Access Control

Discretionary Access Control

Lecture 4

Introduction

- "Access control" is where security engineering meets computer science.
- Its function is to control which (active) subject have access to a which (passive) object with some specific access operation.

Access Control

- Discretionary Access Control (DAC)
  - Access Matrix Model
  - Implementation of the Access Matrix
  - Vulnerabilities of the Discretionary Policies
  - Additional features of DAC

Discretionary Access Control

- Access to data objects (files, directories, etc.) is permitted based on the identity of users.
- Explicit access rules that establish who can, or cannot, execute which actions on which resources.
- Discretionary: users can be given the ability of <u>passing on their privileges to other users</u>, where **granting** and **revocation** of privileges is regulated by an administrative policy.

Discretionary Access Control

- DAC is flexible in terms of policy specification
- This is the form of access control widely implemented in standard multi-user platforms Unix, NT, Novell, etc.

Discretionary Access Control

- Access control matrix
  - Describes protection state precisely
  - Matrix describing rights of subjects
  - State transitions change elements of matrix
- State of protection system
  - Describes current settings, values of system relevant to protection

Access Control

- Discretionary Access Control
  - Access Control Matrix Model
  - Implementation of the Access Matrix
  - Vulnerabilities of the Discretionary Policies
  - Additional features of DAC

Access Control Matrix Model

- Access control matrix
  - Firstly identify the <u>objects</u>, <u>subjects</u> and <u>actions</u>.
  - Describes the protection state of a system.
  - State of the system is defined by a triple (S, O, A)
    - S is the set of subject,
    - O is the set of objects,
    - A is the access matrix
  - Elements indicate the access rights that subjects have on objects
    - Entry A[s, o] of access control matrix is the privilege of s on o

Description

objects (entities)

subjects

	o_1	…	o_m	s_1	…	s_n
s_1						
s_2						
…						
s_n						

- Subjects $S = \{ s_1,\dots,s_n \}$
- Objects $O = \{ o_1,\dots,o_m \}$
- Rights $R = \{ r_1,\dots,r_k \}$
- Entries $A[s_i, o_j] \subseteq R$
- $A[s_i, o_j] = \{ r_x, \dots, r_y \}$ means subject s_i has rights $r_x, \dots, r_y$ over object o_j

	File 1	File 2	File 3	Program 1
Ann	own read write	read write		execute
Bob	read		read write	
Carl		read		execute read

Boolean Expression Evaluation

- ACM controls access to database fields
  - Subjects have attributes
  - Action/Operation/Verb define type of access
  - Rules associated with objects, action pair
- Subject attempts to access object
  - Rule for object, action evaluated, grants or denies access

Example

- Subject Annie
  - Attributes role (artist), groups (creative)
- Verb paint
  - Default 0 (deny unless explicitly granted)
- Object picture
  - Rule:

    Annie paint picture if:

    ‘artist’ in subject.role and
    ‘creative’ in subject.groups and
    time.hour ≥ 0 and time.hour < 5

ACM at 3AM and 10AM

At 3AM, time condition met; ACM is:

… picture …

… annie …			
		paint	

At 10AM, time condition not met; ACM is:

… picture …

… annie …			

Access Controlled by History

- Statistical databases need to
  - answer queries on groups
  - prevent revelation of individual records
- Query-set-overlap control
  - Prevent an attacker to obtain individual piece of information using a set of queries C
  - A parameter *r* (=2) is used to determine if a query should be answered

Name	Position	Age	Salary
Alice	Teacher	45	40K
Bob	Aide	20	20K
Cathy	Principal	37	60K
Dilbert	Teacher	50	50K
Eve	Teacher	33	50K

Access Controlled by History

- Query 1:
  - sum_salary(position = teacher)
  - Answer: 140K

Name	Position	Age	Salary
Celia	Teacher	45	40K
Leonard	Teacher	50	50K
Matt	Teacher	33	50K

- Query 2:
  - sum_salary(age > 40 & position = teacher)
  - Should not be answered as Matt's salary can be deduced

Name	Position	Age	Salary
Celia	Teacher	45	40K
Leonard	Teacher	50	50K

- Can be represented as an ACM

Solution: Query Set Overlap Control (Dobkin, Jones & Lipton '79)

- Query valid if intersection of *query coverage* and *each previous query* < r
- Can represent as access control matrix
  - Subjects: entities issuing queries
  - Objects: *Powerset* of records
  - $O_s(i)$: objects referenced by s in queries $1..i$
  - M[s,o] = read iff $\forall_{q \in O_s(i-1)} |q \cap o| < r$

$$\mathrm{M}[s,o] = \text{read iff} \quad \forall_{q \in O_s^{(i-1)}} |q \cap o| < r$$

- Query 1: O_1 = {Celia, Leonard, Matt} so the query can be answered. Hence
  - M[asker, Celia] = {read}
  - M[asker, Leonard] = {read}
  - M[asker, Matt] = {read}
- Query 2: O_2 = {Celia, Leonard} but $|\, O_2 \cap O_1 \,| = 2$; so the query cannot be answered
  - M[asker, Celia] = $\varnothing$
  - M[asker, Leonard] = $\varnothing$

Access Control

- Discretionary Access Control
  - Access Matrix Model
  - Implementation of the Access Control Matrix
  - Vulnerabilities of the Discretionary Policies
  - Additional features of DAC

ACM Implementation

- ACM is an **abstract** model
  - Rights may vary depending on the object involved
- ACM is implemented primarily in three ways
  - Authorization Table
  - Capabilities (rows)
  - Access control lists (columns)

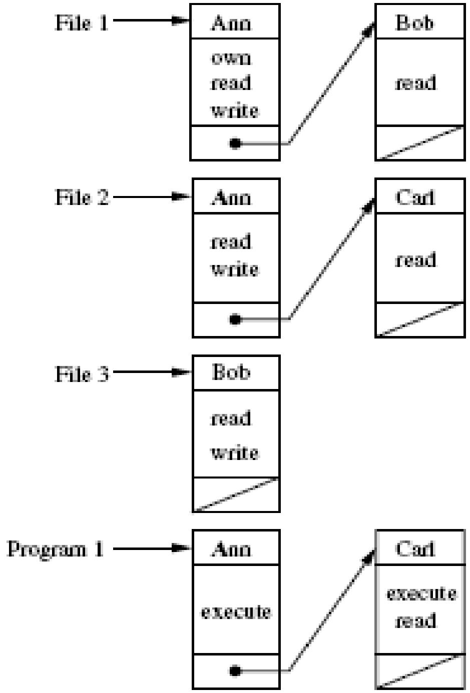
Authorization Table

- Three columns: subjects, actions, objects
- Generally used in DBMS systems

USER	ACCESS MODE	OBJECT
Ann	own	File 1
Ann	read	File 1
Ann	write	File 1
Ann	read	File 2
Ann	write	File 2
Ann	execute	Program 1
Bob	read	File 1
Bob	read	File 3
Bob	write	File 3
Carl	read	File 2
Carl	execute	Program 1
Carl	read	Program 1

