Introduction to Computer Security

Chapter 4: Access Control

Chi-Yu Li (2019 Spring)
Computer Science Department
National Chiao Tung University

Definition of Computer Security (RFC 4949)

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

Access control: the central element of computer security

Principal Objectives of Computer Security

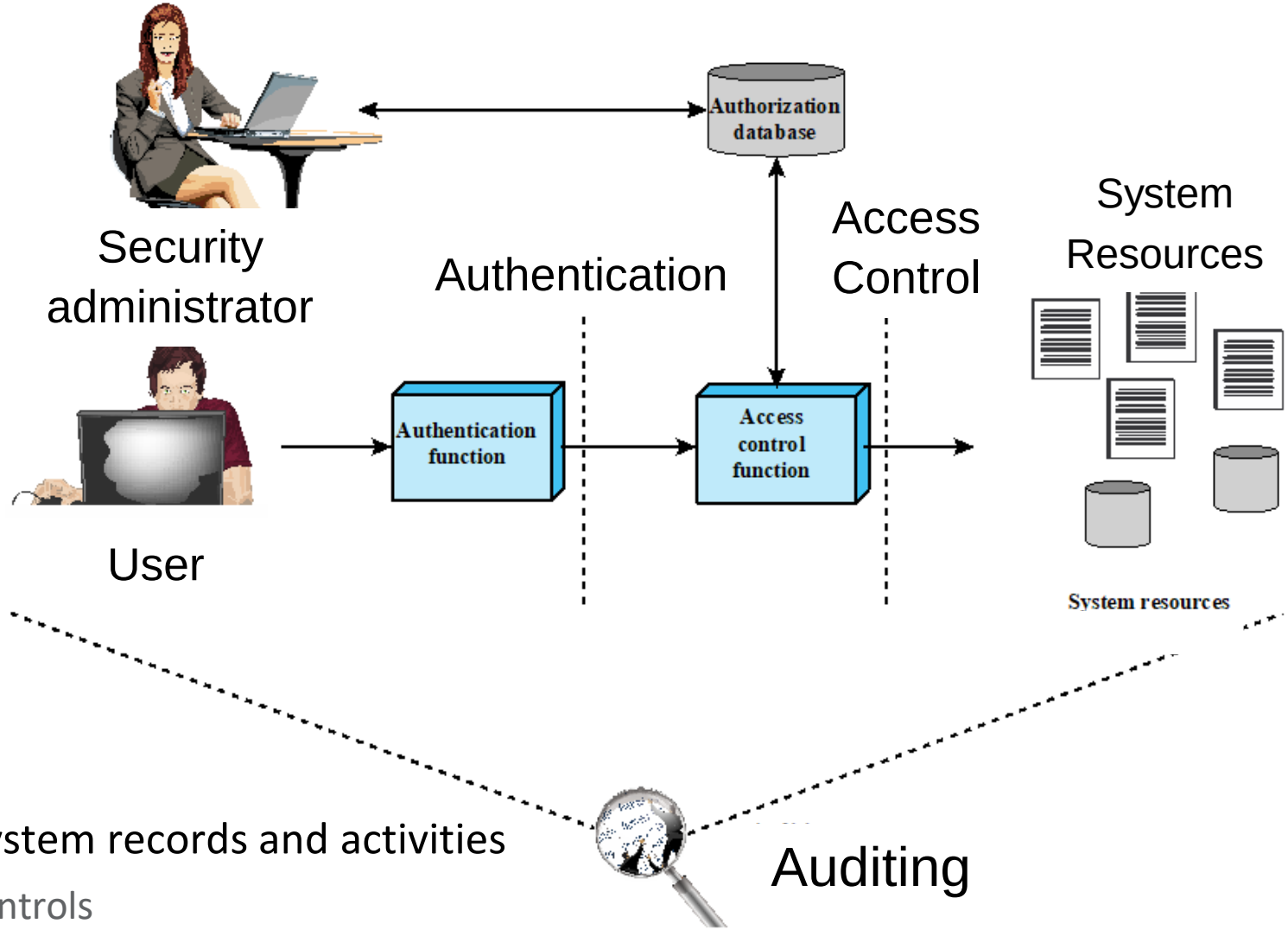
- Prevent unauthorized users from gaining access to resources
- Prevent legitimate users from accessing resources in an unauthorized manner
- Enable legitimate users to access resources in an authorized manner

Outline

- Access Control Principles
- Subjects, Objects, and Access Rights
- Discretionary Access Control
- Example: Unix File Access Control
- Role-Based Access Control
- Attribute-Based Access Control
- Case Study: RBAC System for a Bank

Access Control Context

- Authentication
  - Verification that user/system credentials are valid
- Authorization
  - The granting of a right or permission to a system entity to access a system resource
- Audit
  - An independent examination of system records and activities
    - To test for adequacy of system controls
    - To ensure compliance with established policy and operational procedures
    - To detect breaches in security
    - To recommend any indicated changes in control, policy and procedures



Access Control Policies

- Discretionary access control (DAC)
  - Based on the identity of the requestor, and on access rules stating what requestors are (or are not) allowed to do
  - Why discretionary?
    - An entity might have access rights to enable another entity to access some resource
- Mandatory access control (MAC)
  - Based on security clearances of system entities, and on security labels of resources
  - Why mandatory?
    - An entity that has clearance to access a resource may not enable another entity to access that resource

Access Control Policies (Cont.)

- Role-based access control (RBAC)
  - Based on the roles that users have, and on rules stating what accesses are allowed to given roles
- Attribute-based access control (ABAC)
  - Based on attributes of the user, the resource to be accessed, and current environmental conditions

Basic Elements

Subject

- An entity capable of accessing objects
- Three classes
  - Owner
  - Group
  - World (include all users)

Object

- A resource to which access is controlled

Access Rights

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

Discretionary Access Control (DAC)

- A general approach: access matrix
  - Subjects vs. Objects
  - Each entry: access right

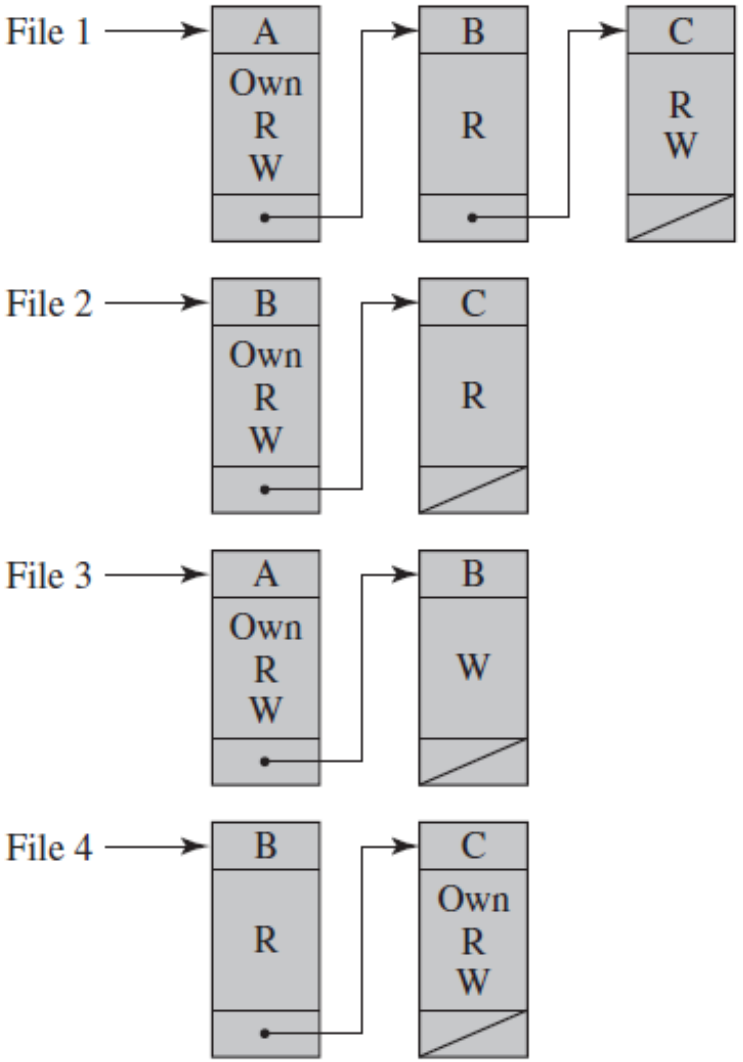
In practice, an access matrix is usually sparse!

