# Access Control

The central element of computer security to achieve the goals of

- enabling legitimate users to access resources in an authorized manner
- preventing legitimate users from accessing resources in an unauthorized manner
- preventing unauthorized users from gaining access to resources

# Learning Objectives

- Explain how access control fits into the **broader context** that includes authentication, authorization, and audit
- Distinguish among **subjects**, **objects**, and **access rights**
- Define the three **major categories** of access control policies
- Discuss the principal concepts of the Discretionary Access Control (**DAC**) model
- Describe the UNIX file access control model
- Discuss the principal concepts of Role-Based Access Control (**RBAC**) model
- Discuss the principal concepts of attribute-based access control (**ABAC**) model
- Explain the **identity**, **credential**, and **access management** model
- Understand the concept of identity federation and its relationship to a **trust framework**

# Index

- Access Control Principles
- Subjects, Objects, and Access Rights
- Discretionary Access Control
- Example: UNIX File Access Control
- Role-Based Access Control
- Attribute-Based Access Control
- Identity, Credential, and Access Management
- Trust Frameworks
- Case Study: RBAC System for a Bank

# Index

# Two definitions of access control

## Are useful in understanding its **scope**

The process of granting or denying specific requests to:

(1) obtain and use information and related information processing services

(2) enter specific physical facilities

[NIST IR 7298, *Glossary of Key Information Security Terms* 2013]

A process by which use of system resources is regulated according to a security policy and is permitted only by authorized entities (users, programs, processes, or other systems) according to that policy

[RFC 4949, *Internet Security Glossary*, Internet Engineering Task Force (IETF)]

# Access Control Security Requirements

NIST SP 800-171 (*Protecting Controlled Unclassified Information in Nonfederal Information Systems and Organizations*, August 2016), provides a useful list of security requirements for access control services

CUI = controlled unclassified information

| | Basic Security Requirements |
|---|---|
| **1** | Limit information system access to authorized users, processes acting on behalf of authorized users, or devices (including other information systems). |
| **2** | Limit information system access to the types of transactions and functions that authorized users are permitted to execute. |
| | **Derived Security Requirements** |
| **3** | Control the flow of CUI in accordance with approved authorizations. |
| **4** | Separate the duties of individuals to reduce the risk of malevolent activity without collusion. |
| **5** | Employ the principle of least privilege, including for specific security functions and privileged accounts. |
| **6** | Use non-privileged accounts or roles when accessing nonsecurity functions. |
| **7** | Prevent non-privileged users from executing privileged functions and audit the execution of such functions. |
| **8** | Limit unsuccessful logon attempts. |
| **9** | Provide privacy and security notices consistent with applicable CUI rules. |
| **10** | Use session lock with pattern-hiding displays to prevent access and viewing of data after period of inactivity. |
| **11** | Terminate (automatically) a user session after a defined condition. |
| **12** | Monitor and control remote access sessions. |
| **13** | Employ cryptographic mechanisms to protect the confidentiality of remote access sessions. |
| **14** | Route remote access via managed access control points. |
| **15** | Authorize remote execution of privileged commands and remote access to security-relevant information. |
| **16** | Authorize wireless access prior to allowing such connections. |
| **17** | Protect wireless access using authentication and encryption. |
| **18** | Control connection of mobile devices. |
| **19** | Encrypt CUI on mobile devices. |
| **20** | Verify and control/limit connections to and use of external information systems. |
| **21** | Limit use of organizational portable storage devices on external information systems. |
| **22** | Control CUI posted or processed on publicly accessible information systems. |

# Access Control Principles

- All of computer security is concerned with access control

RFC 4949 (*Internet Security Glossary*) defines *computer security* as follows: **measures that implement and assure security services in a computer system, particularly those that assure access control service**

- We deal with a more specific concept of access control:

Access control implements a **security policy** that specifies **who** or **what** (e.g., 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 Context

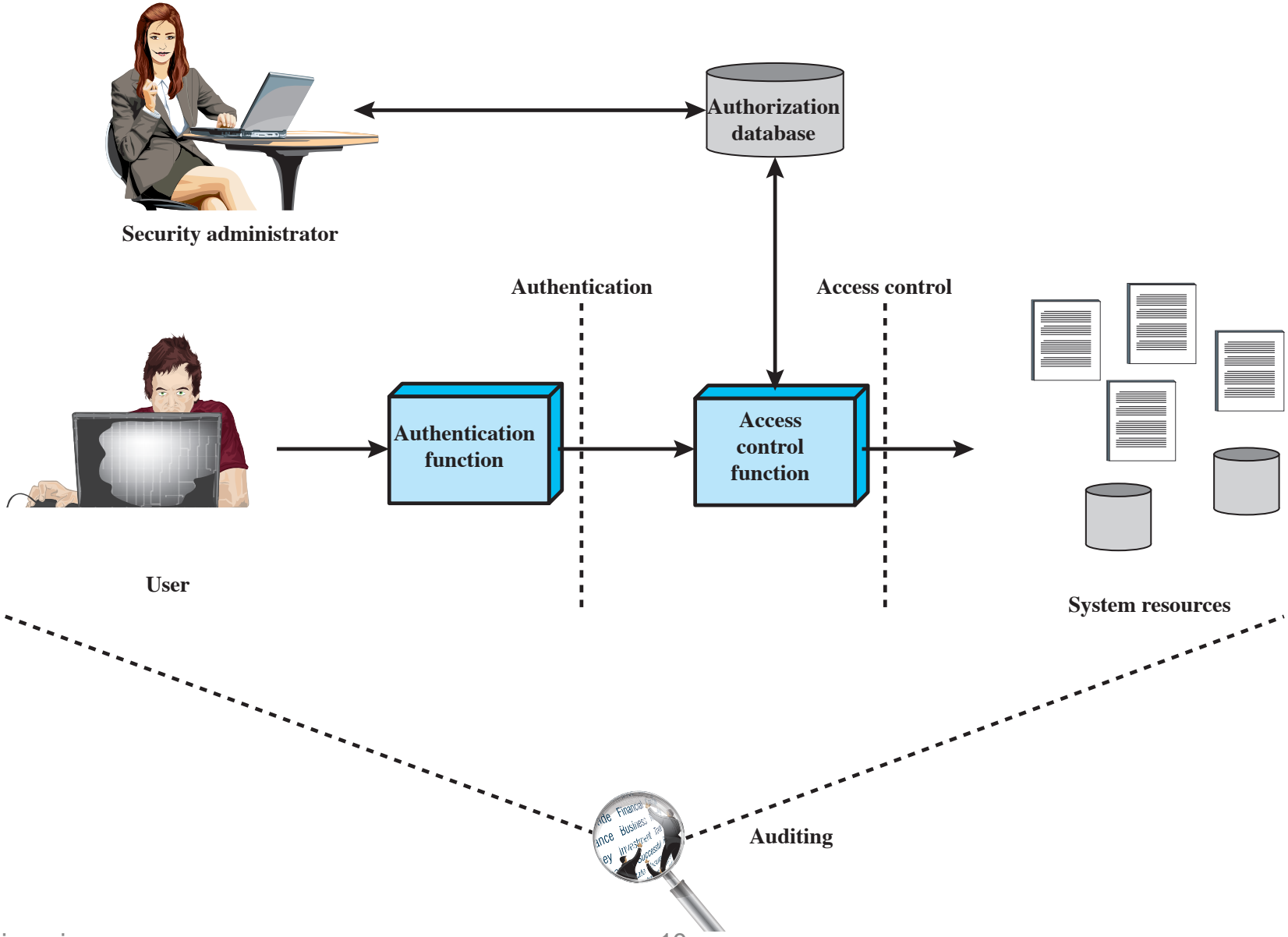
The *access control context* also involves the following functions:

- Authentication: Verification that the credentials of a user or other system entity are valid
- Authorization: The 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 examination of system records and activities in order to
  - test for adequacy of system controls
  - ensure compliance with established policies and operational procedures
  - detect breaches in security
  - recommend any indicated changes in controls, policies and procedures

# Relationship among Access Control and other Security Functions

- First the authentication function determines whether the user is permitted to access the system at all
- Then the access control function determines if the specific requested access by this user is permitted
  - The access control mechanism *mediates* between the user (or a process executing on behalf of the user) and system resources, such as applications, OSs, firewalls, routers, files, and databases
- A security administrator maintains an authorization database embodying **access control policies** which specify *what type of access to which resources is allowed for this user*
  - The access control function consults this database to determine whether to grant access
- An auditing function monitors and keeps a record of user accesses to system resources

# Relationship among Access Control and other Security Functions



# Relationship among Access Control and other Security Functions

In practice, a *number of components* may contribute to the access control function

- All OSs have at least a rudimentary, and in many cases a quite robust, access control component
- Add-on security packages can supplement the native access control capabilities of the OS
- Specific applications or utilities, such as database management systems, also incorporate access control functions
- External devices, such as firewalls, can also provide access control services

# Access Control Policies

An access control policy dictates what types of access are permitted, under what circumstances, and by whom

- 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

- Mandatory access control (MAC)

  Controls access based on comparing **security labels** (which indicate how sensitive or critical system resources are) with **security clearances** (which indicate if system entities are eligible to access certain resources)

- 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

- Attribute-based access control (ABAC)

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

# Access Control Policies

- DAC is the traditional method of implementing access control
  - *Discretionary* means that an entity might have access rights that permit the entity, by its own volition, to enable another entity to access some resource
- MAC is a concept that evolved out of requirements for military information security
  - *Mandatory* means that an entity that has clearance to access a resource may not, just by its own volition, enable another entity to access that resource
- Both RBAC and ABAC have become increasingly popular
- These four policies are not mutually exclusive
  - An access control mechanism can employ two or more of these policies to cover different classes of system resources

# Index

# The basic elements of access control

## Subject

- An entity (typically a process) capable of accessing objects
- Accountable for the actions they have initiated
- Three classes (typically)
  - Owner
  - Group
  - World

## Object

- A resource to which access is controlled
- E.g. an 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

# Index

# Discretionary Access Control (DAC)

Policies often provided (as e.g. in OS and DBMS) using an access matrix

- One dimension consists of **identified subjects** that may attempt access to the objects
- The other dimension lists the **objects** that may be accessed
- Each entry in the matrix indicates the **access rights** of a specific subject for a specific object

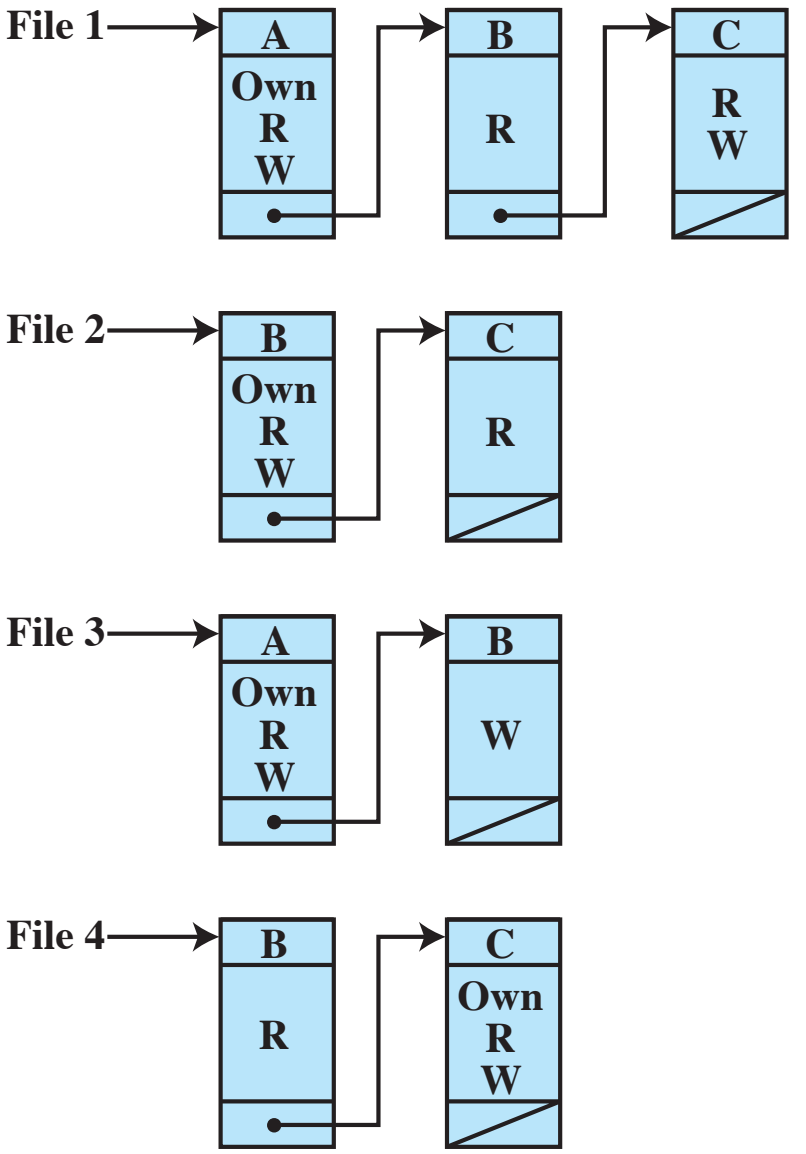
# Access matrix

| | | OBJECTS | | | |
|---|---|---|---|---|---|
| | | File 1 | File 2 | File 3 | File 4 |
| SUBJECTS | User A | Own<br>Read<br>Write | | Own<br>Read<br>Write | |
| | User B | Read | Own<br>Read<br>Write | Write | Read |
| | User C | Read<br>Write | Read | | Own<br>Read<br>Write |

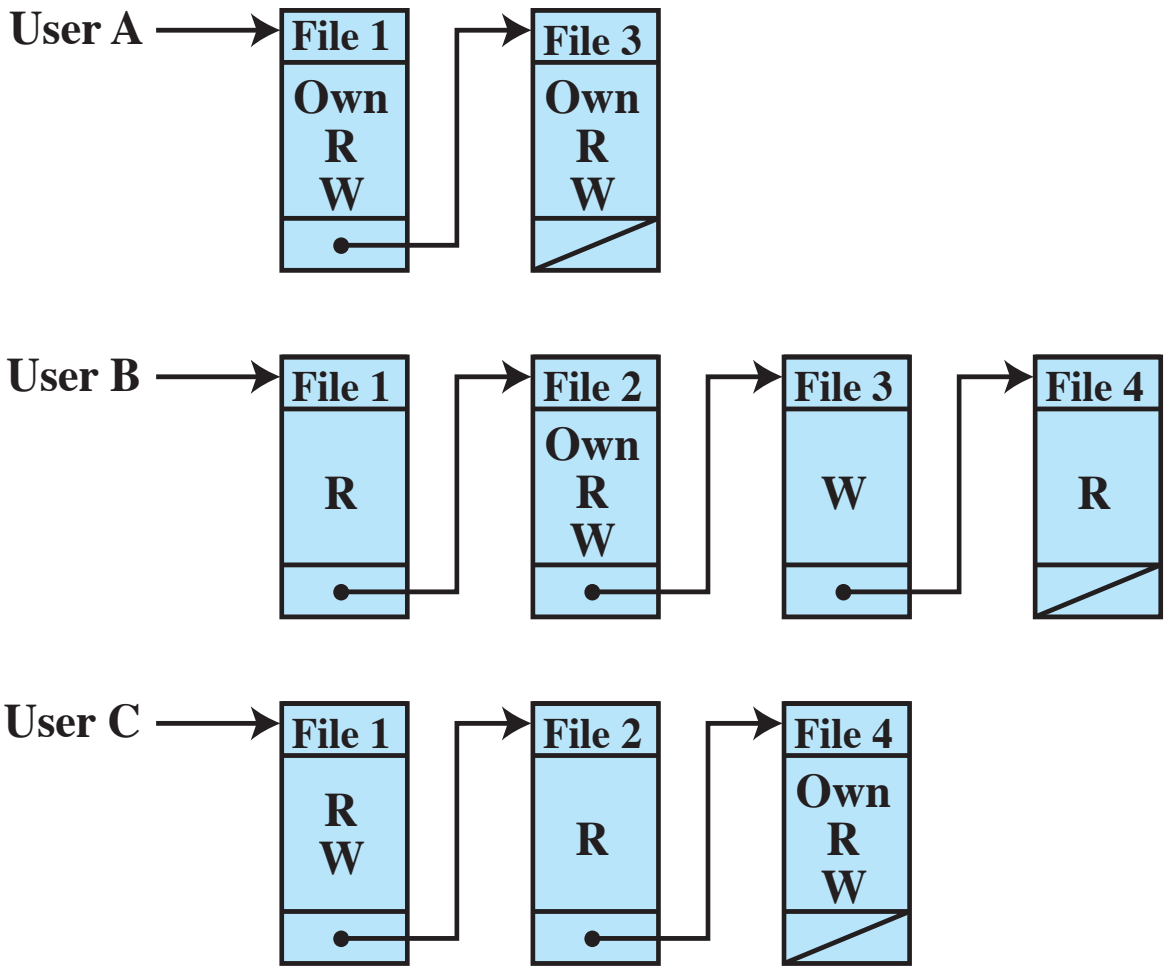
In practice, an access matrix is usually **sparse** and is implemented by decomposition in one of two ways

