Chapter 4

Access Control

Access Control Definition

- RFC 4949 defines **computer security** as:

  "***Measures that implement and assure security services in a computer system,*** particularly those that assure access control service."

- NIST IR 7298 defines **access control** as:

  "a ***process*** by which ***use of system resources is regulated*** according to a security ***policy*** and is **permitted only** by ***authorized entities***"

Authentication vs. Authorization

- **Authentication** — Are you **who** you say you are?
  - Restrictions on **who** (or **what**) can access system
- **Authorization** — Are you **allowed** to do that?
  - Restrictions on **actions** of authenticated users
- **Authorization** is a form of **access control**
- Classic view of authorization…
  - Access Control Lists (**ACL**s)
  - Capabilities (**C-lists**)

  - *Access Control implements a security policy that specifies who (or what in the case of a process) **may have access** to each specific system resource and the type of access that is permitted in each instance.*

Access Control Principles

- **Authentication:** Verification that the credentials of a user or other system entity are valid.

- **Authorization:** Granting of a right or permission to a system entity to access a system resource
  - determines who is trusted for a given purpose.

- **Audit:** An independent **review** and examination of system records and activities in order to
  - test for adequacy of system controls
  - ensure compliance with established policy and operational procedures
  - detect breaches in security
  - to recommend any indicated changes in control, policy and procedures.

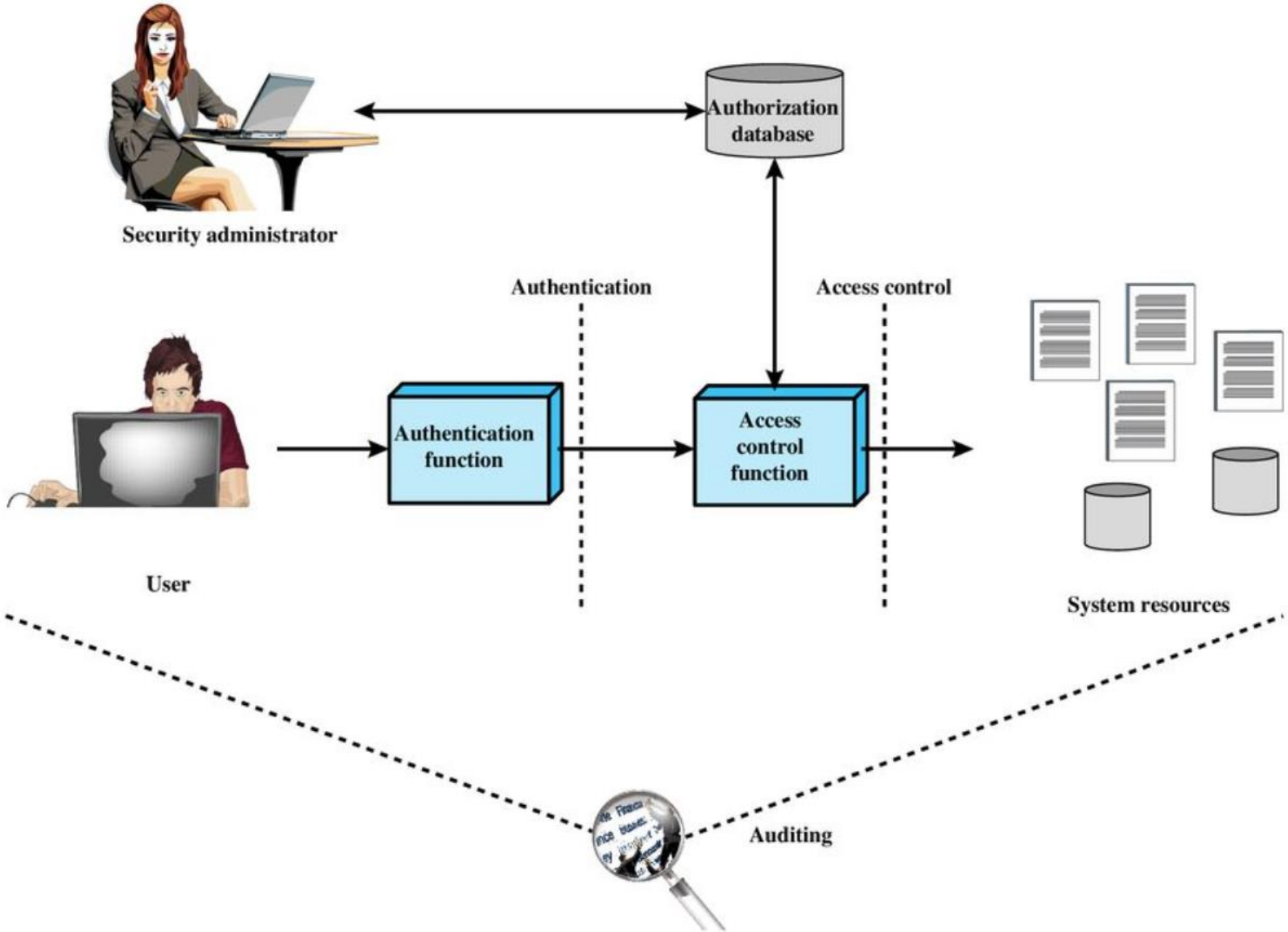


Figure 4.1 Relationship Among Access Control and Other Security Functions

Access Control

- The central element of computer security.