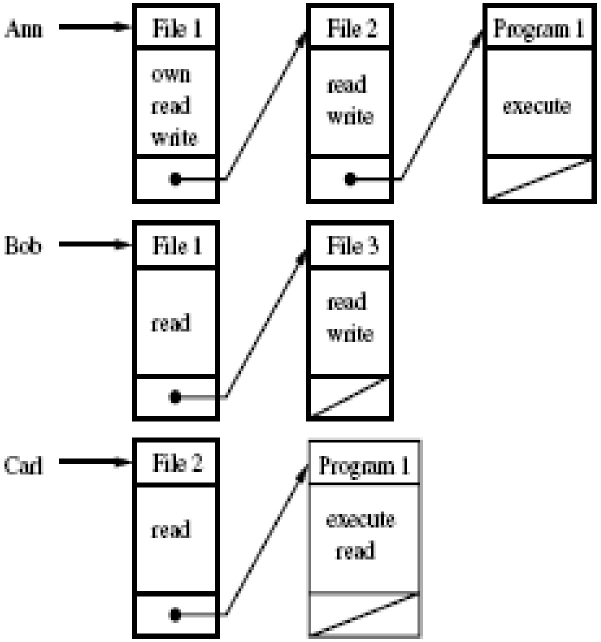
Access Control List (ACL)

- Matrix is stored by column.
- Each object is associated with a list
- Indicate for each subject the actions that the subject can exercise on the object



Capability List

- Matrix is stored by row
- Each user is associated with a capability list
- Indicating for each object the access that the user is allow to exercise on the object



ACLs vs Capability List

- Immediate to check the authorization holding on an object with ACLs. (subject?)
- Immediate to determine the privileges of a subject with Capability lists. (object?)
- Distributed system,
  - authenticate once, access various servers
  - choose which one?
- Limited number of groups of users, small bit vectors, authorization specified by owner.
  - Which one?

Basic Operations in Access Control

- Grant permissions
  - Inserting values in the matrix's entries
- Revoke permissions
  - Remove values from the matrix's entries
- Check permissions
  - Verifying whether the entry related to a subject *s* and an object *o* contains a given access mode

Access Control

Discretionary Access Control

- Access Matrix Model
- State of Protection System
- Implementation of the Access Matrix
- Vulnerabilities of the Discretionary Policies
- Additional features of DAC

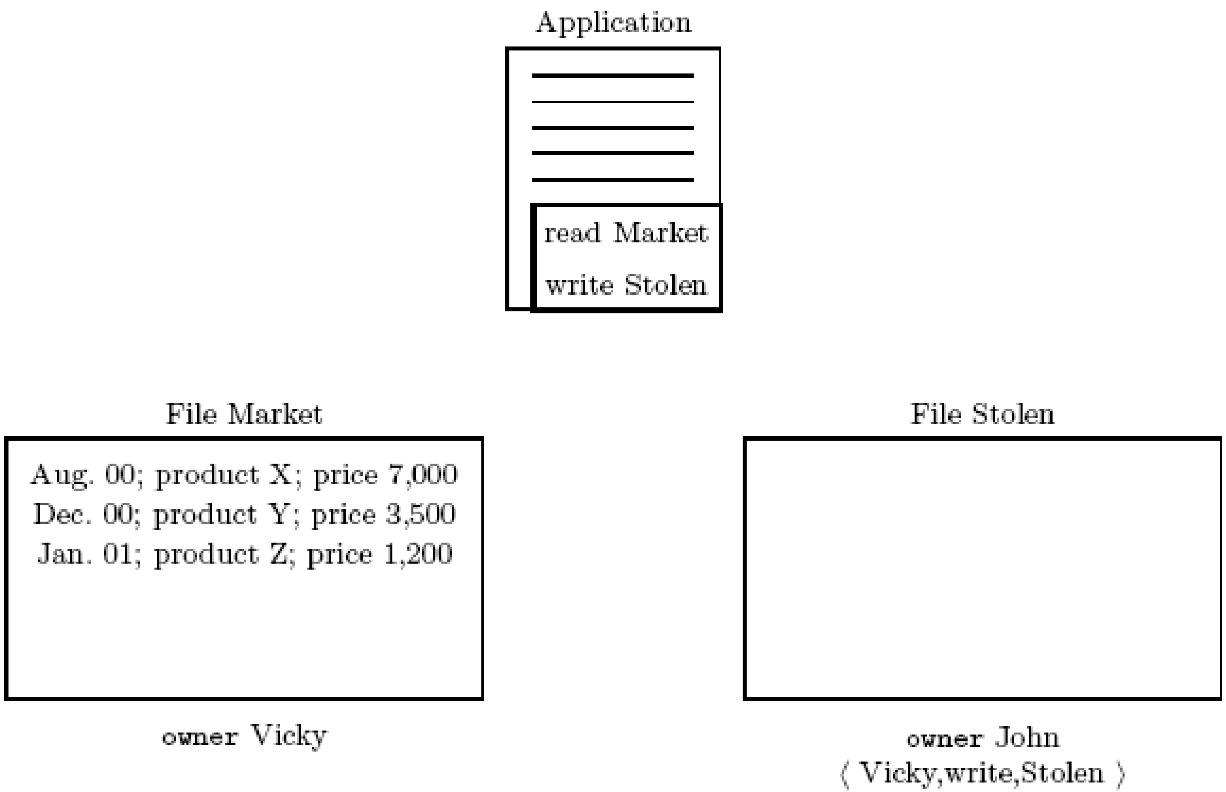
Vulnerabilities of the Discretionary Policies

- No separation of users from subjects
- No control on the flow the information
- Malicious code, i.e., Trojan horse