SUBJECTS	OBJECTS: File 1	File 2	File 3	File 4
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

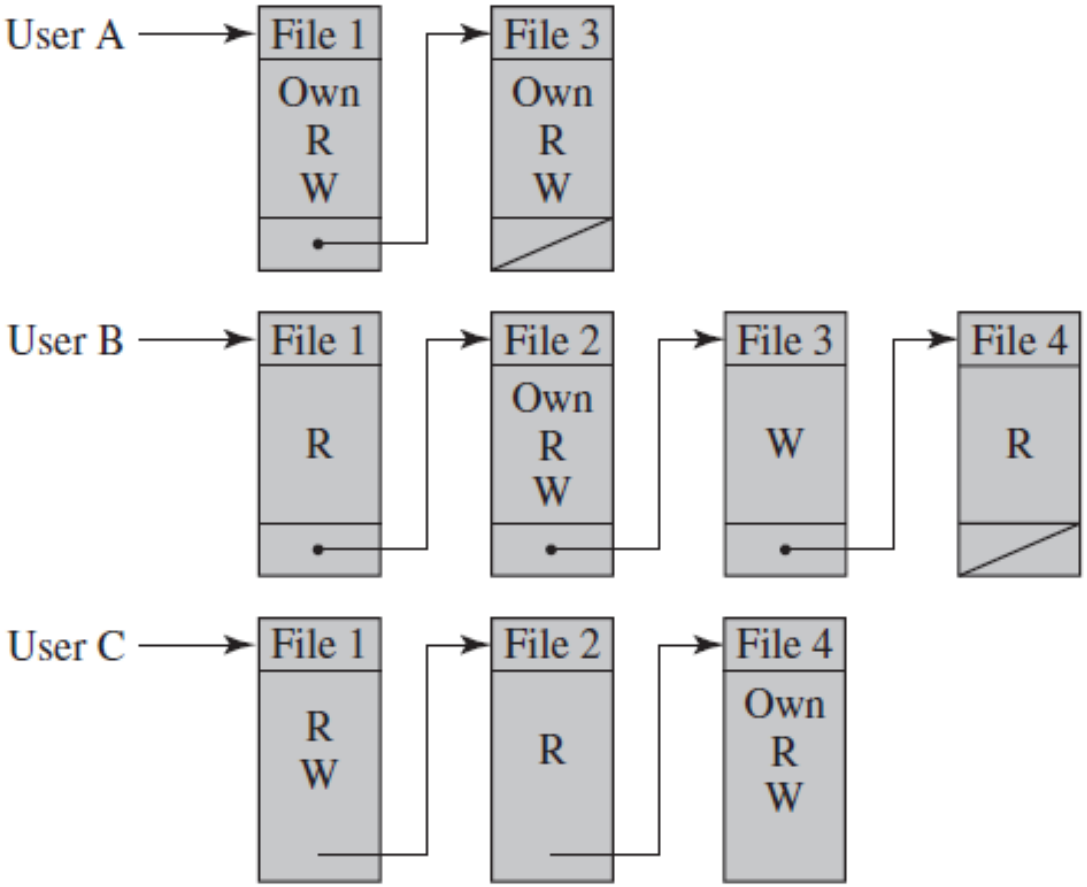
Decomposition Method I

- Access control lists (ACL): decomposed by columns (objects)
  - For each object, an ACL lists users and their permitted access rights
  - Default set of rights: users that are not explicitly listed
  - Convenient: determining which subjects have which access rights to a particular resource
  - Inconvenient: determining the access rights available to a specific user



Decomposition Method II

- Capability tickets: decomposed by rows (subjects)
  - A capability ticket specifies authorized objects and operations for a particular user
  - Convenient/Inconvenient: opposite to ACLs
- Have greater security problem than ACLs. Why?
  - Tickets may be dispersed around the system
    - Integrity of the ticket must be protected, guaranteed, and unforgeable
  - Two solutions
    - OS holds all tickets on behalf of users
    - An unforgeable token in the capability



Another Approach: Authorization Table [SAND94]

- Not sparse and more convenient than either ACLs or capability lists
- A relational database can easily implement an authorization table of this type
- Any drawback?

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 Access Control Model for DAC

- Three requirements
  - Representing the protection state
  - Enforcing access rights
  - Allowing subjects to alter the protection state in certain ways
- Concepts
  - As usual: a set of subjects, objects, and rules
  - New: protection states
- Protection states
  - Processes: delete, stop (block), and wake up
  - Devices: read/write, operation control, and block/unblock
  - Memory locations or regions: read/write
  - Subjects: grant or delete access rights of objects

