separation of Duty Relations

RBAC Reference Model

- The standard begins with an RBAC Reference Model defining a collection of model components.
- It defines a set of basic RBAC elements (i.e. user, roles, permissions, operations, and objects) and relations as types and functions that are included in this standard.
- It serves two purposes:
  - It rigorously defines the scope of RBAC features that are included in the standard.
    - This covers the core set of features to be encompassed in all RBAC systems, aspects of role hierarchies, aspects of static constraint relations, and aspects of dynamic constraint relations.
  - It provides a precise and consistent language, in terms of element sets and functions for use in defining the functional specification

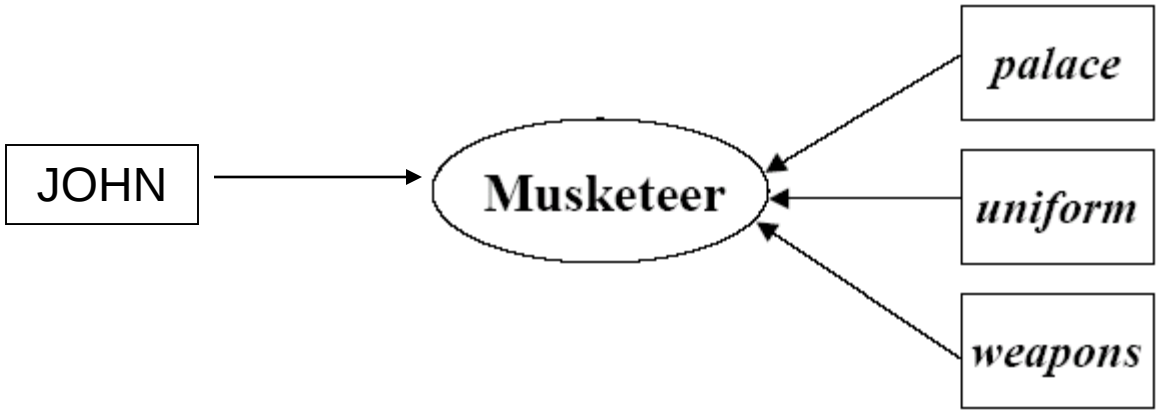
RBAC Reference Model

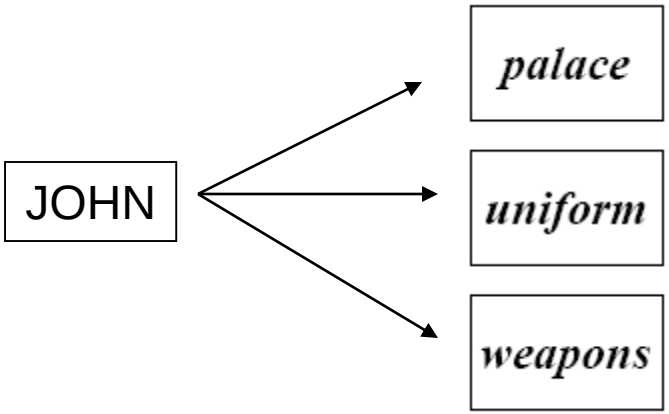
- The NIST RBAC model is defined in terms of four model components .
  - Core RBAC
  - Hierarchical RBAC
  - Static Separation of Duty Relations
  - Dynamic Separation of Duty Relations
- Each Component is defined by subcomponents:
  - Set of basic elements sets
  - A set of RBAC relations involving 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.

Core RBAC

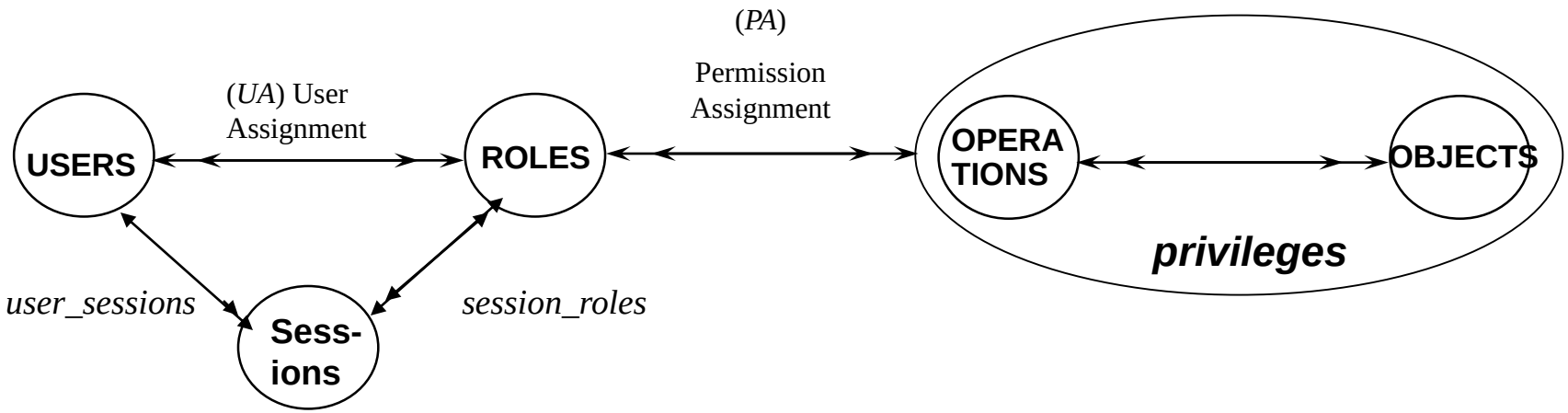
- It embodies the essential aspects of RBAC.
- The basic concept of RBAC is that users are assigned to roles, and users acquire permissions by being members of roles.
- Core RBAC includes requirements that user-role and permission-role assignment can be many-to-many.
- It includes requirements for user-role 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.

Example: (John becomes a Musketeer)





Core RBAC



- Many-to-many relationship among individual users and privileges
- Session is a mapping between a user and an activated subset of assigned roles
- User/role relations can be defined independent of role/privilege relations
- Privileges are system/application dependent
- Accommodates traditional but robust group-based access control

Core RBAC

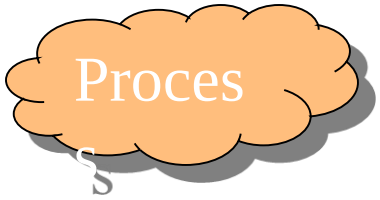
***Definition* 1. Core RBAC.**

- *USERS*, *ROLES*, *OPS*, *and OBS* (users, roles, operations, and objects, respectively).
- *UA USERS ROLES*, a many-to-many mapping user-to-role assignment relation.
- *assigned users*: (*r:ROLES*) $\rightarrow$2^*USERS*, the mapping of role *r* onto a set of users. Formally: *assigned_users*(*r*) = {u ε *USERS* (*u*, *r*) ε *UA}*.
- *PRMS* = 2 ^ (*OPS OBS*), the set of permissions.
- *PA* $\subseteq$ *PRMS x ROLES*, a many-to-many mapping permission-to-role assignment relation.
- *assigned permissions*(*r: ROLES*) $\rightarrow$ 2^*PRMS*, the mapping of role r onto a set of permissions. Formally: *assigned permissions(r)* = {u ε PRMS (p, *r*) ε P*A}*.
- *Ob*(*p*: *PRMS*) $\rightarrow${op $\subseteq$ OPS}, the permission-to-operation mapping, which gives the set of operations associated with permission *p*.
- *Ob*(*p*: *PRMS*) $\rightarrow$ {ob $\subseteq$ OBS}, the permission-to-object mapping, which give the set of objects associated with permission *p*.
- *SESSIONS*, the set of sessions.
- *user sessions* (*u*: *USERS*) $\rightarrow$ 2^*SESSIONS*, the mapping of user *u* onto a set of sessions.
- *session roles* (*s*: *SESSIONS*) $\rightarrow$ 2^*ROLES*, the mapping of session *s* onto a set of roles. Formally: *session roles* (*st*) $\subseteq$ {r ε *ROLES* | (*session_users* (*st*), *r* ε *UA}*.
- We now define role hierarchies as inheritance relationships between roles.

Core RBAC

- The RBAC model as a whole is fundamentally defined in terms of individual users being assigned to roles and permissions being assigned to roles.
- A role is a means for naming many-to-many relationships among individual users and permissions.
- In addition it includes a set of sessions where each session is a mapping between a user and an activated subset of roles that are assigned to user.
- The type of operations and objects that RBAC controls are dependent on the type of the system in which they are implemented.
- The set of objects covered by RBAC includes all the objects listed in the permissions that are assigned to roles.

USERS

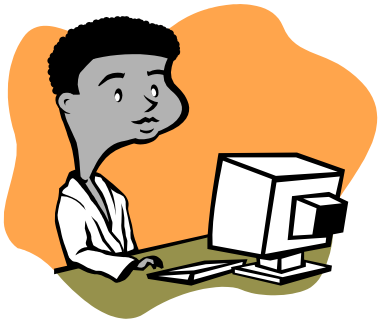


Person

Intelligent Agent

ROLES

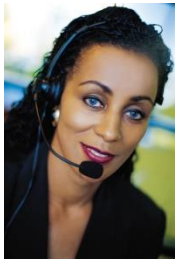
An organizational job function with a clear definition of inherent responsibility and authority (permissions).



Developer

Budget Manager

Director



Help Desk Representative

Relation between USERS & PRMS

OPERATIONS

An execution of an a program specific function that's invocated by a user.

- ✓ Database – Update Insert Append
- ✓ Delete Locks – Open Close
- ✓ Reports – Create View Print
- ✓ Applications - Read Write Execute




OBJECTS

An entity that contains or receives information, or has exhaustible system resources.

- OS Files or Directories
- DB Columns, Rows, Tables, or Views
- Printer
- Disk Space
- Lock Mechanisms

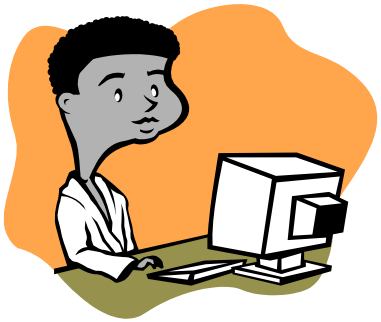
RBAC will deal with all the objects listed in the permissions assigned to roles.

