UnitelmaSapienza
Università degli Studi di Roma

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Role Based Access Control RBAC

Prof. F. Parisi Presicce

Access Control Models

- **Discretionary Access Control** (DAC), 1970
  - Owner controls access
  - But only to the original, not to copies
  - Grounded in pre-computer policies of researchers
- **Mandatory Access Control** (MAC), 1970
  - Synonymous to Lattice-Based Access Control (LBAC)
  - Access based on security labels
  - Labels propagate to copies
  - Grounded in pre-computer military and national security policies
- **Role-Based Access Control** (RBAC), 1995
  - Access based on roles
  - Can be configured to do DAC or MAC
  - Grounded in pre-computer enterprise policies

Role Based Access Control

Motivating Problem: how to administer user-permission relation

- Different from DAC and MAC, which deal with processes in operating systems

Roles as a level of indirection

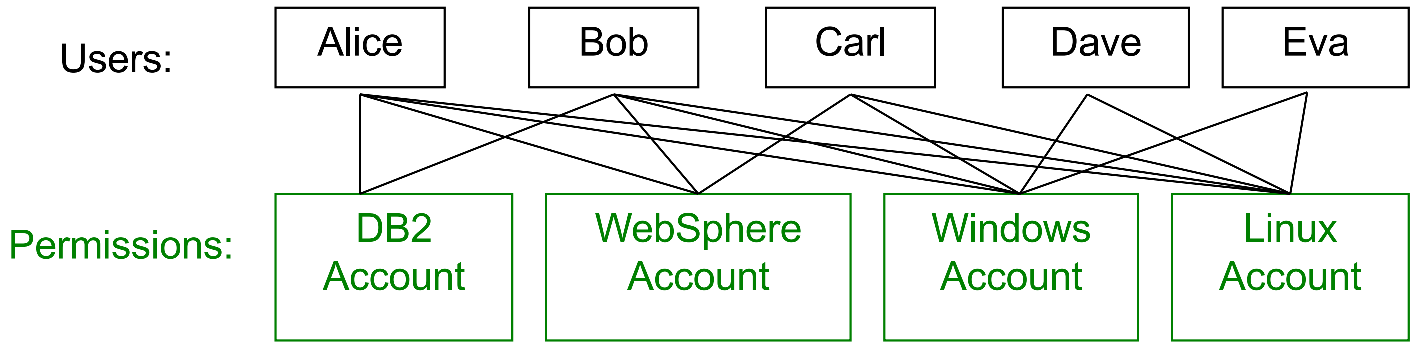
- Butler Lampson: "all problems in Computer Science can be solved by another level of indirection"

RBAC is multi-faceted and open ended

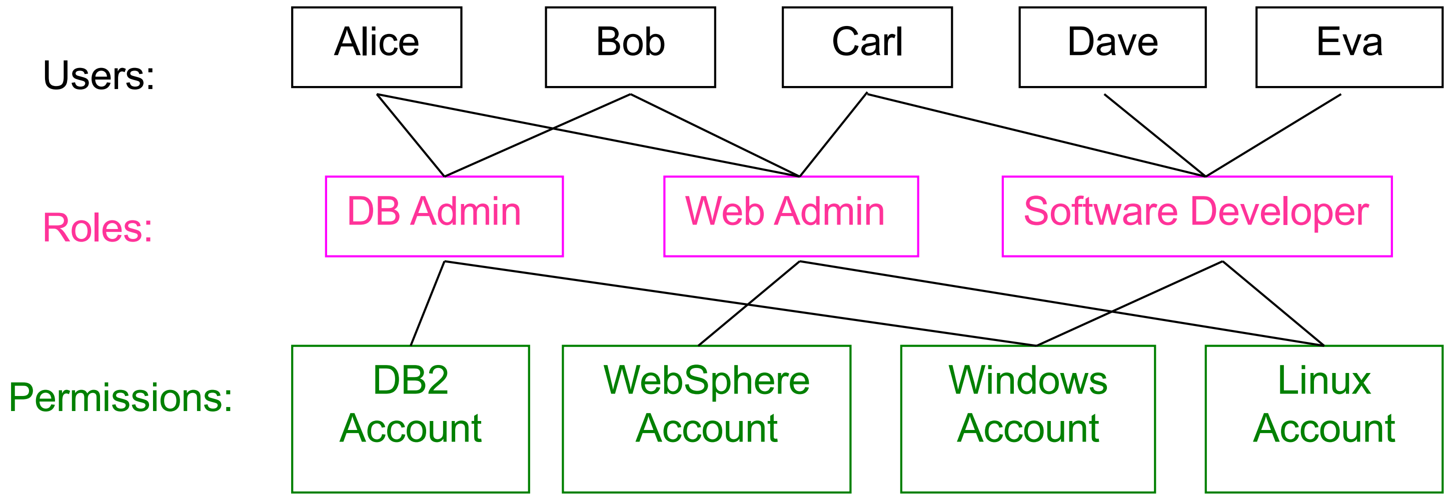
- **Extensions:** ARBAC (administrative), CRBAC (constraint), dRBAC (dynamic), ERBAC (enterprise), fRBAC (flexible), GRBAC (generalized), HRBAC (hierarchical), IRBAC (interoperability), JRBAC (Java), LRBAC (Location), MRBAC (Management), PRBAC (privacy), QRBAC (QoS), RRBAC(Rule), SRBAC(Spatial), TRBAC (temporal), V, W, x.
- **Non extension**: OrBAC (organization)

Access Control based on job function

- Non-role-based systems



- Role-Based Access Control Systems (RBAC)



Why Roles ?