# Access control lists: matrix decomposed by columns

- For each object, an ACL lists **subjects** and their permitted **access rights**
- May contain a **default** entry allowing subjects that are not explicitly listed to have a default set of rights
  - This set of rights should always follow the rule of *least privilege*
- **Convenient** when it is needed to determine which subjects have which access rights to a particular object
- **Inconvenient** for determining the access rights available to a specific subject



# Capability lists (of tickets): matrix decomposition by rows




- A capability ticket specifies authorized **objects** and **operations** for a particular subject
- Each user has a number of tickets and may be authorized to loan or give them to others
- A ticket must be **unforgeable**, e.g. it includes a password or a cryptographic MAC verified by the relevant resource whenever access is requested
- **Vantages** and **disadvantages** of capability lists are the opposite of those for ACLs

# Authorization Table

- A data structure that is **not sparse**, like the access matrix, but is **more convenient** than ACLs and capability tikets
- Contains one row for one access right of one subject to one object
- Sorting or accessing the table **by object** is equivalent to an ACL
- Sorting or accessing the table **by subject** is equivalent to a capability list
- Can be easily **implemented** by a relational database

| Subject | Access Mode | Object |
|---|---|---|
| A | Own | File 1 |
| A | Read | File 1 |
| A | Write | File 1 |
| A | Own | File 3 |
| A | Read | File 3 |
| A | Write | File 3 |
| B | Read | File 1 |
| B | Own | File 2 |
| B | Read | File 2 |
| B | Write | File 2 |
| B | Write | File 3 |
| B | Read | File 4 |
| C | Read | File 1 |
| C | Write | File 1 |
| C | Read | File 2 |
| C | Own | File 4 |
| C | Read | File 4 |
| C | Write | File 4 |

# A general model for DAC

- Developed by Lampson, Graham, and Denning, *provides* a general, logical description of a DAC system
- The model *assumes* a set of subjects, a set of objects, and a set of rules that govern the access of subjects to objects
- Let us define the **protection state** of a system to be the set of information, at a given point in time, that specifies the access rights for each subject with respect to each object
- We can identify three **aspects** to model:
  - representing the protection state
  - enforcing access rights
  - allowing subjects to alter the protection state in certain ways
- To represent the protection state, the universe of objects in the access control matrix is extended to include **Processes, Devices, Memory locations or regions, Subjects**

# An Example of Extended Access Control Matrix

| | | OBJECTS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | subjects | | | files | | processes | | disk drives | |
| | | $S_1$ | $S_2$ | $S_3$ | $F_1$ | $F_2$ | $P_1$ | $P_2$ | $D_1$ | $D_2$ |
| SUBJECTS | $S_1$ | control | owner | owner control | read * | read owner | wakeup | wakeup | seek | owner |
| | $S_2$ | | control | | write * | execute | | | owner | seek * |
| | $S_3$ | | | control | | write | stop | | | |

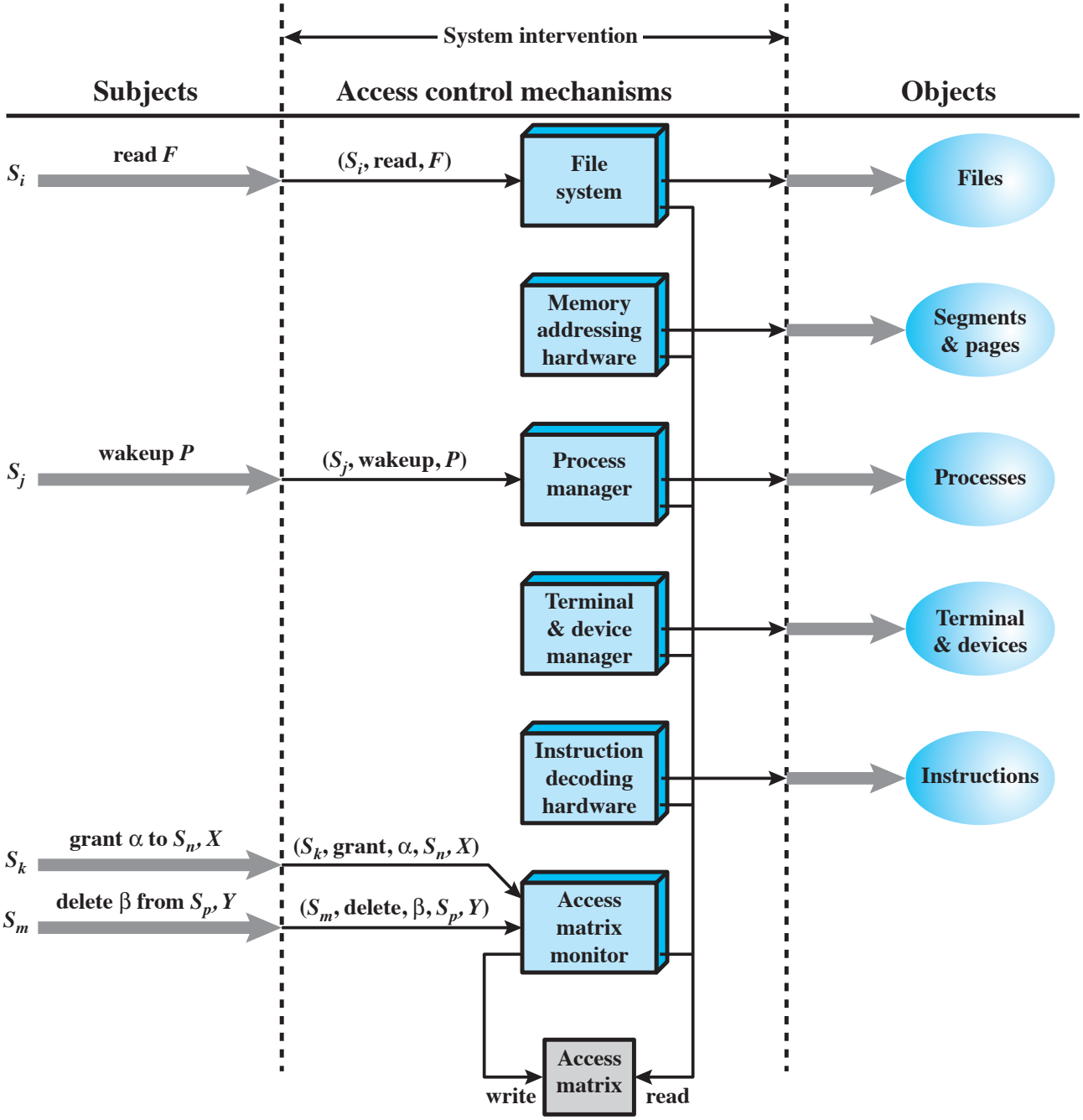
* - copy flag set

N.B. Attached to each attribute is a bit called the *copy flag*

- The presence of the copy flag (*) means that this right can be transferred (with or without copy flag) to another subject

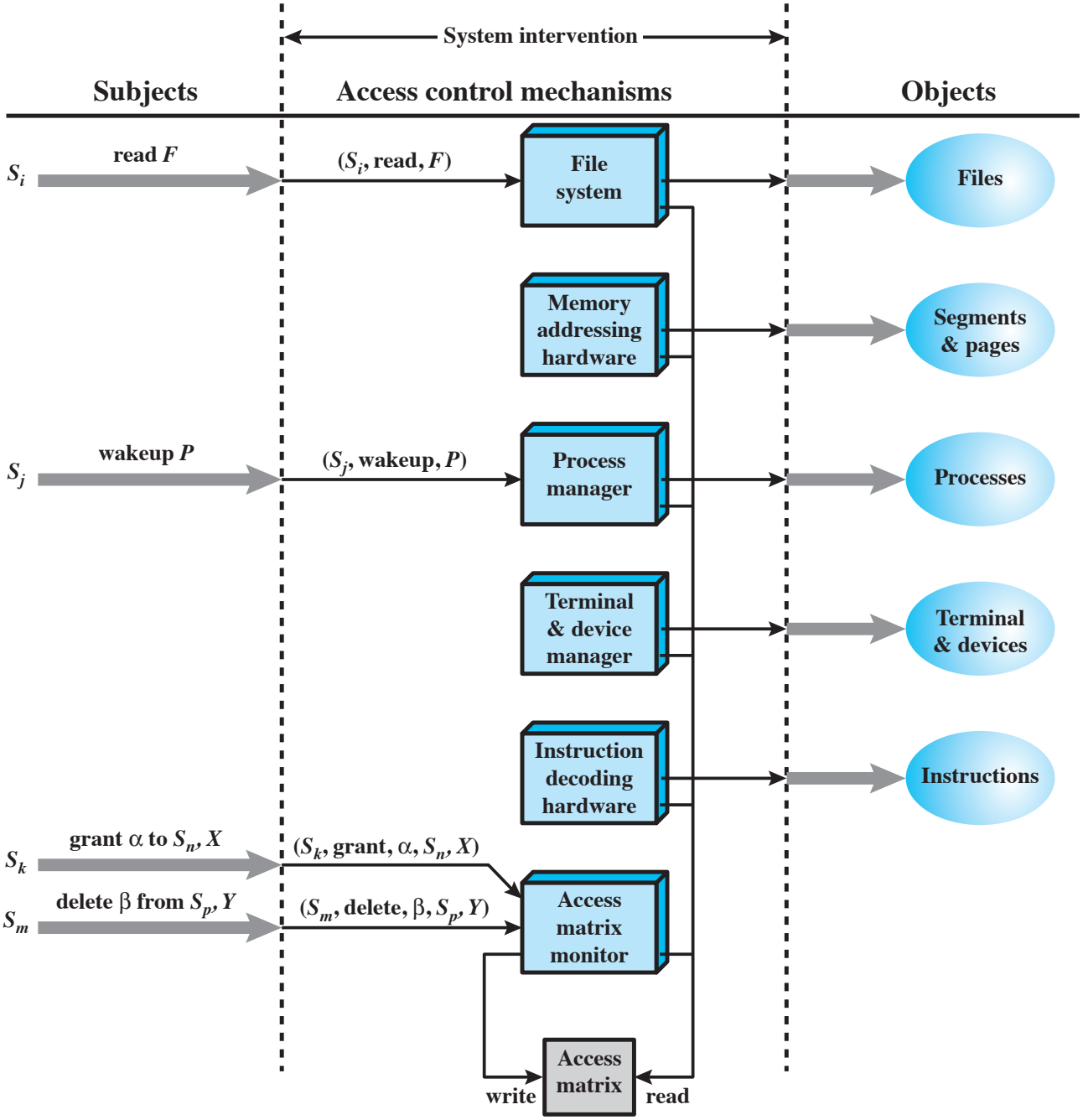
# An Organization of the Access Control Function

- Every access by a subject to an object is mediated by the **controller** for that object
  - The controller's decision is based on the current contents of the matrix



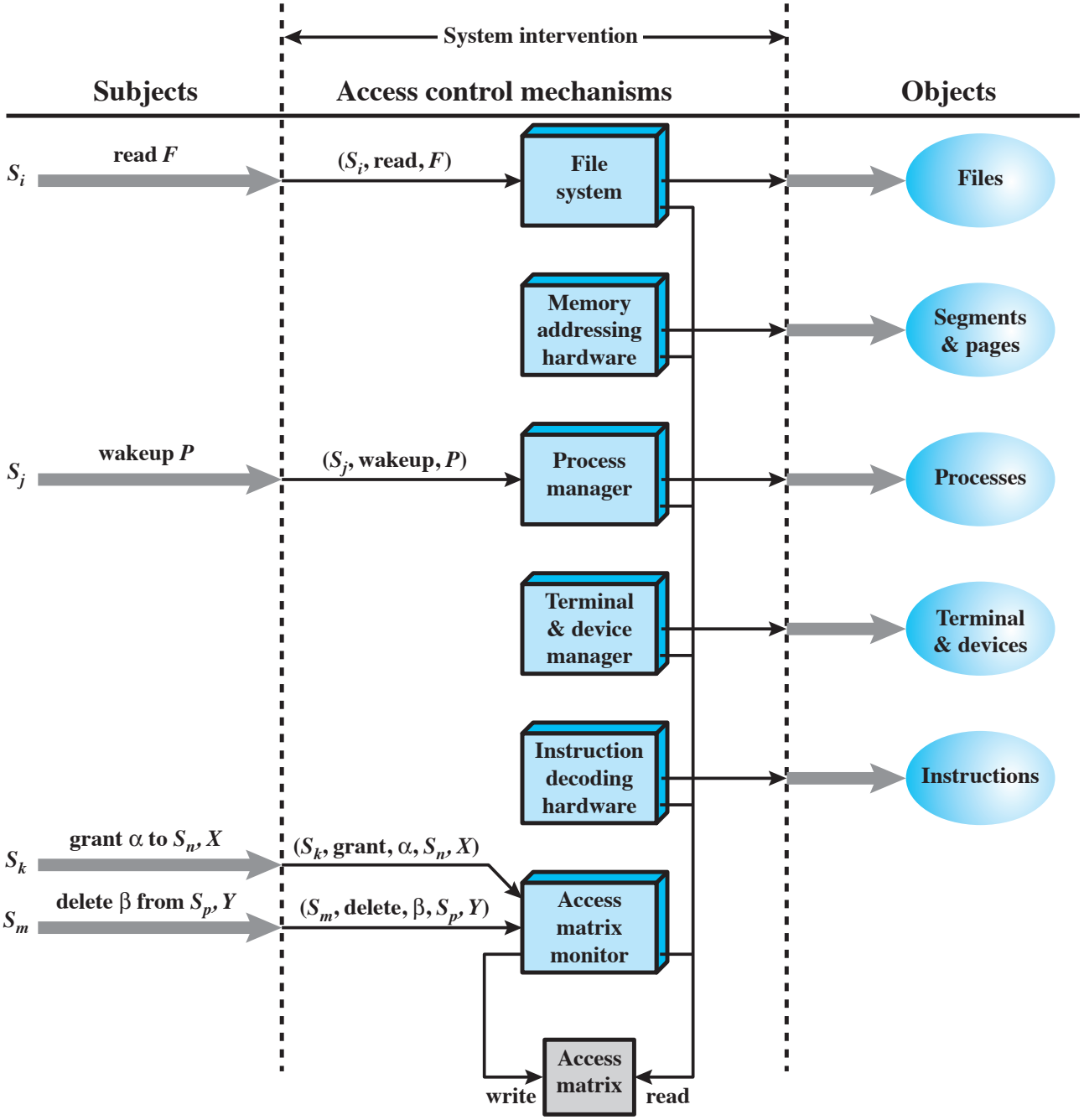
# An Organization of the Access Control Function