- The principal ***objectives*** of computer security are
  - to **prevent** **unauthorized** users from gaining access to resources,
  - to **prevent** **legitimate** users from accessing resources in an **unauthorized** manner,
  - and to **enable** **legitimate** users to access resources in an **authorized** manner

Lampson's Access Control Matrix

- This matrix contains all of the relevant information needed by an OS to make decisions about **which users** are allowed to **do** with the various system **resources**
  - **Subjects** (users) index the *rows*
  - **Objects** (resources) index the *columns*

	OS	Accounting program	Accounting data	Insurance data	Payroll data
Bob	rx	rx	r	—	—
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	rw	rw
Accounting program	rx	rx	rw	rw	rw

The model assumes a **set of subjects**, a **set of objects**, and a **set of rules** that govern the access of subjects to objects.

Access Control Policies

1) **Discretionary** Access Control (**DAC**)

- Controls access based on the **identity of the requestor** and on **access rules** (authorizations) stating what requestors are (or are not) allowed to do (the owner of the access permission can pass it to others).

2) **Mandatory** Access Control (**MAC**)

- Controls access based on comparing security **labels** with security **clearances** (subject clearance and object labels)

3) **Role-Based** Access Control (**RBAC**)

- Controls access based on the **roles** that users have within the system and on **rules** stating what accesses are allowed to users in given roles

4) **Attribute-Based** Access Control (**ABAC**)

- Controls access based on **attributes** of the user, the resource to be accessed, and current environmental conditions

***Note**: these four policies are not mutually exclusive. An access control mechanism can employ two or even all of these policies to cover different classes of system resources.*

Subjects, Objects, and Access Rights

The basic elements of access control are: **subject**, **object**, and **access right.**

Subject

- An entity capable of accessing objects. Held accountable for all actions
- Three classes
  - Owner
  - Group
  - World

Object

- A resource to which access is controlled
- Entity used to contain and/or receive information

Access right

- Describes the way in which a subject may access an object
- Could include:
  - Read
  - Write
  - Execute
  - Delete
  - Create
  - Search

Subjects & Objects

- A **subject** is typically held accountable for the actions they have initiated
  - an audit track may be used to record the association of a subject with security relevant actions performed on an object by the subject.
    - **Owner**: This may be the creator of a resource, such as a file. E.g. a **project administrator** or **leader** may be assigned **ownership**.
    - **Group**: a named group of users may also be granted access rights, E.g. membership in the group is sufficient to exercise these access rights. *a user may belong to multiple groups.*
    - **World**: The *least amount of access* is granted to users who are able to access the system but are not included in the categories owner and group
- An **object** is a resource to which access is controlled.
  - entity used to contain and/or receive information.
  - E.g. records, blocks, pages, segments, files, portions of files, directories, directory trees, mailboxes, messages, and programs

Access Right

- Describes the way in which a subject may access an object
  - **Read**: User may **view** information in a system resource.
    - E.g. a file, selected records in a file, selected fields within a record, or some combination).
    - **Read** access includes the **ability to copy or print**.
  - **Write**: User may **add**, **modify**, or **delete** data in system resource
    - E.g. files, records, programs.
  - **Execute**: User may **execute** specified programs.
  - **Delete**: User may **delete** certain system resources, such as files or records.
  - **Create**: User may **create** new files, records, or fields.
  - **Search**: User may **list** the files in a directory or otherwise **search** the directory.

Discretionary Access Control (DAC)

- Traditional method of implementing access control
- Scheme in which an **entity may enable another entity to access some resource**
  - i.e. applied by operating system or a database management system
- Often provided using an **access matrix** (*Lampson's Access Control Matrix*)
  - One dimension consists of identified **subjects** that may attempt data access to the resources
  - The other dimension lists the **objects** that may be accessed
- Each **entry** in the matrix indicates the **access rights** of a particular subject for a particular object

Simple Example of an Access Matrix

		OBJECTS			
		File 1	File 2	File 3	File 4
SUBJECTS	User A	Own Read Write		Own Read Write	
	User B	Read	Own Read Write	Write	Read
	User C	Read Write	Read		Own Read Write

(a) Access matrix

E.g.

- User A owns files 1 and 3 and has read and write access rights to those files.
- User B has read access rights to file 1, etc.

Access Control Lists (ACLs)

- **ACL**: store Lampson's access control matrix by ***column***
- E.g.: **ACL** for **insurance data** is in **blue**

- **ACLs are preferable** when:
  - users manage their ***own files*** and
  - protection is ***data oriented***.
- With **ACLs**, it's also easy to change rights to a particular resource.

	OS	Accounting program	Accounting data	Insurance data	Payroll data
Bob	rx	rx	r	—	—
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	rw	rw
Accounting program	rx	rx	rw	rw	rw