USER Assignment

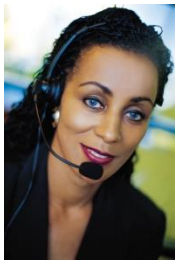
USERS set

A user can be assigned to one or more roles

ROLES set

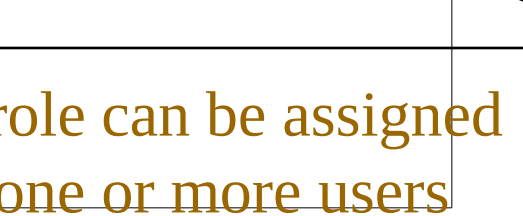


Developer



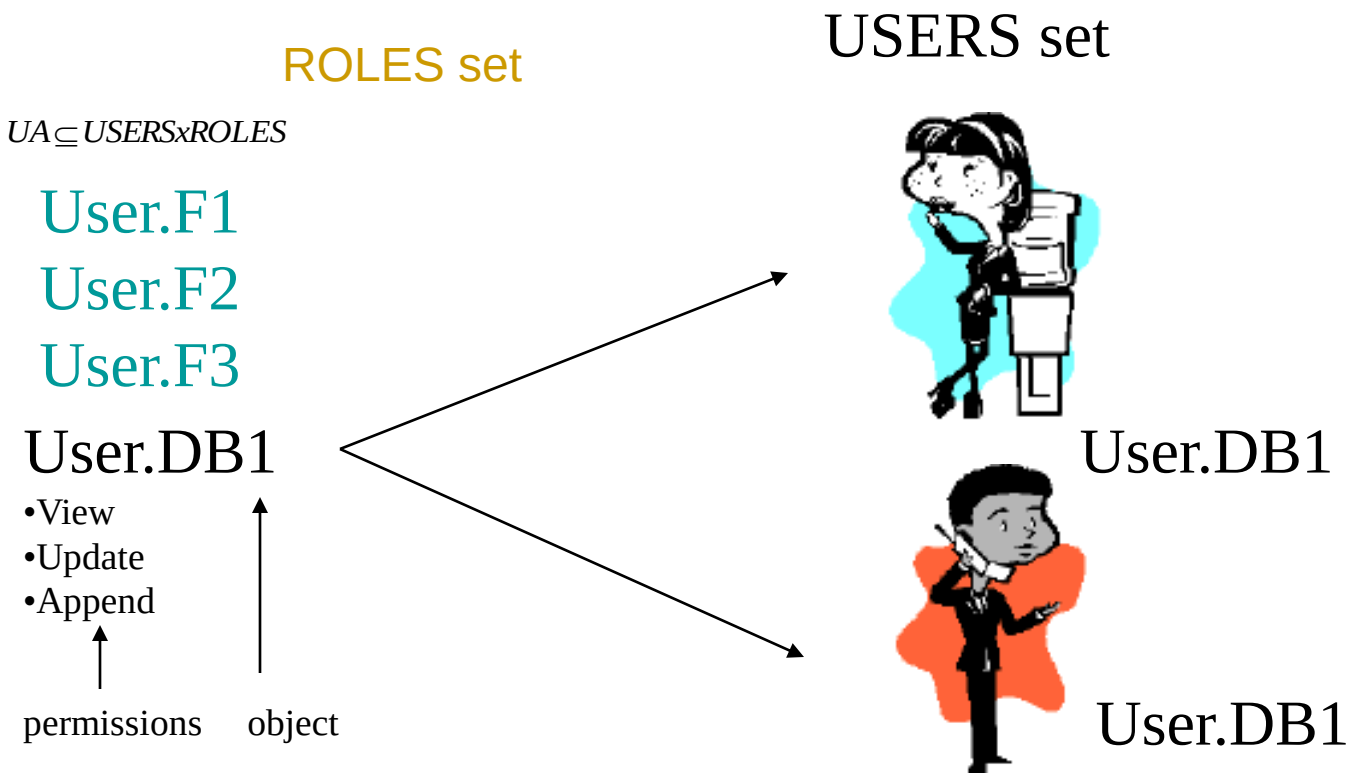
Help Desk Rep

A role can be assigned to one or more users



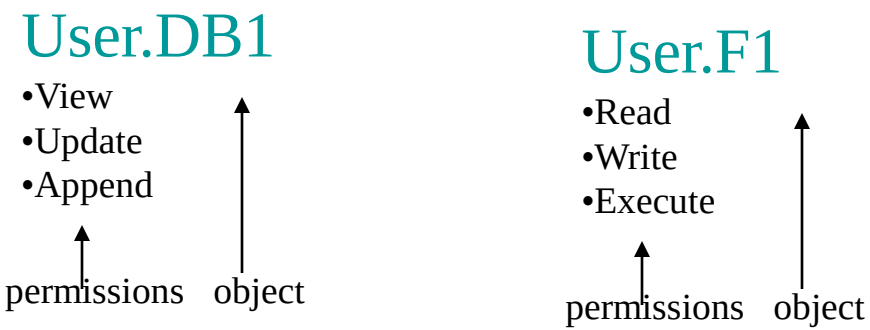
USER Assignment

Mapping of role *r* onto a set of users



PERMISSIONS

The set of permissions that each grant the approval to perform an operation on a protected object.