- An access attempt triggers the following steps:
  - A subject S issues a request of type α for object X
  - The request causes the system (the OS or an access control interface module of some sort) to generate a message of the form (S, α, X) to the controller for X
  - The controller interrogates the access matrix A to determine if α is in A[S,X]. If so, the access is allowed; if not, the access is denied and a *protection violation* occurs. The violation should trigger a warning and appropriate action



# An Organization of the Access Control Function

- Certain subjects have the authority to make specific **changes to the access matrix**
  - A **request to modify the matrix** is treated as an access to the matrix, with the individual entries in the matrix treated as objects
  - Such accesses are mediated by an **access matrix controller**, which controls updates to the matrix on the base of specific rules
- Hence, the model also includes a set of rules that govern **changes to the access matrix**



# An example of rules governing changes to the access matrix

- We introduce the access rights '*owner*' and '*control*' and the concept of a *copy flag*
- The first three rules deal with **transferring**, **granting**, and **deleting** access rights
- The fourth permits a subject to **read** that portion of the matrix that it owns or controls
- The remaining rules govern the **creation** and **deletion** of **subjects** and **objects**

| Rule | Command (by $S_0$) | Authorization | Operation |
|---|---|---|---|
| R1 | **transfer** $\left\{\begin{matrix}\alpha* \\ \alpha\end{matrix}\right\}$ **to** $S, X$ | 'α*' in $A[S_0, X]$ | store $\left\{\begin{matrix}\alpha* \\ \alpha\end{matrix}\right\}$ in $A[S, X]$ |
| R2 | **grant** $\left\{\begin{matrix}\alpha* \\ \alpha\end{matrix}\right\}$ **to** $S, X$ | 'owner' in $A[S_0, X]$ | store $\left\{\begin{matrix}\alpha* \\ \alpha\end{matrix}\right\}$ in $A[S, X]$ |
| R3 | **delete** α **from** $S, X$ | 'control' in $A[S_0, S]$<br>or<br>'owner' in $A[S_0, X]$ | delete α from $A[S, X]$ |
| R4 | $w \leftarrow$ **read** $S, X$ | 'control' in $A[S_0, S]$<br>or<br>'owner' in $A[S_0, X]$ | copy $A[S, X]$ into $w$ |
| R5 | **create object** $X$ | none | add column for $X$ to $A$;<br>store 'owner' in $A[S_o, X]$ |
| R6 | **destroy object** $X$ | 'owner' in $A[S_0, X]$ | delete column for $X$ from $A$ |
| R7 | **create subject** $S$ | none | add row for $S$ to $A$;<br>execute **create object** $S$;<br>store 'control' in $A[S, S]$ |
| R8 | **destroy subject** $S$ | 'owner' in $A[S_0, S]$ | delete row for $S$ from $A$;<br>execute **destroy object** $S$ |

# An Example of Extended Access Control Matrix

| | | OBJECTS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | subjects | | | files | | processes | | disk drives | |
| | | $S_1$ | $S_2$ | $S_3$ | $F_1$ | $F_2$ | $P_1$ | $P_2$ | $D_1$ | $D_2$ |
| SUBJECTS | $S_1$ | control | owner | owner control | read * | read owner | wakeup | wakeup | seek | owner |
| | $S_2$ | | control | | write * | execute | | | owner | seek * |
| | $S_3$ | | | control | | write | stop | | | |

- **Assumptions**:
  - A subject may hold 'owner' access to any object, but 'control' access only to subjects
  - Each subject is owned or controlled by at most one other subject, though other multiply-owned non-subject objects are allowable
- **Observations**:
  - If a subject has 'owner' access to another subject, the former can grant itself 'control' access to the latter, so that 'control' is implied by 'owner'
  - It is undesirable for a subject to be 'owner' of itself, for then (by Rule R3) it can delete other subjects' access to itself
  - Instead, it is required that 'control' be in *A*[S,S] for every subject *S* (which enables use of rules R3 and R4)

# An Example of Extended Access Control Matrix

| | | OBJECTS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | subjects | | | files | | processes | | disk drives | |
| | | $S_1$ | $S_2$ | $S_3$ | $F_1$ | $F_2$ | $P_1$ | $P_2$ | $D_1$ | $D_2$ |
| SUBJECTS | $S_1$ | control | owner | owner control | read * | read owner | wakeup | wakeup | seek | owner |
| | $S_2$ | | control | | write * | execute | | | owner | seek * |
| | $S_3$ | | | control | | write | stop | | | |

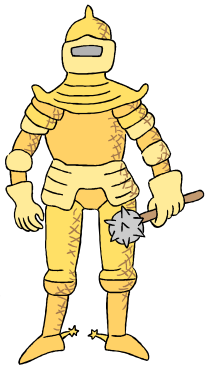
- **Assumption**:
  - The system enforces the requirement that each subject be owned by at most one other subject, i.e., an owner may not grant 'owner,' and 'owner' is either untransferable or is transfer-only
- **Observations:**
  - In this case the relation 'owner' defines naturally a tree hierarchy of subjects
    - $S_1$ owns $S_2$ and $S_3$, so that $S_2$ and $S_3$ are subordinate to $S_1$
    - By the rules previously seen, $S_1$ can grant and delete to $S_2$ access rights that $S_1$ already has

# An Example of Extended Access Control Matrix

| | | OBJECTS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | subjects | | | files | | processes | | disk drives | |
| | | $S_1$ | $S_2$ | $S_3$ | $F_1$ | $F_2$ | $P_1$ | $P_2$ | $D_1$ | $D_2$ |
| SUBJECTS | $S_1$ | control | owner | owner control | read * | read owner | wakeup | wakeup | seek | owner |
| | $S_2$ | | control | | write * | execute | | | owner | seek * |
| | $S_3$ | | | control | | write | stop | | | |

- It is possible to define a hierarchy of subjects based on the 'owner' access right
  - By rule R1, a subject *can transfer only a subset* of its access attributes to a subordinate
  - By rule R2, it *can grant any access* for a subordinate to any other subject
  - By rule R3, it *can delete any access* attribute from a subordinate
  - By rule R4, it *can discover what access* attributes a subordinate has
- Thus, a subject can create another subject with a subset of its own access rights
  - This might be useful, for example, if a subject is invoking an application that is not fully trusted and does not want that application to be able to transfer access rights to other subjects

# Protection Domains



- A more general and more flexible approach is to associate capabilities with protection domains, rather than with users
- A protection domain is a set of objects together with access rights to those objects
  - In terms of the access matrix, a row defines a protection domain rather than a single user
- Users can have associated more protection domains thus, e.g., can spawn processes with different access rights
  - Association between processes and domains can be *static* or *dynamic*
  - The use of protection domains provides a simple means to *minimize* the capabilities that any user or process has at any time
- One form of protection domain has to do with the distinction made in many OSs, such as UNIX, between **user mode** and **kernel mode**
  - In user mode certain areas of memory are protected from use and certain instructions may not be executed
  - In kernel mode privileged instructions may be executed and protected areas of memory may be accessed

# Index

# UNIX File Access Control

## Files are administered using inodes (index nodes)

- An **inode** is a control structure containing key information for a specific file needed by the OS
- Several file names may be associated with a single inode
- An active inode is associated with exactly one file
- File attributes, permissions and control information are stored in the inode
- On the disk there is an inode table, or inode list, that contains the inodes of all the files in the file system
- When a file is opened, its inode is brought into main memory and stored in a memory resident inode table

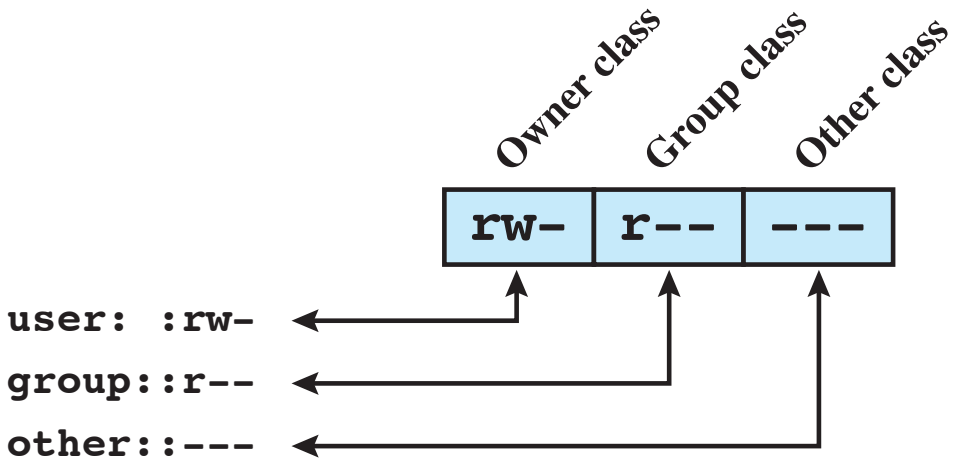
## Directories are structured in a hierarchical tree

- May contain files and/or other directories
- A directory is simply a file (associated with an inode) that contains file names plus pointers to the associated inodes

# Traditional UNIX File Access Control

Each UNIX user

- is assigned a unique user identification number (user ID)
- is member of a *primary group*, and possibly a number of other groups, each identified by a group ID




When a file is created

- is designated as owned by a specific user and marked with that user's ID
- belongs to a specific group
- is assigned 12 protection bits
  - 9 of these bits specify read, write, and execute permission for the owner of the file, members of the group and all other users
- owner ID, group ID, and protection bits are part of the file's inode

When applied to a directory

- the read and write bits grant the right to list and to create/rename/delete files in the directory
- the execute bit grants the right to descend into the directory or search it for a filename

# Traditional UNIX File Access Control

- The remaining three bits define special additional behavior for files or directories
  - **SetUID** (“Set user ID”) & **SetGID** (“Set group ID”)
    - If these are set on an executable file,
      - While a user (with execute privilege for this file) executes the file, the OS temporarily uses rights of the file owner/group *in addition to* the real user’s rights when making access control decisions
      - This mechanism enables the creation and use of privileged programs that may use files normally inaccessible to other users
    - If these are set on a directory,
      - the SetUID permission is ignored
      - the SetGID permission indicates that newly created files will inherit the group of this directory
  - **Sticky** bit
    - When set *on a file*, it is ignored (originally it indicated that the system should retain the file contents, i.e. the program text, in memory following execution)
    - When set *on a directory*, the filesystem treats the files in such directories in a special way so only the file's owner, the directory's owner, or root user can rename or delete the file
      - Without the sticky bit set, any user with write and execute permissions for the directory can rename or delete contained files, regardless of the file's owner
      - Typically this is set on the `/tmp` directory to prevent ordinary users from deleting or moving other users' files

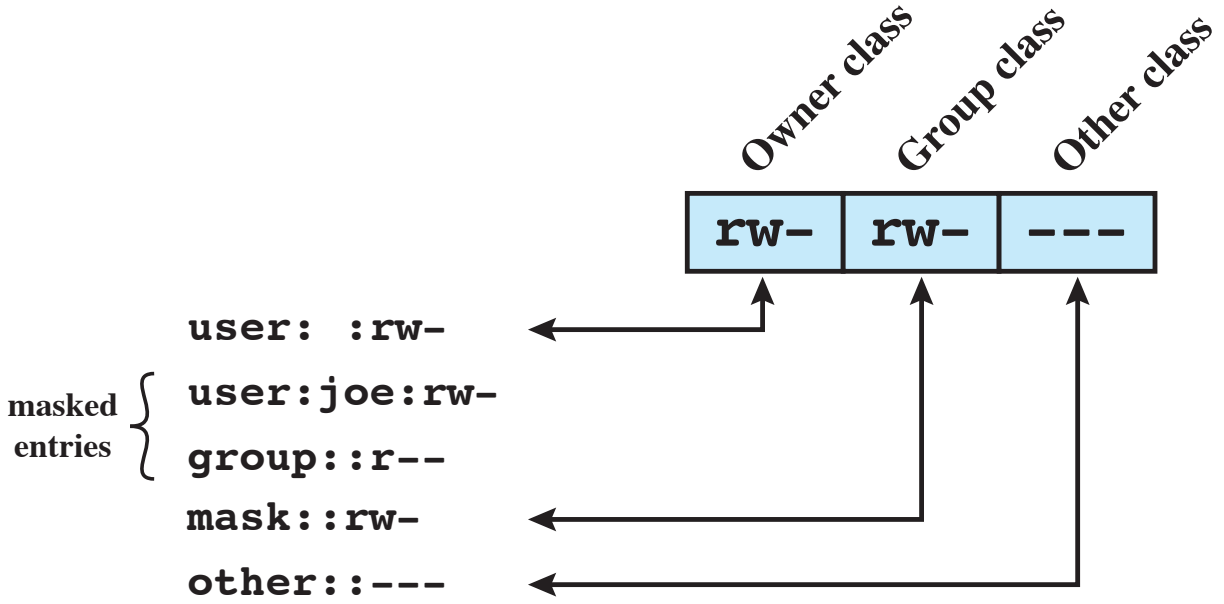
# Traditional UNIX File Access Control

- One particular user ID is designated as **superuser**
  - Has system-wide access, is exempt from usual access control restrictions
  - Any program that is owned by, or SetUID to, the *superuser* potentially grants unrestricted access to the system to any user executing that program
- Each user is associted with a **single protection domain** (switching the domain corresponds to changing the user ID temporarily)

# Access Control Lists (ACLs) in UNIX

- Modern UNIX systems, e.g. FreeBSD, OpenBSD, Linux, support ACLs
  - It is referred to as extended ACL, while the traditional UNIX approach is referred to as *minimal* ACL
- These scheme permits to manage a large number of groups which may be needed to, e.g., provide many different groupings of users requiring a range of access rights to different files
- **FreeBSD** allows the administrator to assign a list of UNIX user IDs and groups to a file by using the `setfacl` command
  - Any number of users and groups can be associated with a file, each with three protection bits (read, write, execute)
  - A file does not need to have an ACL (but may be protected solely by the traditional UNIX file access mechanism)
  - Includes an additional protection bit that indicates whether the file has an extended ACL
- Steps performed when a process requests access to a file system object:
  - Step 1: selects the most appropriate ACL entry (looked at in the following order: owner, named users, (owning or named) groups, others)
  - Step 2: checks if the matching entry contains sufficient permissions

# Extended Access Control List




- The owner class and other class entries in the 9-bit permission field have the same meaning as in the minimal ACL case
- The group class entry specifies the permissions for the owner group for this file
  - These permissions represent the maximum permissions that can be assigned to named users or named groups, other than the owning user
  - In this latter role, the group class entry functions as a mask
- Additional named users/groups may be associated with the file, each with a 3-bit permission field
  - The permissions listed for a named user/group are compared to the mask field
  - Any permission for the named user/group that is not present in the mask field is disallowed