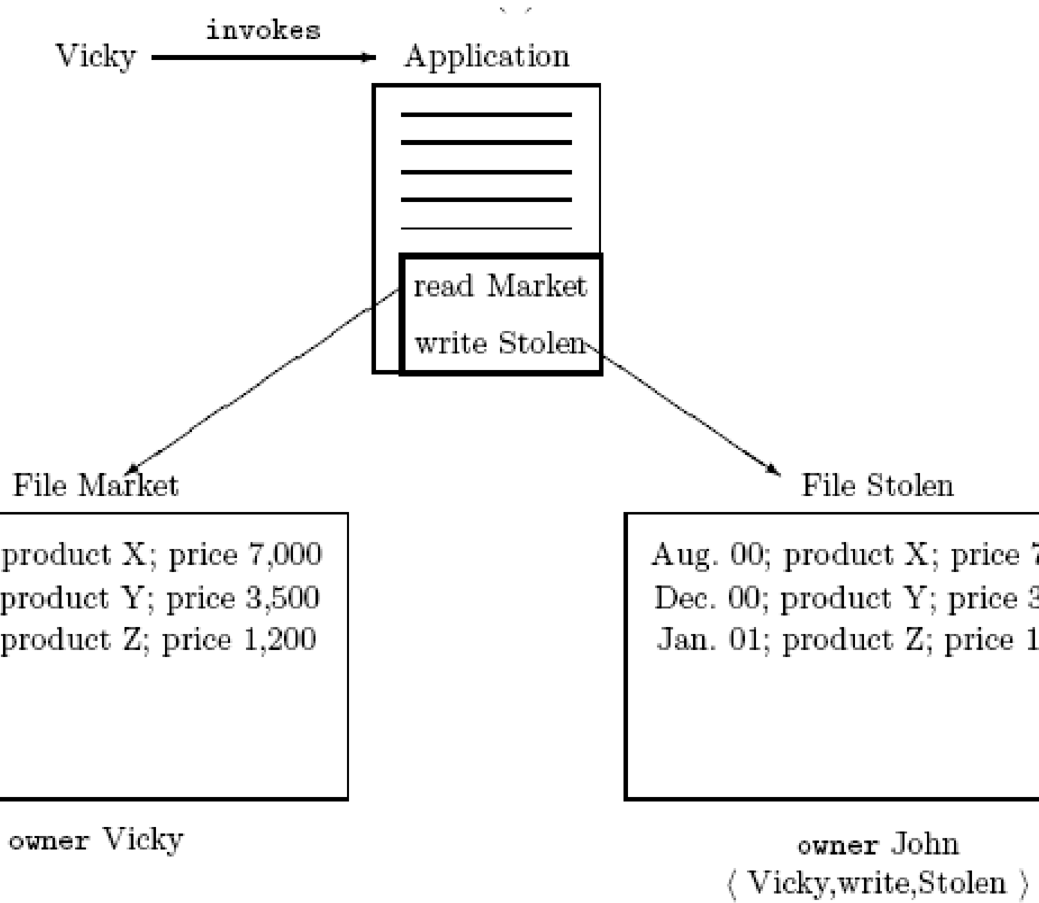
Example

- Vicky, a top-level manager
- A file Market on the new products release
- John, subordinate of Vicky
- A file called "Stolen"
- An application with two hidden operations
  - Read operation on file Market
  - Write operation on file Stolen

Example (cond)



Example (cond)



- Restriction should be enforced on the operations that processes themselves can **execute**.
- Mandatory policies provide a way to enforce **information flow control** through the use of labels

Access Control

- Discretionary Access Control
  - Access Matrix Model
  - State of Protection System
  - Implementation of the Access Matrix
  - Vulnerabilities of the Discretionary Policies
  - Additional features of DAC

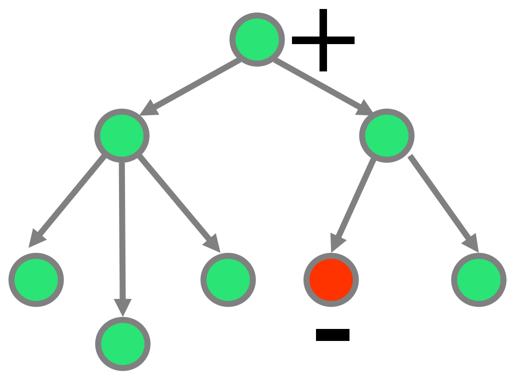
DAC – additional features and recent trends

- Flexibility is enhanced by supporting different kinds of permissions
  - Positive vs. negative
  - Strong vs. weak
  - Implicit vs. explicit
  - Content-based

Positive and Negative Permissions

- Positive permissions → Give access
- Negative permissions → Deny access
- Useful to specify exceptions to a given policy and to enforce stricter control on particular crucial data items

Positive and Negative Permissions



Main Issue: Conflicts

Authorization Conflicts

- Main solutions:
  - No conflicts
  - Negative permissions take precedence
  - Positive permissions take precedence
  - Nothing take precedence
  - Most specific permissions take precedence

Weak and Strong Permissions

- Strong permissions cannot be overwritten
- Weak permissions can be overwritten by strong and weak permissions

Implicit and Explicit Permissions

- Some models support implicit permissions
- Implicit permissions can be derived:
  - by a set of *propagation rules* exploiting the subject, object, and privilege hierarchies
  - by a set of user-defined *derivation rules*

Derivation Rules: Example

- Ann can read file F1 from a table if Bob has an explicit denial for this access
- Tom has on file F2 all the permissions that Bob has
- Derivation rules are a way to concisely express a set of security requirements

Derivation Rules

- Derivation rules are often expressed according to logic programming
- Several research efforts have been carried out to compare the expressive power of such languages
- We need languages based on SQL and/or XML

Content-based Permissions

- Content-based access control conditions the access to a given object based on its content
- This type of permissions are mainly relevant for database systems
- As an example, in a RDBMS supporting content-based access control it is possible to authorize a subject to access information only of those employees whose salary is not greater than 30K

Content-based Permissions

- Two most common approaches to enforce content-based access control in a DBMS are done:
  - by associating a predicate (or a Boolean combination of predicates) with the permission
  - by defining a *view* which selects the objects whose content satisfies a given condition, and then granting the permission on the view instead of on the basic objects