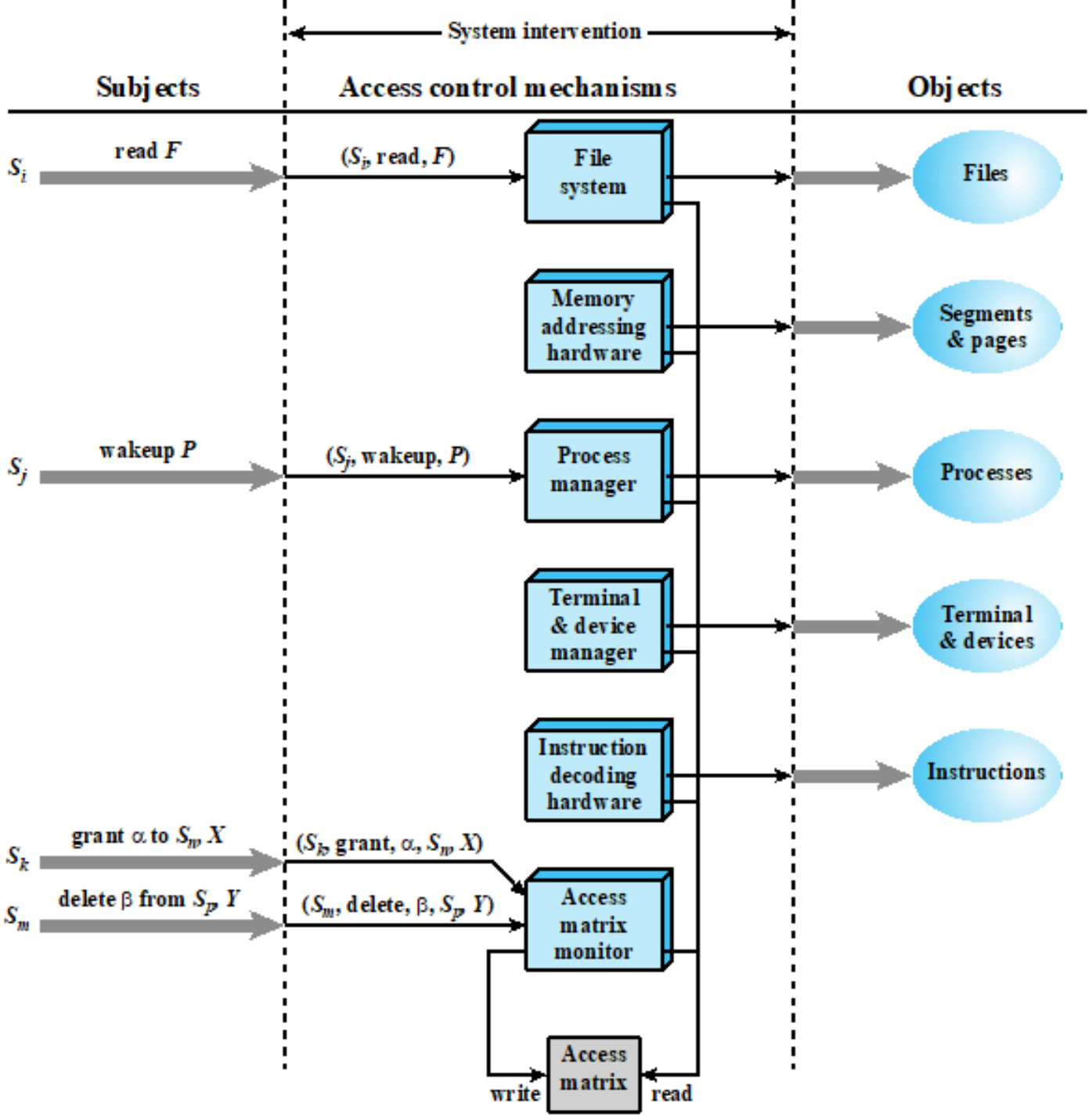
Example: Extended Access Control Matrix

		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

Example: Access Control Function

- Every access by a subject to an object is mediated by the controller for that object
- Decisions are based on access matrix monitor



More Flexible Model: Protection Domains

- A set of objects together with access rights to those objects
  - e.g., the access matrix
    - A row defines a protection domain
    - Each user has a protection domain → Any processes spawned by the user have access rights of the same domain

  Do the processes really need all the access rights?

- Recall security design principles: Least privilege
  - Every process and every user of the system should operate using the least set of privileges necessary to perform the task

Protection Domains (Cont.)

- More general concept: minimize the access rights that any user or process has at any one time
  - e.g., A user: spawns processes with a subset of the access rights of the user
    - Limit the capability of the processes
- Association between a process and a domain can be static or dynamic
  - e.g., A process: a sequence of procedures require different access rights
- One form: distinction mode in many OSes (e.g., UNIX)
  - User mode: certain areas of memory are protected and certain instructions may not be executed
  - Kernel mode

Example: UNIX File Access Control

UNIX files are administered using inodes (index nodes)

- An inode: a control structure with key information for a particular file
- Several file names may be associated with a single inode; inode and file are 1-1 mapping
- File attributes, permissions and control information are sorted 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

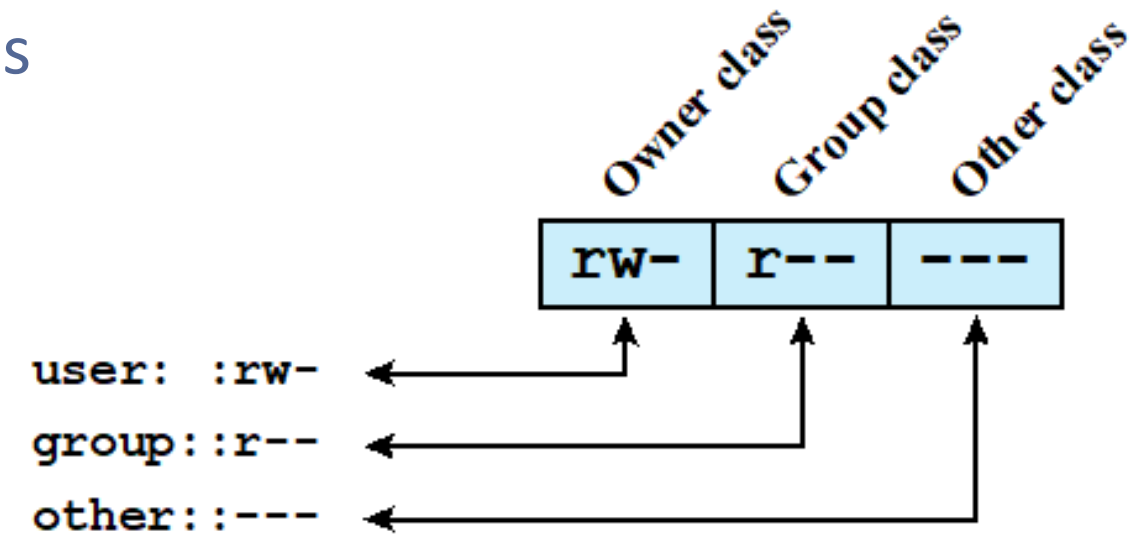
Directories are structured in a hierarchical tree

- May contain files and/or other directories
- Simply a file: contains file names plus pointers to associated inodes

Traditional UNIX File Access Control

- UNIX user: a unique user identification number (user ID)
  - A member of a primary group, and possibly other groups
  - Each group is identified by a group ID

- Each file/directory: 12 protection bits
  - First 9 bits: read, write, execute
  - Last 3 bits: setUID, setGID, and sticky



Traditional UNIX File Access Control (Cont.)

- SetUID/SetGID bits
  - Known as the “effective user ID” and “effective group ID”
  - System temporarily grants a real user with the rights of the file owner/group in addition to the real user’s rights
  - For executable files
    - Only effective while the program is being executed
    - Allows users to run programs with temporarily elevated privileges to perform a specific task
    - e.g., the `ping` command: need access to networking privileges that a normal user cannot access
  - For directories
    - SetGID: newly created files will inherit the group of this directory, rather than the primary group ID of the user who created this file
    - SetUID is ignored
  - Security risk?

Traditional UNIX File Access Control (Cont.)

- Examples: passwd and ping

setuid: 4
setgid: 2

```
chiyuli@linux1:~ [83x25]
Connection  Edit  View  Window  Option  Help
[chiyuli@linux1 ~]$ stat -c "%a %U:%G %n" /usr/bin/passwd
4755 root:root /usr/bin/passwd
[chiyuli@linux1 ~]$ stat -c "%a %U:%G %n" /etc/passwd
644 root:root /etc/passwd
[chiyuli@linux1 ~]$ stat -c "%a %U:%G %n" /etc/shadow
0 root:root /etc/shadow
[chiyuli@linux1 ~]$ passwd
Please enter your old LDAP(Linux/FreeBSD) password:
chiyuli@linux1:~ [83x25]
Connection  Edit  View  Window  Option  Help
[chiyuli@linux1 ~]$ stat -c "%a %U:%G %n" /bin/ping
755 root:root /bin/ping
[chiyuli@linux1 ~]$ getcap /bin/ping
/bin/ping = cap_net_admin,cap_net_raw+p
[chiyuli@linux1 ~]$ getcap /usr/bin/passwd
[chiyuli@linux1 ~]$
```

Traditional UNIX File Access Control (Cont.)