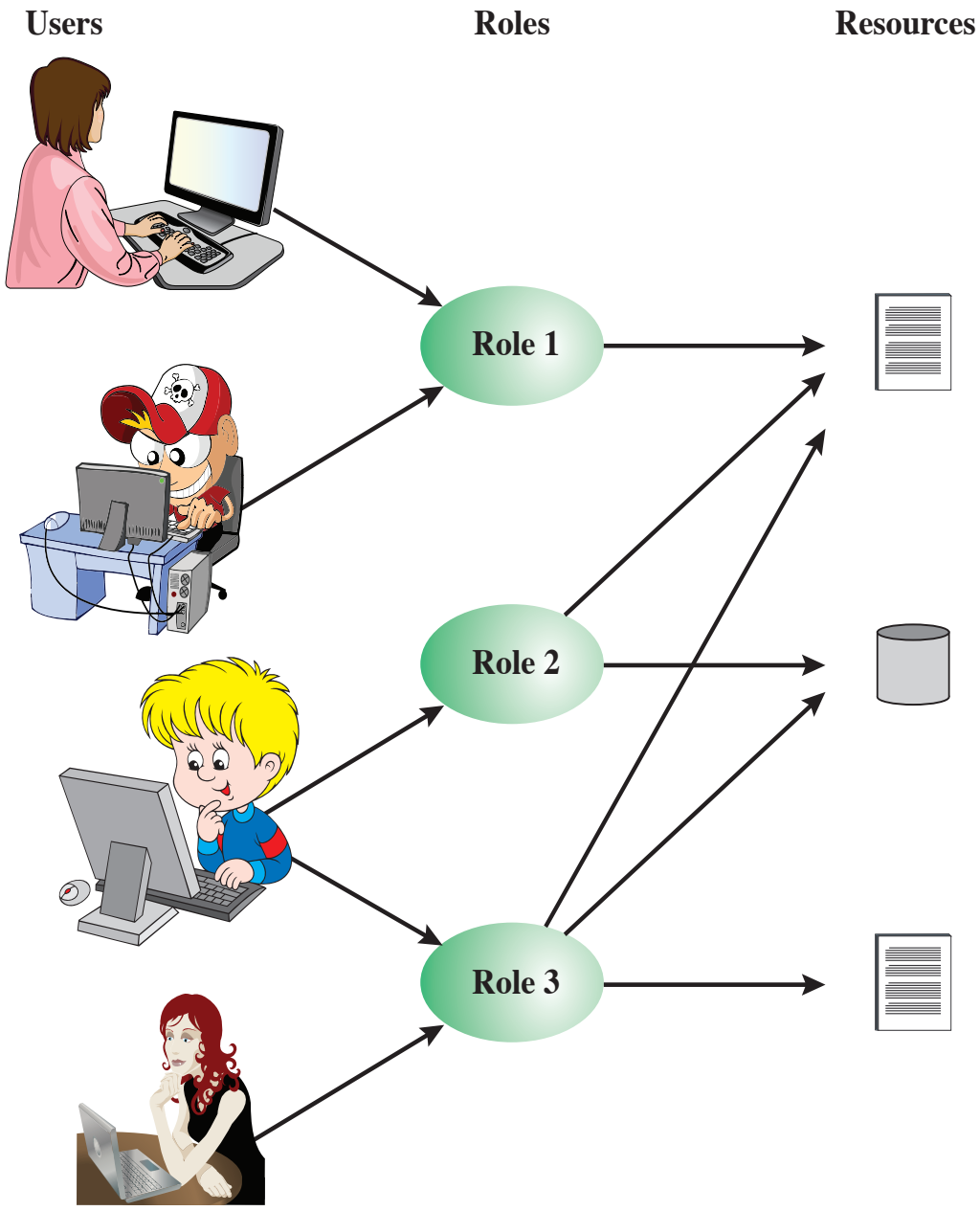
# Index

# Role-Based Access Control

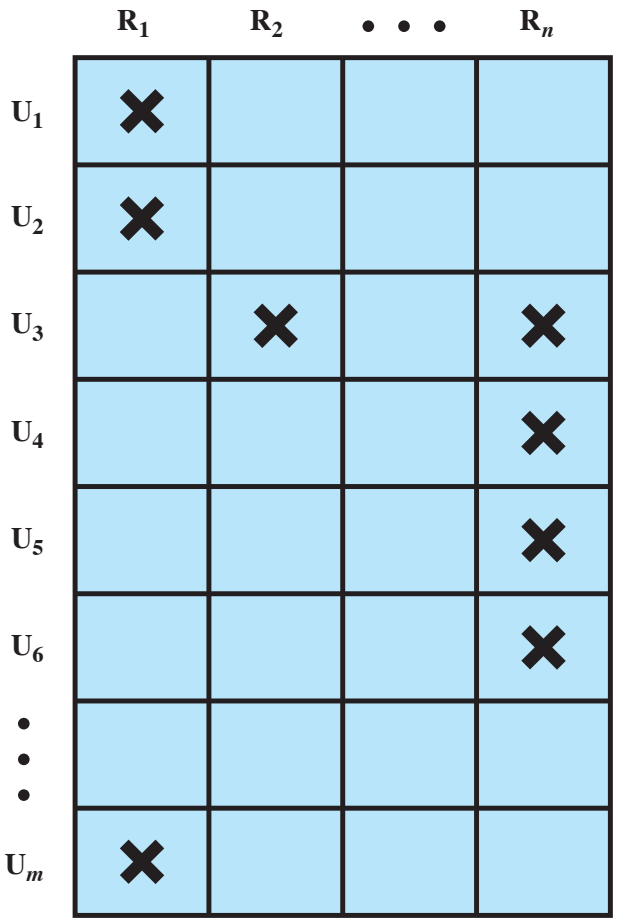
- RBAC is based on the roles that users assume in a system rather than the user's identity
  - Typically, roles correspond to job functions within an organization
  - Access rights are assigned to roles **instead of** individual users
  - In turn, users are assigned to different roles, either **statically** or **dynamically**, according to their responsibilities
- RBAC allows to effectively implement the least privilege principle
  - Each role should contain the *minimum set* of access rights needed for that role
  - A user is assigned to a role that enables him or her *to perform only what is required* for that role
  - Multiple users assigned to the same role *share* the same minimal set of access rights
- RBAC enjoys widespread commercial use and remains an area of active research

# Users, Roles, and Resources

- The relationship of users to roles is **many to many**, as is the relationship of roles to resources, or system objects
- The *set of users* changes, in some environments frequently, and the assignment of a user to one or more roles may also be **dynamic**
- The *set of roles* in the system in most environments is relatively **static**, with only occasional additions or deletions
- Each role will have **specific access rights** to one or more resources
- The *set of resources* and the specific access rights associated with a particular role are also likely to change **infrequently**



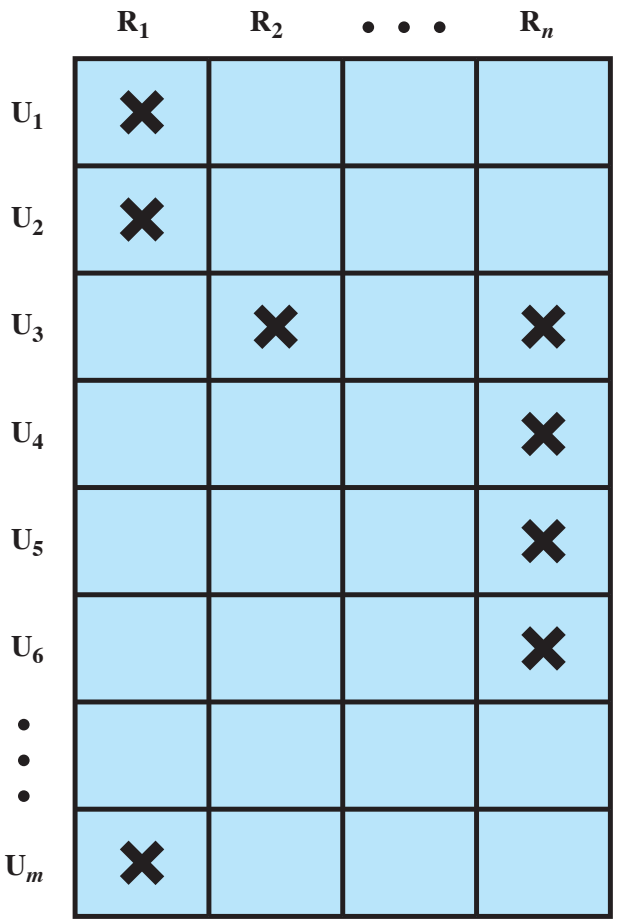
# Access Matrix Representation of RBAC




This matrix *relates individual users to roles*

- Typically there are many more users than roles
- Each matrix entry is either blank or marked, the latter indicating that this user is assigned to this role

# Access Matrix Representation of RBAC




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 | | | |

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

- Typically, there are few roles and many objects, or resources
- The entries are the specific access rights enjoyed by the roles
- A role can be treated as an object, allowing the definition of role hierarchies

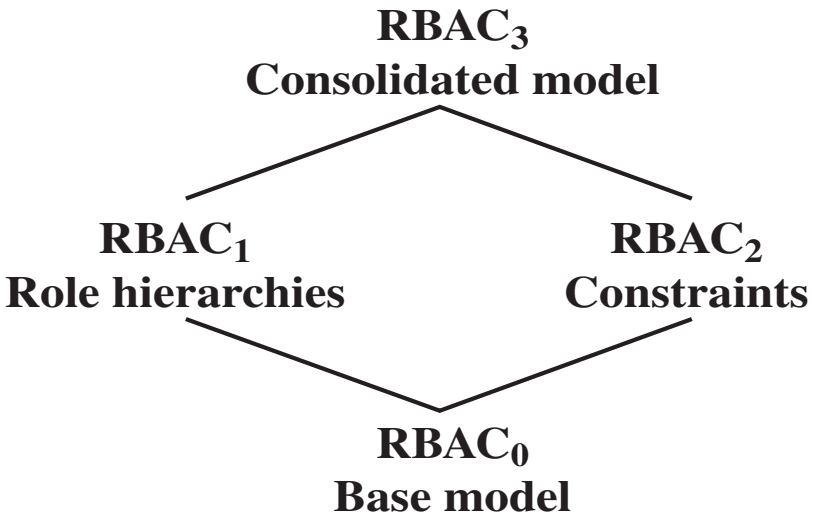
This matrix relates individual users to roles

- Typically there are many more users than roles
- Each matrix entry is either blank or marked, the latter indicating that this user is assigned to this role

# RBAC Reference Models

There is a family of reference models, related to each other, that has served as the basis for ongoing standardization efforts

- $RBAC_0$ 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 $RBAC_2$



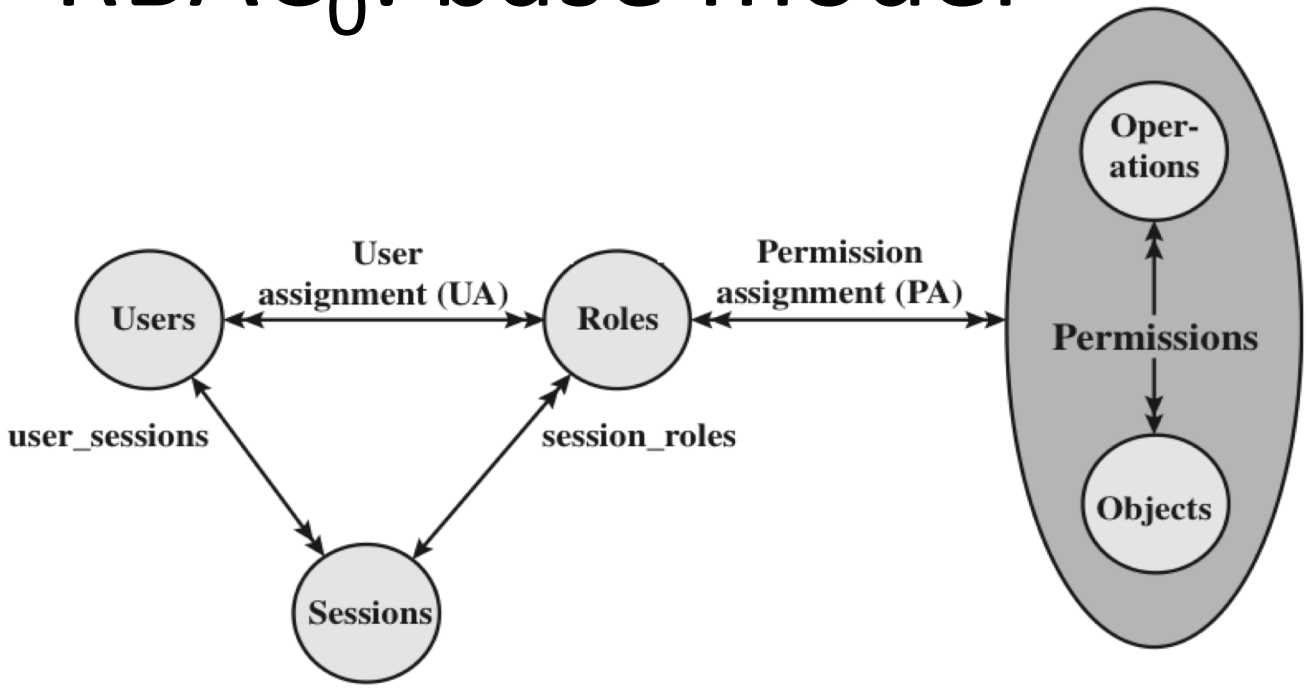

| Models | Hierarchies | Constraints |
|---|---|---|
| $RBAC_0$ | No | No |
| $RBAC_1$ | Yes | No |
| $RBAC_2$ | No | Yes |
| $RBAC_3$ | Yes | Yes |

# $RBAC_0$: base model

An $RBAC_0$ system contains four types of entities:

- User**:** An **individual** that has access to this computer system. Each individual has an associated user ID
- Role**:** A named **job function** within the organization that controls the computer system
  - Typically, associated with each role is a *description of the authority and responsibility* conferred on this role, and on any user who assumes this role
- Permission**:** An **approval** of a particular *mode of access* to one or more objects
  - Equivalent terms are *access right*, *privilege*, and *authorization*
- Session**:** A **mapping** between a user and an activated subset of the set of roles to which the user is assigned

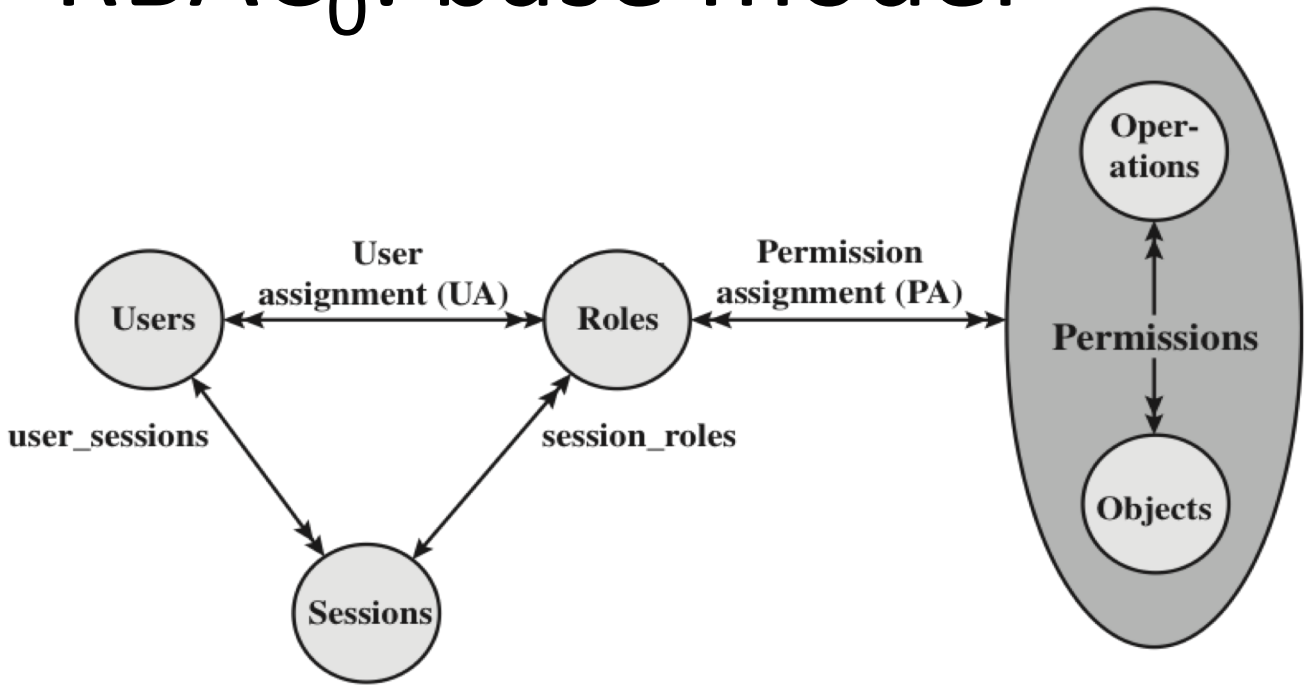
# $RBAC_0$: base model




The arrowed lines indicate **relationships**, or mappings, with a single arrowhead indicating one and a double arrowhead indicating many