DAC models - DBMS vs OS

- Increased number of objects to be protected
- Different granularity levels (relations, tuples, single attributes)
- Protection of logical structures (relations, views) instead of real resources (files)
- Different architectural levels with different protection requirements
- Relevance not only of data physical representation, but also of their semantics

Access Control -- RBAC

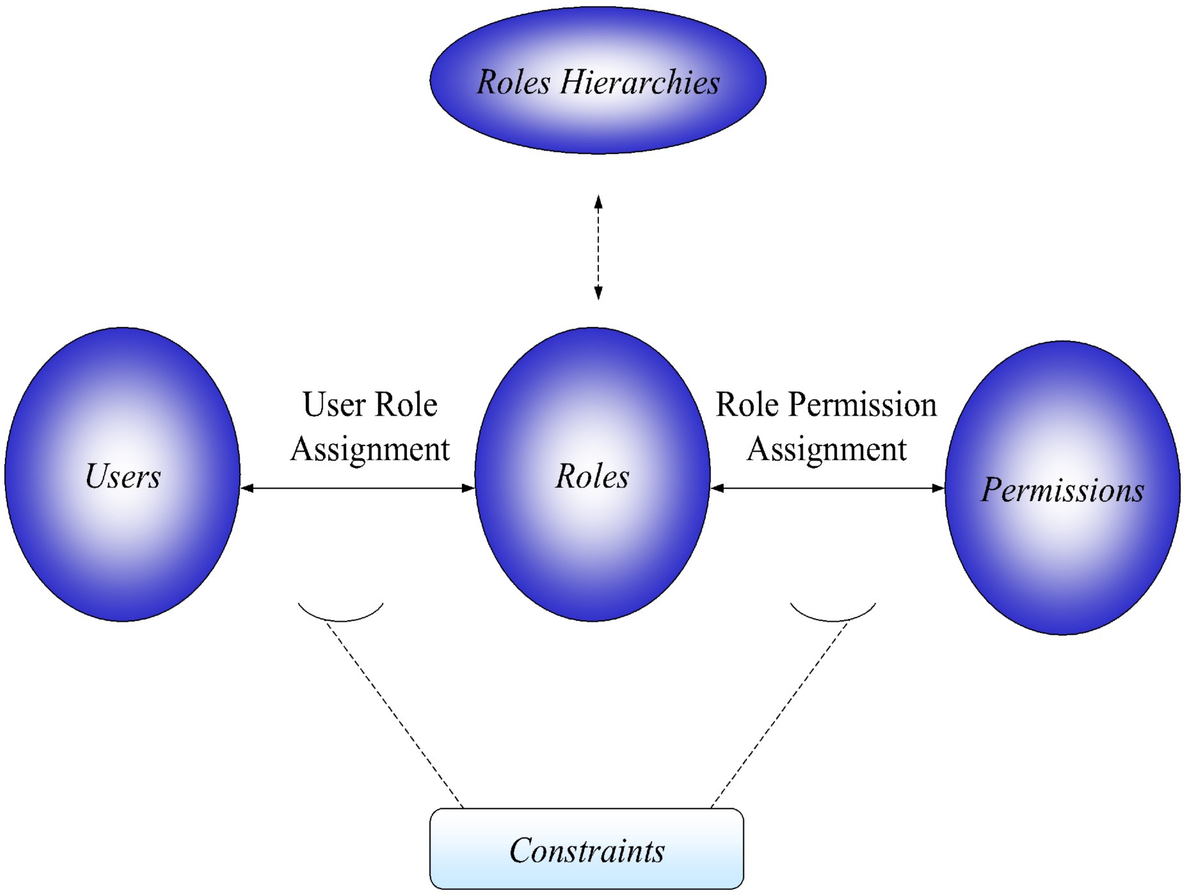
Lecture 4

RBAC

- Many organizations base access control decisions on "**the roles** that individual users take on as part of the organization".
- They prefer to <u>centrally control and maintain</u> access rights that reflect the organization's protection guidelines.
- With RBAC, <u>role-permission relationships can be predefined</u>, which makes it simple to assign users to the predefined roles.
- The combination of <u>users and permissions</u> tend to change over time, the permissions associated with a role are more stable.
- RBAC concept supports three well-known security principles:
  - Least privilege
  - Separation of duties
  - Data abstraction

Role Based Access Control (RBAC)

- Access control in organizations is based on "roles that individual users take on as part of the organization"
- A role is "is a collection of permissions"