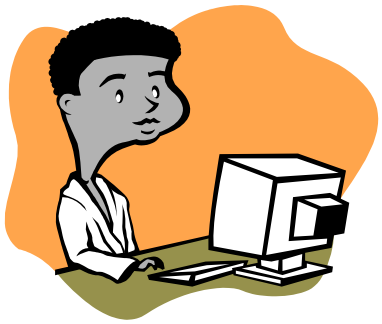


Permissions Assignment

PRMS set

A prms can be assigned to one or more roles

ROLES set



Admin.DB1

Create
Delete
Drop

View
Update
Append

A role can be assigned to one or more prms

User.DB1

Permissions Assignment

Mapping of role *r* onto a set of permissions

ROLES set

PRMS set

$UA \subseteq USERS \times ROLES$

User.F1

User.F2

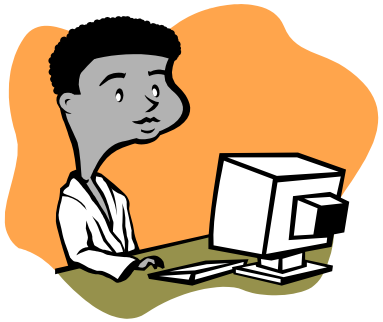
User.F3

Admin.DB1

•Read
•Write
•Execute

•View
•Update
•Append
•Create
•Drop

SQL



Permission Assignments

Mapping of permissions to objects

Gives the set of objects associated with the prms

PRMS set

Objects

Open
Close

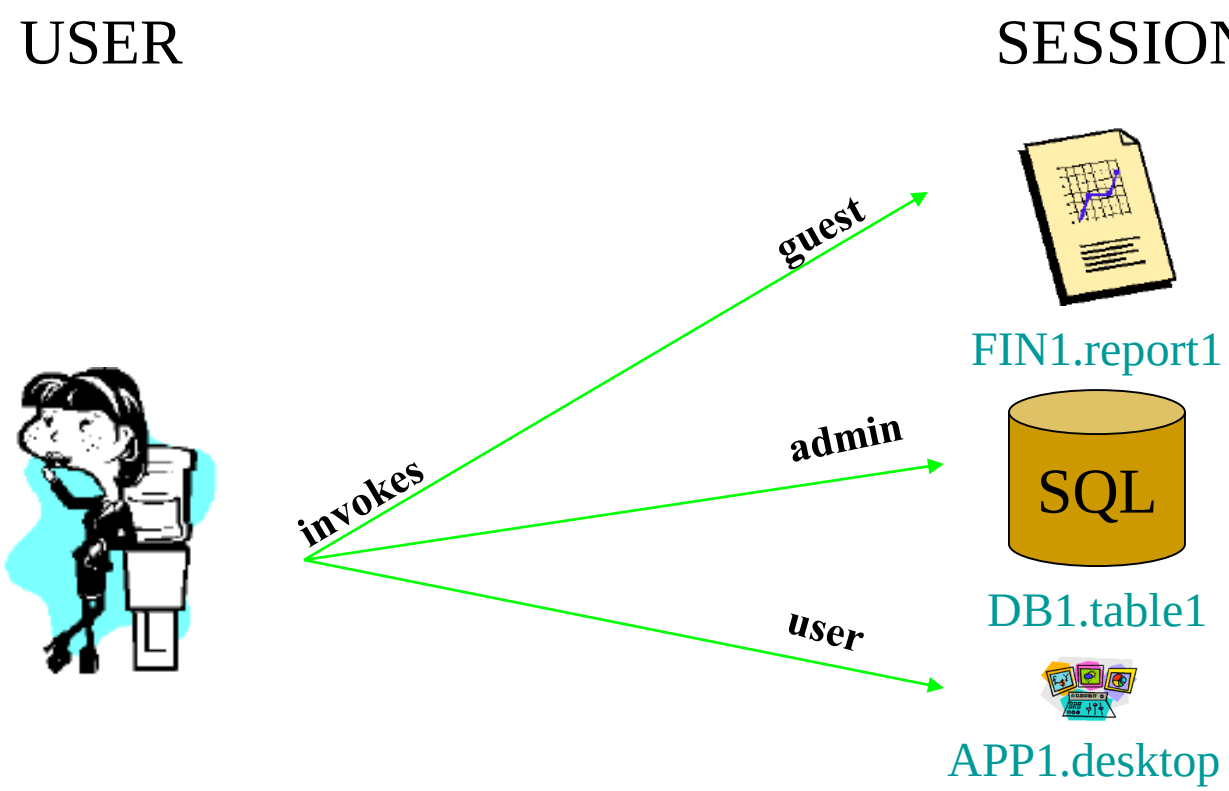
BLD1.door2

View
Update
Append
Create
Drop

DB1.table1

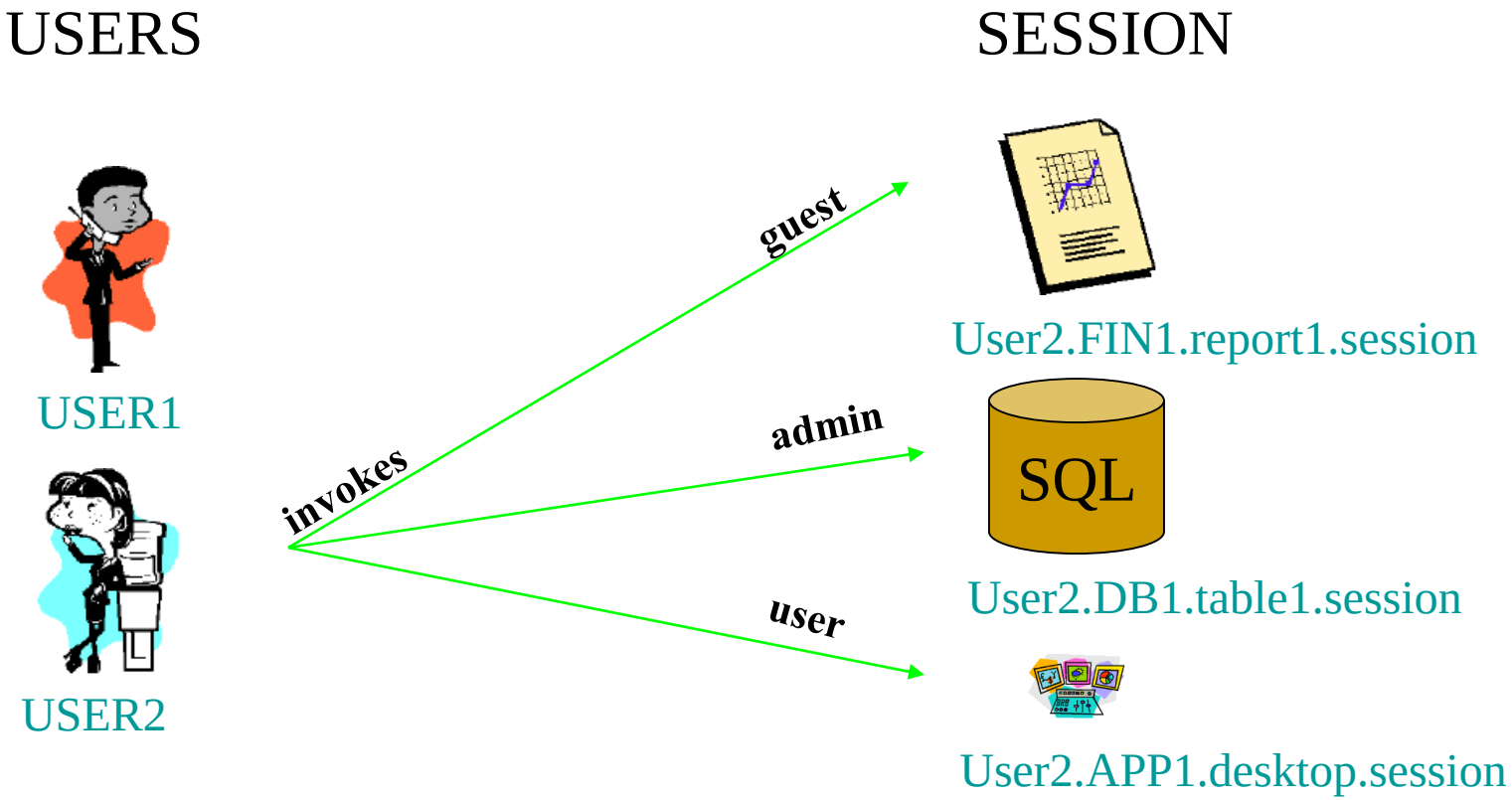
SESSIONS

The set of sessions that each user invokes.



SESSIONS

The mapping of user *u* onto a set of sessions.



SESSIONS

SESSION

ROLES

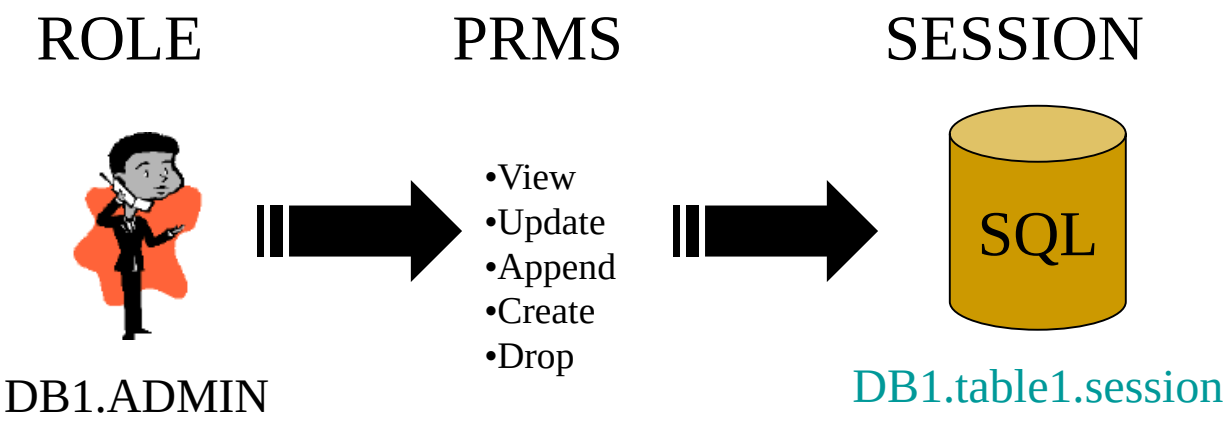
- Admin
- User
- Guest

DB1.table1.session




SESSIONS

Permissions available to a user in a session.

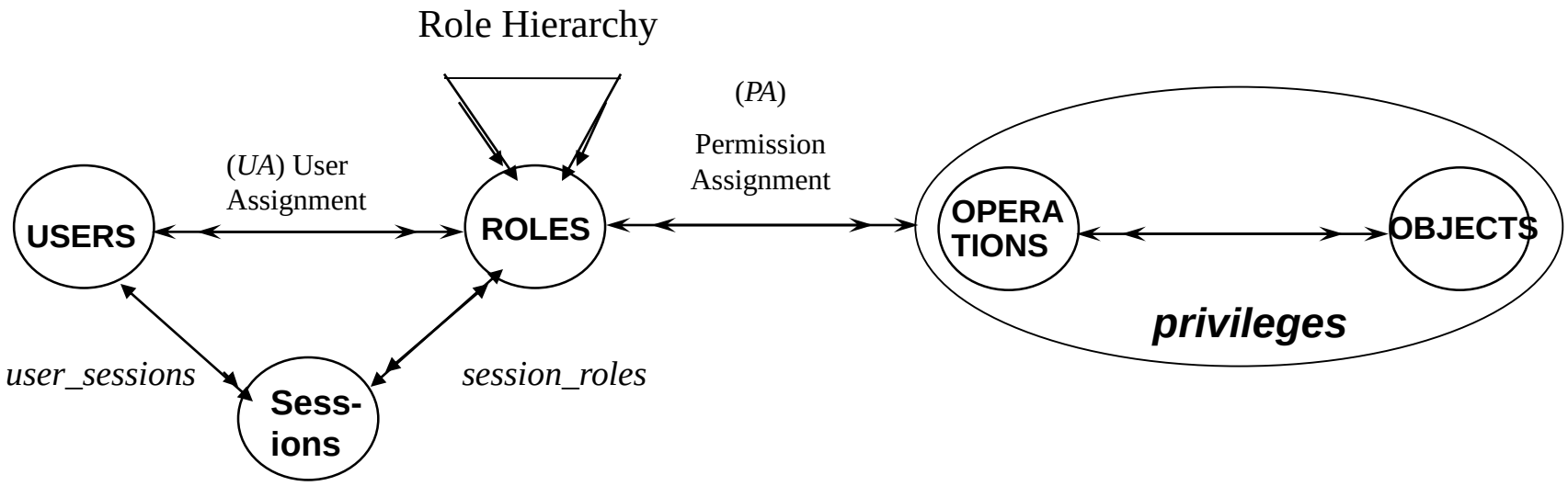


Hierarchical RBAC

- It adds requirements for supporting role hierarchies. A hierarchy is mathematically a partial order defining a seniority relation between roles, whereby the seniors roles acquire the permission of their juniors, and junior roles acquire the user membership of their seniors. This standard recognizes two types of role hierarchies

  - General Hierarchical RBAC: In this case, there is support for an arbitrary partial order to serve as role hierarchy, to include the concept of multiple inheritance of permissions and user membership among roles.
  - Limited Hierarchical RBAC: Some systems may impose restrictions on the role hierarchy. Most commonly, hierarchies are limited to simple structures such as trees and inverted trees

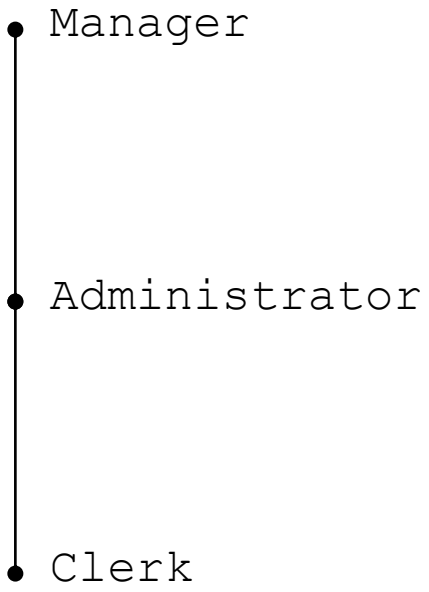
Hierarchical RBAC



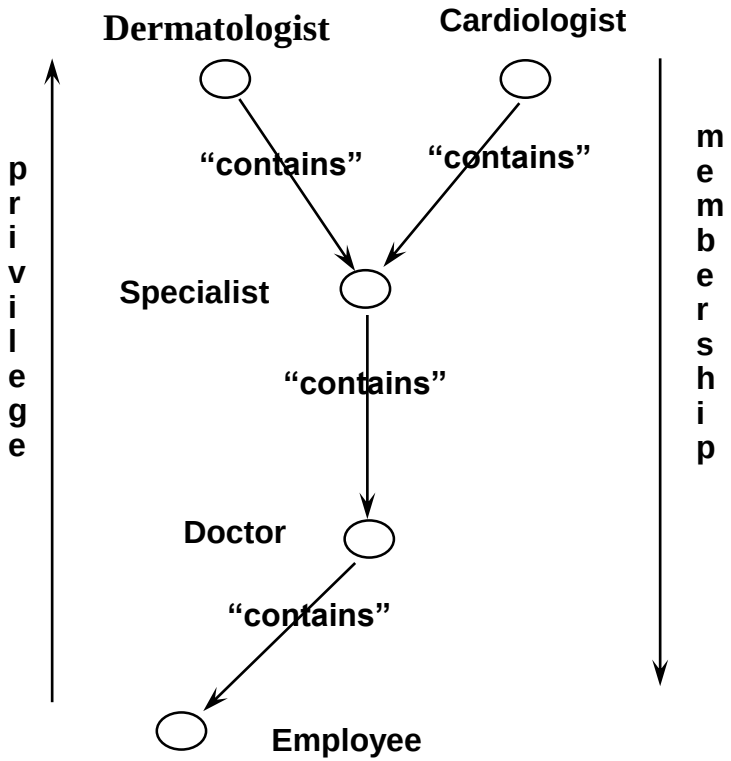
- Role/role relation defining user membership and privilege inheritance
- Reflects organizational structures and functional delineations
- Two types of hierarchies:
  - - Limited hierarchies
  - - General hierarchies