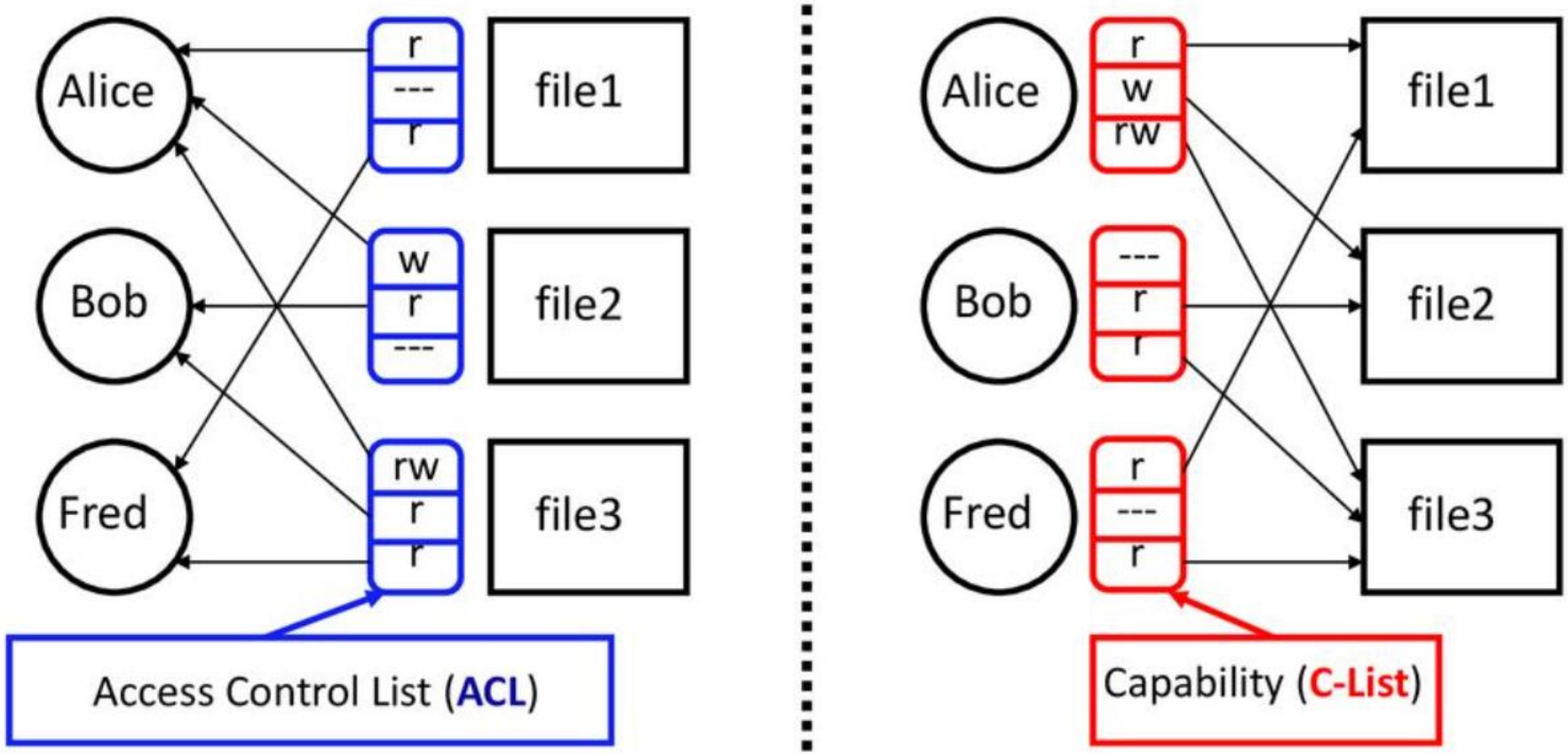
Capabilities (or **C-Lists**)

- Store access control matrix by ***row***
- E.g.: Capability (**C-List**) for **Alice** is in **red**

- With **C-Lists**, it is easy to delegate (and sub-delegate and sub-sub-delegate, and so on), and it is easier to **add** or **delete** users.

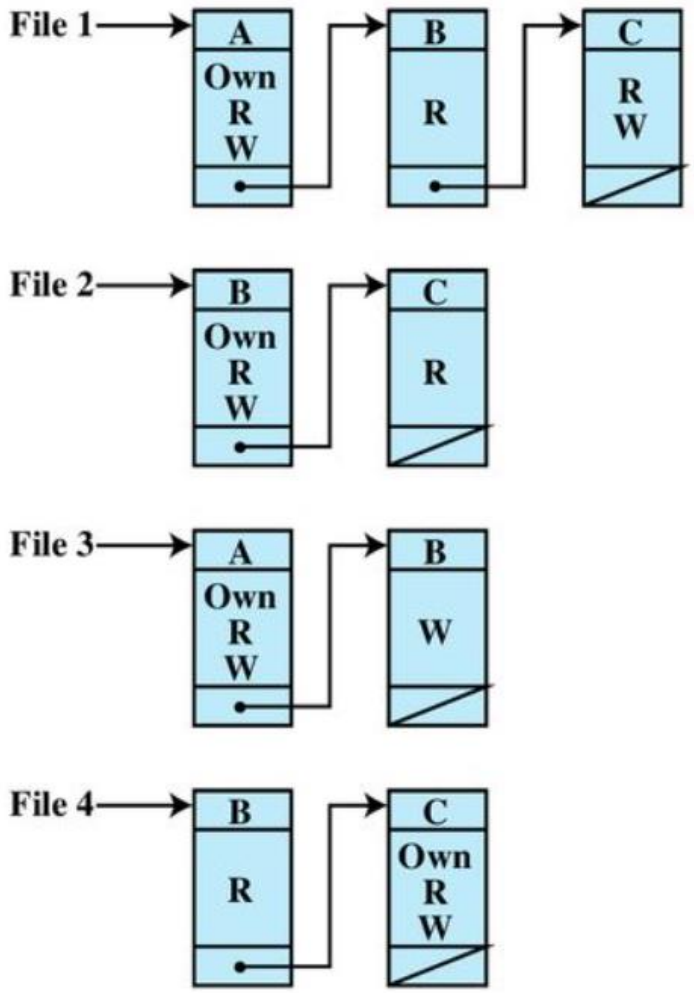
	OS	Accounting program	Accounting data	Insurance data	Payroll data
Bob	rx	rx	r	—	—
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	rw	rw
Accounting program	rx	rx	rw	rw	rw