- Fewer relationships to manage
  - possibly from O(mn) to O(m+n), where m is the number of users and n is the number of permissions
- Roles add a useful level of abstraction
- Organizations operate based on roles, so RBAC can directly support the security policies of the organization
- A role may be more stable than
  - ➢ the collection of users and the collection of permissions that are associated with it

Groups vs. Roles

- Depending on the precise definition, can be the same or different.
- Some differences that may or may not be important, depending on the situation
  1. sets of users vs. sets of users as well as permissions
  2. roles can be activated and deactivated, groups cannot
     - Groups can be used to prevent access with negative authorization.
     - Roles can be deactivated for least privilege
  3. can easily enumerate permissions that a role has, but not for groups

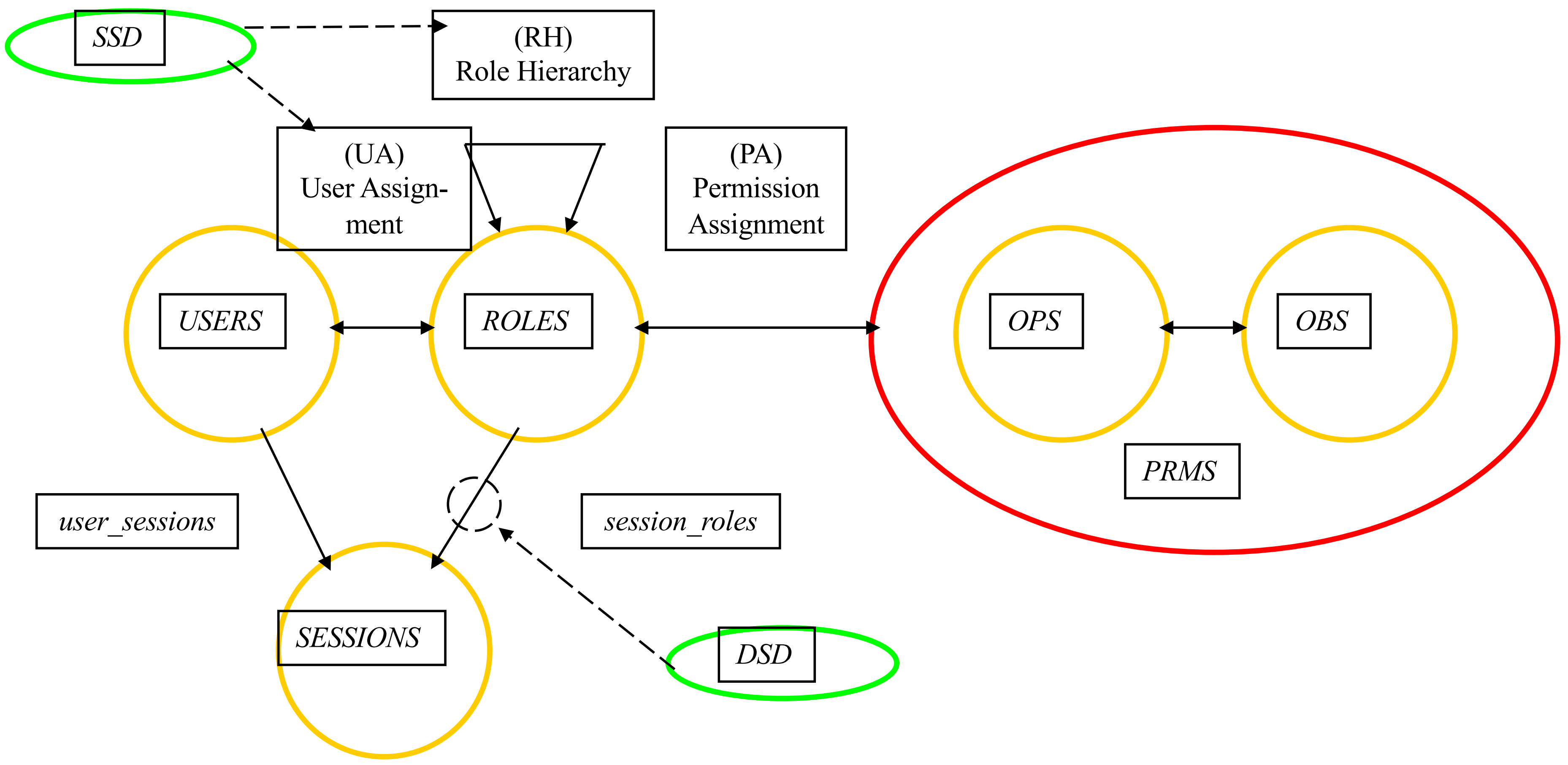
RBAC (http://csrc.nist.gov/rbac/)

A policy-neutral model, that can express both DAC (role as identity) and MAC (role as clearance)

Access/right often depends on role (job function), not on identity

- Example:
  - Allison, bookkeeper, has access to financial records.
  - Bob hired to replace Allison as the new bookkeeper
  - Bob now has access automatically to those records
- The role of “bookkeeper” determines access, not the identity of the individual, and ‘connects’ the subject to the permission(s).