The Role Hierarchy

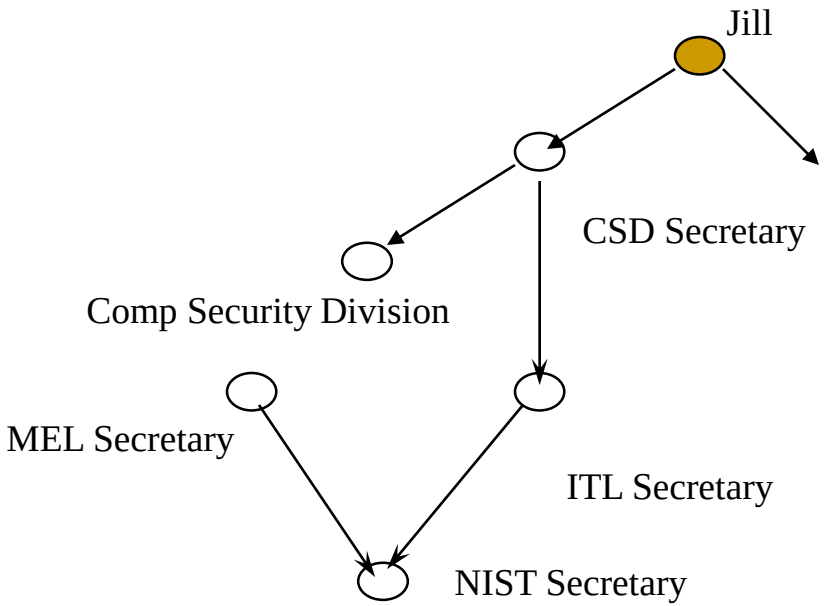
- The set of roles is partially ordered
- Models hierarchical structure of enterprise
- Aggregates permissions and implicitly assigns users to roles
  - Further simplifies administration
  - The `Manager` role inherits the permissions of the `Administrator` and `Clerk` roles
  - A user assigned to the `Manager` role can activate the `Administrator` or `Clerk` role
- Separation of duty can be defined on roles
  - No user can be assigned to both the purchase order clerk and financial clerk roles



The Role Hierarchy



a-Limited Hierarchies

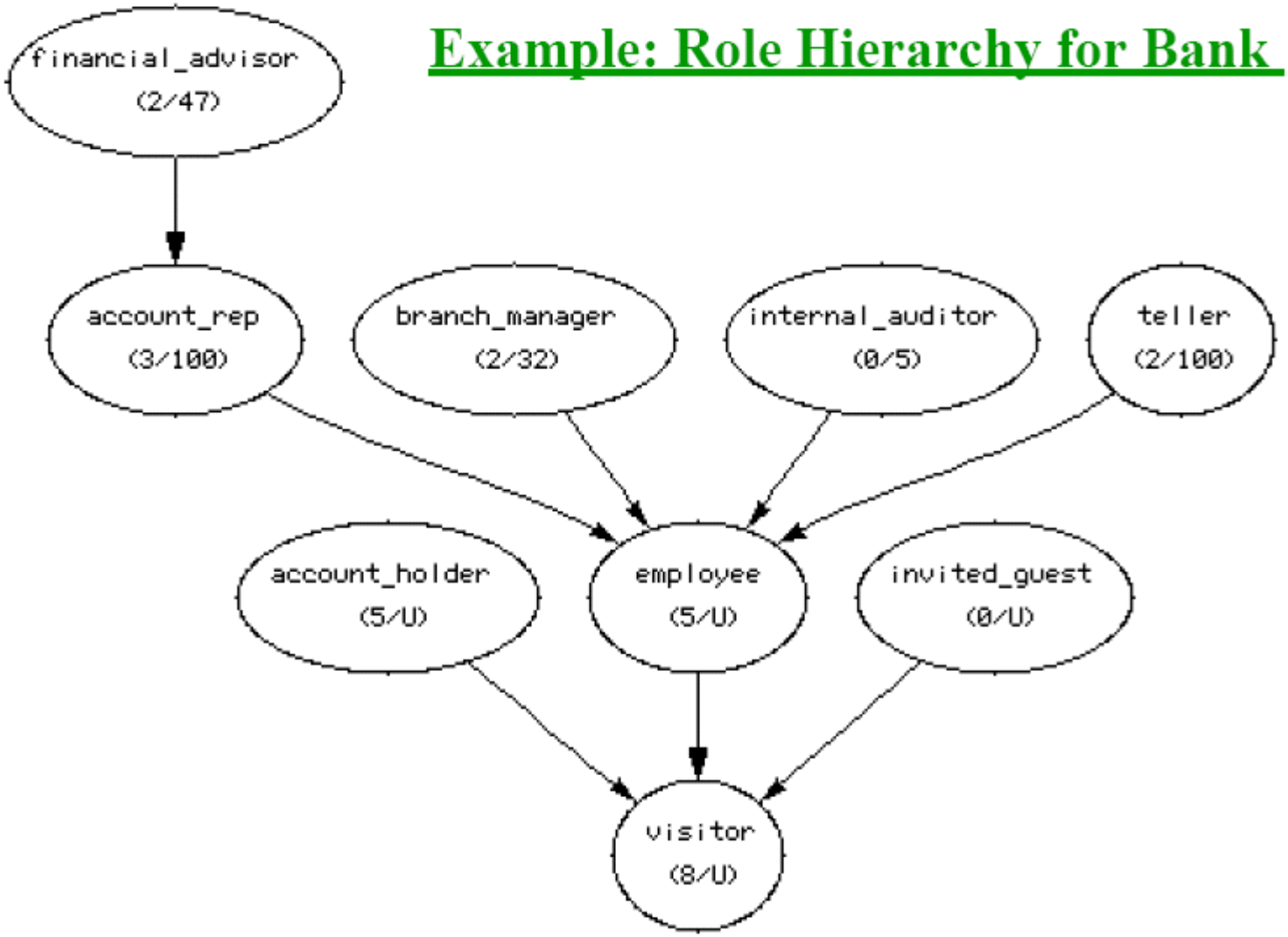


Added Advantages:

- User's can be included on edges of graph
- Role's can be defined from the privileges of two or more subordinate roles

b-General Hierarchies

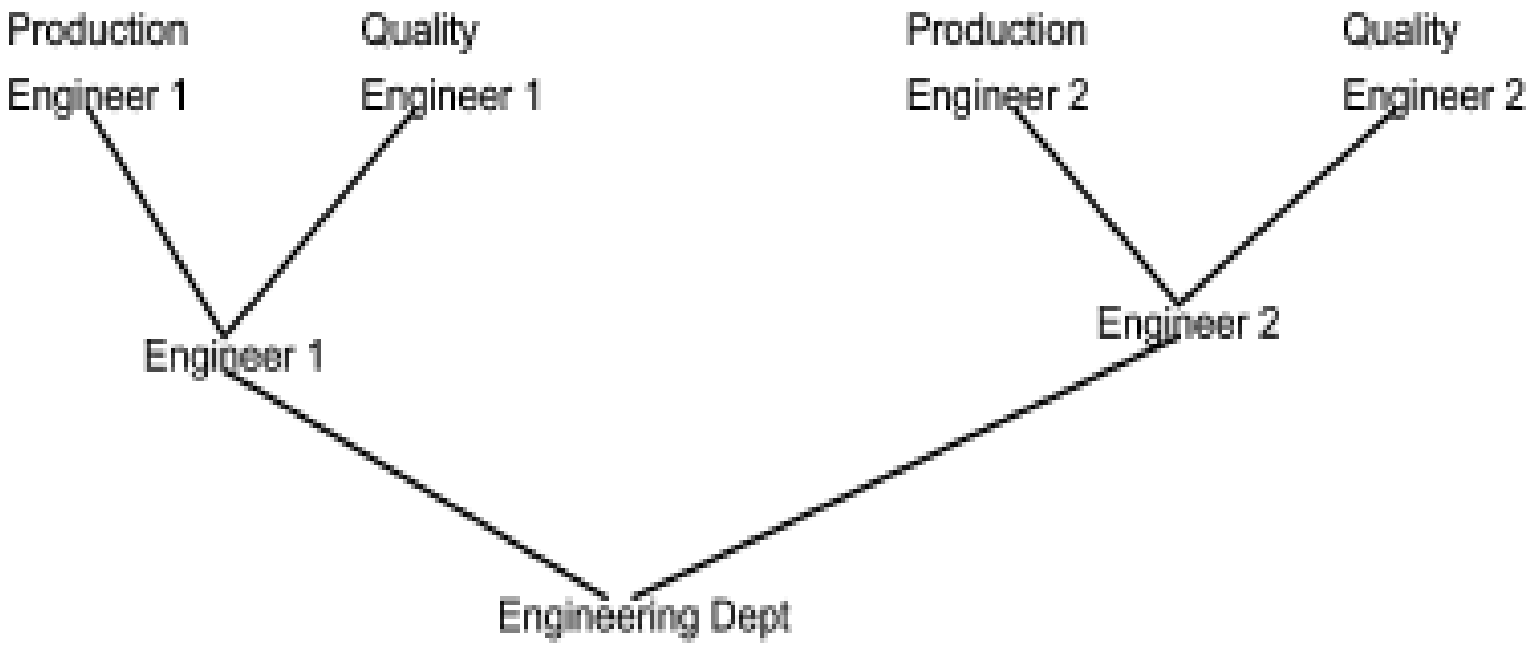
Example: Role Hierarchy for Bank



Hierarchal RBAC

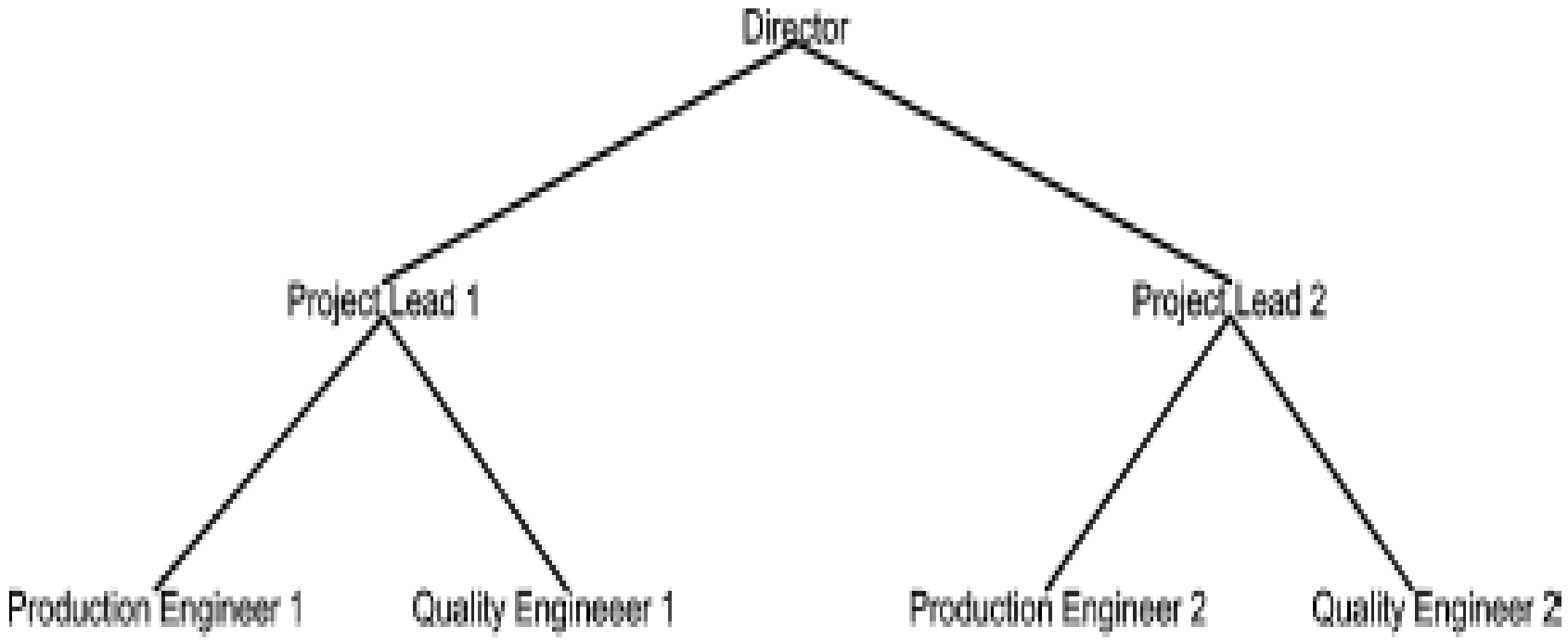
- Role hierarchies define an inheritance relation among roles. Inheritance has been described in terms of permissions that is *r*1 “inherits” role *r*2 if all privileges of *r*2 are also privileges of *r*1.
- This standard recognizes two type of hierarchies
  - General role Hierarchy
  - Limited role Hierarchy
- General role hierarchies provide support for an arbitrary partial order to serve as the role hierarchy, to include the concept of multiple inheritances of permissions and user membership among roles.
- Limited role hierarchies impose restrictions resulting in a simpler tree structure (i.e., a role may have one or more immediate ascendants, but is restricted to a single immediate descendant).