ACLs vs. Capabilities



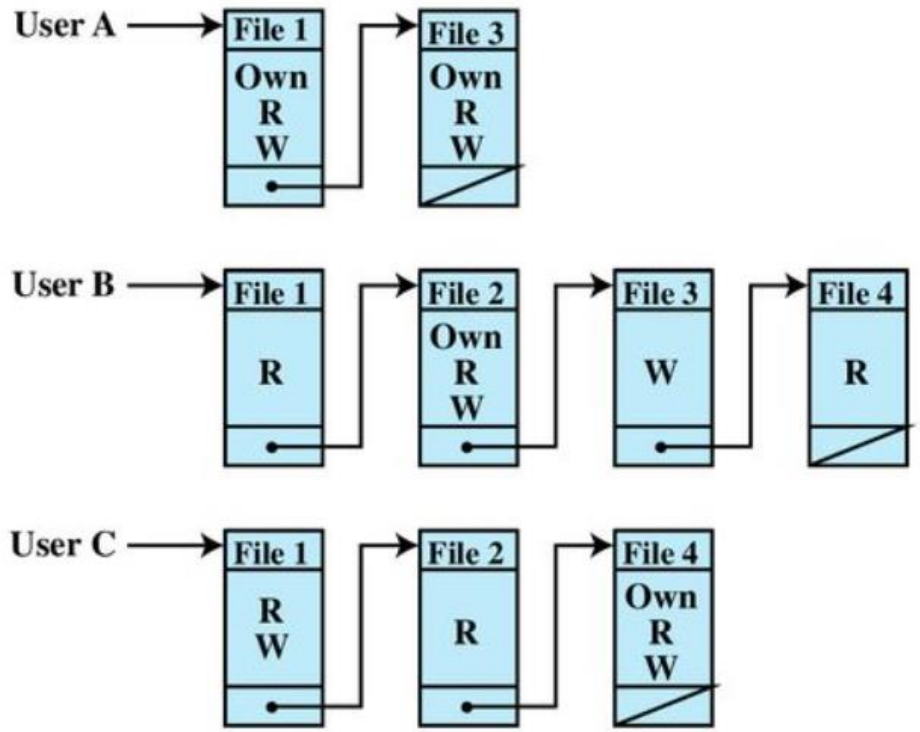
- Note that arrows point in opposite directions…
- With ACLs, still need to associate users to files

ACL



(b) Access control lists for files of part (a)

C-List



(c) Capability lists for files of part (a)

In practice, an access matrix is usually sparse and is implemented by decomposition in one of two ways.

Figure 4.2 Example of Access Control Structures

Role-Based Access Control (RBAC)

- **Traditional DAC systems define the access rights of individual users and groups of users. In contrast, RBAC** is based on:
  - **Roles** that users assume in a system (***instead*** *of their Identity*)
  - Role is a job function within an organization. A role will have specific access rights to one or more resources.
  - Assign Access Rights to Roles (***instead*** *of individual users.*)
  - Users assigned to **different** Roles according to their Responsibilities.
  - Users-to-Roles are Many-to-Many.

- The set of **Users changes** frequently (***dynamic***), and the assignment of a user to one or more roles may also be ***dynamic***.
- The set of **Roles is relatively *static***, with only occasional additions or deletions.
- The set of **Resources** and the specific access rights associated with a particular role are also likely to change infrequently (***relatively static***).

- Access rights are assigned to **Roles** *instead* of **individuals**
- Users are assigned to **Roles**. (statically or dynamically, Based on responsibilities)
- Users to Roles are Many-to-Many
- Users may change frequently
- Often, **Roles** are static
- A **Role** has specific access rights