- Sticky bit
  - Files: the system should retain the file contents in memory following execution (no longer used)
  - Directories: only the owner of any file in the directory can rename, move, or delete that file
    - Useful for managing files in shared temporary directories
- superuser
  - Exempts from the usual file access control constraints
  - Needs great care on the programs owned by and setuid set to “superuser”

Traditional UNIX File Access Control (Cont.)

- What issues does this access scheme have?
  - Consider one scenario
    - Read access for file X to Users A and B
    - Read access for file Y to Users B and C
  - Need at least two user groups
  - What if there are a large number of different groupings of users requiring a range of access rights to different files?
- No scalability: unwieldly and difficult to manage

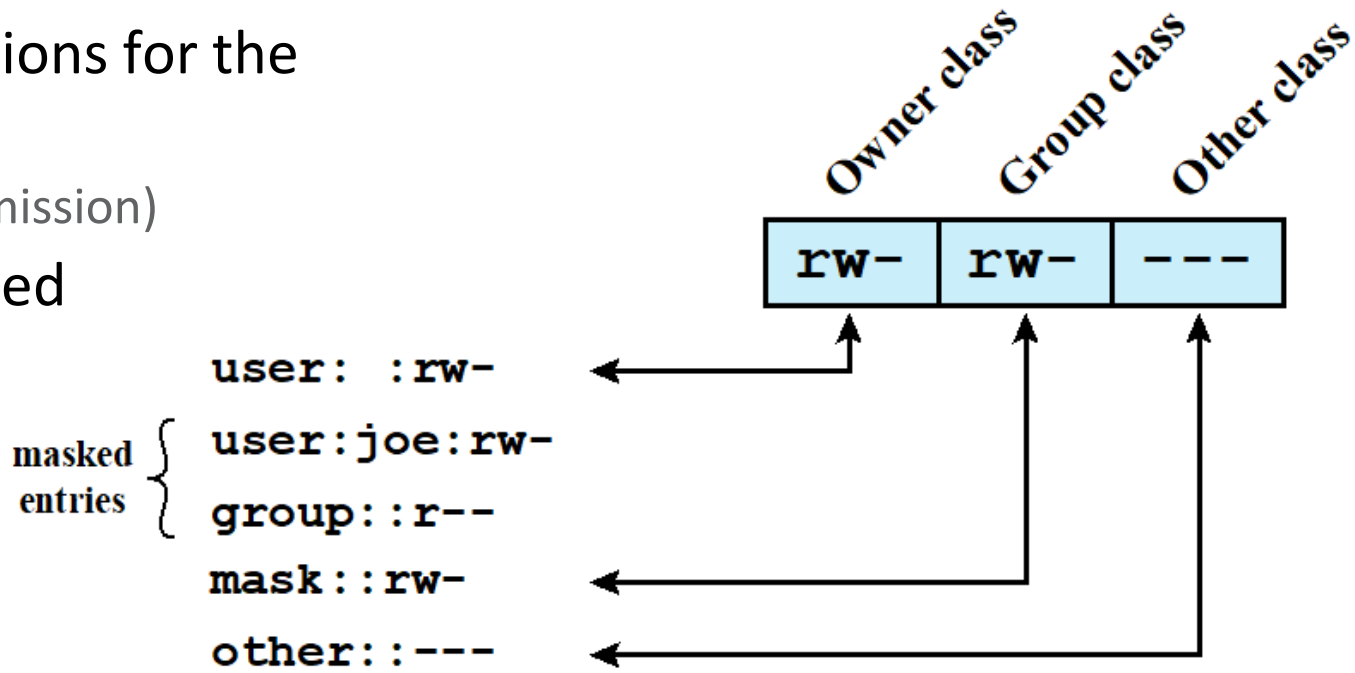
Modern UNIX Access Control: Access Control Lists (ACLs)

- Supported by many modern UNIX-based OSes
  - e.g., FreeBSD, OpenBSD, Linux, and Solaris
  - Extended ACL vs. minimal ACL (traditional)

- FreeBSD
  - Any number of users and groups can be assigned to a file
    - Each with three protection bits
  - A file need not have an ACL; may be protected solely by traditional access control
  - An additional protection bit: whether the file has an extended ACL

Modern UNIX Access Control (Cont.)

- Extended ACLs are used with the following strategies
  - Owner and other classes remain the same
  - Group class specifies the permissions for the owner group for this file
    - Functions as a mask (maximum permission)
  - Additional named users and named groups may be associated with the file
    - Each with a 3-bit permission field



Examples

```
[root@linux ~]# setfacl -m u:bob:rwx test
[root@linux ~]# getfacl test
# file: test
# owner: root
# group: root
user::rwx
user:bob:rwx
group::r--
mask::rwx
other::r--
```

Step 1

```
[root@linux ~]# setfacl -m g:cs:rx test
[root@linux ~]# getfacl test
# file: test
# owner: root
# group: root
user::rwx
user:bob:rwx
group::r--
group:cs:r-x
mask::rwx
other::r--
```

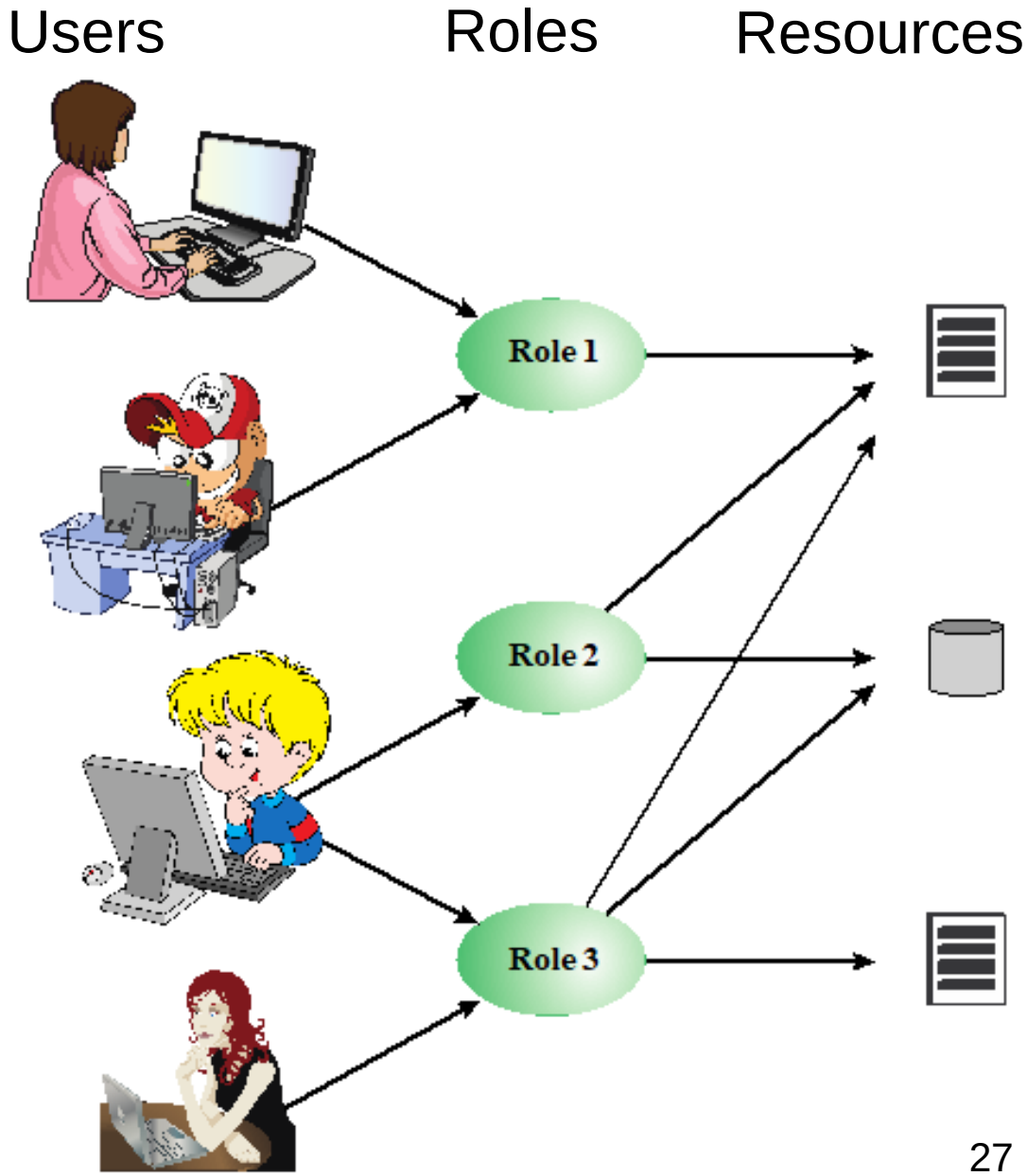
Step 2

```
[root@linux ~]# setfacl -m m:r test
[root@linux ~]# getfacl test
# file: test
# owner: root
# group: root
user::rwx
user:bob:rwx        effective: ?
group::r--
group:cs:r-x        effective: ?
mask::r--
other::r--
```