Hierarchal RBAC



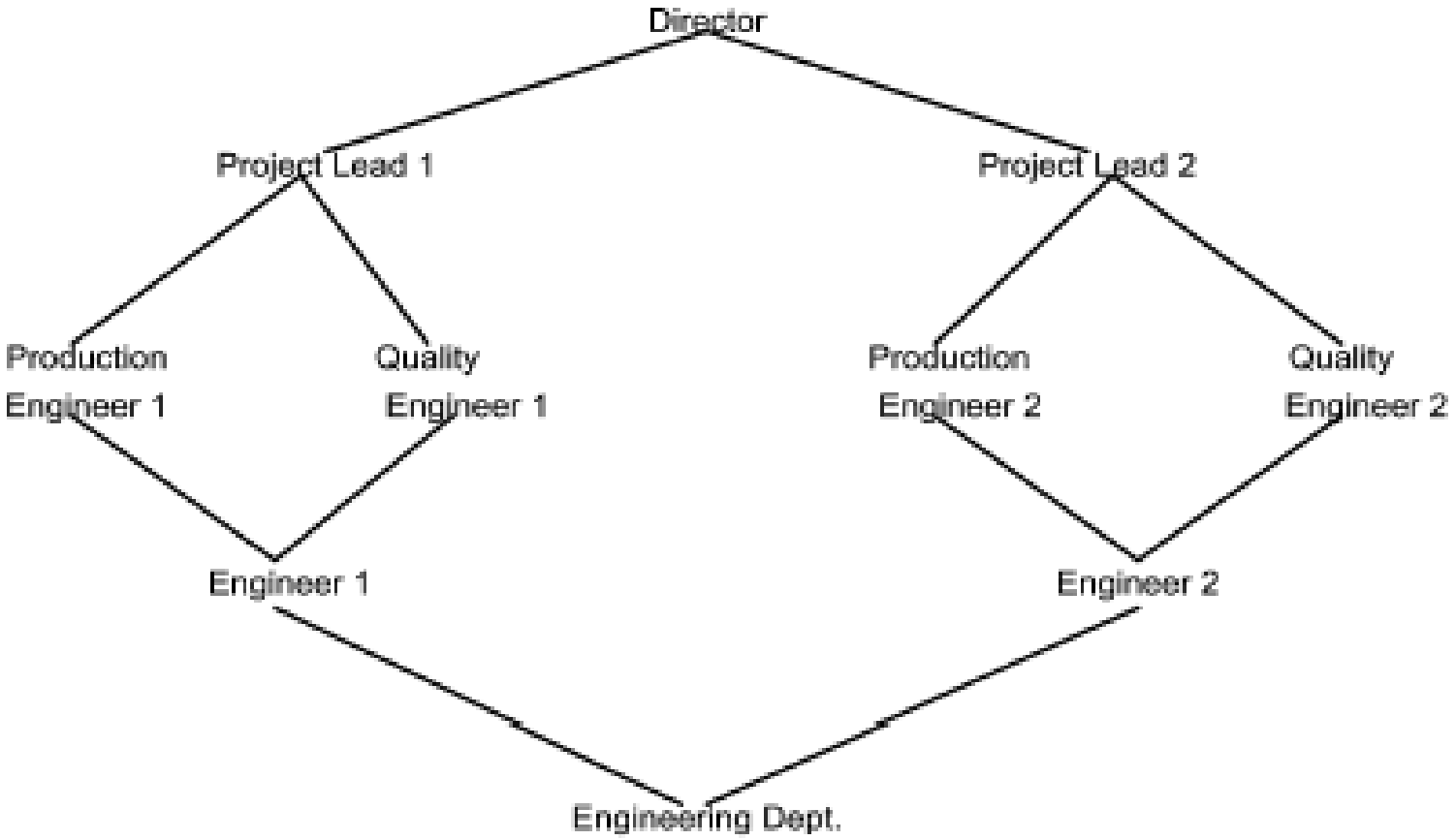
(a)

Hierarchal RBAC



(b)

Hierarchal RBAC

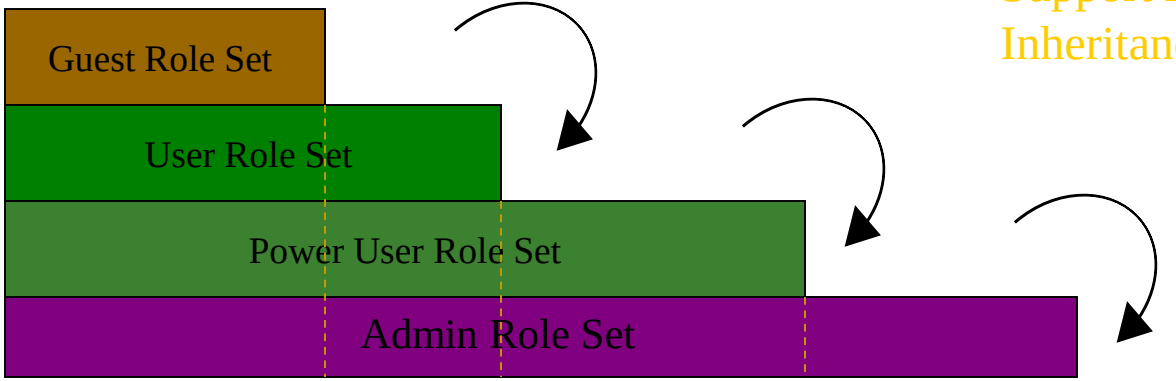


(c)

General Role Hierarchies

- General role hierarchies support the concept of *multiple inheritance*, which provides the ability to inherit permission from two or more role sources and to inherit user membership from two or more role sources. Multiple inheritances provide important hierarchy properties.
  - The first is the ability to compose a role from multiple subordinate roles (with fewer permissions) in defining roles and relations that are characteristic of the organization and business structures, which these roles are intended to represent.
  - Second, multiple inheritances provide uniform treatment of user/role assignment relations and role/role inheritance relations. Users can be included in the role hierarchy, using the same relation to denote the user assignment to roles, as well as permission inheritance from a role to its assigned users.

General RH



Support Multiple Inheritance

Only if all permissions of r_1 are also permissions of r_2

i.e. r1 *inherits* r2

Only if all users of r_1 are also users of r_2

User
r-w-h

Guest
-r-

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

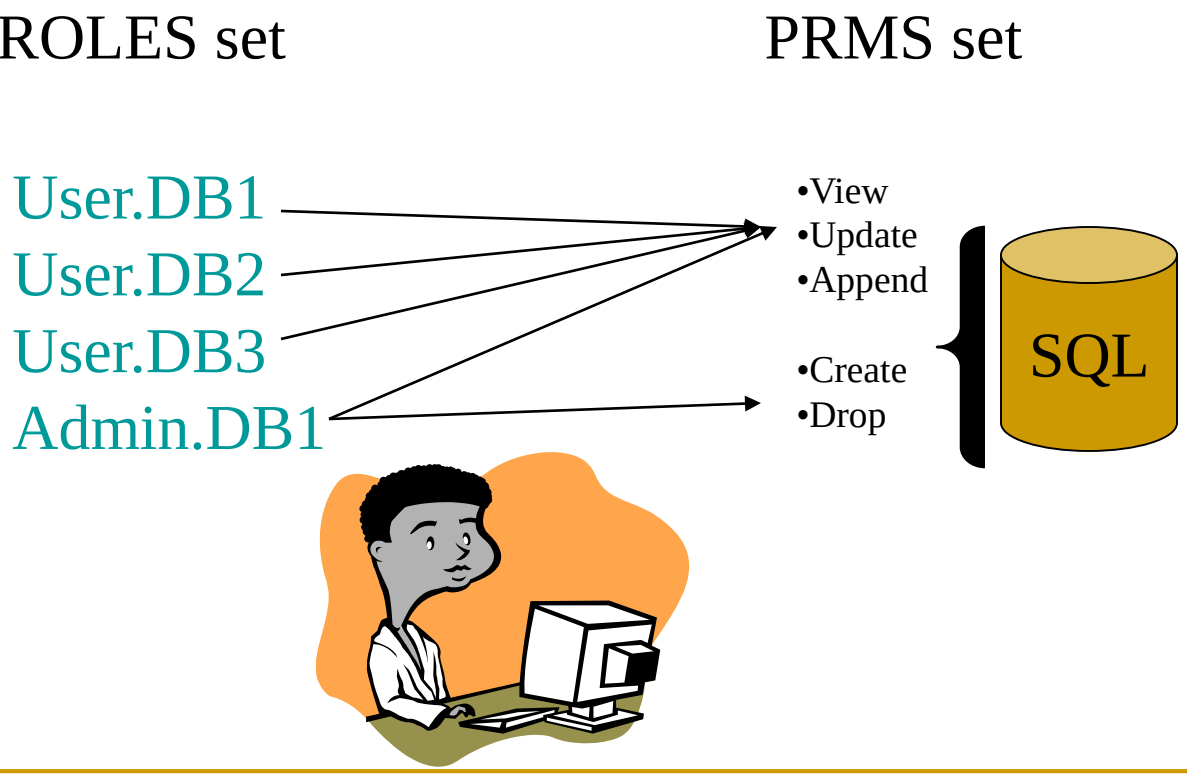
First Tier USERS set

User.DB1

User.DB1

Authorized Permissions

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

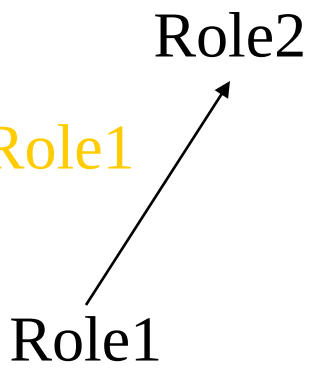


Limited Role Hierarchies

- Roles in a limited role hierarchy are restricted to a single *immediate descendant*. Although limited role hierarchies do not support multiple inheritances, they nonetheless provide clear administrative advantages over Core RBAC.
- We represent $r1$ as an immediate descendent of $r2 > > r1$ $r2$, if $r1 \geq r2$, but no role in the role hierarchy lies between $r1$ and $r2$. That is, there exists no role $r3$ in the role hierarchy such that $r \geq r3 \geq r2$, where $r1 \neq r2$ and $r2 \neq r3$.
- Definition of limited Role Hierarchy:
- $\forall\ r, r1, r2\ \varepsilon\ ROLES, r \geq r1\ \nabla\ r \geq r2\ )r1 = r2:$