RBAC (http://csrc.nist.gov/rbac/)



Role-Based AC

- A user has access to an object based on the assigned role.
- Roles are defined based on job functions.
- Permissions are defined based on job authority and responsibilities within a job function.
  - Permissions are positive, no negative permissions
  - Closed policy: access denied unless explicitly authorized
- Operations on an object are invoked based on the permissions.
- The object is concerned with the user’s role and not the user.

Privilege

- Roles are engineered based on the principle of least privilege.
- A role contains the minimum amount of permissions to instantiate an object.
- A user is assigned to a role that allows him or her to perform only what is required for that role.
- No single role is given more permission than the same role for another user.
- No duties or Obligations
    - Can access patient records but must notify patient or delete record after 20 days

Role-Based AC Framework

- Core Components
- Constraining Components
  - Hierarchical RBAC
    - General
    - Limited
  - Separation of Duty Relations
    - Static
    - Dynamic

Core Components

Defines:

- USERS {process, intelligent agent, human}
- ROLES
- OPERATIONS (*ops*)
- OBJECTS (*obs*)
- User Assignments (*ua*)
  - assigned_users

Core Components cont.

- Permissions (*prms*)
  - Assigned Permissions
  - Object Permissions
  - Operation Permissions
- Sessions
  - User Sessions
  - Available Session Permissions
  - Session Roles

Constraint Components

- Role Hierarchies (*rh*)
  - General
  - Limited
- Separation of Duties Constraints
  - Static
  - Dynamic

RBAC Transition

Least Privilege Separation of Duties

Most Complex

Models	Hierarchies	Constraints
$RBAC_0$	No	No
$RBAC_1$	Yes	No
$RBAC_2$	No	Yes
$RBAC_3$	Yes	Yes

RBAC Functional Specification

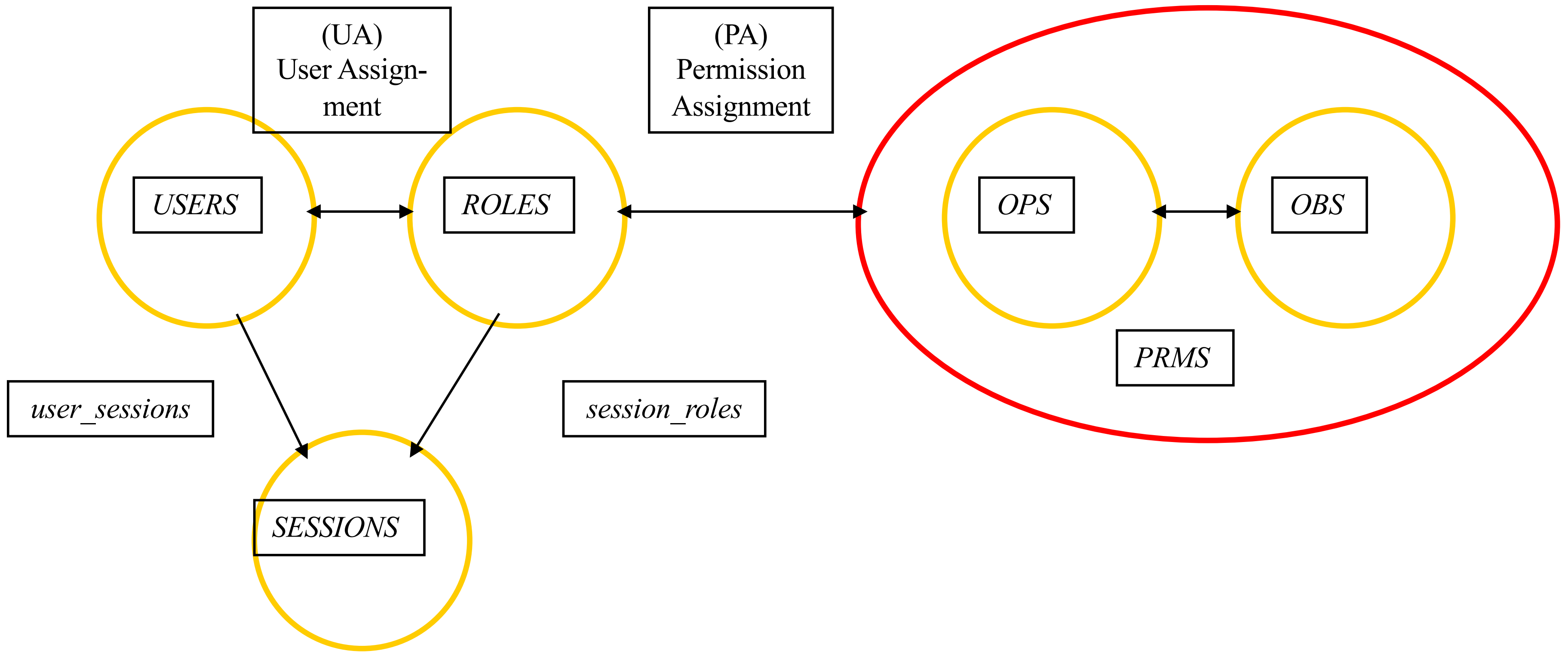
It defines the features required of an RBAC system.
These features fall into three categories

- 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.