Step 3

Role-based Access Control (RBAC)

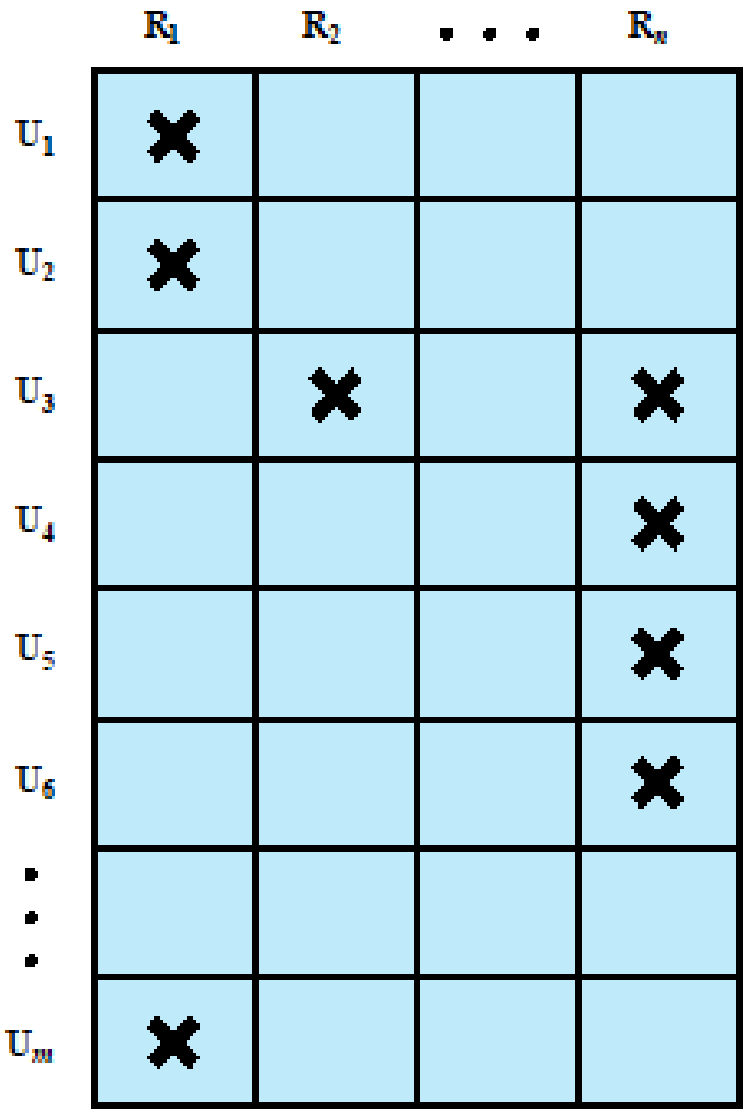
- Based on the roles that users assume, instead of their identities
- Widespread commercial use and an area of active research
- Many-to-many relationship
  - users to roles
  - roles to resources



Access Control Matrix for RBAC

- RBAC implementation: obeys principle of "least privilege"
  - Each role contains the minimum set of access rights needed for that role

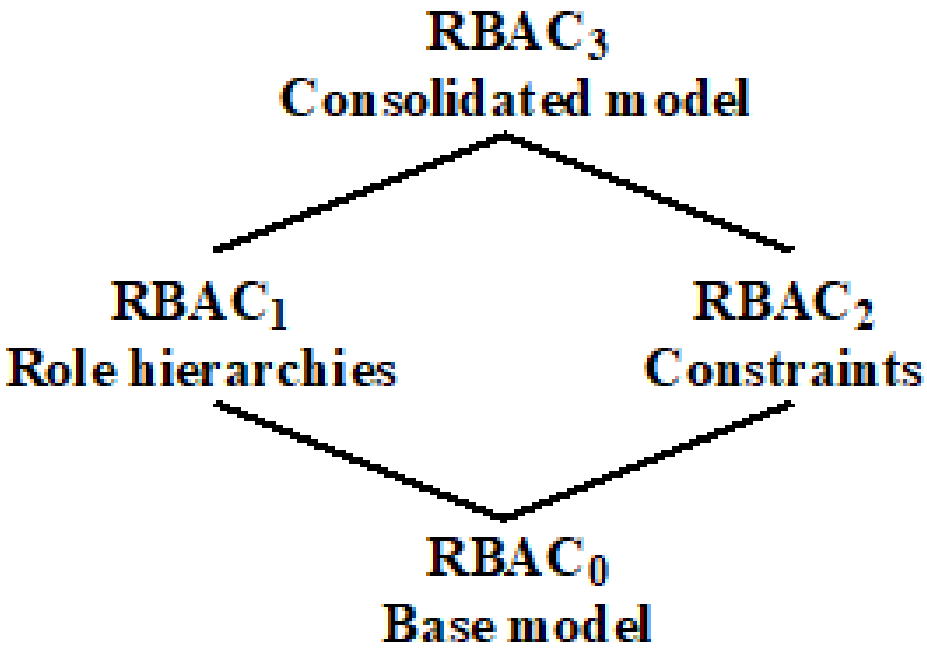
ROLES \ OBJECTS	R_1	R_2	R_n	F_1	F_1	P_1	P_2	D_1	D_2
R_1	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
R_2		control		write *	execute			owner	seek *
...									
R_n			control		write	stop			