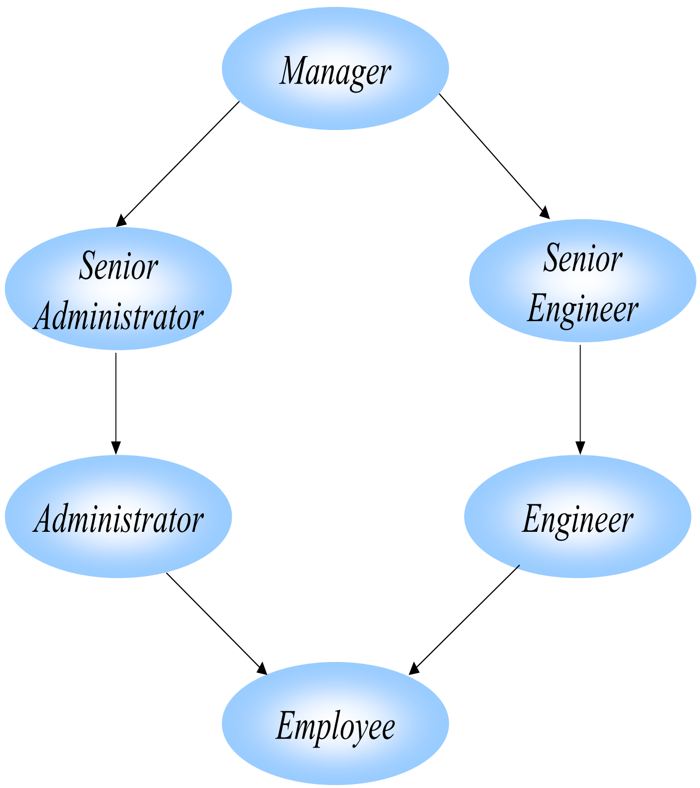
RBAC

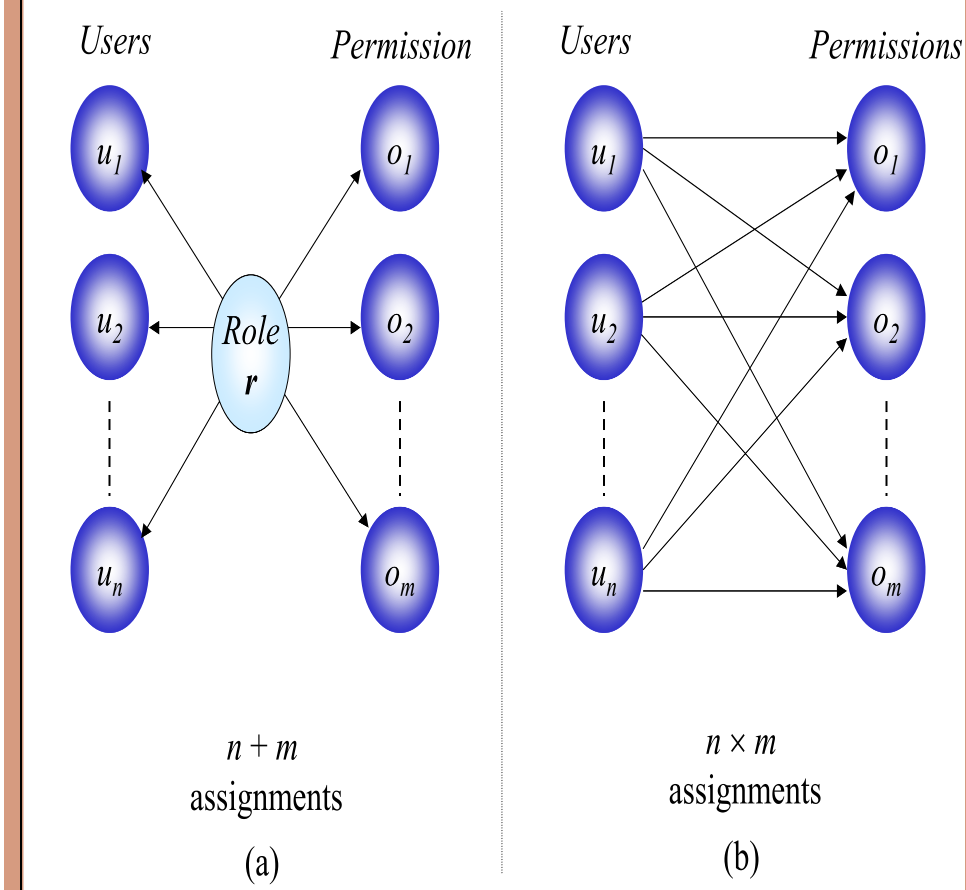
- Access depends on role/function, not identity
  - Example: Allison is **bookkeeper** for Math Dept. She has access to financial records. If she leaves and Betty is hired as the new **bookkeeper**, Betty now has access to those records. The role of "bookkeeper" dictates access, not the identity of the individual.

Advantages of RBAC

- Allows Efficient Security Management
  - Administrative roles, Role hierarchy
- Principle of least privilege allows minimizing damage
- Separation of Duties constraints to prevent fraud
- Allows grouping of objects
- Policy-neutral - Provides generality
- Encompasses DAC and MAC policies