Definitions

- Role *r* : collection of job functions
  - developer, director, manager, …
  - an organizational job function with a clear definition of inherent responsibility and authority (permissions).
- M-T-M relation between USERS and PRMS (going through roles)

Core RBAC


(UA)
User Assign-
ment
(PA)
Permission
Assignment
USERS
ROLES
OPS
OBS
PRMS
user_sessions
session_roles
SESSIONS

RBAC0: Formal Model

- Vocabulary: U, R, P, S (users, roles, permissions, and sessions)
- Static relations:
  - $PA \subseteq P \times R$ (permission assignment)
  - $UA \subseteq U \times R$ (user assignment)
- Dynamic relations:
  - user: $S \rightarrow U$ each session has one user
  - roles: $S \rightarrow 2^R$ and some activated roles
    - requires $roles(s) \subseteq \{ r \mid (user(s), r) \in UA \}$

- Session s has permissions

$$\bigcup_{r \in roles(s)} \{ p \mid (p, r) \in PA \}$$

UA (user assignment)

- A user can be assigned to one or more roles
- A role can be assigned to one or more users

$$assigned\_user : (r : ROLES) \rightarrow 2^{users}$$

$$assigned\_user(r) = \{u \in USERS \mid (u, r) \in UA\}$$

PA (permission assignment)

- A permission can be assigned to one or more roles
- A role can be assigned to one or more permissions

$$assigned\_permissions(r : ROLES) \rightarrow 2^{PRMS}$$

$$assigned\_permissions(r) = \{p \in PRMS \mid (p, r) \in PA\}$$

$$Op(p : PRMS) \rightarrow \{op \subseteq OPS\}$$

$$Ob(p : PRMS) \rightarrow \{ob \subseteq OBS\}$$

Sessions

- Each session is associated to a number of roles
- Each session is associated to only one user
- Each user is associated to a set of sessions

$$session\_roles(s : SESSIONS) \rightarrow 2^{ROLES}$$

$$session\_roles(s_i) \subseteq \{r \in ROLES \mid (session\_users(s_i), r) \in UA)\}$$

$$user\_sessions(u : USERS) \rightarrow 2^{SESSIONS}$$

$$avail\_session\_persm(s : SESSIONS) \rightarrow 2^{PRMS}$$

$$\bigcup_{r \in session\_roles(s)} assigned\_permissions(r)$$

Notation

- Role *r*
  - *trans*(*r*): set of authorized transactions for *r*
- Active role of user *u* : the role *r* is currently in
  - *actr*(*u*)
- Authorized roles of *u* : set of roles *u* can assume
  - *authr*(*u*)
- *canexec*(*u*, *t*) is true if and only if subject *u* can execute transaction *t* at current time

Axioms (mandatory style)

U the set of users/subjects; T the set of transactions.

Rule of role assignment:

- $(\forall u \in U)(\forall t \in T)\ [canexec(u, t) \rightarrow actr(u) \neq \emptyset]$.
  - If u can execute a transaction, it has an active role
  - This ties transactions to roles, not users

Rule of role authorization:

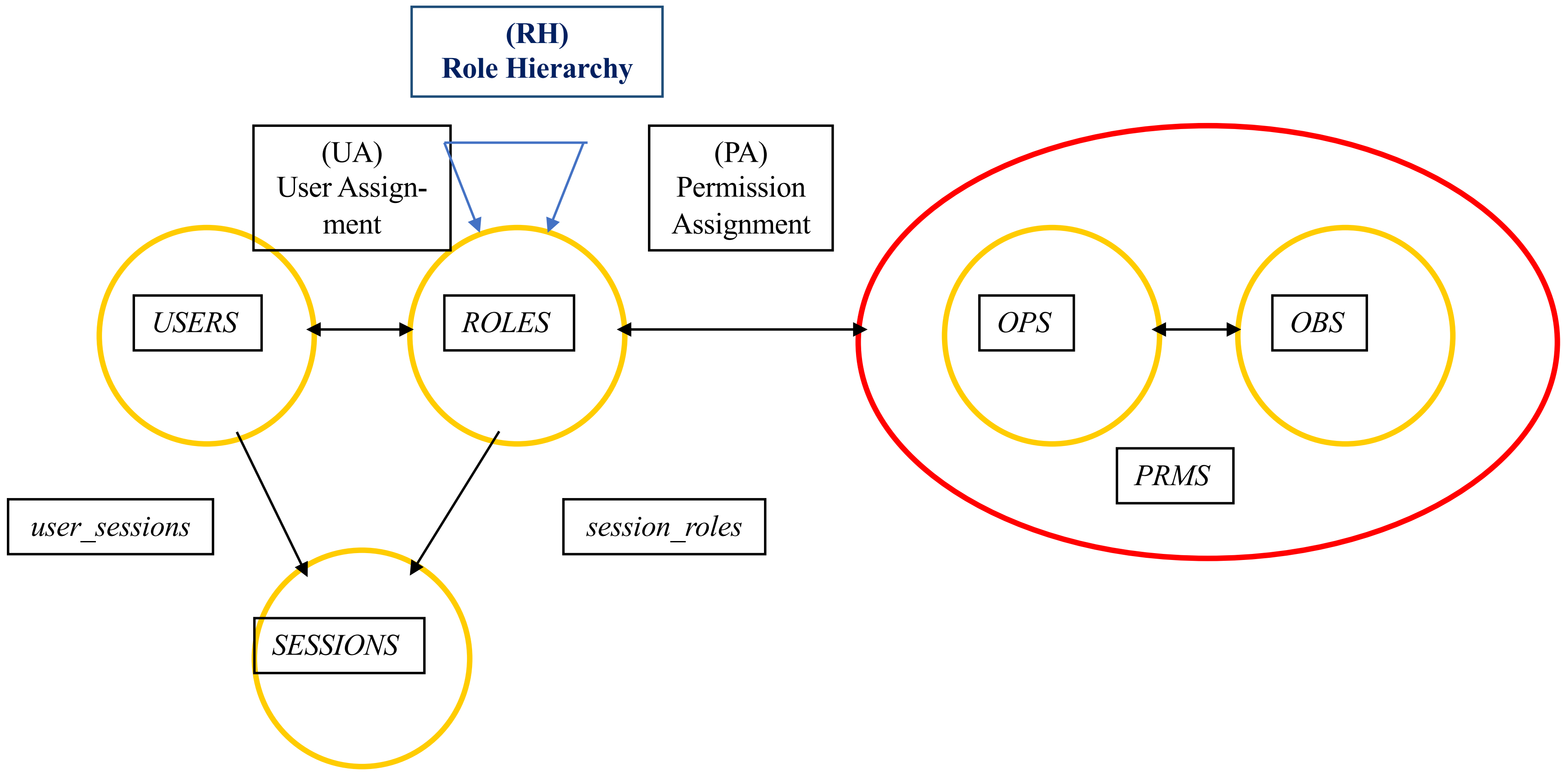
- $(\forall u \in U)\ [actr(u) \subseteq authr(u)]$.
  - Subject/user must be authorized to assume an active role (otherwise, any subject could assume any role)