LIMITED RH

Restriction on the immediate descendants of the general role hierarchy

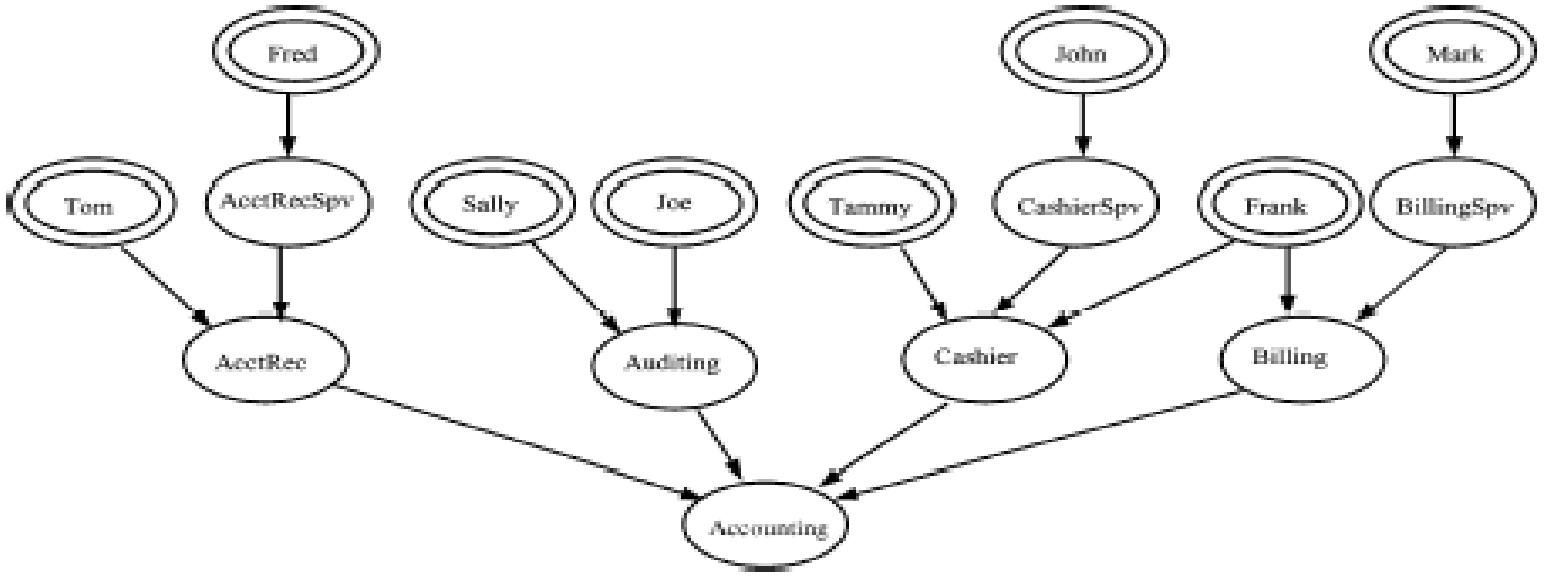


Role2 inherits from Role1

- Role3

Role3 does not inherit from Role1 or Role2

Limited Role Hierarchies



Notice that Frank has two roles: Billing and Cashier

This requires the union of two distinct roles and prevents Frank from being a node to others

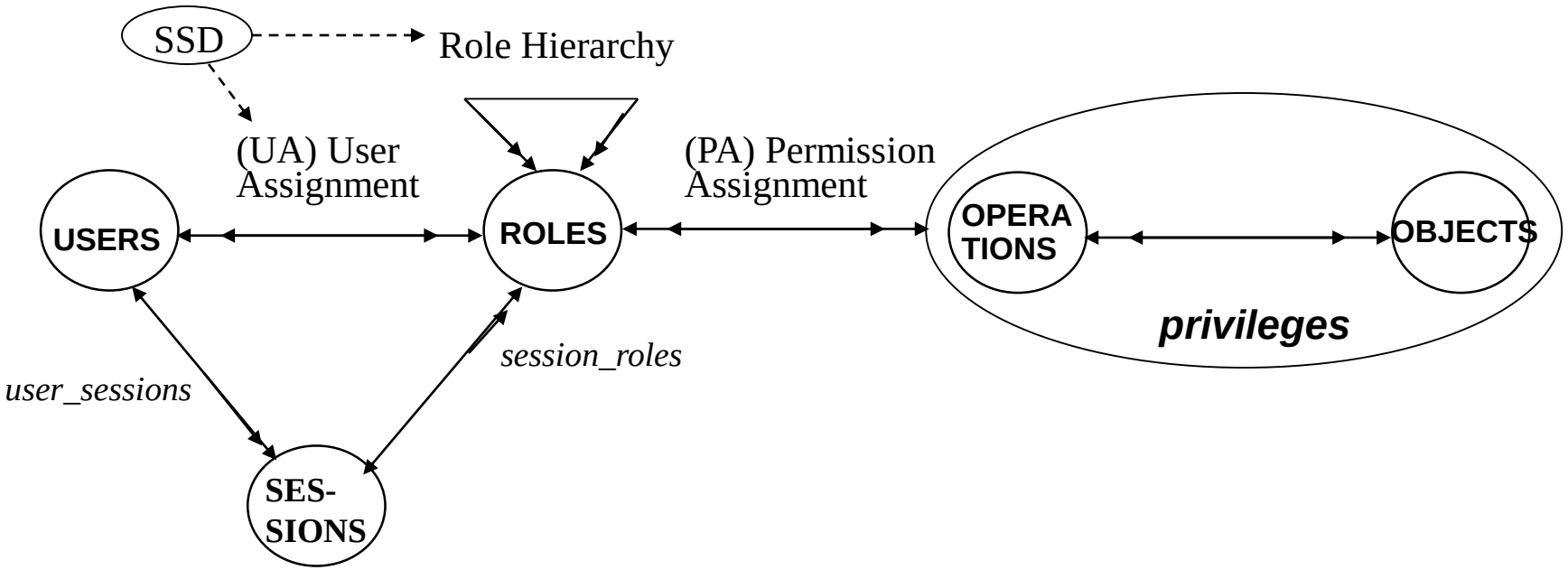
Constrained RBAC

- It adds separation of duty relations to the RBAC model.
- As a security principle, SOD has long been recognized for its wide application in business, industry, and government.
- Its purpose is to ensure that failures of omission or commission within an organization can be caused only as a result of collusion among individuals.
- To minimize the likelihood of collusion, individuals of different skills or divergent interests are assigned to separate tasks required in the performance of a business function.
- The motivation is to ensure that fraud and major errors cannot occur without deliberate collusion of multiple users.
- This RBAC standard allows for both static and dynamic separation of duty

Static Separation of Duty Relations

- Separation of duty relations are used to enforce conflict of interest policies. Conflict of interest in a role-based system may arise as a result of a user gaining authorization for permissions associated with conflicting roles.
- One means of preventing this form of conflict of interest is though *static separation of duty* (SSD), that is, to enforce constraints on the assignment of users to roles.
  - An example of such a static constraint is the requirement that two roles be mutually exclusive; for example, if one role requests expenditures and another approves them, the organization may prohibit the same user from being assigned to both roles.
- The SSD policy can be centrally specified and then uniformly imposed on specific roles. Because of the potential for inconsistencies with respect to static separation of duty relations and inheritance relations of a role hierarchy, we define SSD requirements both in the presence and absence of role hierarchies.

Static Separation of Duty Relations



SoD policies deter fraud by placing constrains on administrative actions and there by 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

Static Separation of Duty Relations

- ***Static Separation of Duty*.** SSD relations place constraints on the assignments of users to roles. Membership in one role may prevent the user from being a member of one or more other roles, depending on the SSD rules enforced.

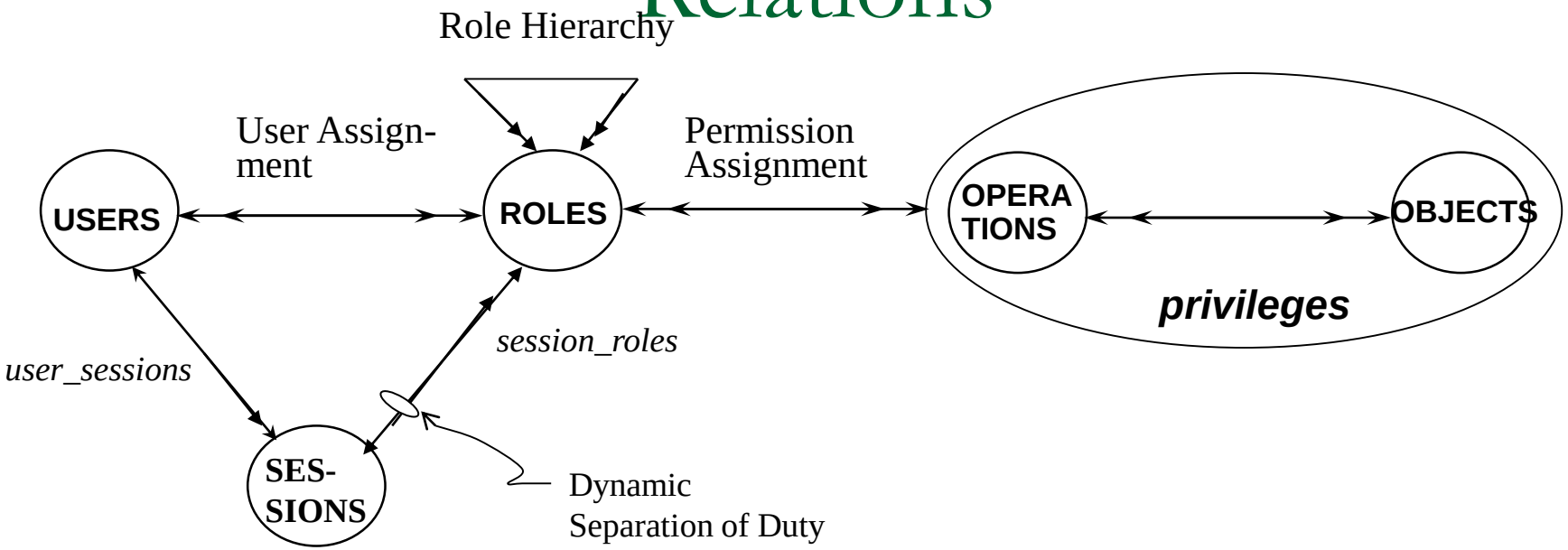
- ***Static Separation of Duty in the Presence of a Hierarchy***. This type of SSD relation works in the same way as basic SSD except that both inherited roles as well as directly assigned roles are considered when enforcing the constraints.