RBAC Reference Models

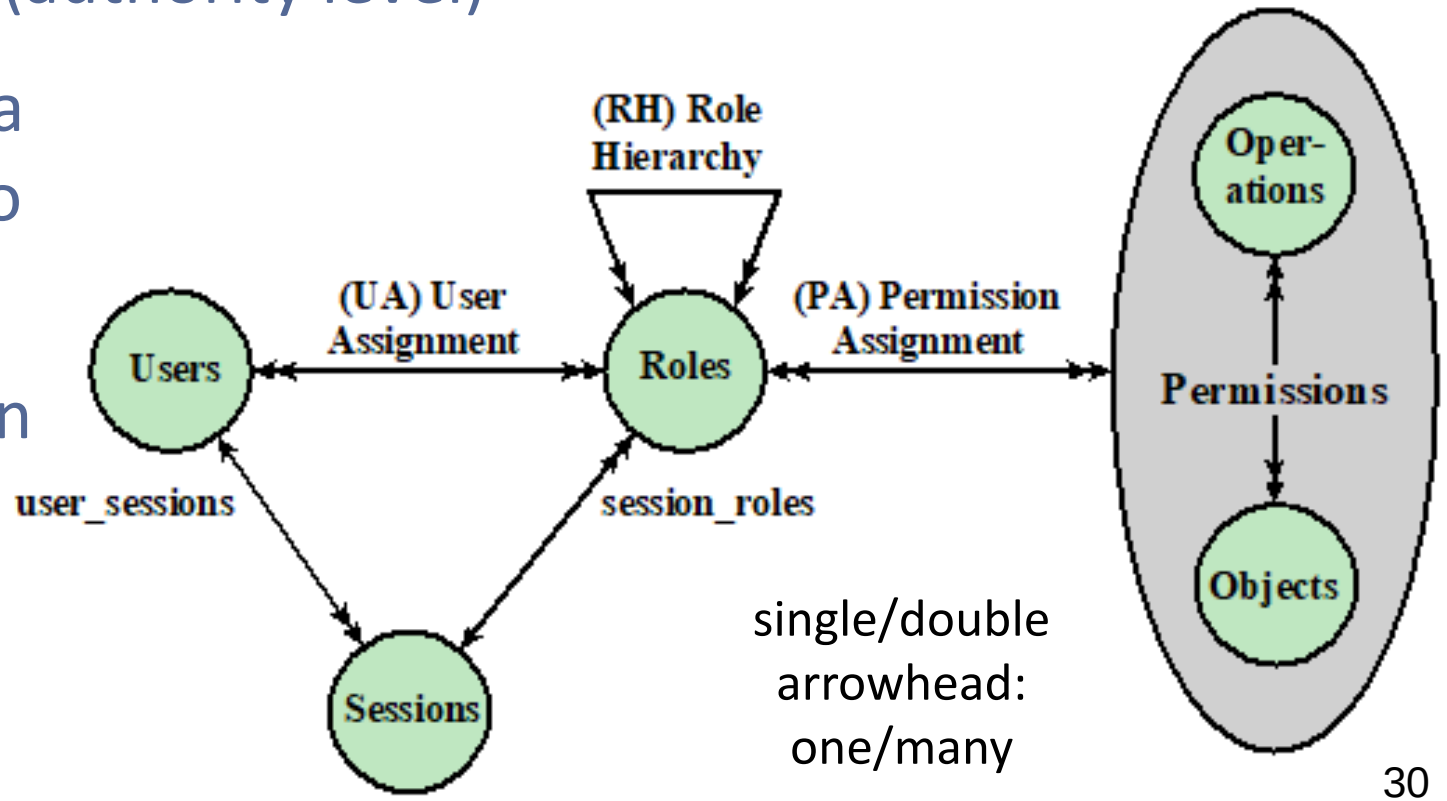
- A family of reference models have been defined as the basis for ongoing standardization efforts [SAND96]

- Four models
  - $RBAC_0$: minimum functionality
  - $RBAC_1$: RBAC0 + role hierarchies
  - $RBAC_2$: RBAC0 + constrains
  - $RBAC_3$: RBAC0 + RBAC1 + RBAC2



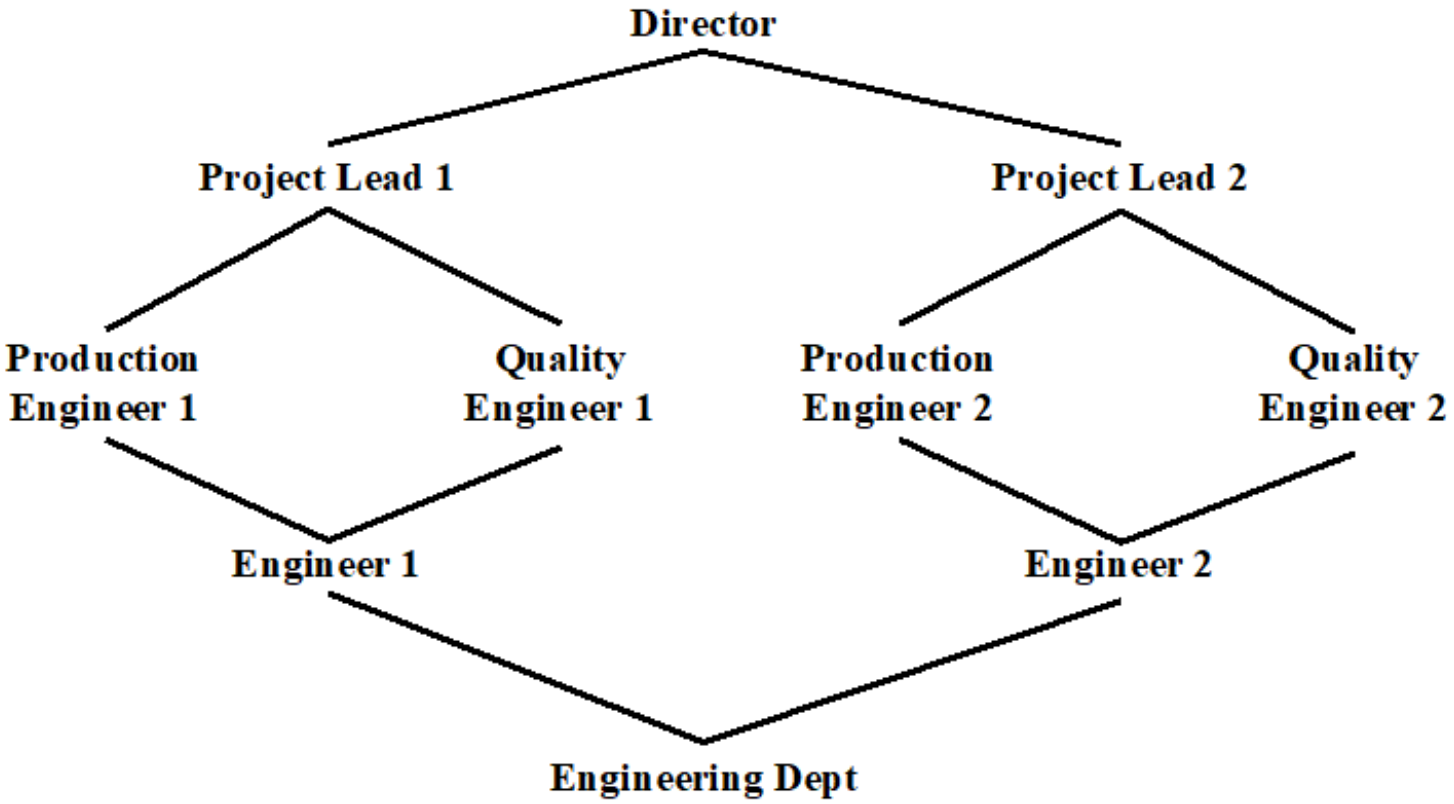
$RBAC_0$: Base Model

- User: an individual that has access to this computer system
  - Has an associated user ID
- Role: a named job function (authority level)
- Permission: an approval of a particular mode of access to one or more objects
- Session: a mapping between a user and set of roles to which a user is assigned