- There is a **many-to-many** relationship between users and roles: one user may have multiple roles, and multiple users may be assigned to a single role
- Similarly, there is a **many-to-many** relationship between roles and permissions
- A session is used to define a temporary **one-to-many** relationship between a user and one or more of the roles to which the user has been assigned
  - The user establishes a session with *only the roles needed for a particular task*; this is an example of the concept of least privilege

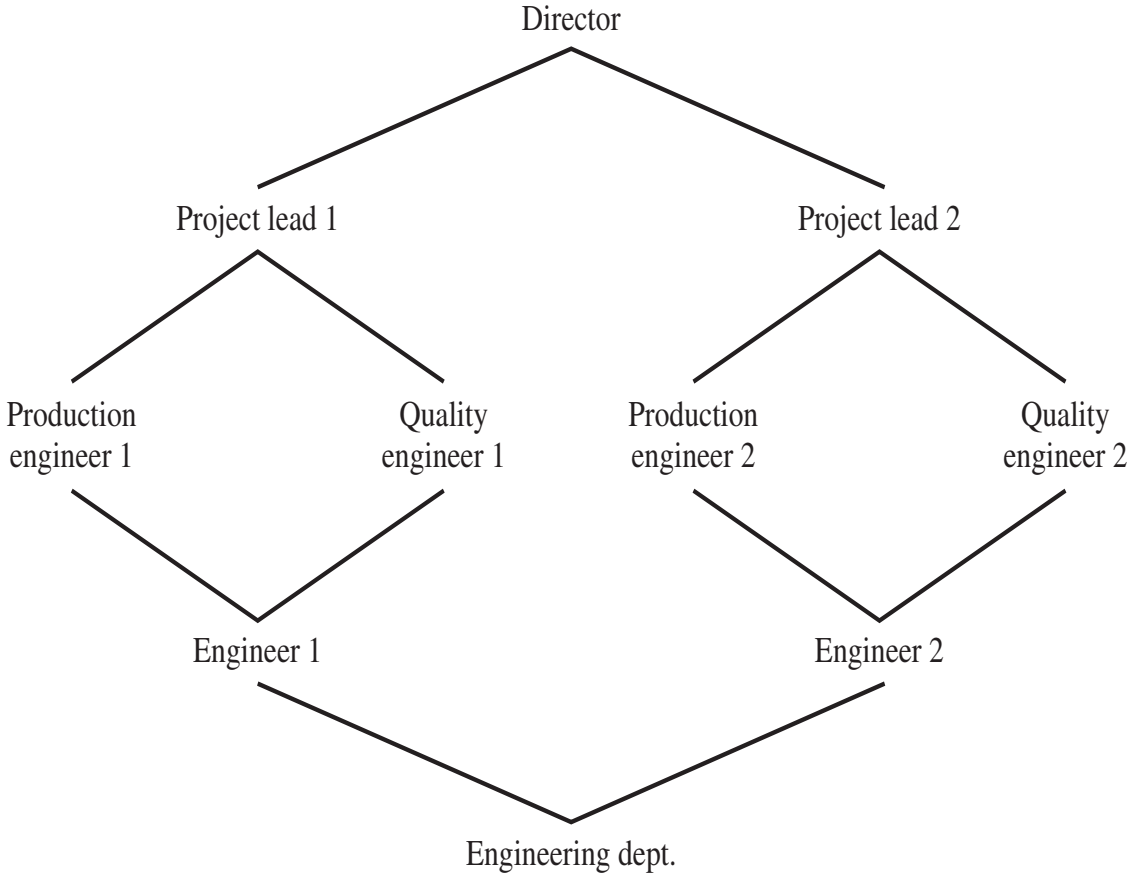
# $RBAC_0$: base model




Wrt DAC schemes, it provides a greater **flexibility** and **granularity** of assignment

- This permit greater control over the types of access that can be allowed thus avoiding to grant more access to resources than is needed

# $RBAC_1$: role hierarchies




- Subordinate roles are lower in the diagram
- A line between two roles implies that *the upper role includes all of the access rights of the lower role*, as well as other access rights not available to the lower role
- One role can inherit access rights from *multiple* subordinate roles

- Role hierarchies provide a means of reflecting the **structure** of roles in an organization
  - Job functions with greater responsibility have greater authority to access resources
  - A subordinate job function may have a subset of the access rights of the superior job function
- Role hierarchies make use of the concept of inheritance to enable one role to implicitly include access rights associated with a subordinate role

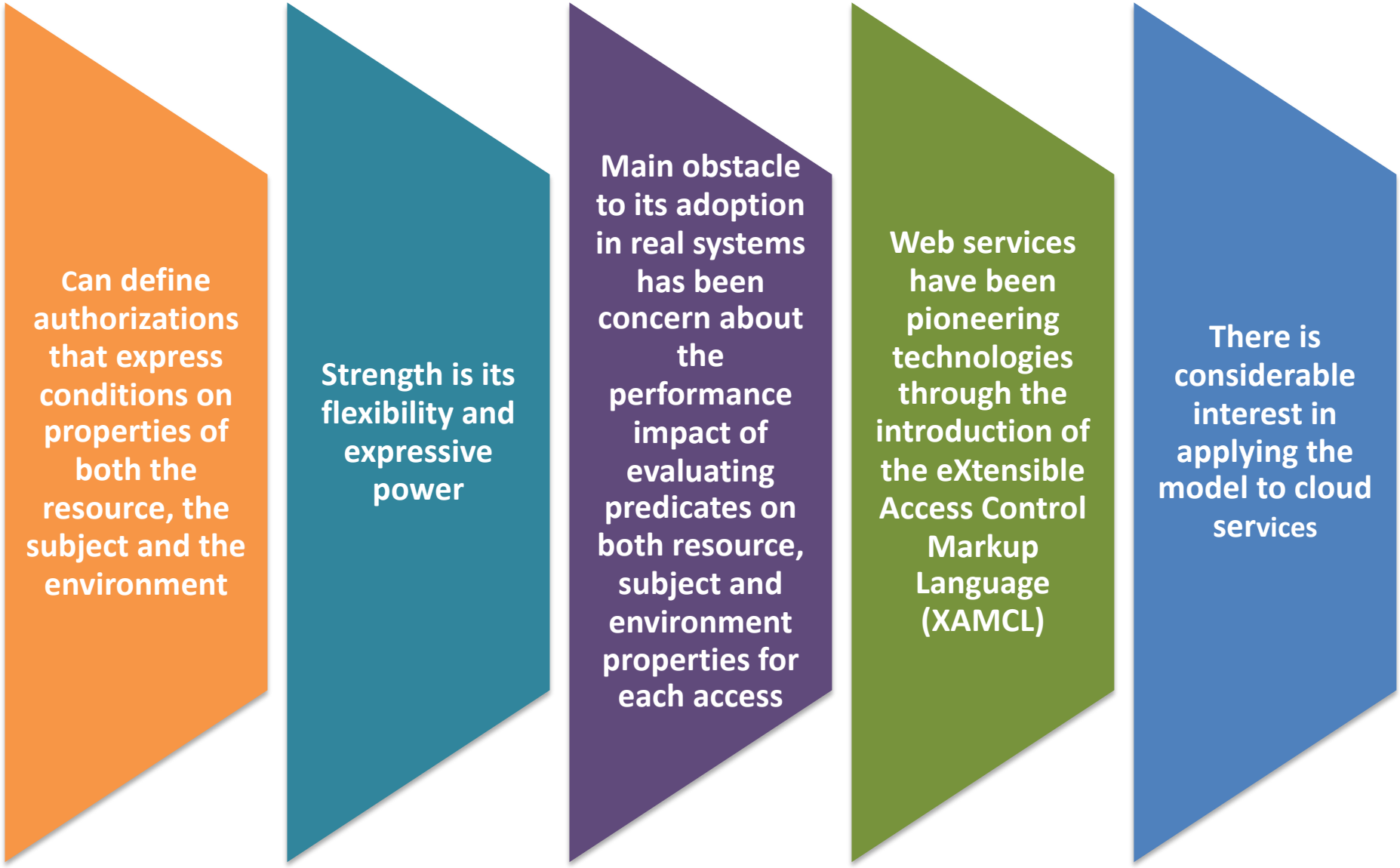
# $RBAC_2$: constraints

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

| Mutually exclusive roles | Cardinality | Prerequisite roles |
|---|---|---|
| • E.g. a user can only be assigned to only one role (either during a session or statically)<br>• E.g. any permission can be granted to only one role<br>• Supports *separation of duties* and *capabilities*<br>• Aims at increasing the difficulty of *collusion* among individuals of different skills or divergent job functions to thwart security policies | • Setting a *maximum* number with respect to roles, e.g.<br>  • users that can be assigned to a given role<br>  • roles that a user is assigned to<br>  • roles a user can activate for a single session<br>  • roles that can be granted a particular permission | • A user can be assigned to a particular role *only if* it is already assigned to some other specified role<br>• E.g. it might be required that a user can be assigned to a senior (higher) role only if it is already assigned an immediately junior (lower) role<br>• The use of prerequisites tied to the concept of hierarchy requires the $RBAC_3$ model |

# Index

# Attribute-Based Access Control (ABAC)



# ABAC Model: key elements

There are three key elements of an ABAC model:

- attributes, which are defined for entities in a configuration
- the architecture model, which applies to policies that enforce access control
- a policy model, which defines the ABAC policies

# Attributes

- Are characteristics that define **specific aspects** of the subject, object, environment conditions, and/or requested operations
- Contain information that indicates
  - the class of information given by the attribute
  - a name
  - a value
- E.g.:

```
Class=HospitalRecordsAccess
Name=PatientInformationAccess
Value=BusinessHoursOnly
```

# Types of attributes

## Subject attributes

- A subject is an **active entity** (e.g., a user, an application, a process, or a device) that causes information to flow among objects or changes the system state
- Attributes define the characteristics of the subject, e.g. identifier, name, organization, job title, role

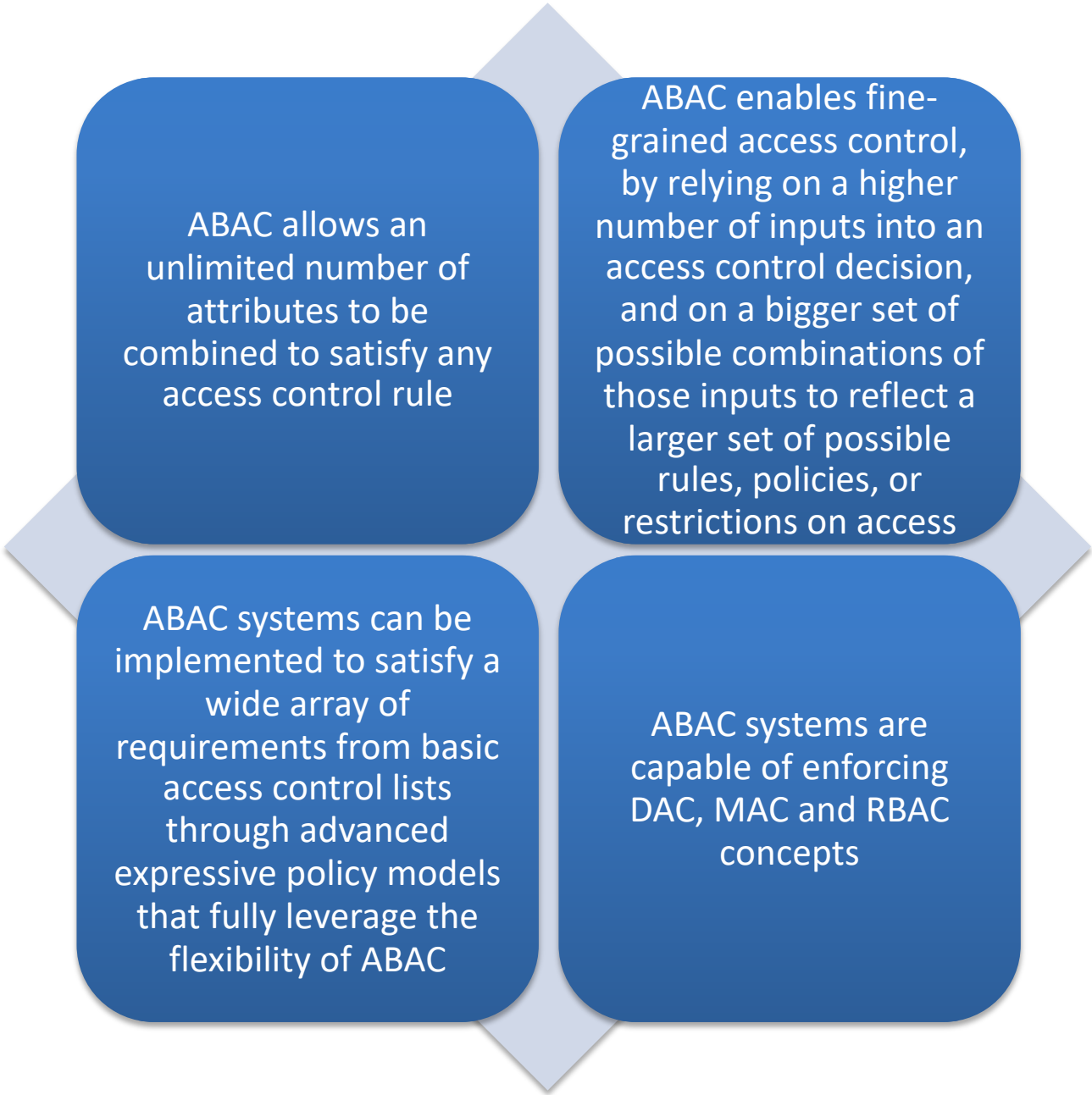
## Object attributes

- An object (or resource) is a **passive entity** (e.g., devices, files, records, tables, processes, programs, networks, domains) containing or receiving information
- Objects have attributes, often extracted from its metadata, that can be leveraged to make access control decisions (e.g. a Microsoft Word document has title, subject, date, and author)

## Environment attributes

- Describe the operational, technical, and even situational **environment** or **context** in which the information access occurs
- E.g. current date and time, current virus/hacker activities, network's security
- These attributes have so far been **largely ignored** in most access control policies