Dynamic Separation of Duty Relations

- Dynamic separation of duty (DSD) relations, like SSD relations, limit the permissions that are available to a user. However DSD relations differ from SSD relations by the context in which these limitations are imposed.

- DSD requirements limit the availability of the permissions by placing constraints on the roles that can be activated within or across a user's sessions.

Dynamic Separation of Duty Relations



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

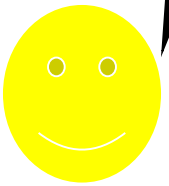
Dynamic Separation of Duty Relations

Roles

inherits

Cashier Supervisor

Reduce COI



Supervisor

Cashier

Closes Cashier Role session

Close Cash Drawer

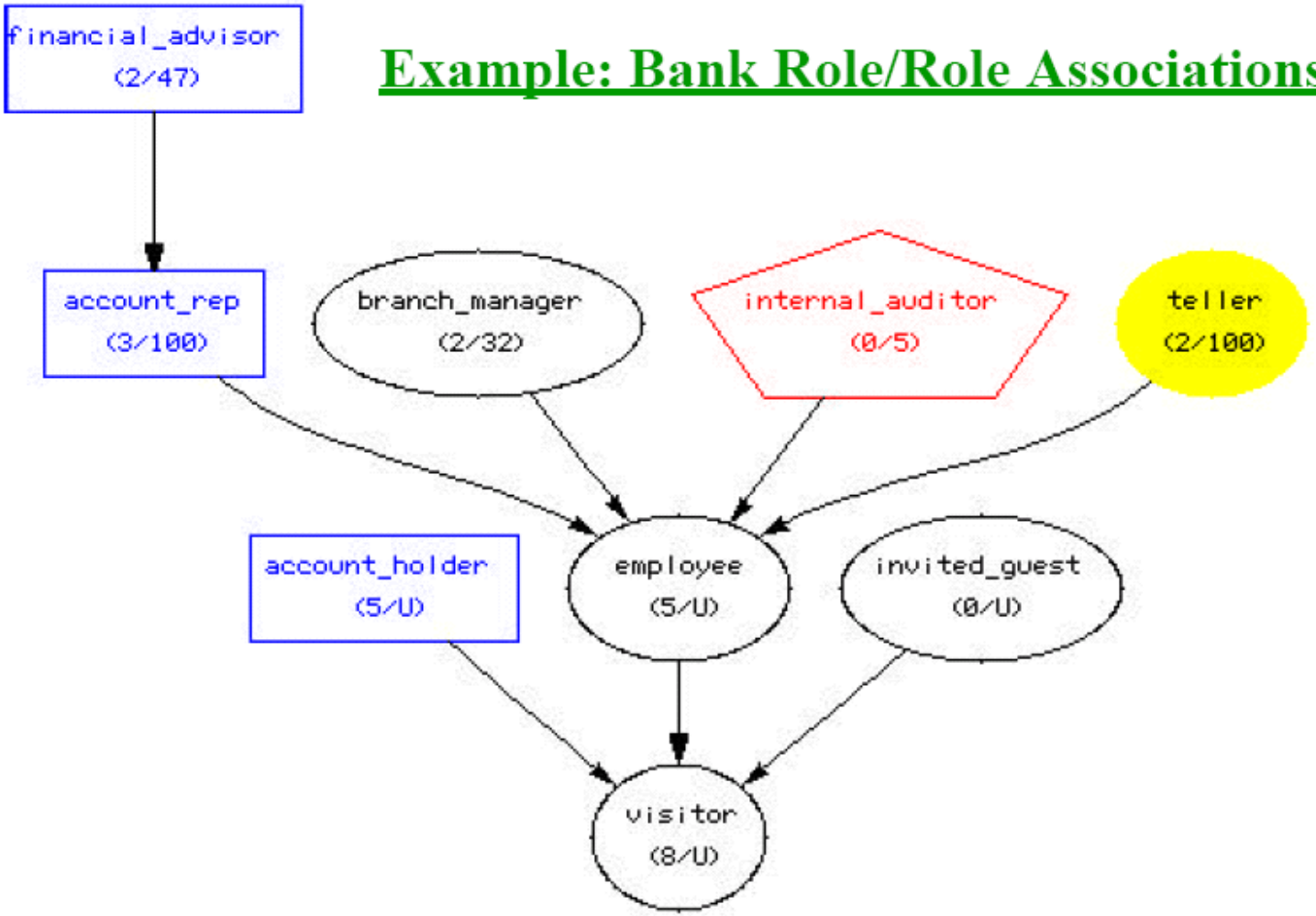
Opens Supv Role session

Open Cash Drawer

Accounting Error

Correct Error

Example: Bank Role/Role Associations



RBAC Functional Specification

- It defines the features required of an RBAC system.
- These features fall into three categories
  - Administrative Operations
    - Administrative operations define requirements in terms of an administrative interfaces and an associated set of semantics that provide the capability to create, delete and maintain RBAC elements and relations.
  - Administrative Reviews
    - The administrative review features define requirements in terms of an administrative interfaces and an associated set of semantics that provide the capability to perform query operations on RBAC elements and relations.
  - System level functionality
    - The System level functionality defines features for the creations of user sessions to include role activation/deactivation, the enforcement of constraints on role activation, and for calculation of an access decision.

Functional Specification of Core RBAC

Administrative Functions

- Creation and Maintenance of Element Sets
  - The basic element sets in Core RBAC are USERS, ROLES, OPERATIONS, and OBJECTS. Of these element sets, OPERATIONS and OBJECTS are considered predefined by the underlying information system for which RBAC is deployed. Administrators create and delete USERS and ROLES, and establish relationships between roles and existing operations and objects. Required administrative functions for USERS are AddUser and DeleteUser, and for ROLES are AddRole and DeleteRole.
- *Creation and Maintenance of Relations*.
  - The main relations of Core RBAC are (a) user-to-role assignment relation (UA) and (b) permission-to-role assignment relation (PA). Functions to create and delete instances of UA relations are AssignUser and DeassignUser. For PA the required functions are Grant- Permission and RevokePermission.

Functional Specification of Core RBAC

Supporting System Functions

- Supporting system functions are required for session management and in making access control decisions.
- An active role is necessary for regulating access control for a user in a session. The function that creates a session establishes a default set of active roles for the user at the start of the session. The composition of this default set can then be altered by the user during the session by adding or deleting roles.
- Functions relating to the adding and dropping of active roles and other auxiliary functions are:

  —CreateSession: creates a User Session and provides the user with a default set of active roles;

  —AddActiveRole: adds a role as an active role for the current session;

  —DropActiveRole: deletes a role from the active role set for the current session

  —CheckAccess: determines if the session subject has permission to perform the requested operation on an object.

Functional Specification of Core RBAC

Review Functions

- When user-to-role assignment and permission to-role relation instances have been created, it should be possible to view the contents of those relations from both the user and role perspectives.
- It should be possible to view the results of the supporting system functions to determine some session attributes such as the active roles in a given session or the total permission domain for a given session.
  - —AssignedUsers (M): returns the set of users assigned to a given role;
  - —AssignedRoles (M): returns the set of roles assigned to a given user;
  - —RolePermissions (O): returns the set of permissions granted to a given role;
  - —UserPermissions (O): returns the set of permissions a given user gets through his or her assigned roles;
  - —SessionRoles(O): returns the set of active roles associated with a session;
  - —SessionPermissions (O): returns the set of permissions available in the session
  - —RoleOperationsOnObject (O): returns the set of operations a given role may perform on a given object
  - —UserOperationsOnObject (O): returns the set of operations a given user may perform on a given object (obtained either directly or through his or her assigned roles).

Functional Specification for Hierarchical RBAC

Administrative Functions

- The administrative functions required for hierarchical RBAC include all the administrative functions that were required for Core RBAC.
- The additional administrative functions required for the Hierarchical RBAC model pertain to creation and maintenance of the partial order relation (RH) among roles.
  - —AddInheritance: establish a new immediate inheritance relationship between two existing roles;
  - —DeleteInheritance: delete an existing immediate inheritance relationship between two roles;
  - —AddAscendant: create a new role and add it as an immediate ascendant of an existing role;
  - —AddDescendant: create a new role and add it as an immediate descendant of an existing role.

Functional Specification for Hierarchical RBAC

Supporting System Functions

- The Supporting System Functions for Hierarchical RBAC are the same as for Core RBAC and provide the same functionality.
- Due to the presence of a role hierarchy, the functions *CreateSession* and *AddActiveRole* have to be redefined.
- In a role hierarchy, a given role may inherit one or more of the other roles.
- ***CreateSession function***, the active role set created as a result of the new session shall include not only roles directly assigned to a user but also some or all of the roles inherited by those “directly assigned roles” (that were previously included in the default Active Role Set) as well.
- ***AddActiveRole function***, a user can activate a directly assigned role or one or more of the roles inherited by the “directly assigned role.”

Functional Specification for Hierarchical RBAC

Review Functions

- In addition to the review functions mentioned in Core RBAC, the user membership set for a given role includes not only users directly assigned to that *given role* but also those users assigned *to roles that inherit the given role*.
- To capture this expanded “User Memberships for Roles” and “Role Memberships for a User” the following functions are defined.
  - —AuthorizedUsers: returns the set of users directly assigned to a given role as well as those who were members of those “roles that inherited the given role.”
  - —AuthorizedRoles: returns the set of roles directly assigned to a given user as well as those “roles that were inherited by the directly assigned roles.”

Functional Specification for Hierarchical RBAC

Review Functions

- Because of the presence of partial order among the roles, the permission set for a given role includes not only the permissions directly assigned to a given role but also permissions obtained from the roles that the given role inherited.
- “Permissions Review” functions are listed below:
  - —RolePermissions: returns the set of all permissions either directly granted to or inherited by a given role;
  - —UserPermissions: returns the set of permissions of a given user through his or her authorized roles (union of directly assigned roles and roles inherited by those roles);
  - —RoleOperationsOnObject: returns the set of operations a given role may perform on a given object (obtained either directly or by inheritance); and
  - —UserOperationsOnObject: returns the set of operations a given user may perform on a given object (obtained directly or through his or her assigned roles or through roles inherited by those roles).