Axioms (mandatory style)

Rule of transaction authorization:

- $(\forall u \in U)(\forall t \in T)$
  $[canexec(u, t) \rightarrow t \in trans(actr(u))].$
  - A subject/user *u* can execute a transaction t only if the transaction is authorized for the role *u* has assumed (active)

Hierarchical RBAC



Semantics of Role Hierarchies

User inheritance

- r1 ≥ r2 means every user that is a member of r1 is also a member of r2

Permission inheritance

- r1 ≥ r2 means every permission that is authorized for r2 is also authorized r1

Activation inheritance

- r1 ≥ r2 means that activating r1 will also activate r2

Permission and Activation inheritance have different effect when there are constraints about activation.

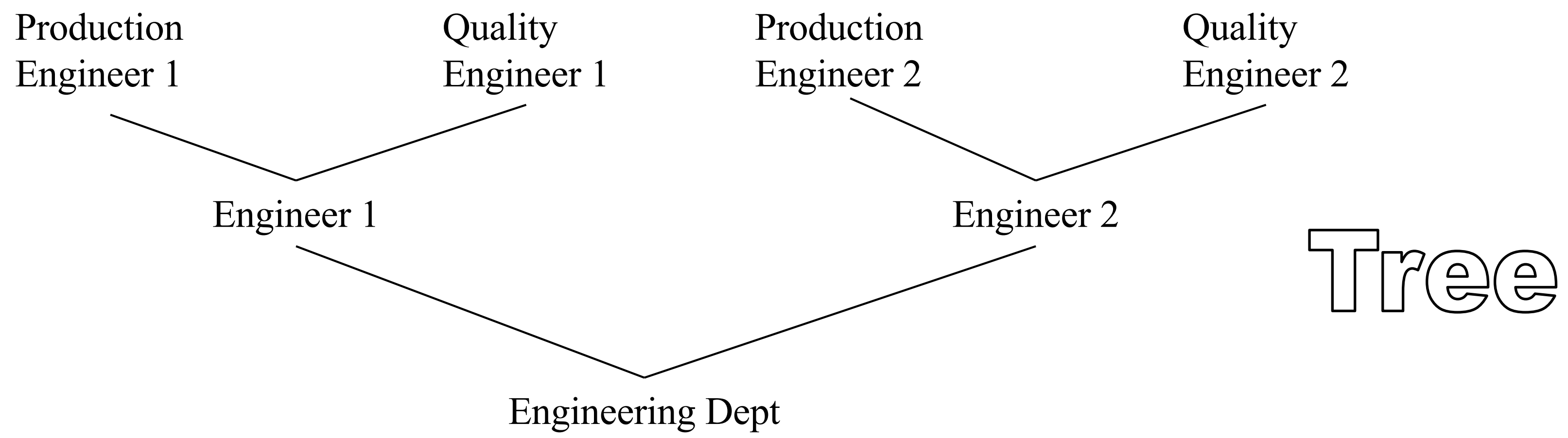
RBAC1: Formal Model

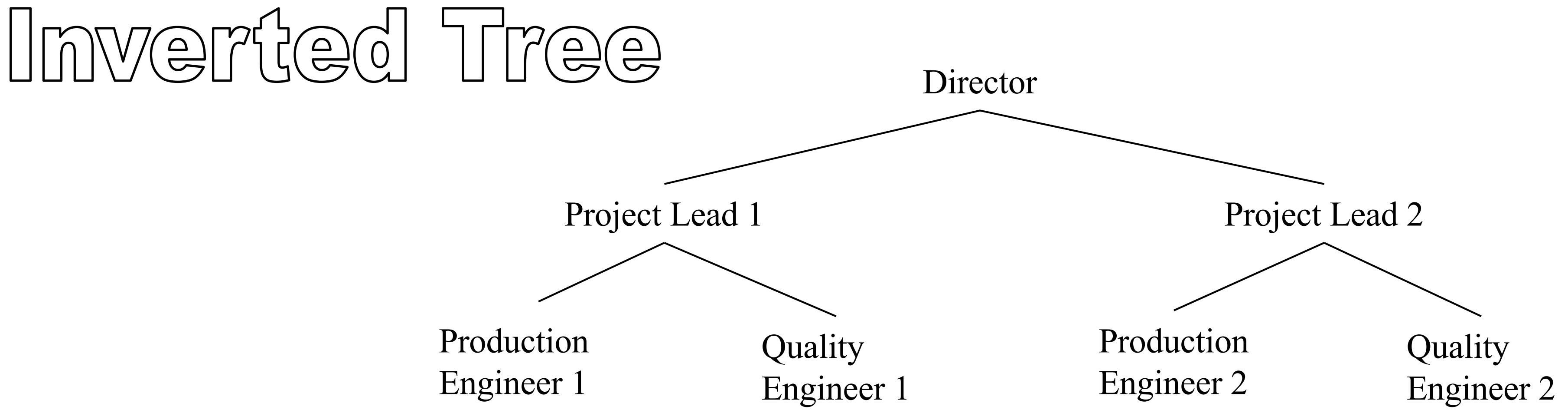
- U, R, P, S, PA, UA unchanged from RBAC0
- $RH \subseteq R \times R$: a partial order on R, written as $\geq$
  - When $r1 \geq r2$, we say r1 is a senior than r2, and r2 is a junior than r1
- roles: $S \rightarrow 2^R$
  - requires $roles(s) \subseteq \{ r \mid \exists r' [(r' \geq r) \ \& \ (user(s), r') \in UA] \}$

- Session s includes permissions

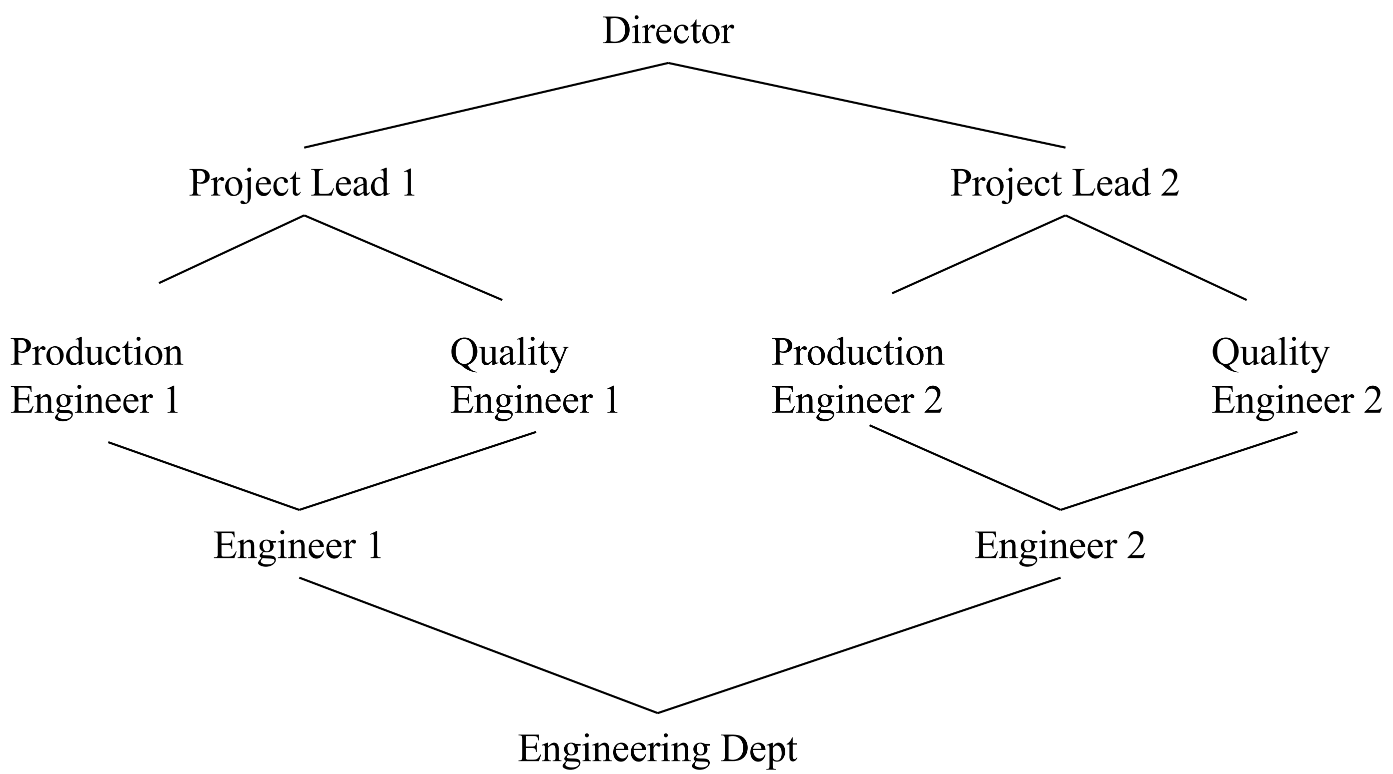
$$\bigcup_{r \in roles(s)} \{ p \mid \exists r'' [(r \geq r'') \ \& \ (p, r'') \in PA] \}$$