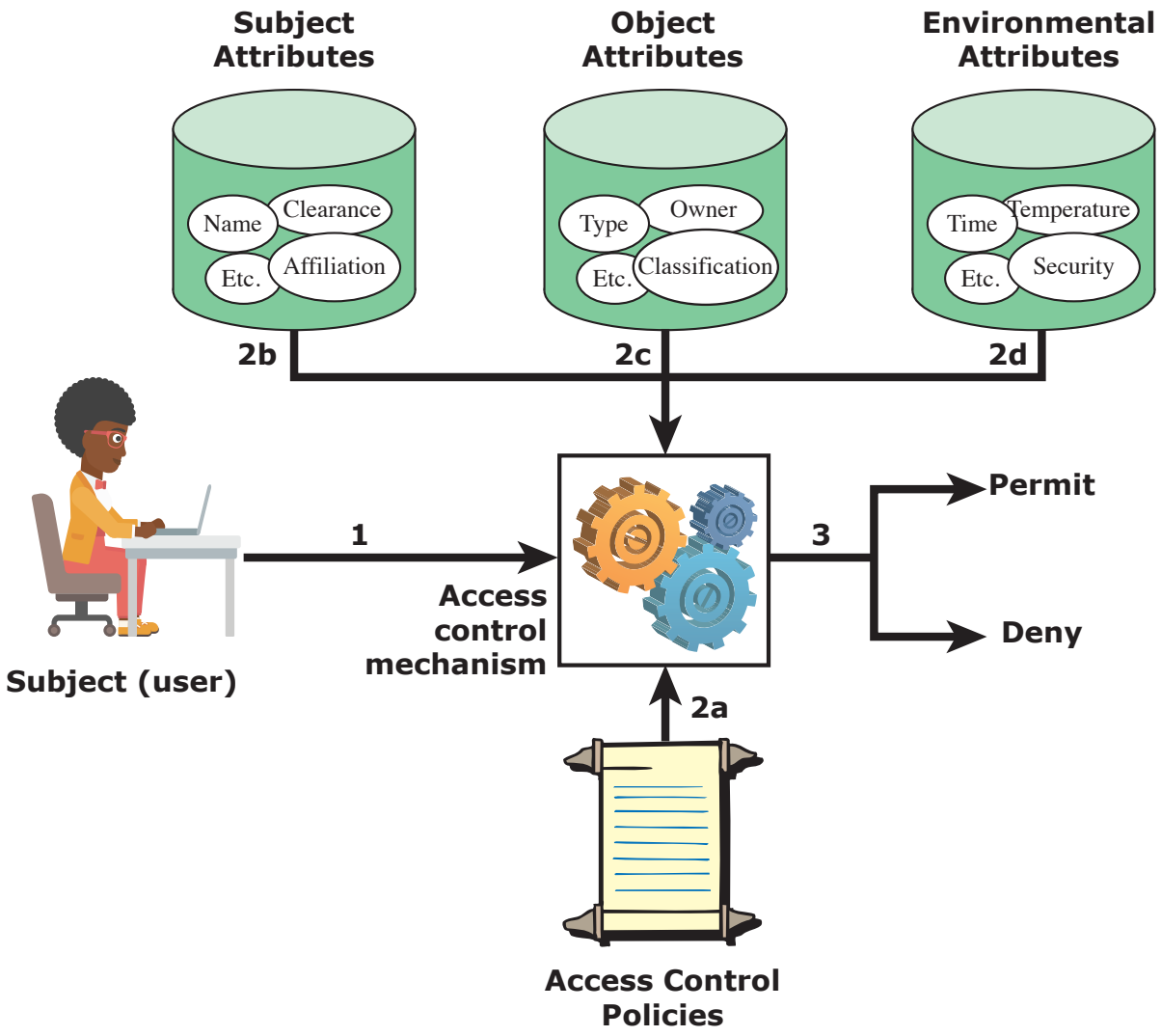
# ABAC features



# ABAC Logical Architecture

Steps for an **access by a subject to an object**:

1. The subject *request* is routed to an access control mechanism
2. The access control mechanism is governed by a *set of rules* (2a) that are defined by a preconfigured access control *policy*. Based on these rules, the access control mechanism assesses the attributes of the subject (2b), object (2c), and current environmental conditions (2d) to determine *authorization*
3. The access control mechanism *grants* the subject access to the object if access is authorized and *denies* access if it is not authorized

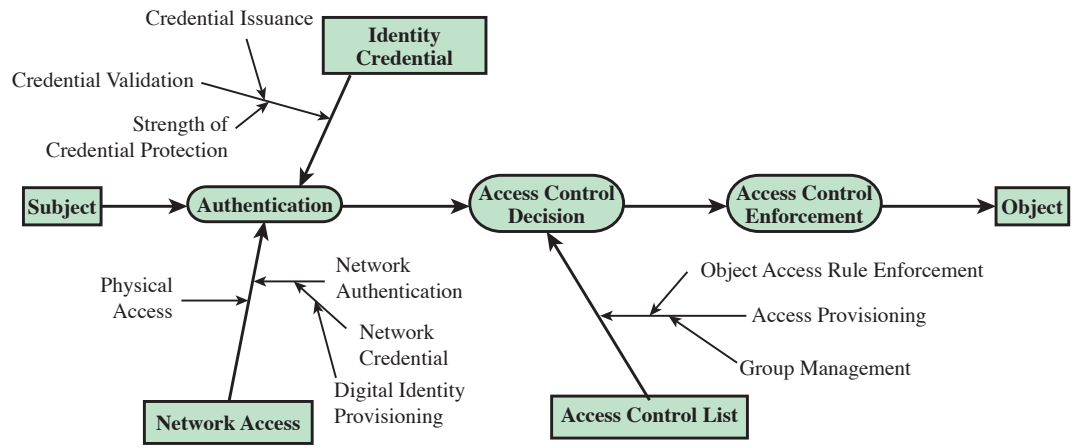


# Considerations

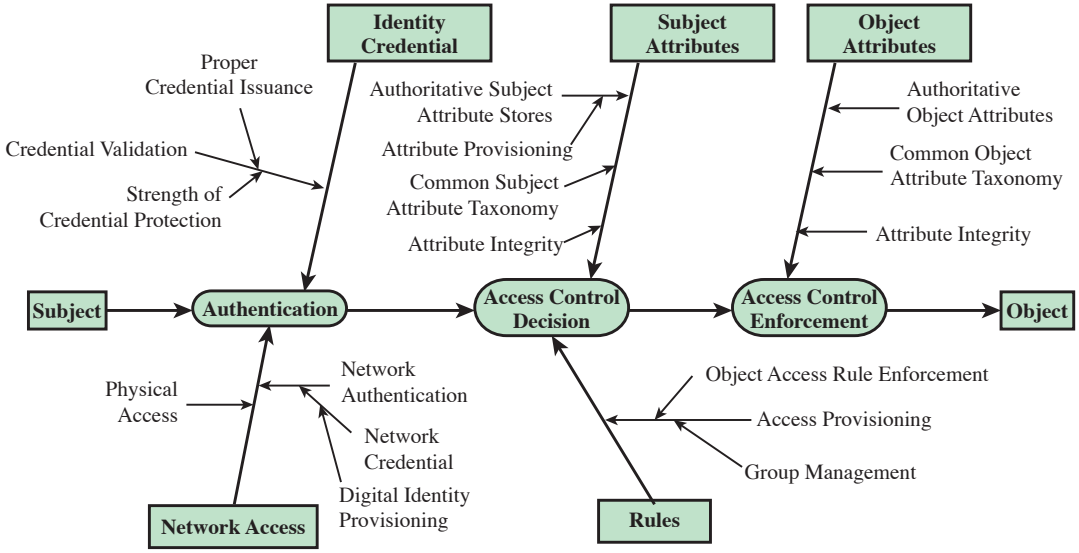
- There are **four independent sources of information** used for the access control decision:
  1. set of rules (i.e. access control policies)
  2. attributes of the subject
  3. attributes of the object
  4. current environmental conditions
- **Tradeoff** that the system authority must make
  - This approach is **very powerful** and **flexible**
  - However, the **cost**, both in terms of the **complexity of the design** and **implementation** and in terms of the **performance** impact, is likely to exceed that of other access control approaches

# ACL Trust Chain

- The figure is a useful way of grasping the scope of an ABAC model compared to a DAC model using access control lists (ACLs)
- This figure not only illustrates the relative complexity of the two models, but also clarifies the **trust requirements** of the two models
- A comparison of representative trust relationships (indicated by arrowed lines) for ACL use and ABAC use shows that there are many more complex trust relationships required for ABAC to work properly

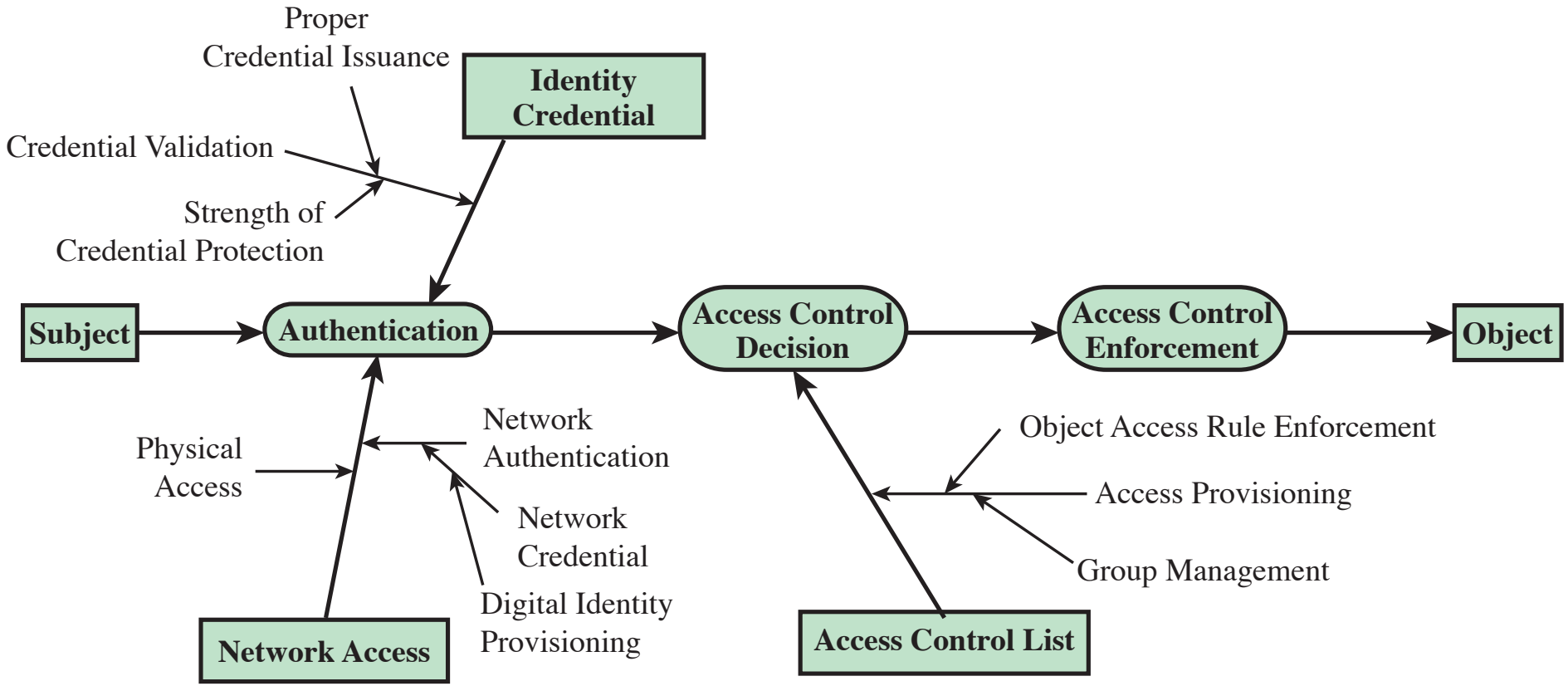



(a) ACL Trust Chain



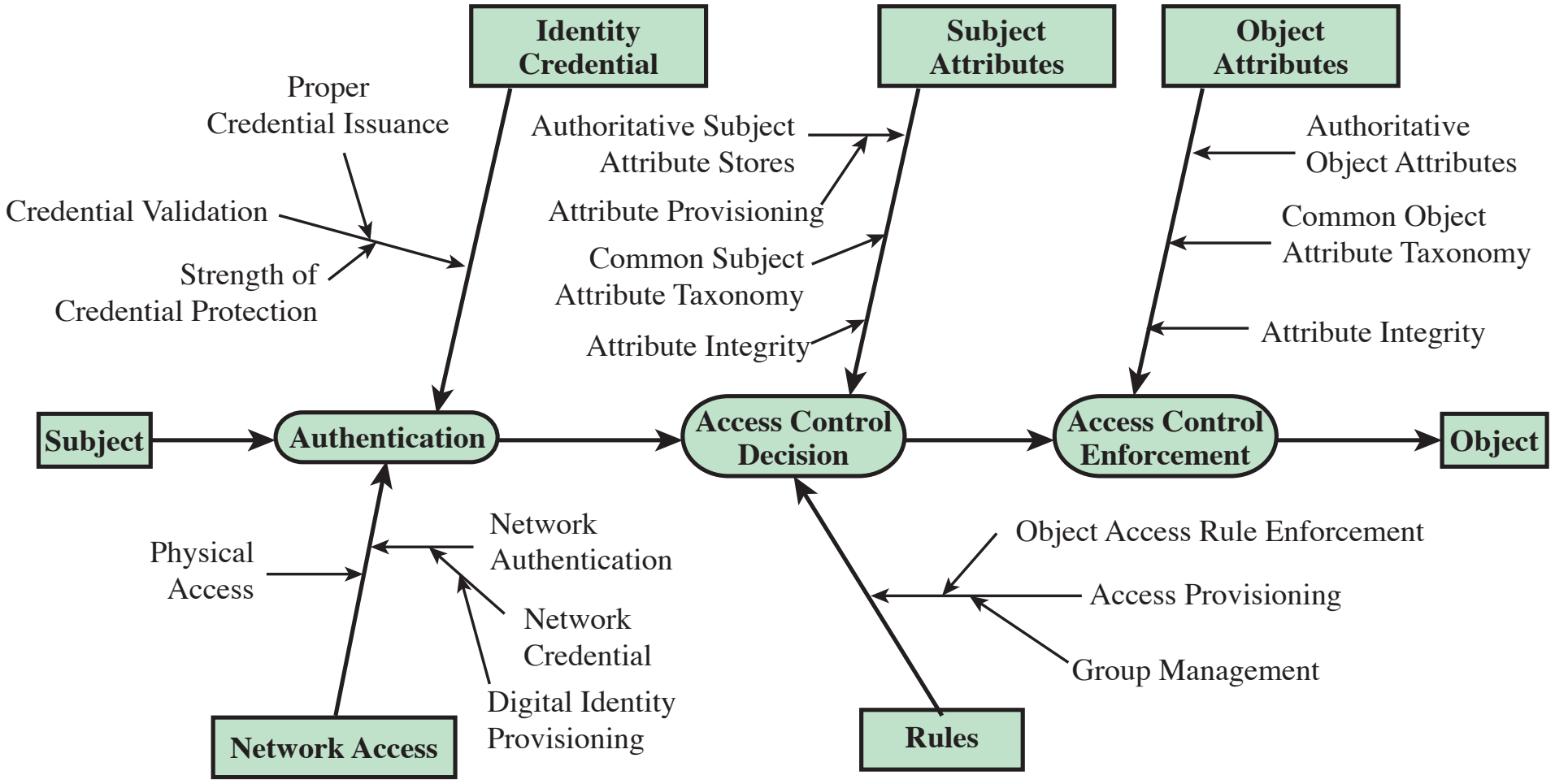

(b) ABAC Trust Chain

# ACL Trust Chain




The **root of trust** is with the object owner, who ultimately enforces the object access rules by provisioning access to the object through addition of a user to an ACL

# ABAC Trust Chain




The **root of trust** is derived also from many sources of which the object owner has no control, such as Subject Attribute Authorities, Policy Developers, and Credential Issuers

# ABAC Policies

A **policy** is a set of rules and relationships that govern allowable behavior within an organization, based on the privileges of subjects and how resources or objects are to be protected under which environment conditions

Typically written from the perspective of the object that needs protection and the privileges available to subjects

**Privileges** represent the authorized behavior of a subject and are defined by an authority and embodied in a policy

Other terms commonly used instead of privileges are: rights, authorizations, and entitlements

# An ABAC policy model

1. S, O, and E are subjects, objects, and environments, respectively
2. $SA_k$ ($1 \le k \le K$), $OA_m$ ($1 \le m \le M$), and $EA_n$ ($1 \le n \le N$) are the predefined attributes for subjects, objects, and environments, respectively
3. `ATTR(s), ATTR(o), and ATTR(e)` are attribute assignment relations for subject *s*, object *o*, and environment *e*, respectively:

$$ATTR(s) \subseteq SA_1 \times SA_2 \times \ldots \times SA_K$$

$$ATTR(o) \subseteq OA_1 \times OA_2 \times \ldots \times OA_M$$

$$ATTR(e) \subseteq EA_1 \times EA_2 \times \ldots \times EA_N$$

   We also use the function notation for the value assignment of individual attributes. E.g.