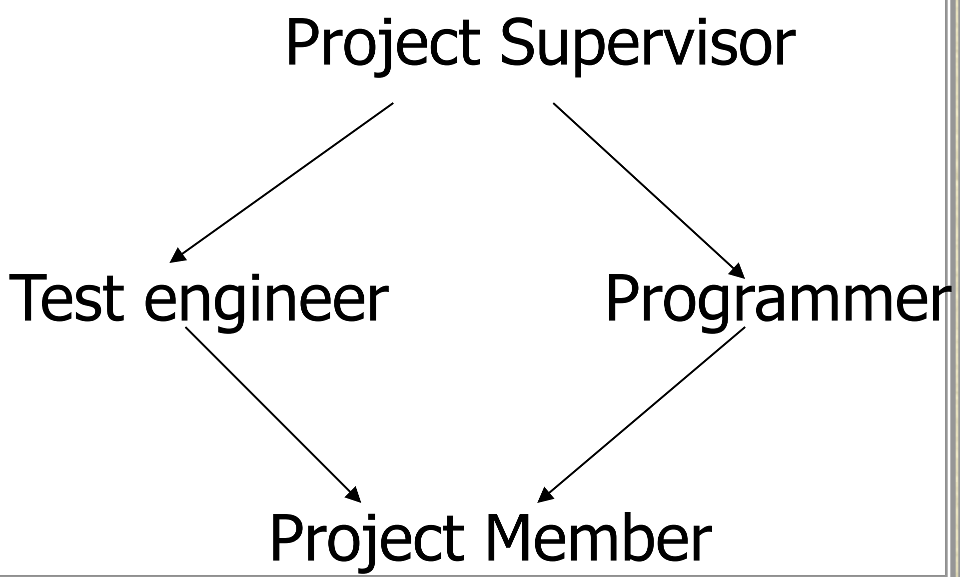
RBAC


Manager
Senior Administrator
Senior Engineer
Administrator
Engineer
Employee

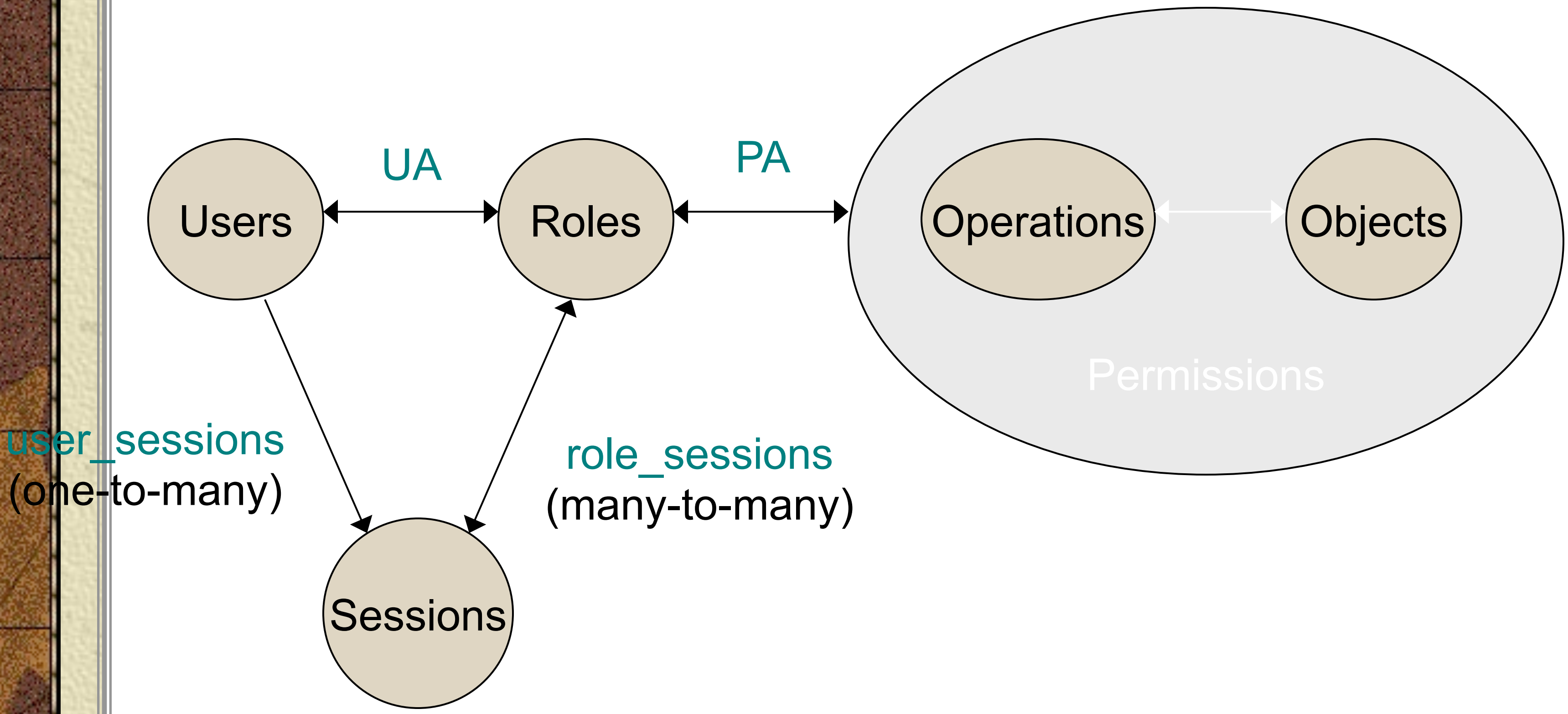

Users
Permission
u_1
u_2
u_n
Role r
o_1
o_2
o_m
$n + m$ assignments
(a)
Users
Permissions
u_1
u_2
u_n
o_1
o_2
o_m
$n \times m$ assignments
(b)

RBAC (cont'd)

- Is RBAC a discretionary or mandatory access control?
  - RBAC is policy neutral; however individual RBAC configurations can support a mandatory policy, while others can support a discretionary policy.
- Role Hierarchies
- Role Administration



RBAC (NIST Standard)

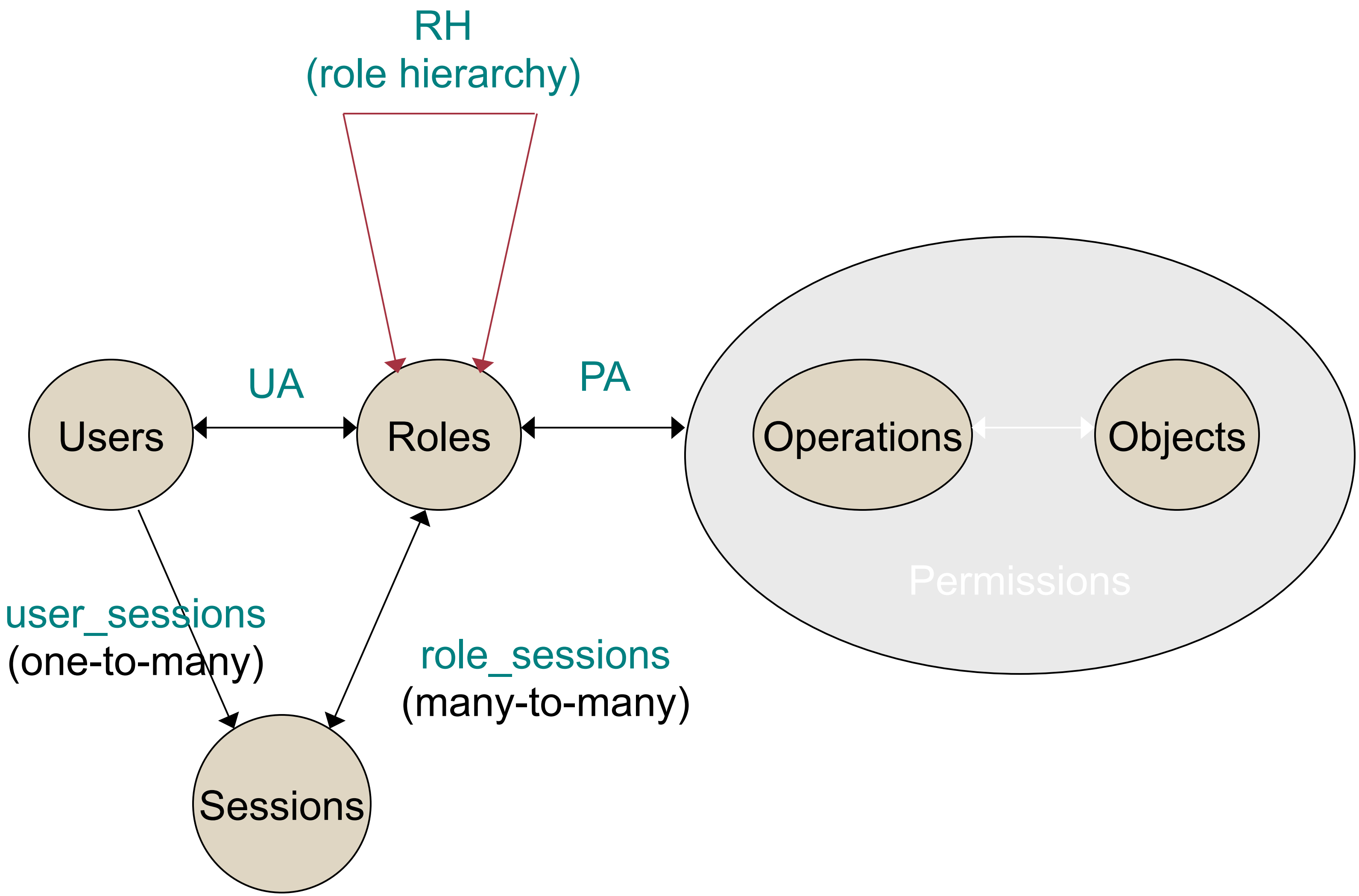


An important difference from classical models is that Subject in other models corresponds to a Session in RBAC