Functional Specification for SSD Relation

Administrative Functions

- —CreateSSDSet: creates a named instance of an SSD relation;
- —DeleteSSDSet: deletes an existing SSD relation;
- —AddSSDRoleMember: adds a role to a named SSD role set;
- —DeleteSSDRoleMember: deletes a role from a named SSD role set; and
- —SetSSDCardinality: sets the cardinality of the subset of roles from the named SSD role set for which common user membership restriction applies.

Supporting System Functions.

- The Supporting System Functions for an SSD RBAC Model are the same as those for the Core RBAC Model.

Functional Specification for SSD Relation

Review Functions

- All the review functions for the Core RBAC model are needed for implementation of the SSD RBAC model.

- —SSDRoleSets: returns the set of named SSD relations created for the SSD RBAC model;
- —SSDRoleSetRoles: returns the set of roles associated with a named SSD role set; and
- —SSDRoleSetCardinality: returns the cardinality of the subset within the named SSD role set for which common user membership restriction applies.

Functional Specification for DSD Relation

Administrative Functions

- The semantics of creating an instance of a DSD relation are identical to that of an SSD relation.
- While constraints associated with an SSD relation are enforced during user assignments, the constraints associated with DSD are typically enforced only at the time of role activation within a user session.
  - —CreateDSDSet: creates a named instance of a DSD relation;
  - —DeleteDSDSet: deletes an existing DSD relation;
  - —AddDSDRoleMember: adds a role to a named DSD role set;
  - —DeleteDSDRoleMember: deletes a role from a named DSD role set;
  - —SetDSDCardinality: sets the cardinality of the subset of roles from the named DSD role set for which user activation restriction within the same session applies.

Functional Specification for DSD Relation

Supporting System Functions

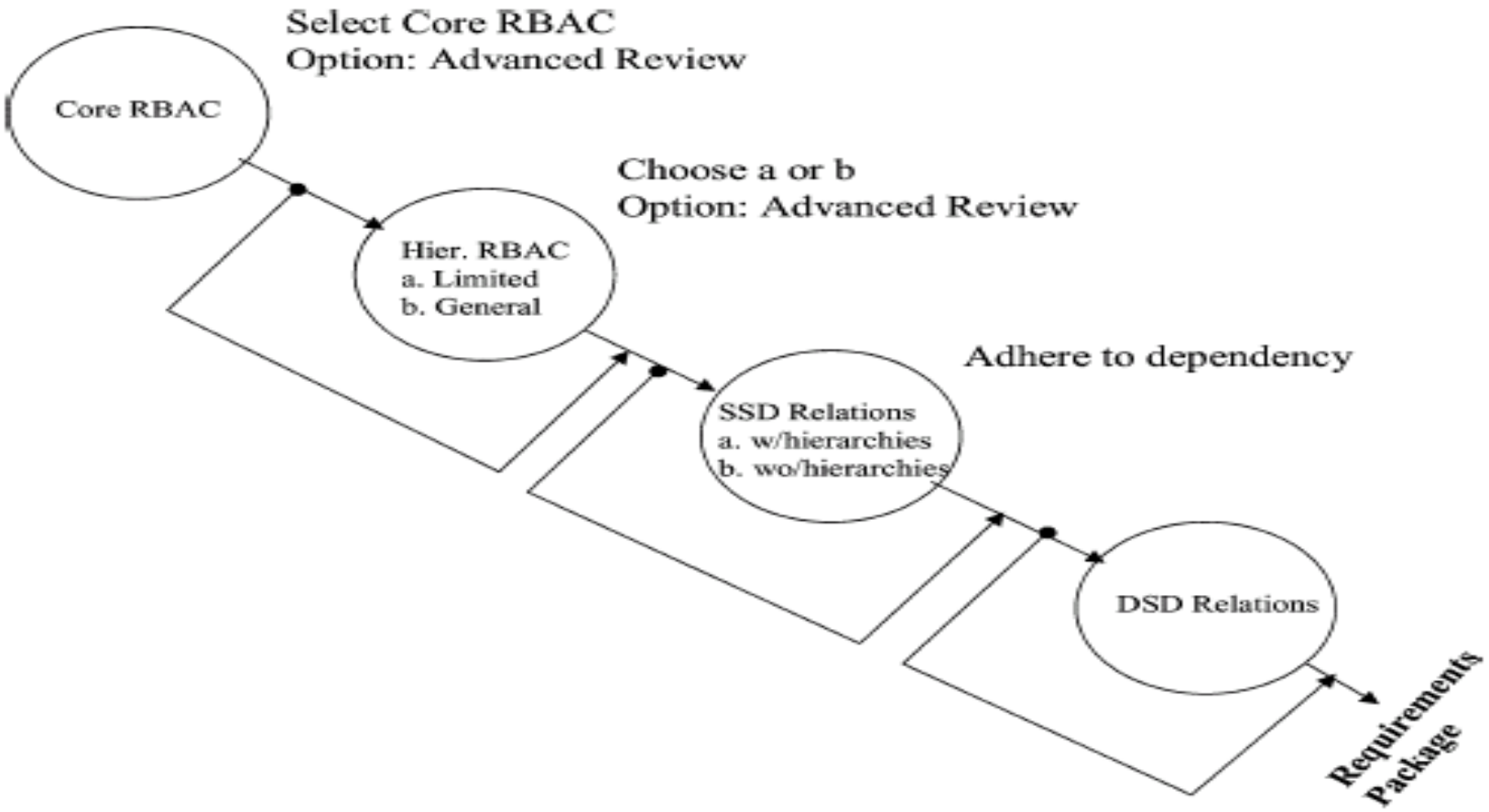
- The additional functionality required of these functions in the DSD RBAC model context is that they should enforce the DSD constraints.
  - —CreateSession: creates a user session and provides the user with a default set of active roles;
  - —AddActiveRole: adds a role as an active role for the current session;
  - —DropActiveRole: deletes a role from the active role set for the current session.
- The semantics of the Supporting System Functions for a DSD RBAC model with role hierarchies are the same as those for corresponding functions for hierarchical RBAC
  - —CreateSession: creates a user session and provides the user with a default set of active roles;
  - —AddActiveRole: adds a role as an active role for the current session;
  - —DropActiveRole: deletes a role from the active role set for the current session.

Functional Specification for DSD Relation

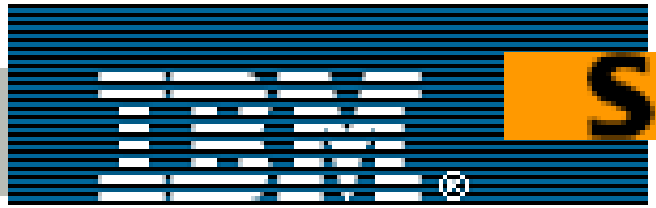
Review Functions

- All the review functions for the Core RBAC model are needed for implementation of the DSD RBAC model. In addition, functions to view the results of administrative functions shall also be provided.
  - —DSDRoleSets: returns the set of named SSD relations created for the DSD RBAC model;
  - —DSDRoleSetRoles: returns the set of roles associated with a named DSD role set; and
  - —DSDRoleSetCardinality: returns the cardinality of the subset within the named DSD role set for which user activation restriction within the same session applies.

Methodology for creating functional packages.



Some of the Vendors Offering RBAC Products



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.**

Questions…Comments??

Thank You!!!!!!!!!!!!!!!!!!

Core RBAC

- User: Is defined as human being. The concept of user can be extended to include machines, networks and so on…
- Role: Is a job function within the contest of an organization within the context of an organization with some associated semantics regarding the authority and responsibility conferred on the user assigned to that role.
- Permissions: Is an approval to perform an operation on one or more RBAC protected objects.
- Operation: Is an executable image of the program, which upon invocation executes some function for the user.
- Object: Is an entity that contains or receives information.

General Role Hierarchies

—*RH* $\subseteq$ *ROLES*×*ROLES* is a partial order on *ROLES* called the inheritance relation, written as ≥, where *r*1 ≥ *r*2 only if all permissions of *r*2 are also permissions of *r*1, and all users of *r*1 are also users of *r*2. Formally: *r*1 ≥ *r*2) $\Rightarrow$ *authorized _permissions*(*r*2) $\subseteq$ *authorized_permissions*(*r*1) ^ *authorized users*(*r*1) $\subseteq$ *authorized_users*(*r*2).

—*authorized_users*(*r* :*ROLES*) $\rightarrow$ 2 ^*USERS*, the mapping of role *r* onto a set of users in the presence of a role hierarchy. Formally: *authorized_users*(*r*) = {*u* $\in$*USERS*| *r* ‘ ≥ *r* (*u*, *r* ‘) $\in$ *UA*}.

—*authorized_permissions*(*r* : *ROLES*) $\rightarrow$ 2 ^ *PRMS*, the mapping of role *r* onto a set of permissions in the presence of a role hierarchy. Formally: *authorized_permissions*(*r*)= { *p* $\in$*PRMS* | *r* ‘≥ *r* (*p*, *r* ‘) $\in$ *PA*} .

Static Separation of Duty Relations

Static Separation of Duty.

—$SSD \subseteq (2^{ROLES} \times N)$ is a collection of pairs (rs, n) in *Static Separation of Duty*, where each rs is a role set, t a subset of roles in rs, and n is a natural number ≥ 2, with the property that no user is assigned to n or more roles from the set rs in each $(rs, n) \in SSD$. Formally: $\forall (rs, n) \in SSD, \forall t \subseteq rs : |t| \geq n \Rightarrow \bigcap_{r \in t} assigned\_users(r) = \emptyset$.

Static Separation of Duty in the Presence of a Hierarchy.

—In the presence of a role hierarchy *static separation of duty* is redefined based on authorized users rather than assigned users as follows.

$$\forall (rs, n) \in SSD, \forall t \subseteq rs: |t| \geq n \Rightarrow \bigcap_{r \in t} authorized\_users(r) = \emptyset.$$

Dynamic Separation of Duty Relations

—$DSD \subseteq (2^{ROLES} \times \mathrm{N})$ is collection of pairs (rs, n) in *Dynamic Separation of Duty*, where each rs is a role set and n is a natural number ≥ 2, with the property that no subject may activate n or more roles from the set rs in each $dsd \in DSD$. Formally:

$$\forall rs \in 2^{ROLES}, n \in \mathrm{N}, (rs, n) \in DSD \Rightarrow n \geq 2 \wedge |rs| \geq n, \text{ and}$$

$$\forall s \in SESSIONS, \forall rs \in 2^{ROLES}, \forall role\_subset \in 2^{ROLES}, \forall n \in \mathrm{N}, (rs, n) \in DSD,$$

$$role\_subset \subseteq rs, role\_subset \subseteq session\_roles(s) \Rightarrow |role\_subset| < n.$$