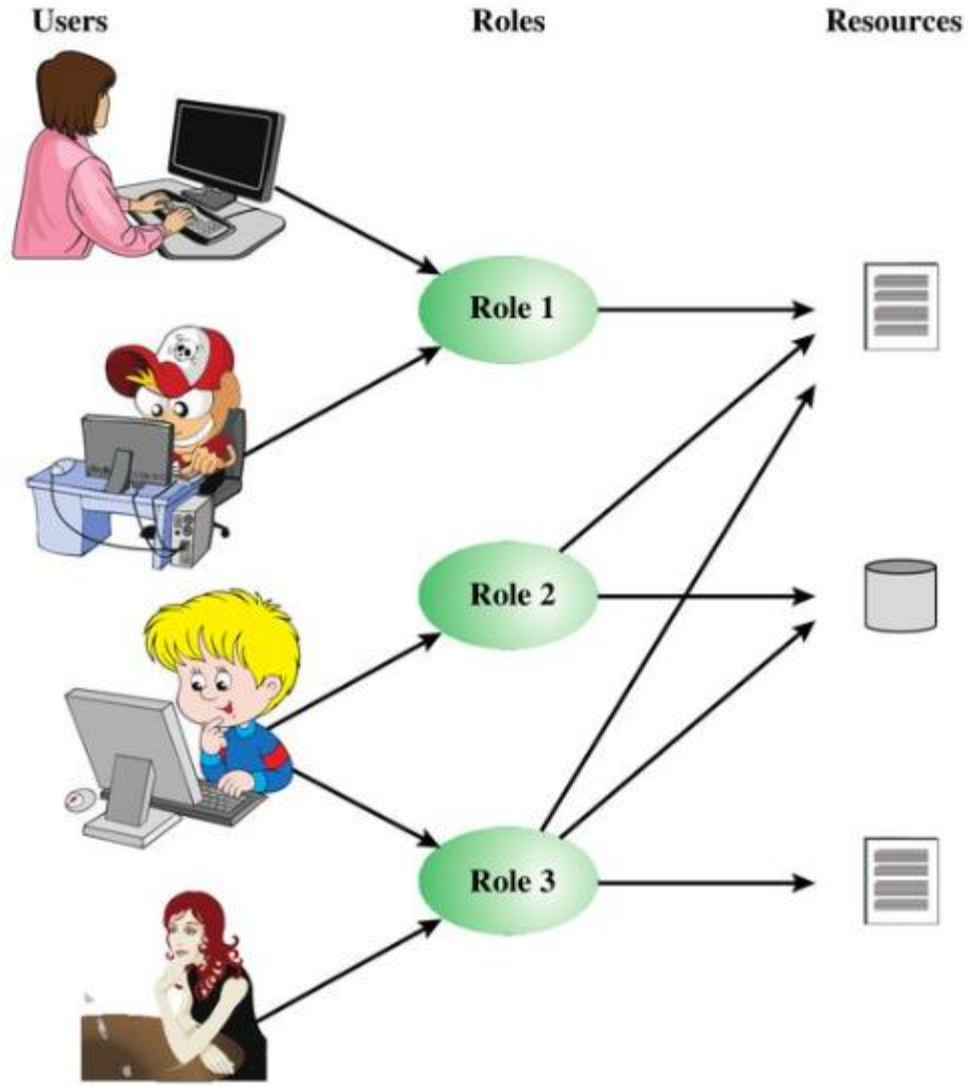


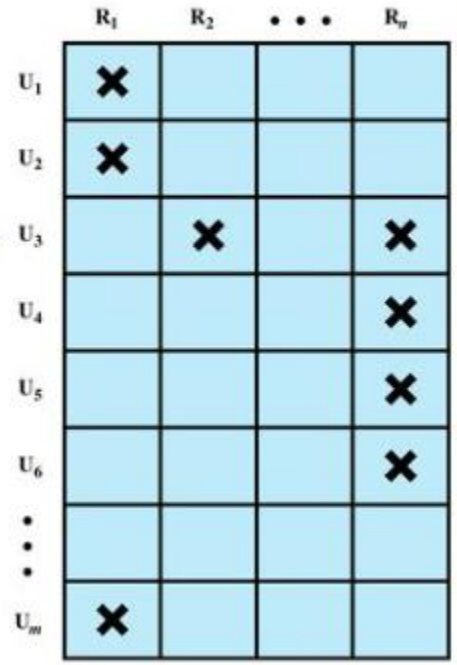
Figure 4.6 Users, Roles, and Resources

Best practice for using RBAC

- **RBAC allows to**
  - Segregate duties within a team and
  - Grant only the amount of access to users that they need to perform the jobs.
- Instead of giving everybody (group) unrestricted permissions on a resource, you can allow only **certain actions at a particular scope**.
- Planning the access control strategy, it's a best practice to **grant users the least privilege** to get their work done.
  - Each role should contain the ***minimum set of access rights needed for that role***.
- A role assignment consists of three elements:
  - **Security principal**, (object that represents a user, group, service principal)
  - **Role definition**, (collection of permissions.)
  - **Scope**, (set of resources that the access applies to)

- Relates individual users to roles
- Typically there are many more users than roles
- Each entry is either blank or marked
- A user may be assigned multiple roles

Typically, few **Roles** & many **Users**,



- A role contains the minimum set of access rights.
- A user is assigned to a role that enables him/her to perform only what is required.
- Multiple users may be assigned to the same Role.

- has the same structure as the **DAC** access control matrix, with **roles as subjects**

OBJECTS

ROLES	R_1	R_2	R_n	F_1	F_1	P_1	P_2	D_1	D_2
R_1	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
R_2		control		write *	execute			owner	seek *
• • •									
R_n			control		write	stop			

Figure 4.7 Access Control Matrix Representation of RBAC

Constraints - RBAC

- Provide a means of adapting **RBAC** to the specifics of **administrative** and **security policies** of an organization
- A **constraint** is a **defined relationship among roles** or a condition related to roles. Types:

Mutually Exclusive Roles

- A **user** can only be assigned to **one role** in the set (*either during a **session** or **statically***)
- Any **permission** (access right) can be granted to only **one role** in the set
- **Separation** of duties and capabilities (no collusion among individuals, **roles have non-overlapping permissions**)

Cardinality

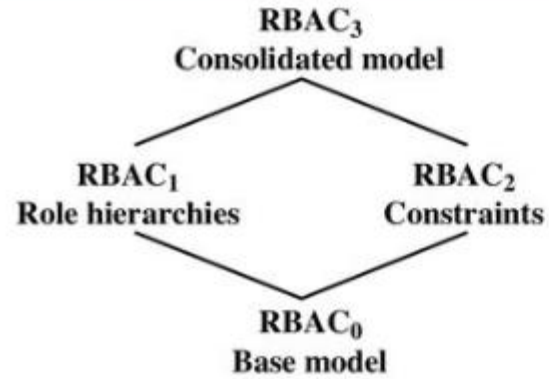
- Setting a maximum **number** with respect to roles. E.g. Constraint
- the maximum number of **users** that can be assigned to a **given role**
- the number of roles that a user is assigned to,
- the number of roles a user can activate for a single session

Prerequisite Roles

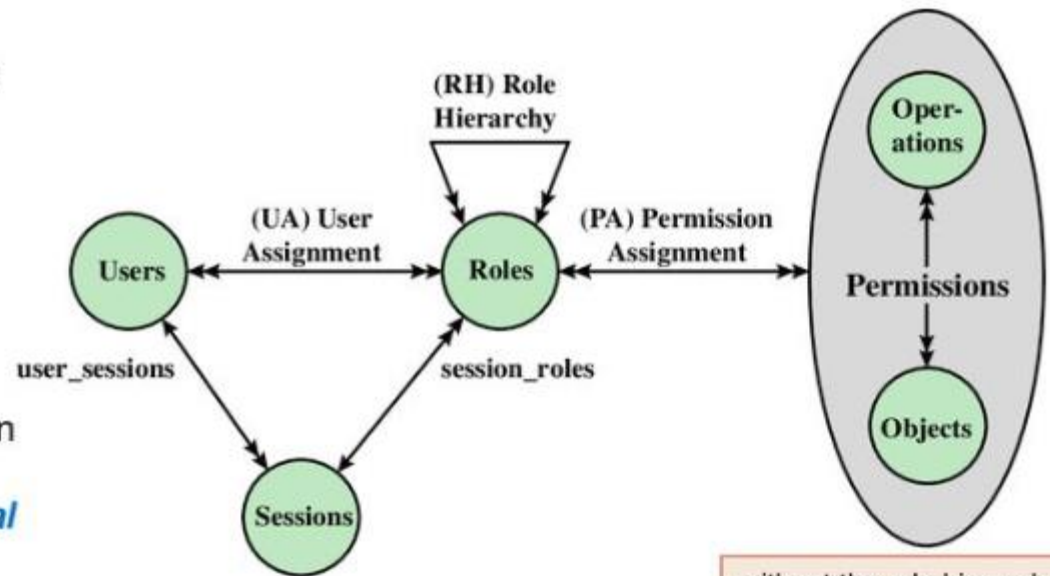
- **Dictates** that a user can only be assigned to a particular role if it is already assigned to some other specified role
- E.g. can be used to structure the impl. of the **least privilege concept**