Tree Hierarchies





Lattice Hierarchy



- Supports multiple inheritance

General RH

$$UA \subseteq USERS x ROLES$$

$$r_1 \succeq r_2 \Rightarrow authorized\_permissions(r_2) \subseteq authorized\_permissions(r_1)$$
$$\wedge\, authorized\_users(r_1) \subseteq authorized\_users(r_2)$$

$$assigned\_user(r) = \{u \in USERS \mid (u,r) \in UA\}$$

$$authorized\_users(r) = \{u \in USERS \mid r' \succeq r \wedge (u,r') \in UA\}$$

$$authorized\_permissions(r : ROLES) \rightarrow 2^{PRMS}$$
$$authorized\_permissions(r) = \{p \in PRMS \mid r \succeq r', (p,r') \in PA\}$$

More axioms

Rule of containment of roles:

- Trainer can do all the transactions that trainee can do (and then some). This means role *r'* contains role *r* (*r'* > *r*). So:
  - $(\forall u \in U)[\, r' \in authr(u) \wedge r' > r \rightarrow r \in authr(u)\,]$
  - $(\forall t \in T)[\, t \in trans(r) \wedge r' > r \rightarrow t \in trans(r')\,]$

RBAC2 = RBAC0 + constraints

No formal model specified

Example constraints

- Separation of Duty (e.g. mutual exclusion)
- Pre-condition: Must satisfy some condition to be member of some role
    - E.g., a user must be an graduate student before being assigned the Sapienza Tutor role
- Cardinality

Why Using Constraints?

For laying out higher level organization policy

- Only a tool for convenience and error checking when admin is centralized
  - Not absolutely necessary if admin is always vigilant, as admin can check all organization policies are met when making any changes to RBAC policies
  - Reduces chances of mistake
  - Like “assert” statements in C/C++/Java programs
- A tool to enforce high-level policies when admin is decentralized

Separation of Duties

- Enforces conflict of interest policies employed to prevent users from exceeding a reasonable level of authority for their position.
- Ensures that failures of omission or commission within an organization can be caused only as a result of collusion among individuals.

Two Types:

- Static Separation of Duties (SSD)
- Dynamic Separation of Duties (DSD)

Mutual Exclusion Constraints

Conflict of Interest

- **Static Exclusion**: No user can hold both roles
  - Preventing a single user from having too much permissions
  - Example: controller and controlled
- **Dynamic Exclusion**: No user can activate both roles in the same session
  - Interact with role hierarchy interpretation
  - Example: professor and student

Mutual Exclusion (static)

For r a role, the predicate $meauth(r)$ (for *m*utually exclusive *auth*orizations) is the set of roles that a subject u, for which $r \in auth(u)$, cannot assume because of some separation of duty requirement.

Separation of duty constraint:

$$(\forall r_1, r_2 \in R)\ [\ r_2 \in meauth(r_1) \rightarrow$$
$$[\ (\forall u \in U)\ [\ r_1 \in authr(u) \rightarrow r_2 \notin authr(u)\ ]\ ]\ ]$$

Cardinality Constraints

On User-Role Assignment

- at most k users can belong to the role
- at least k users must belong to the role
- exactly k users must belong to the role

On activation

- at most k users can activate a role
- ...

SSD

$$SSD \subseteq (2^{ROLES} x N)$$

- SSD places restrictions on the set of roles and in particular on their ability to form *UA* relations.
- No user is assigned to *n* or more roles from the same role set, where *n* or more roles conflict with each other.
- A user may be in one role, but not in another—mutually exclusive.
- Prevents a person from submitting and approving their own request.

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

SSD in the presence of RH

- A constraint on the authorized users of the roles that have an SSD relation.
- Based on the authorized users rather than assigned users.
- Ensures that inheritance does not undermine SSD policies.
- Reduce the number of potential permissions that can be made available to a user by placing constraints on the users that can be assigned to a set of roles.