   Role(*s*) = "Service Consumer" ServiceOwner(*o*) = "XYZ, Inc." CurrentDate(*e*) = "01-23-2005"

4. In the most general form, a Policy Rule, which decides on whether a subject *s* can access an object *o* in a particular environment *e*, is a Boolean function of the attributes of *s*, *o*, and *e*:

$$Rule: can\_access\ (s, o, e) \leftarrow f(ATTR(s), ATTR(o), ATTR(e))$$

   Given all the attribute assignments of *s*, *o*, and *e*, if the function's evaluation is true, then the access to the resource is granted; otherwise the access is denied

5. A policy rule base or policy store may consist of a number of policy rules, covering many subjects and objects within a security domain
   - The access control decision process in essence amounts to the evaluation of applicable policy rules in the policy store

# An online entertainment store: RBAC

- Streams movies to users for a flat monthly fee
- Enforces the following AC policy based on the user's age and the movie's content rating

| Movie Rating | Users Allowed Access |
|---|---|
| R | Age 17 and older |
| PG-13 | Age 13 and older |
| G | Everyone |

- **RBAC approach**
  - Every user would be assigned one of *three roles*: Adult, Juvenile, or Child, possibly during registration
  - There would be *three permissions* created: Can view R-rated movies, Can view PG-13-rated movies, and Can view G-rated movies
  - The *Adult role* gets assigned with all three permissions
  - The *Juvenile role* gets Can view PG-13-rated movies and Can view G-rated movies permissions
  - The *Child role* gets the Can view G-rated movies permission only
  - Both the user-to-role assignment and the permission-to-role assignment are *manual administrative tasks*

# An online entertainment store: ABAC

- Streams movies to users for a flat monthly fee
- Enforces the following AC policy based on the user's age and the movie's content rating

| Movie Rating | Users Allowed Access |
|---|---|
| R | Age 17 and older |
| PG-13 | Age 13 and older |
| G | Everyone |

- **ABAC approach**
  - Does not need to explicitly define roles
  - Whether a user *u* can view a movie *m* (the security environment *e* is ignored here) is resolved by evaluating a policy rule such as the following:

```
R1: can_access(u, m, e) ←
    (Age(u) ≥ 17 ∧ Rating(m) ∈ {R, PG-13, G}) ∨
    (Age(u) ≥ 13 ∧ Age(u) < 17 ∧ Rating(m) ∈ {PG-13, G}) ∨
    (Age(u) < 13 ∧ Rating(m) ∈ {G})
```

    Age and Rating are the *subject attribute* and the *object attribute*, respectively
  - The **advantage** of the ABAC model shown here is that it eliminates the definition and management of *static* roles, hence eliminating the need for the *administrative tasks* for user-to-role assignment and permission-to-role assignment

# ABAC advantages

- But the advantage of ABAC is more clearly seen when we impose finer-grained policies
  - Movies are classified as either New Release or Old Release, based on *release date* compared to the current date
  - Users are classified as Premium User and Regular User, based on the *fee they pay*
  - We would like to enforce a **policy** that *only premium users can view new movies*
  - For the RBAC approach, we would have to **double the number of roles**, to distinguish each user by age and fee, and **double the number** of separate permissions as well
    - Thus we can see that to accommodate finer-grained policies, *as the **number of attributes** increases the number of roles and permissions grows exponentially*
  - In contrast, the ABAC approach deals with additional attributes in an **efficient way**
    - The policy R1 defined previously still applies, but we need two new policy rules:
    - R2: can_access($u$, $m$, $e$) ←
      (MembershipType($u$) = Premium) ∨
      (MembershipType($u$) = Regular ∧ MovieType($m$) = OldRelease)
    - R3: can_access(u, m, e) ← R1 ∧ R2
- With the ABAC approach, it is also easy to add environmental attributes
  - Suppose we wish to add a **new policy rule** that is expressed in words as follows:
    *Regular users are allowed to view new releases in **promotional periods***
  - This would be *difficult* to express in an RBAC model
  - In an ABAC model, we only need *add a rule* that checks to see the environmental attribute *today's date* falls within a promotional period and compose this rule with the previous ones

# Some frameworks for ABAC

We now examine some frameworks that are relevant to an access control approach **centered on attributes**

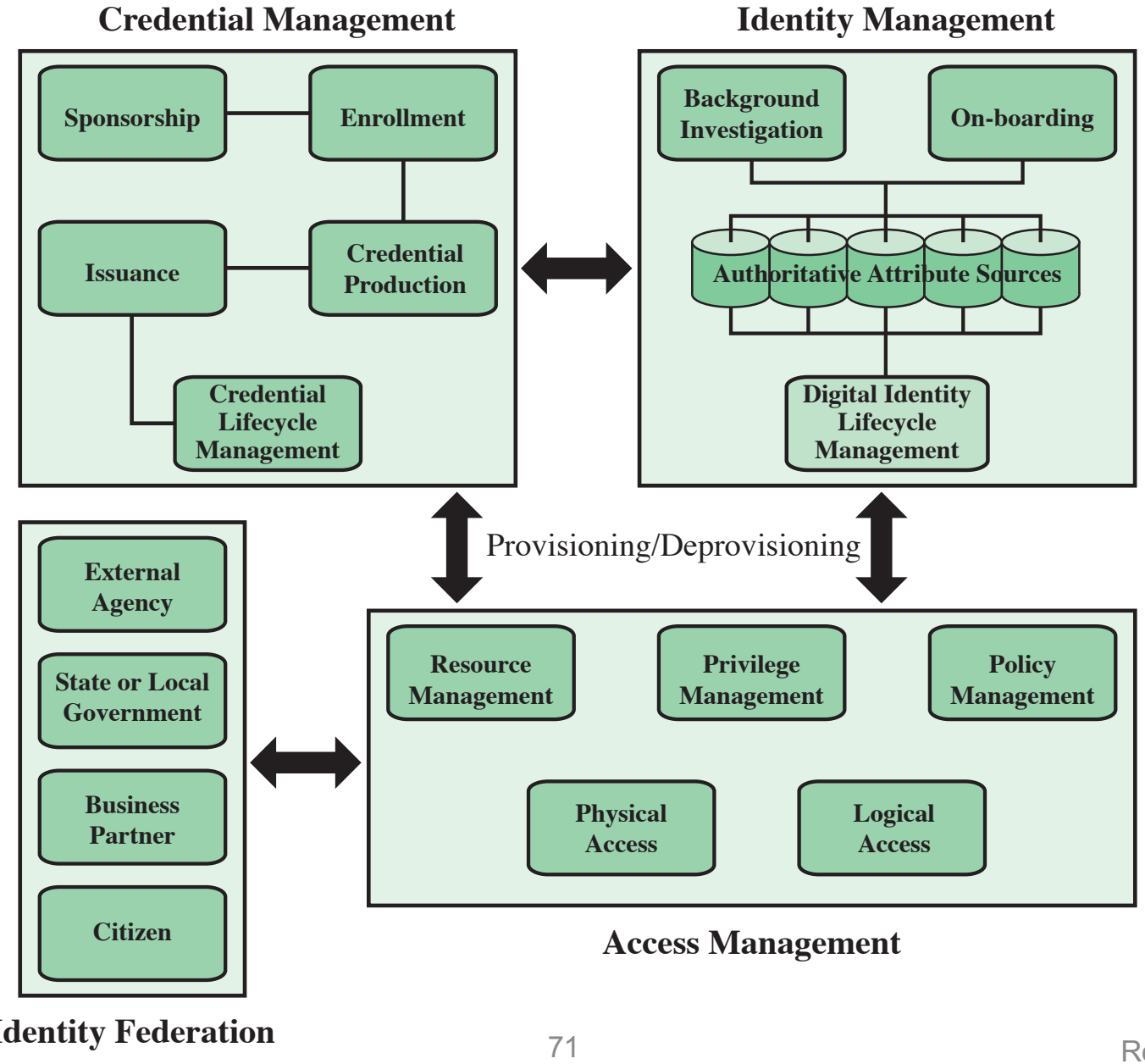
- ICAM frameworks for **managing** digital identities, credentials, and access control
- Trust frameworks for **exchanging** digital identities and attributes

# Index

# Identity, Credential, and Access Management (ICAM)

- A **comprehensive approach** to managing and implementing digital identities (and associated attributes), credentials, and access control
- Developed by the **U.S. government**, but applicable also to enterprises
- Design principles:
  - **Create trusted digital identity representations** of individuals and nonperson entities (NPEs, as e.g. processes, applications, and automated devices) seeking access to a resource (*identity management*)
  - **Bind those identities to credentials** that may serve as a proxy for the individual or NPE in access transactions (*credential management*)
  - **Use the credentials** to provide authorized access to resources (*access management*)

# An overview of the logical components of an ICAM architecture



# Digital Identity Management

- **Purpose** is to establish a trustworthy digital identity, connected to an individual or NPE, that is independent of a specific application or context
- Establishment of a digital identity typically begins with **collecting** identity data as part of an on-boarding process or background investigation
- A digital identity is **comprised of a set of attributes** that may be stored in various authoritative sources and when aggregated uniquely identify a user within a system or an enterprise
- This digital identity may then be **provisioned** into applications in order to support physical and logical access (part of Access Management) and **de-provisioned** when access is no longer required
- Final element is **lifecycle management** which includes:
  - Mechanisms, policies, and procedures for protecting identity information
  - Controlling access to identity information
  - Techniques for sharing identity information with applications that need it
  - Revocation of an identity

# Credential Management

- A **credential** is an object or data structure that authoritatively binds an identity (and optionally, additional attributes) to a token possessed and controlled by a subscriber
  - E.g. smart cards, private/public cryptographic keys, digital certificates
- **Credential management** encompasses five logical elements
  - An authorized individual **sponsors** an individual/NPE for getting a credential
  - The sponsored individual/NPE **enrolls for** the credential
    - Enrollment typically consists of identity proofing and the capture of biographic and biometric data
  - A credential is **produced**
    - This process may involve encryption, the use of a digital signature, the production of a smart card or other functions
  - The credential is **issued** to the individual/NPE
  - A credential must be **maintained** over its life cycle
    - Might include revocation, reissuance/replacement, reenrollment, expiration, reset, suspension, or reinstatement

# Access Management

- Deals with the management and control of the **ways individuals/NPEs are granted access to resources**
  - Covers both *logical* and *physical* access
  - May be internal or external to a system
- **Purpose** is to ensure that the proper identity verification is made when an individual/NPE attempts to access a security sensitive asset (e.g. building, computer systems, or data)
- The access control function makes use of the **credentials** and the **digital identity** of the requestor
- An enterprise-wide access control facility requires three supporting elements:
  - Resource management
  - Privilege management
  - Policy management

# Access management supporting elements

## Resource management

- Concerned with the definition of rules for resources needing access control
- Rules include credential requirements and what user, resource and environment attributes are required for accessing a given resource for a given function

## Privilege management

- Concerned with establishing and maintaining the privileges that are part of an individual's access profile
- Privileges are considered attributes that can be linked to a digital identity
- These attributes represent features of an individual that can be used as the basis for determining access decisions to both physical and logical resources

## Policy management

- Concerned with what is allowable and unallowable in an access transaction
- Given the identity and attributes of the requestor, the resource and the environment, a policy specifies what actions this user can perform on this object

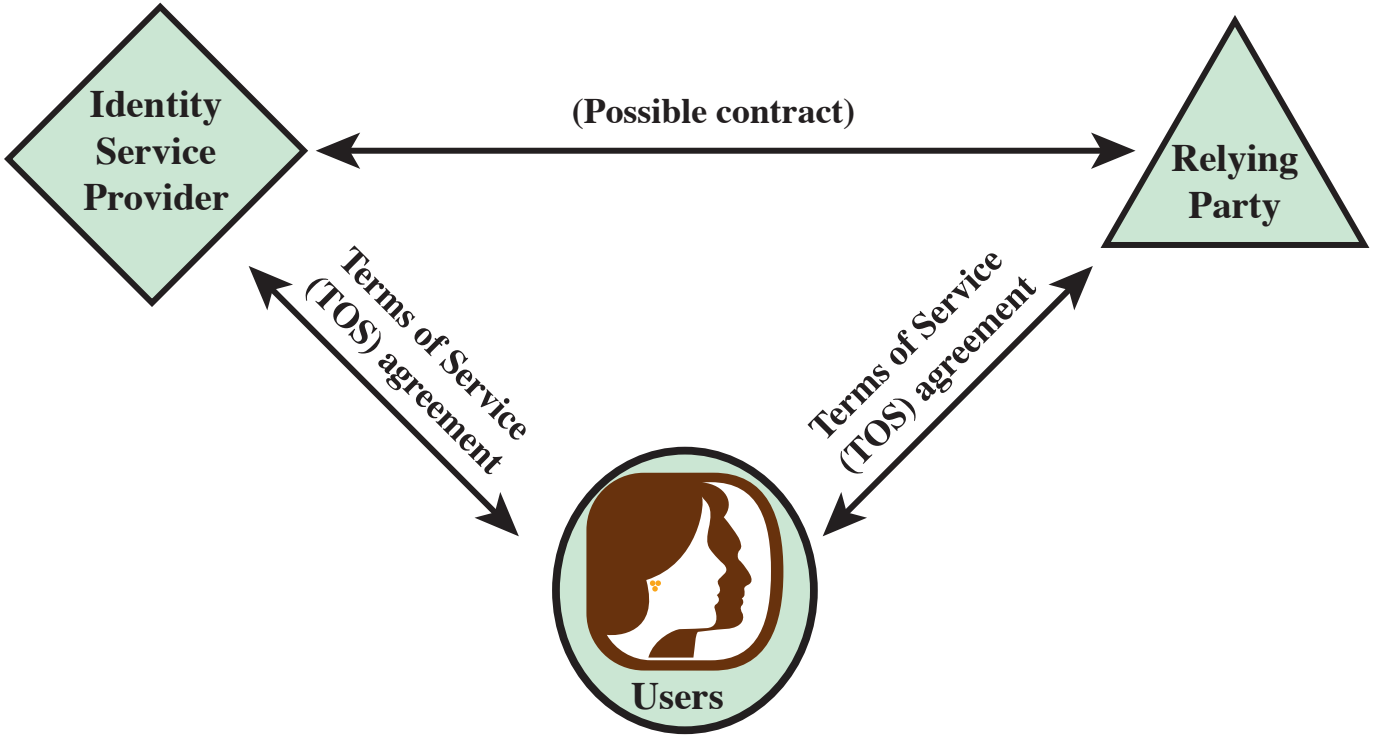
# Identity Federation

- Refers to the *technology, standards, and procedures* that allow an organization to trust digital identities, identity attributes, and credentials created and issued by other organizations
- Addresses two issues:
  - How do you **trust** identities of individuals from external organizations who need access to your systems?
  - How do you **guarantee** identities of individuals in your organization when they need to collaborate with external organizations?
- This leads to the concepts of trust frameworks for digital identity exchange

# Index

# Traditional Identity exchange approach

- Online or network transactions involving parties from different organizations, or between an organization and an individual user such as an online customer, generally require the **sharing of identity information**
- The exchange of identity information requires **users** to develop arrangements
  - with an **identity service provider** (ISP) to procure digital identity and credentials
  - with **relying parties** (or **service providers**), i.e. entities relying on the identity and credentials generated by the ISP, to provide end-user services and applications