Abstract models of RBAC functionality

- **Role**: A named **job** function within the organization that controls this computer system. (***authority*** & ***responsibility***)
- **Permission**: An **approval** of a particular mode of **access** to one or more objects. (***access right***, ***privilege***, ***authorization***).
- **Session**: A mapping between a ***user*** and an ***activated subset*** **of the** ***set of roles*** to which the user is assigned.
  - **Temporary** one-to-many relationship
  - **Least privilege**: Only needed roles
- One user may have multiple roles, and multiple users may be assigned to a single role (*many-to-many*).
- Flexibility and granularity: the **many-to-many** relationships between users and roles and between roles and permissions (***not found in conventional DAC schemes***).
  - Without this flexibility and granularity, there is a risk that a user may be granted more access to resources than is needed



(a) Relationship among RBAC models



(b) RBAC models

without the role hierarchy and constraints, contains the four types of entities in an $RBAC_0$ system:

Figure 4.8 A Family of Role-Based Access Control Models.

Table 4.3 Scope RBAC Models

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

- **$RBAC_0$** contains the minimum functionality for an RBAC system
- **$RBAC_1$** includes the **$RBAC_0$** functionality and adds role hierarchies, which enable one role to inherit permissions from another role
- **$RBAC_2$** includes **$RBAC_0$** and adds constraints, which restrict the ways in which the components of a **RBAC** system may be configured
- **$RBAC_3$** contains the functionality of **$RBAC_0$**, **$RBAC_1$**, and **$RBAC_2$**