$RBAC_1$: Role Hierarchies

- Roles with greater responsibility: greater authority to access resources
  - A subordinate job function may have a subset of the access rights of the superior job function



$RBAC_2$: Constraints

- Adapting RBAC to the specifics of administrative and security policies in an organization
  - Mutually exclusive roles
    - A user can be assigned to only 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
    - Non-overlapping permissions, if two users are assigned to different roles in the set
  - Cardinality
    - Setting a maximum number of users w.r.t. roles
    - e.g., a project leader role or a department head role might be limited to a single user
  - Prerequisite role
    - A user can only be assigned to a particular role if it is already assigned to some other specified role
    - e.g., a user can be assigned to a senior (higher) role only if it is already assigned an immediately junior (lower) role

Attribute-based Access Control (ABAC)

- Define authorizations that express conditions on properties of both the resource and the subject
  - e.g., Alice (subject attr.) can access the HR database (resource attr.) during week days (environment attr.)
- Strength: *flexibility* and *expressive power*
- Drawback: the *performance impact* of evaluating predicates on both resource and user properties for each access
  - However, increased performance cost is less noticeable for Web services and cloud computing

ABAC Model: Attributes

- Subject 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
    - e.g., name, organization, job title
- Object attributes
  - An object (or resource) is a passive system-related entity containing or receiving information
  - Objects have attributes that can be leveraged to make access control decisions
    - e.g., file name, file size, creator

ABAC Model: Attributes (Cont.)

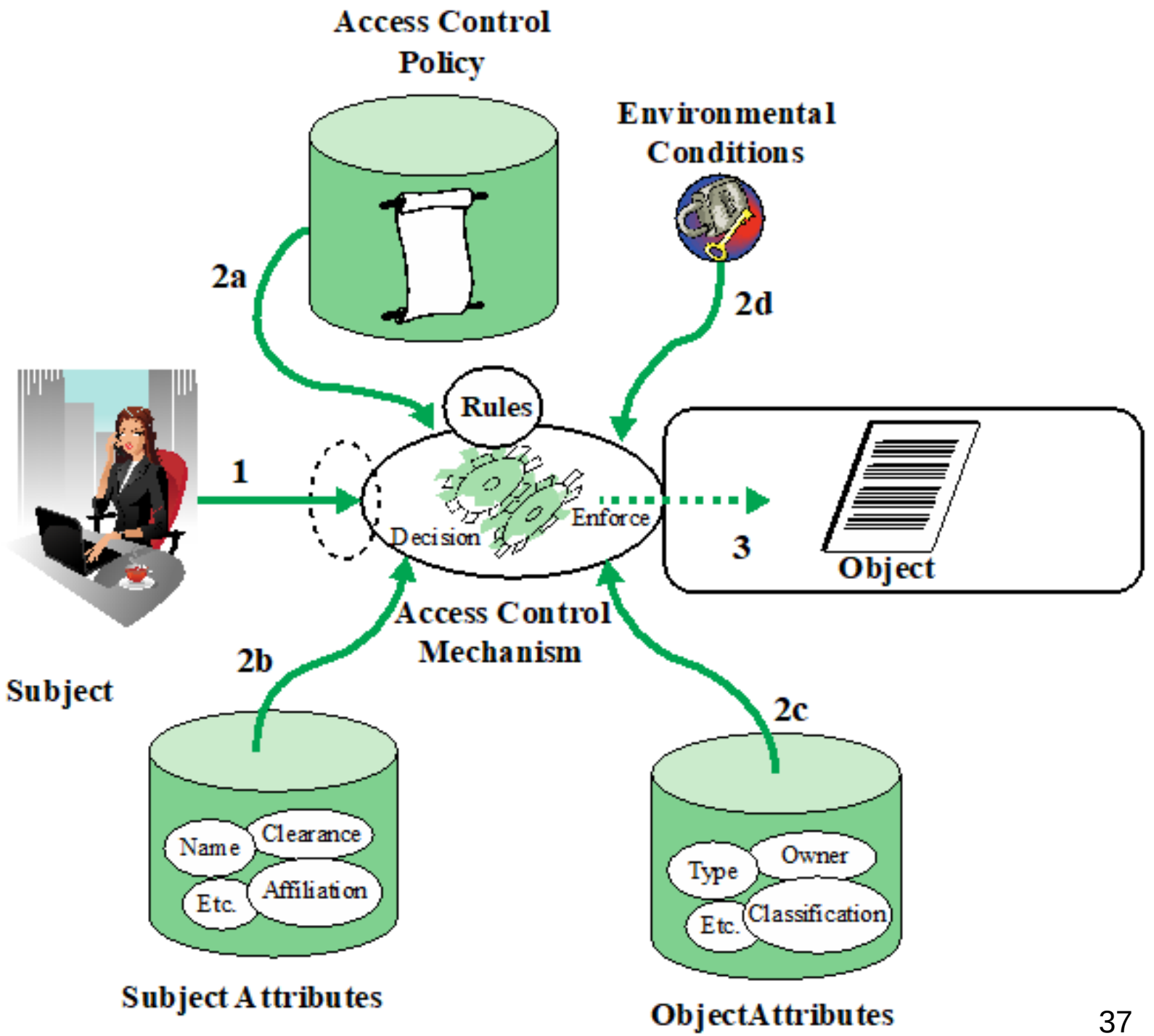
- Environment attributes
  - The operational, technical, and even situational environment or context in which the information access occurs
    - e.g., current date, time, network type, etc.
  - These attributes have so far been largely ignored in most access control policies