# Traditional Identity exchange approach

- The **relying party** requires that
  - the user has been authenticated to some degree of assurance
  - the attributes imputed to the user by the ISP are accurate
  - the ISP is authoritative for those attributes
- The **ISP** requires assurance that
  - it has accurate information about the user
  - if it shares information, the relying party will use it in accordance with contractual terms and conditions, and the law
- The **user** requires assurance that the ISP and relying party
  - can be entrusted with sensitive information
  - respect user's preferences and privacy
- *All the parties want to know*
  - if the practices described by the other parties are actually those implemented by the parties
  - how reliable those parties are

# Trust Frameworks

- A trust framework is an open, standardized approach to trustworthy exchange digital identities and attributes
- It functions as a certification program
  - Enables a party who accepts a digital identity/credential (called the *relying party*) to trust the identity, security, and privacy policies of the party who issues the identity/credential (called the *identity service provider*), and vice versa
- It is developed by a community whose members have similar goals and perspectives
  - Defines the rights and responsibilities of that community's participants
  - Specifies the policies and standards specific to the community
  - Defines the community-specific procedures that provide assurance
- *Different* trust frameworks exist, and sets of participants *can tailor* trust frameworks to meet their particular needs

# OITF: Open Identity Trust Framework

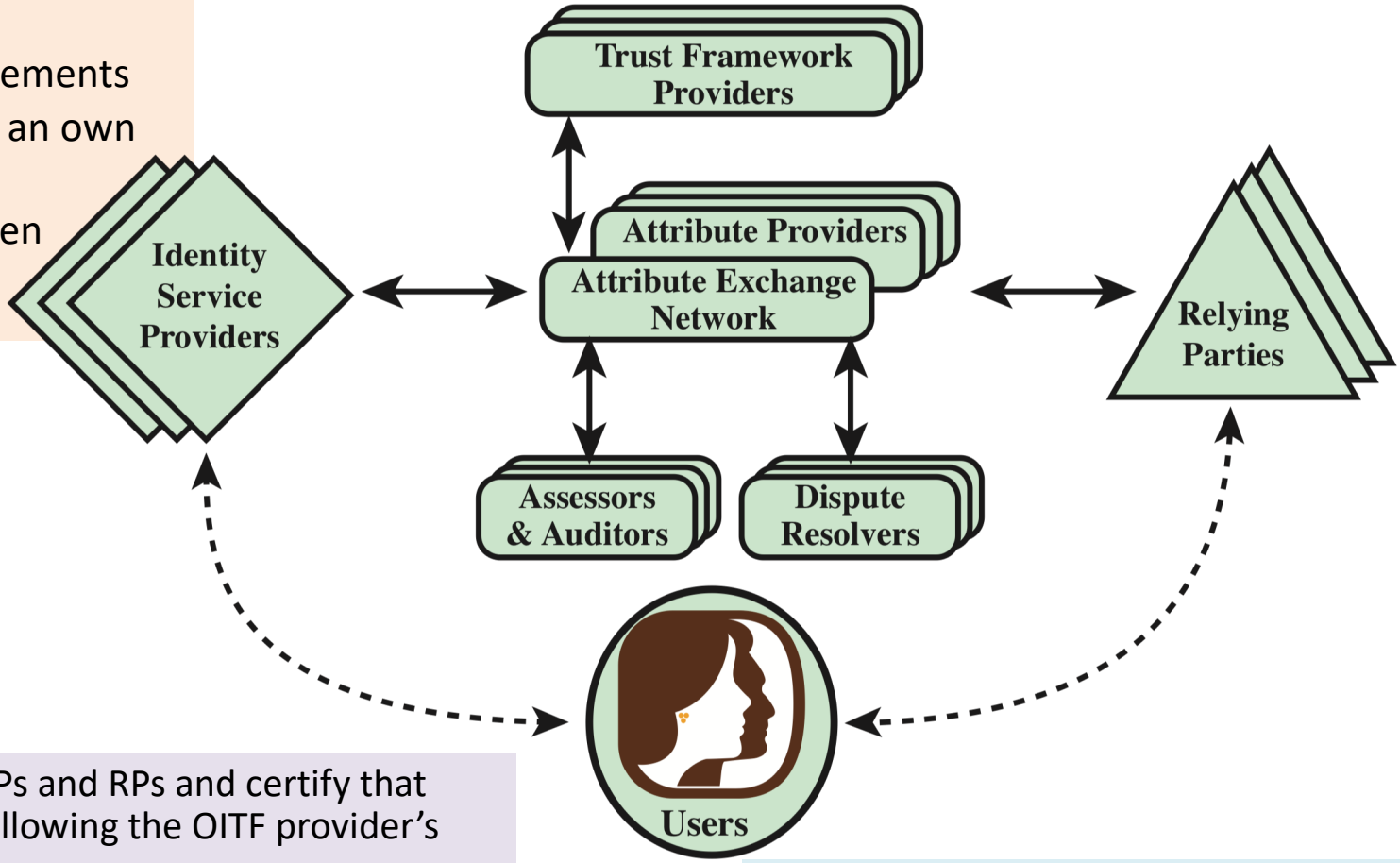
- Is the specification of an open, standardized trust framework
- It can be **integrated within the ICAM framework** to provide an identity federation functionality that allows organizations to exchange trustworthy digital identities and credentials

# OITF operation

- Responsible persons within participating organizations determine the technical, operational, and legal requirements for exchanges of identity information that fall under their authority
  - They then select OITF providers to implement these requirements
- These OITF providers translate the requirements into a **blueprint** for a trust framework that may include additional conditions of the OITF provider
- The OITF provider **inspects** identity service providers and relying parties and **contracts** with them to follow its trust framework requirements when conducting exchanges of identity information
  - The contracts carry provisions relating to *dispute resolvers* and *auditors* for contract interpretation and enforcement

# Elements involved in the OITF

- **Trust framework provider** is an organization that translates the requirements of policymakers into an own blueprint for a trust framework that it then proceeds to build




- **Assessors** evaluate ISPs and RPs and certify that they are capable of following the OITF provider's blueprint
- **Auditors** may be called on to check that parties' practices have been in line with what was agreed for the OITF
- **Dispute resolvers** provide arbitration and dispute resolution

- The solid arrowed lines indicate agreements with the trust framework provider for implementing technical, operations, and legal requirements
- The dashed arrowed lines indicate other agreements potentially affected by these requirements

# Index

# RBAC System for a Bank

- We consider the RBAC system implemented by the Dresdner Bank
- Originally, a simple **DAC scheme** was used on each server
  - Administrators maintained a **local access control file** on each host and defined the access rights for each employee on each application on each host
  - Cumbersome, time-consuming, and error-prone!
- After 1990, the bank introduced a systemwide **RBAC scheme**
  - **Roles** within the organization are defined by a combination of official position and job function

# Roles: combination of job function and official position

| Role | Function | Official Position |
|---|---|---|
| A | financial analyst | Clerk |
| B | financial analyst | Group Manager |
| C | financial analyst | Head of Division |
| D | financial analyst | Junior |
| E | financial analyst | Senior |
| F | financial analyst | Specialist |
| G | financial analyst | Assistant |
| • • • | • • • | • • • |
| X | share technician | Clerk |
| Y | support e-commerce | Junior |
| Z | office banking | Head of Division |

# Permission Assignments

| Role | Application | Access Right |
|---|---|---|
| A | money market instruments | 1, 2, 3, 4 |
| | derivatives trading | 1, 2, 3, 7, 10, 12 |
| | interest instruments | 1, 4, 8, 12, 14, 16 |
| B | money market instruments | 1, 2, 3, 4, 7 |
| | derivatives trading | 1, 2, 3, 7, 10, 12, 14 |
| | interest instruments | 1, 4, 8, 12, 14, 16 |
| | private consumer instruments | 1, 2, 4, 7 |
| • • • | • • • | • • • |

This table shows the (absolute) assigment of access rights to roles according to the application

# Permission Assignments

| Role | Application | Access Right |
|---|---|---|
| A | money market instruments | 1, 2, 3, 4 |
| | derivatives trading | 1, 2, 3, 7, 10, 12 |
| | interest instruments | 1, 4, 8, 12, 14, 16 |
| B | money market instruments | 1, 2, 3, 4, 7 |
| | derivatives trading | 1, 2, 3, 7, 10, 12, 14 |
| | interest instruments | 1, 4, 8, 12, 14, 16 |
| | private consumer instruments | 1, 2, 4, 7 |
| • • • | • • • | • • • |

To simplify the assigment, the inheritance **hierarchy** based on official position can be exploited: one role is superior to another if its position is superior and their functions are identical

# Hierarchy based on official position

| Role | Function | Official Position |
|---|---|---|
| A | financial analyst | Clerk |
| B | financial analyst | Group Manager |
| C | financial analyst | Head of Division |
| • • • | • • • | • • • |
| X | share technician | Clerk |
| Y | support e-commerce | Junior |
| Z | office banking | Head of Division |

The positions Head of Division, Group Manager, and Clerk are in *descending order* (thus role C *inherits* from B, which *inherits* from A)

# Permission Assignments with Inheritance

## Permission Assignments

| Role | Application | Access Right |
|---|---|---|
| A | money market instruments | 1, 2, 3, 4 |
| | derivatives trading | 1, 2, 3, 7, 10, 12 |
| | interest instruments | 1, 4, 8, 12, 14, 16 |
| B | money market instruments | 1, 2, 3, 4, 7 |
| | derivatives trading | 1, 2, 3, 7, 10, 12, 14 |
| | interest instruments | 1, 4, 8, 12, 14, 16 |
| | private consumer instruments | 1, 2, 4, 7 |
| • • • | • • • | • • • |

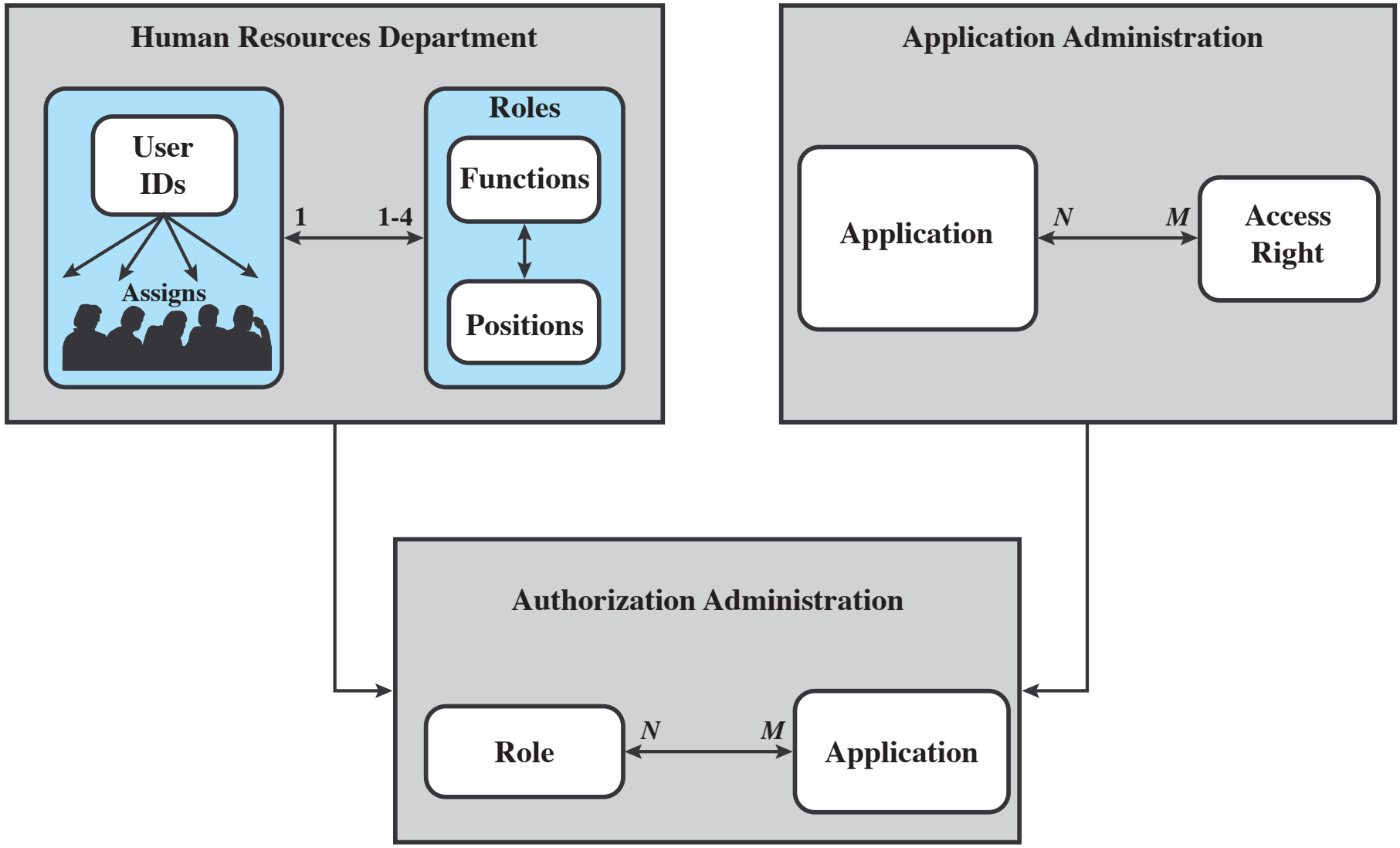
## PA with Inheritance

| Role | Application | Access Right |
|---|---|---|
| A | money market instruments | 1, 2, 3, 4 |
| | derivatives trading | 1, 2, 3, 7, 10, 12 |
| | interest instruments | 1, 4, 8, 12, 14, 16 |
| B | money market instruments | 7 |
| | derivatives trading | 14 |
| | private consumer instruments | 1, 2, 4, 7 |
| • • • | • • • | • • • |

The positions Head of Division, Group Manager, and Clerk are in *descending order* (thus role C *inherits* from B, which *inherits* from A)

# Access Control Administration

For greater security, the determination of access rights is compartmentalized into **three different administrative units**



# Access Control Administration

- The Application Administration determines (once and for all) the set of access rights associated with each individual application
- The Human Resource Department statically assigns a unique User ID and one or more roles (at most 4) to each employee
  - In principle, each user may be statically assigned up to four roles and may select a given role for use in invoking a particular application (this corresponds to the NIST concept of *session*)
  - In practice, most users are statically assigned a single role based on the user's official position and job function
- The Authorization Administration associates access rights with the use of an application on the basis of the user's role
- *When a user invokes an application*, the Application Administration determines what subset of the application's access rights are in force for this user in this role, and grants or denies access accordingly
  - E.g. role A has numerous access rights, but only a subset of those rights are applicable to each of the three applications that role A may invoke

# Some figures about this system

- There are 65 official positions, ranging from a Clerk in a branch, through the Branch Manager, to a Member of the Board
- These positions are combined with 368 different job functions provided by the Human Resources Department database
- Potentially, there are 23,920 (=65x368) different roles, but the number of roles in current use is about 1,300
  - This is in line with the experience of other RBAC implementations
- On average, 42,000 security profiles are distributed to applications each day by the Authorization Administration module

# Summary

- Access Control Principles
  - Access Control Context
  - Access Control Policies
- Subjects, Objects, and Access Rights
- Discretionary Access Control
  - An Access Control Model
  - Protection Domains
- Example: Unix File Access Control
  - Traditional UNIX File Access Control
  - Access Control Lists in UNIX
- Role-Based Access Control
  - RBAC Reference Models
- Attribute-Based Access Control
  - Attributes
  - ABAC Logical Architecture
  - ABAC Policies
- Identity, Credential, and Access Management
  - Identity Management
  - Credential Management
  - Access Management
  - Identity Federation
- Trust Frameworks
  - Traditional Identity Exchange Approach
  - Open Identity Trust Framework
- Case Study: RBAC System for a Bank