Core RBAC (relations)

- Permissions = $2^{\text{Operations x Objects}}$
- UA $\subseteq$ Users x Roles
- PA $\subseteq$ Permissions x Roles
- *assigned_users*: Roles $\rightarrow 2^{\text{Users}}$
- *assigned_permissions*: Roles $\rightarrow 2^{\text{Permissions}}$
- *Op*(p): set of operations associated with permission p
- *Ob*(p): set of objects associated with permission p
- *user_sessions*: Users $\rightarrow 2^{\text{Sessions}}$
- *session_user*: Sessions $\rightarrow$ Users
- *session_roles*: Sessions $\rightarrow 2^{\text{Roles}}$
  - *session_roles*(s) = $\{r \mid (\text{session\_user}(s), r) \in \text{UA})\}$
- *avail_session_perms*: Sessions $\rightarrow 2^{\text{Permissions}}$

RBAC with General Role Hierarchy



RBAC with General Role Hierarchy

- *authorized_users*: Roles$\rightarrow 2^{\text{Users}}$

  $authorized\_users(r) = \{u \mid r' \geq r \;\&(r', u) \in UA)$

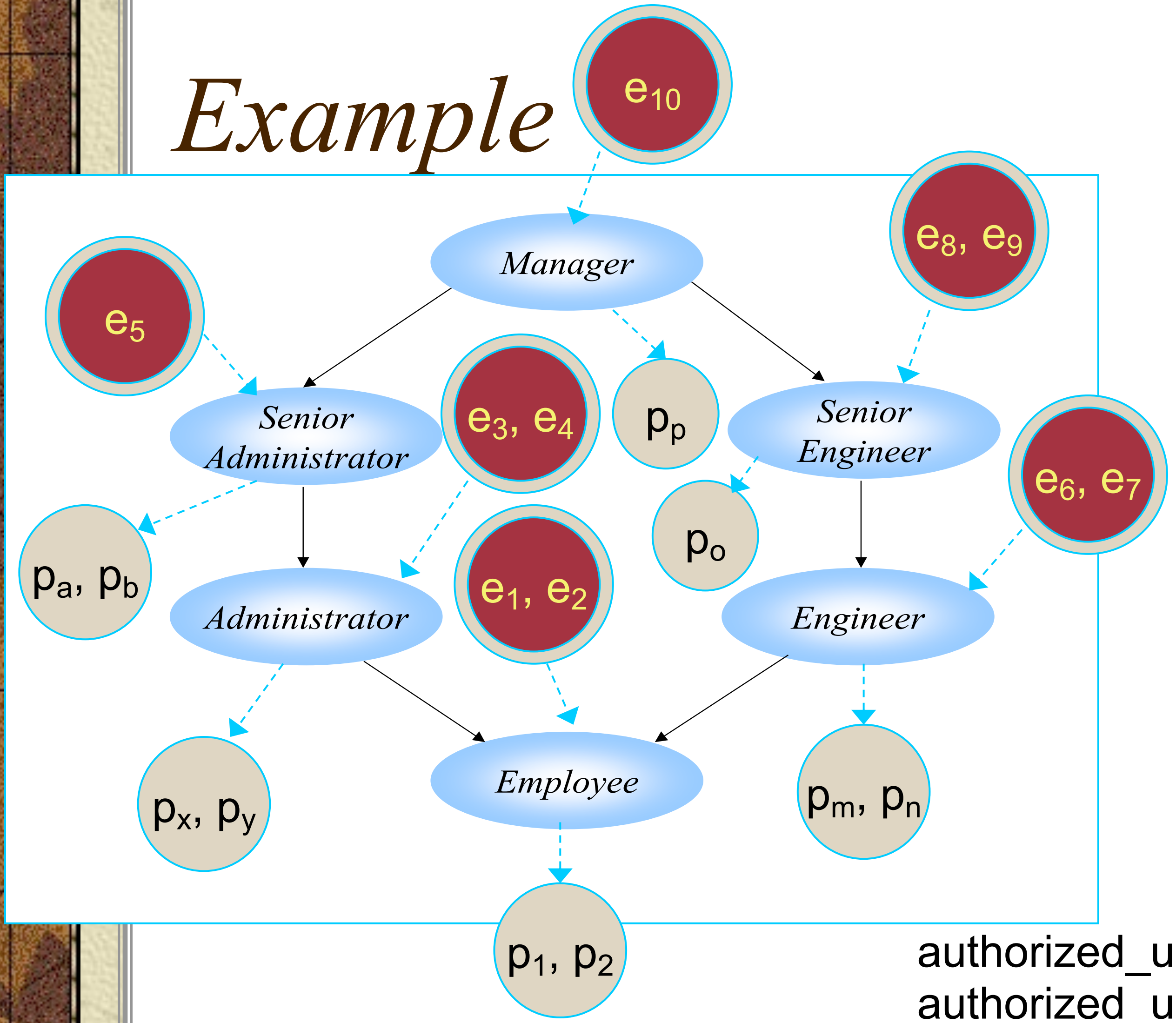
- *authorized_permissions*: Roles$\rightarrow 2^{\text{Permissions}}$

  $authorized\_users(\mathrm{r}) = \{p \mid r' \geq r \;\&(p, r') \in PA)$

- RH$\subseteq$ Roles x Roles is a partial order
  - called the inheritance relation
  - written as $\geq$.