An example of a diagram of a role *Hierarchy*

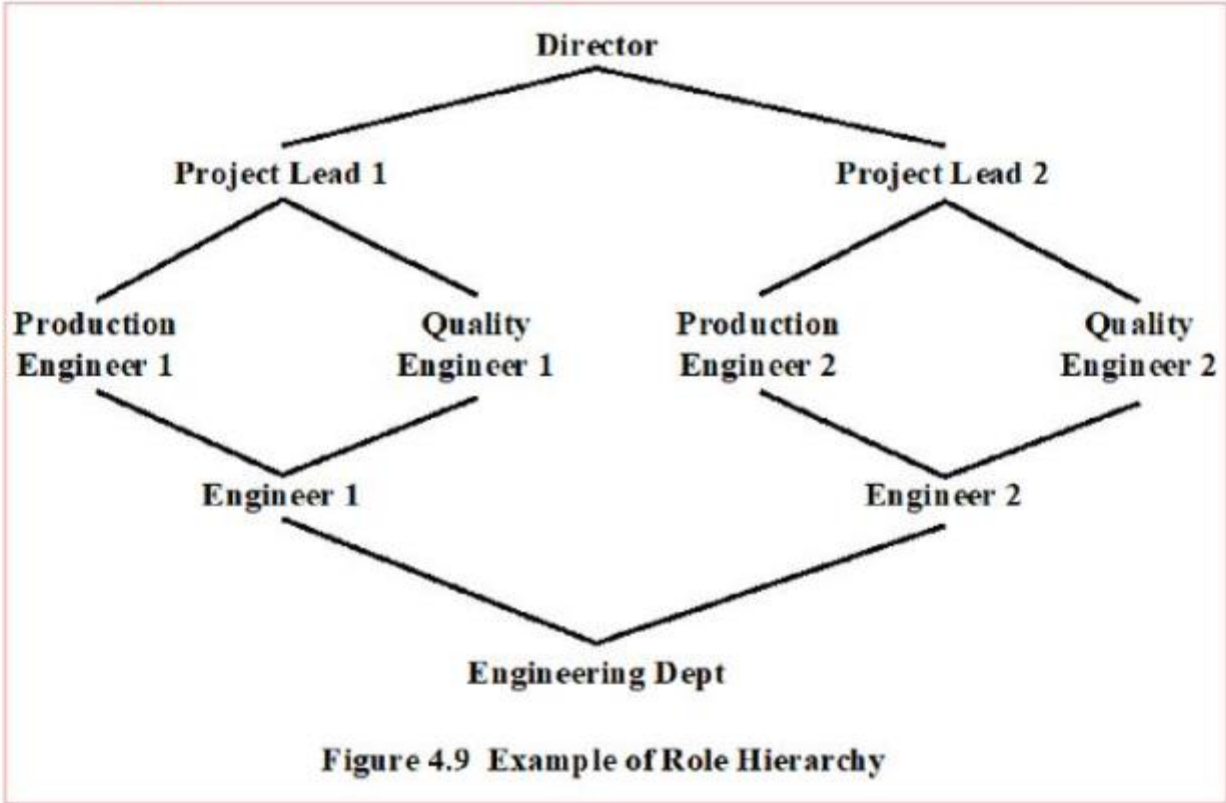


Figure 4.9 Example of Role Hierarchy

- Role hierarchies reflect the hierarchical structure of roles in an organization.
- A line between two roles implies that the upper role includes all of the access rights of the lower role,
- Typically, job functions with greater responsibility have greater authority to access resources
- Role hierarchies make use of the concept of inheritance to enable one role to implicitly include access rights associated with a subordinate role.

```
CREATE USER user
   { {
       IDENTIFIED

        {
           BY password [ [HTTP] DIGEST { ENABLE | DISABLE } ]
           | EXTERNALLY [ AS 'certificate_DN'  |  AS 'kerberos_principal_name' ]
           | GLOBALLY [ AS '{ directory_DN | { AZURE_USER | AZURE_ROLE }=value
                                          | {IAM_GROUP_NAME | IAM_PRINCIPAL_NAME}=value  }']
           }

        }
          | NO AUTHENTICATION
   }
```

```
CREATE ROLE role
   [ NOT IDENTIFIED
   | IDENTIFIED { BY password
                | USING [ schema. ] package
                | EXTERNALLY
                | GLOBALLY AS domain_name_of directory_group
        }
   ] [ CONTAINER = { CURRENT | ALL } ] ;
```

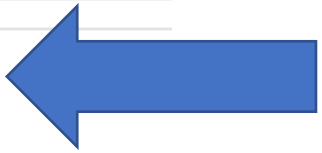
```
GRANT {privilege | role}[,{privilege | role} ...]

    TO {username | rolename | PUBLIC}[,{username | rolename | PUBLIC} ...]
    [WITH ADMIN OPTION]
```

Learn Oracle CREATE ROLE Statement By Practical Examples
https://www.oracletutorial.com/oracle-administration/oracle-create-role/

Grant - Oracle privileges - Oracle - SS64.com
https://ss64.com/ora/grant.html

Create Role - Oracle - SS64.com
https://ss64.com/ora/role_c.html

Create User - Oracle - SS64.com
https://ss64.com/ora/user_c.html



See Week # 7 materials for these links

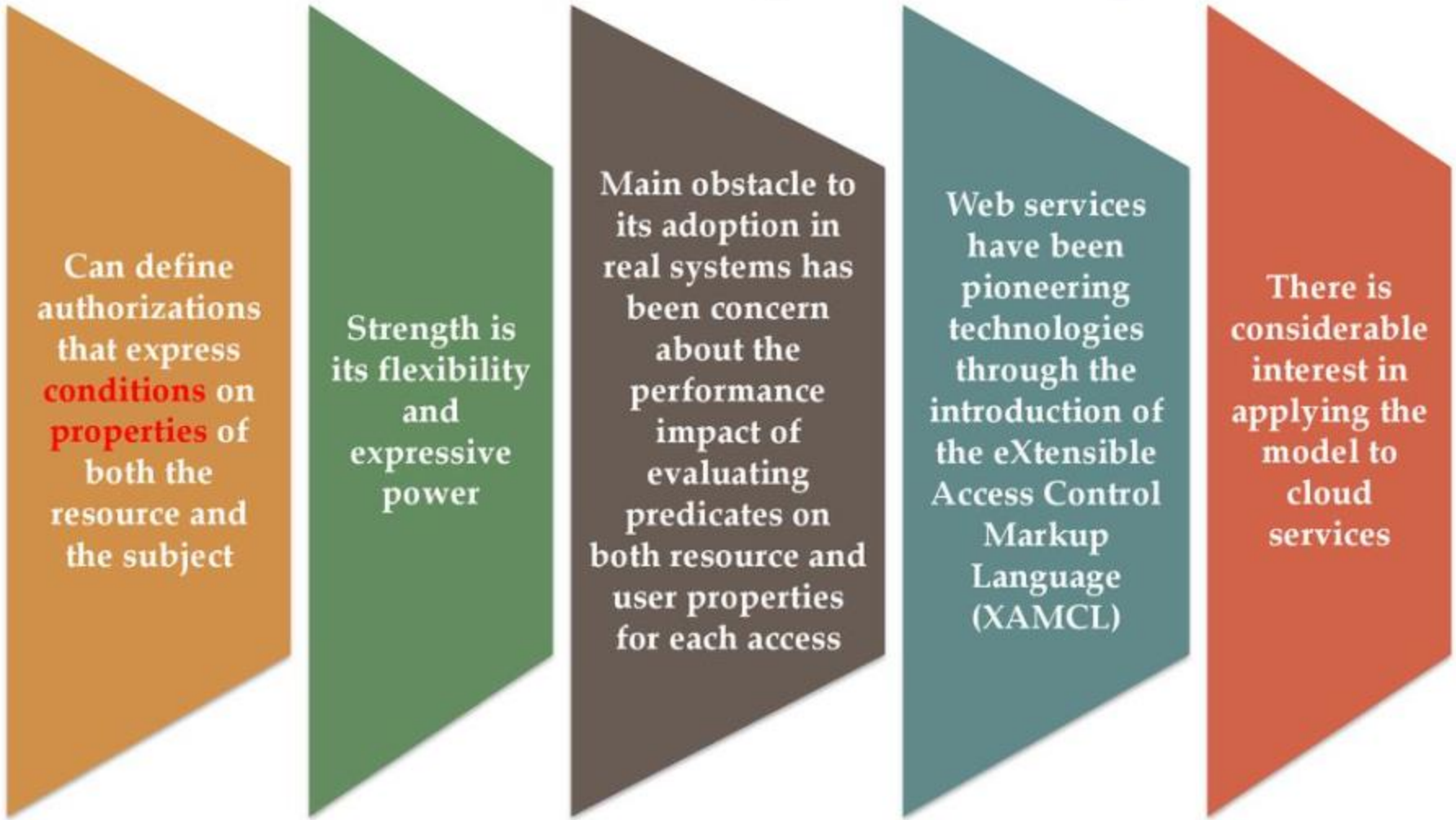
Advantage of RBAC

- Once implemented RBAC simplifies system administration
- Strong support for separation of duties
- Good auditing support
- Considered best practice by many