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

DSD

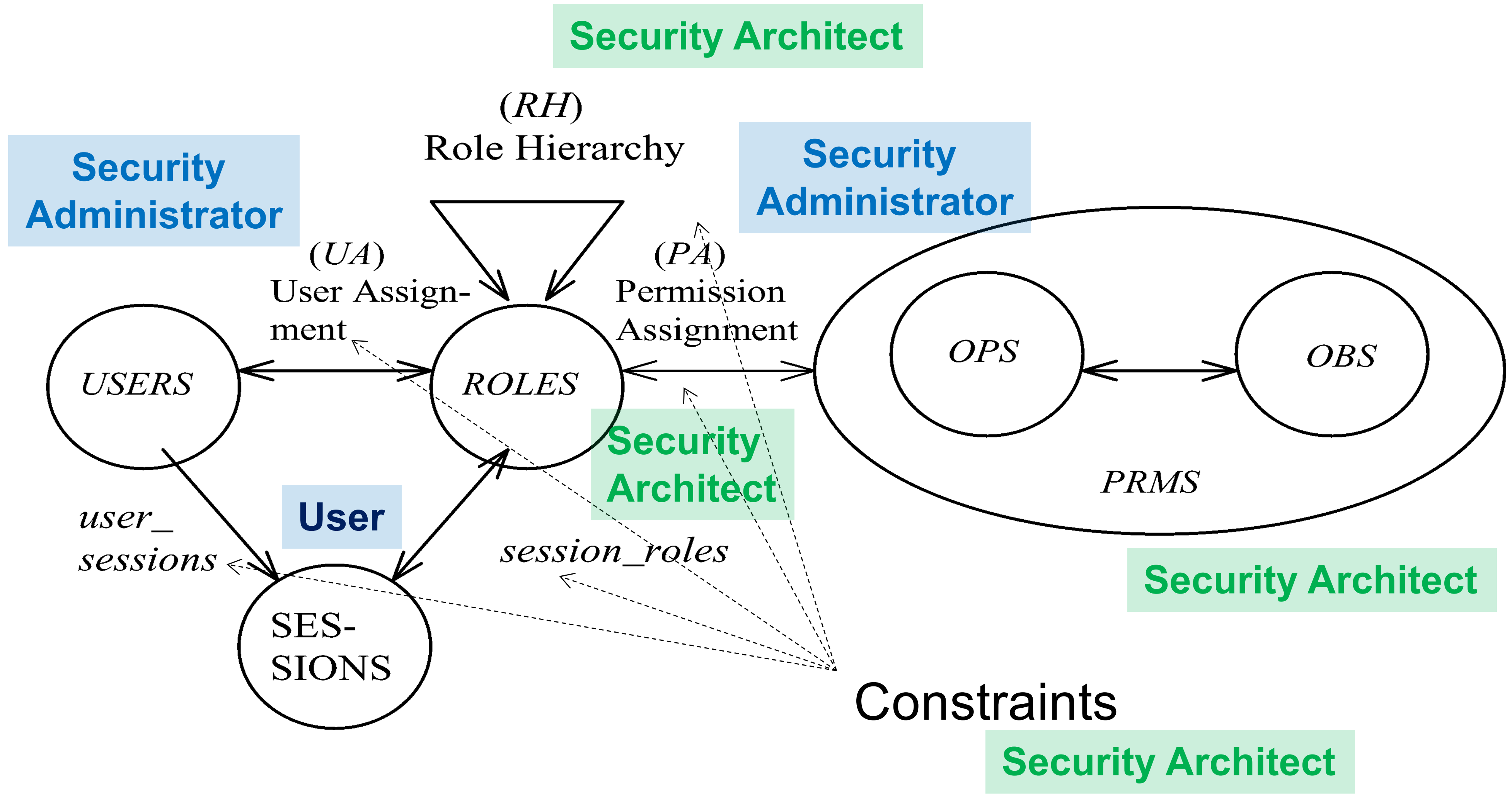
- Places constraints on the users that can be assigned to a set of roles, thereby reducing the number of potential permissions that can be made available to a user.
- Constraints are across or within a user's session.
- No user may activate *n* or more roles from the roles set in each user session.
- *Timely Revocation of Trust* ensures that permissions do not persist beyond the time that they are required for performance of duty.

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

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

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

RBAC Configuration Points



RBAC shortcomings

- Role granularity is not adequate leading to role explosion
  - Researchers have suggested several extensions such as parameterized privileges, role templates, parameterized roles (1997-)
- Role design and engineering is difficult and expensive
  - Substantial research on role engineering top down or bottom up (1996-), and on role mining (2003-)
- Assignment of users/permissions to roles is cumbersome
  - Researchers have investigated decentralized administration (1997-), attribute-based implicit user-role assignment (2002-), role-delegation (2000-), role-based trust management (2003-), attribute-based implicit permission-role assignment (2012-)
- Adjustment based on local/global situational factors is difficult
  - Temporal (2001-) and spatial (2005-) extensions to RBAC proposed
- RBAC does not offer an extension framework
  - Every shortcoming seems to need a custom extension
  - ABAC may unify these extensions in a common open-ended framework

Attribute-Based Access Control (ABAC)

An access control method where subject requests to perform operations on objects are granted or denied based on

- assigned attributes of the subject,
    - E.g., job role, clearance, division/unit, location
- assigned attributes of the object,
    - E.g., readable, completed, shared, …
- environment conditions,
    - E.g., time, state of emergency
- and a set of policies that are specified in terms of those attributes and conditions.
    - E.g., a list of rules, as in firewall policies,

Attribute-Based Access Control

- Attributes are (name:value) pairs
  - possibly chained
  - values can be complex data structures
- Associated with
  - actions
  - users
  - subjects
  - objects
  - contexts
  - policies
- Converted by policies into rights “just in time”
  - policies specified by security architects
  - attributes maintained by security administrators
  - but also possibly by users OR reputation and trust mechanisms
- Inherently extensible

Relationship-Based Access Control (ReBAC) motivation

- Users in Online Social Networks (OSNs) are connected by social relationships (user-to-user relationships U2U)
- Owner of a resource can control its release based on U2U relationships between the access requester and the owner
- OSNs keep massive resources and support varied activities for users
- Users want to regulate access to their resources and activities related to them (as a requester or target)
- Some related users also expect control on how the resource or user can be exposed

Relationship-Based Access Control (ReBAC) motivation

What current friend-of-friend approach cannot do?

- User who is tagged in a photo wants to keep her image private (Related User's Control)
- Mom doesn't want her kid to become friend with her colleagues (Parental Control)
- Employee promotes his resume to headhunters without letting his current employer know (Allowing farther users but preventing closer users)