  $(r_1 \geq r_2) \rightarrow authorized\_users(r_1) \subseteq authorized\_users(r_2)$ &
  $authorized\_permisssions(r_2) \subseteq authorized\_permisssions(r_1)$

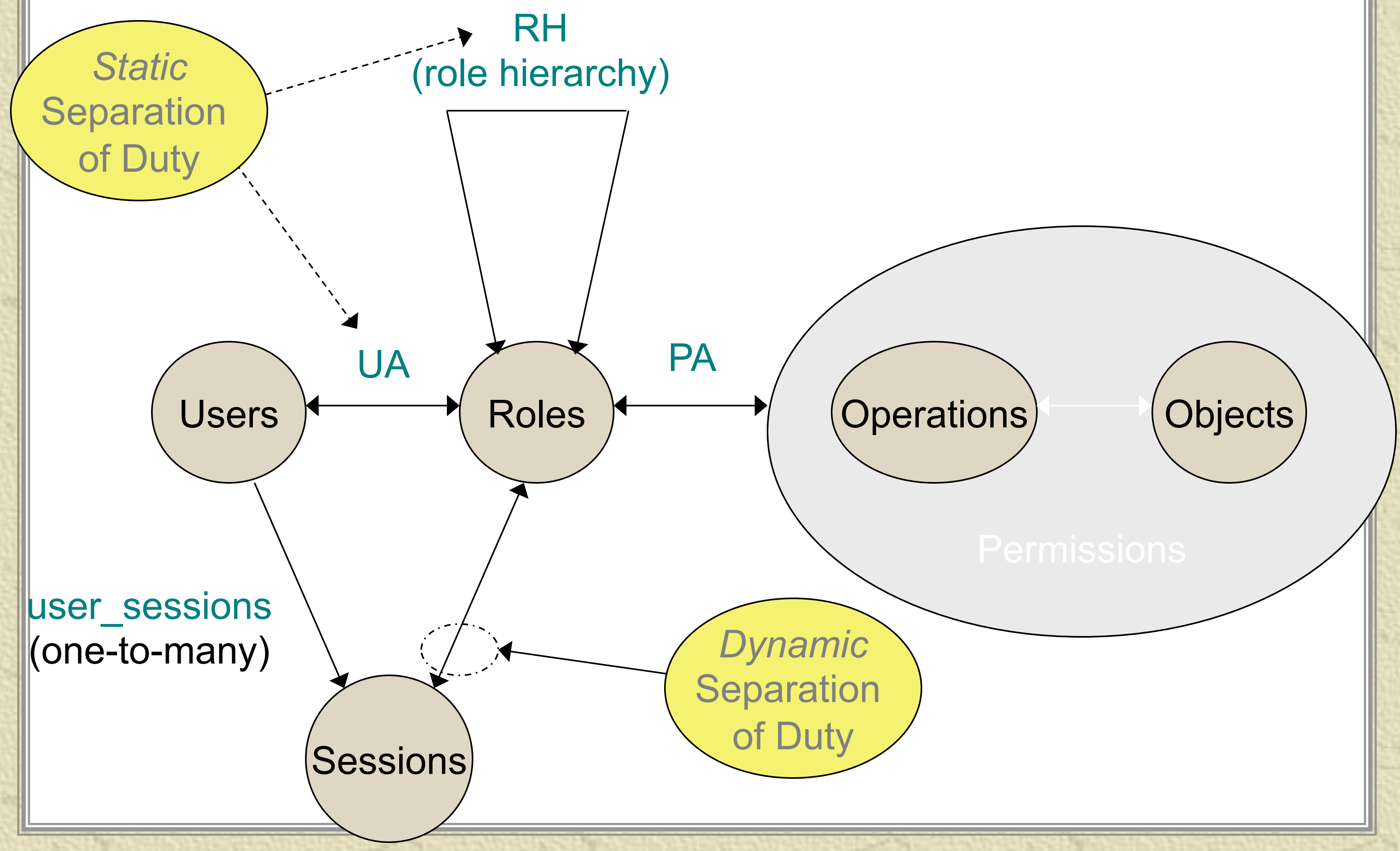
Example



authorized_users(Employee)?
authorized_users(Administrator)?

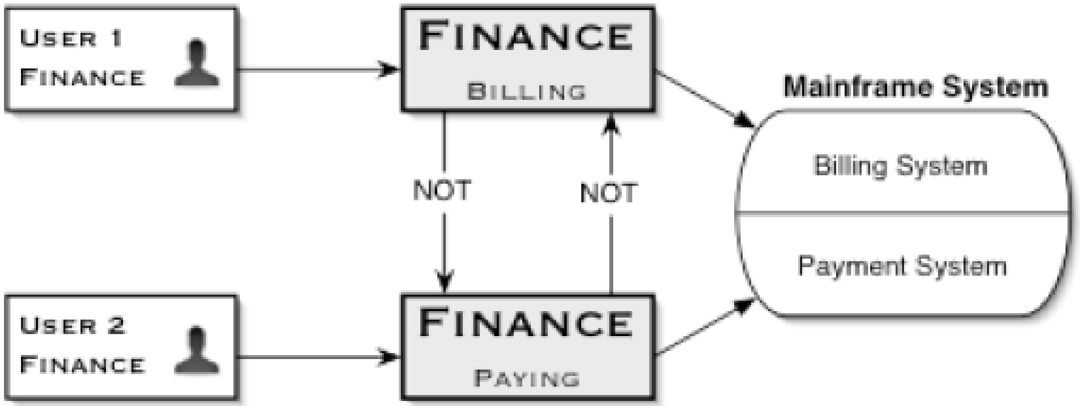
authorized_permissions(Employee)?
authorized_permissions(Administrator)?

Constrained RBAC



Separation of Duties

- No user should be given enough privileges to misuse the system on their own.
- Statically: defining the conflicting roles
- Dynamically: Enforcing the control at access time



RBAC's Benefits

TABLE 1: **ESTIMATED TIME (IN MINUTES) REQUIRED FOR ACCESS ADMINISTRATIVE TASKS**

TASK	RBAC	NON-RBAC	DIFFERENCE
Assign existing privileges to new users	6.14	11.39	5.25
Change existing users' privileges	9.29	10.24	0.95
Establish new privileges for existing users	8.86	9.26	0.40
Termination of privileges	0.81	1.32	0.51

Cost Benefits

- Saves about 7.01 minutes per employee, per year in administrative functions
  - Average IT admin salary - $59.27 per hour
  - The annual cost saving is:
    - $6,924/1000; $692,471/100,000
- Reduced Employee downtime
  - if new transitioning employees receive their system privileges faster, their productivity is increased
  - 26.4 hours for non-RBAC; 14.7 hours for RBAC
  - For average employee wage of $39.29/hour, the annual productivity cost savings yielded by an RBAC system:
    - $75000/1000; $7.4M/100,000