Attribute-based Access Control (ABAC)

- **ABAC** is a logical access control model that **controls access to objects by evaluating rules against the attributes of entities** (subject and object), operations, and the environment relevant to a request.
- Define **authorizations** that express **conditions** on **properties** of both ***the resource and the subject***
  - E.g. consider a configuration in which each resource has an **attribute** that identifies the subject that created the resource.
  - Then, a single access rule can specify the ownership privilege for all the creators of every resource
- **Pros:**
  - The strength of the ABAC approach is its flexibility and expressive power
- **Cons:**
  - Main obstacle to its adoption in real systems has been concern about the performance impact of evaluating predicates on both resource and user properties for each access.

Attribute-Based Access Control (ABAC)



ABAC Model: Attributes

- A **subject** is an **active entity** that causes information to flow among objects or changes the system state
- **Attributes** define the **identity** and **characteristics** of the subject

- An **object** (or **resource**) is a **passive information system-related entity** containing or receiving information
- **Objects** have **attributes** that can be leveraged to **make access control decisions**

- Describe the **operational**, technical, and even situational environment or context in which the information access occurs
- These attributes have so far been largely ignored in most access control policies

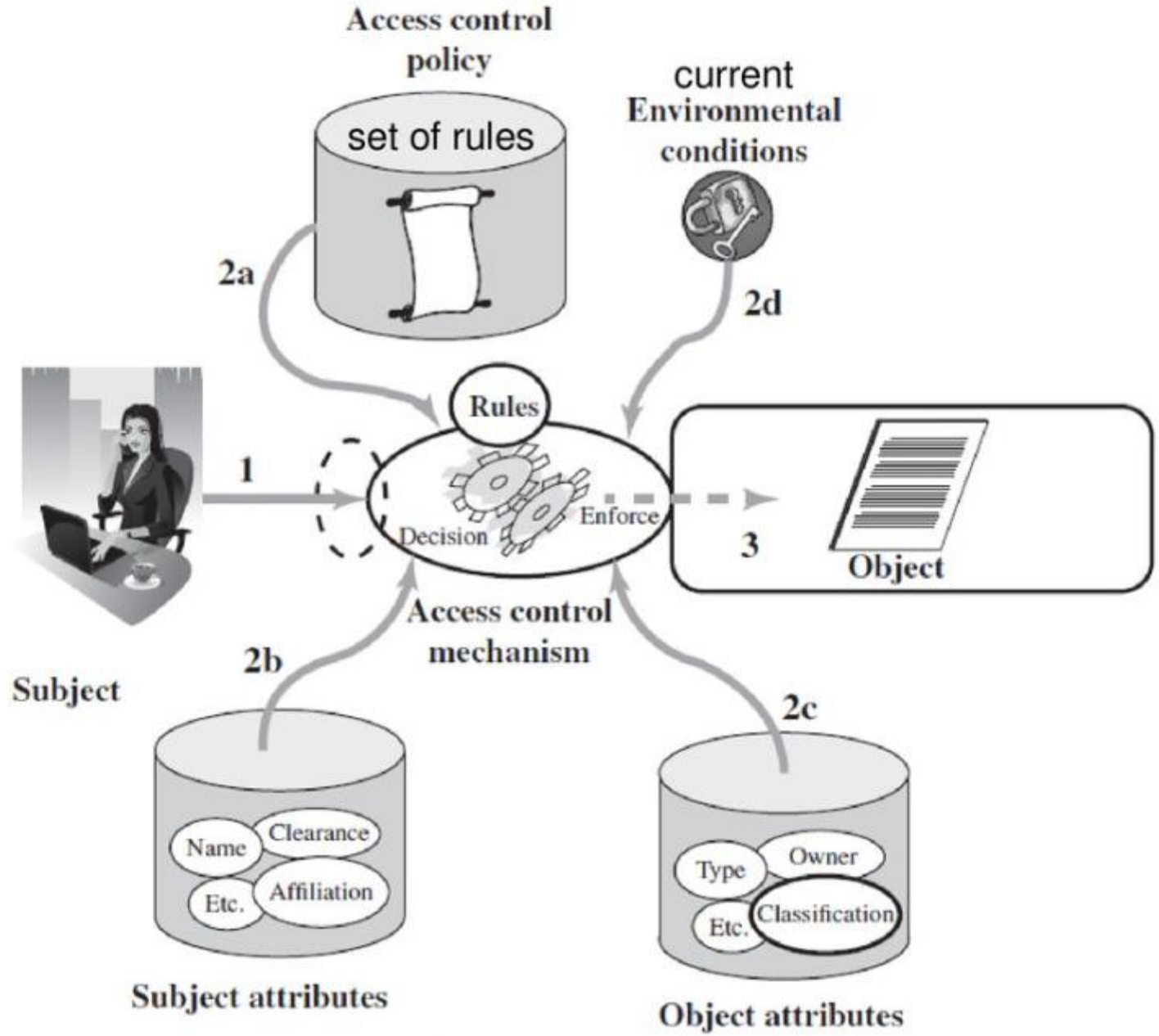


Figure 4.10 **Simple ABAC Scenario to determine authorization**

Attribute-based access control

1. fine-grained access control
2. context-aware access control
3. dynamic access control

DistriNet
iMinds
KU LEUVEN