ABAC Model: Distinguishable

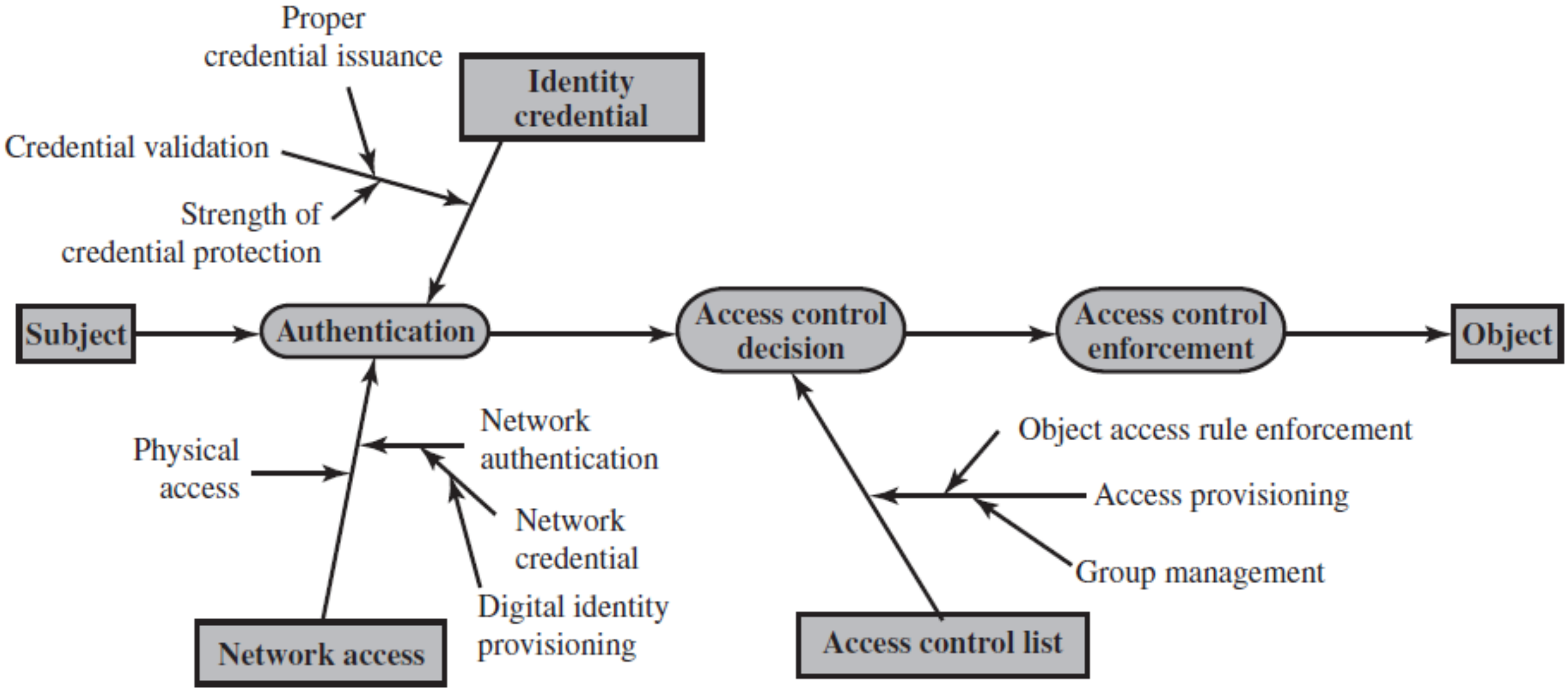
- Controls access to objects by evaluating rules against the attributes of entities (subject and object), operations, and the environment
  - Attributes may be considered characteristics of anything that may be defined
- Capable of enforcing DAC, RMAC, and MAC concepts
- Fine-grained access control: allows an unlimited number of attributes to be combined to satisfy any access control rule

ABAC Logical Architecture

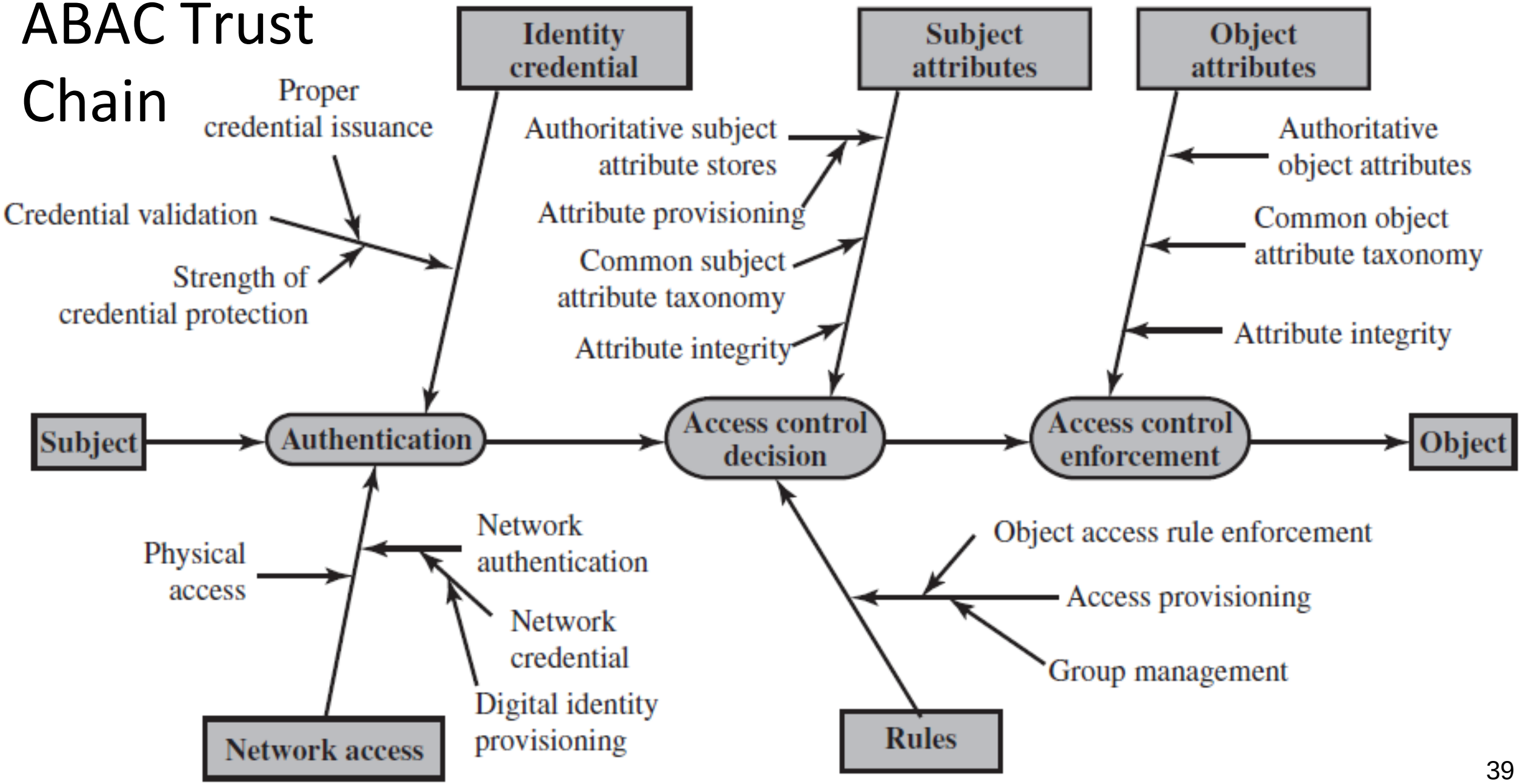
- Four independent sources of information used for the access control decision
- It is very powerful and flexible, but the cost is larger than that of other access control approaches



ACL Trust Chain



ABAC Trust Chain



ABAC Policies

- A policy is a set of rules and relationships that govern allowable behavior within an organization
  - Based on (1) privileges of subjects; (2) how resources or objects are to be protected; (3) under which environment conditions
- An ABAC policy model [YUAN05]

Subject attributes $ATTR(s) : SA_1 \times SA_2 \times \cdots \times SA_K$

Object attributes $ATTR(o) : OA_1 \times OA_2 \times \cdots \times OA_M$

Environment attributes $ATTR(e) : EA_1 \times EA_2 \times \cdots \times EA_N$

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

Case Study: RABC System for a Bank

- Dresdner bank uses a variety of computer applications over servers and mainframe computers
- In 1990, a simple DAC system was used
- For each server and mainframe computer, administrators maintained a local access control file on each host
  - Defining access rights for each employee on each application installed on the host
- However, it was cumbersome, time-consuming, and error-prone

How to solve it?

Case Study: RABC System for a Bank (Cont.)

- Dresdner bank then introduced an RBAC scheme
- The determination of access rights is compartmentalized into three different administrative units
  - Roles: a combination of official position and job function
  - Difference from NIST: a role is defined by a job function

(a) Functions and Official Positions

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

Case Study: Functions and Roles for Banking

Role A: Financial analyst/Clerk

Role B: Financial analyst/Group manager

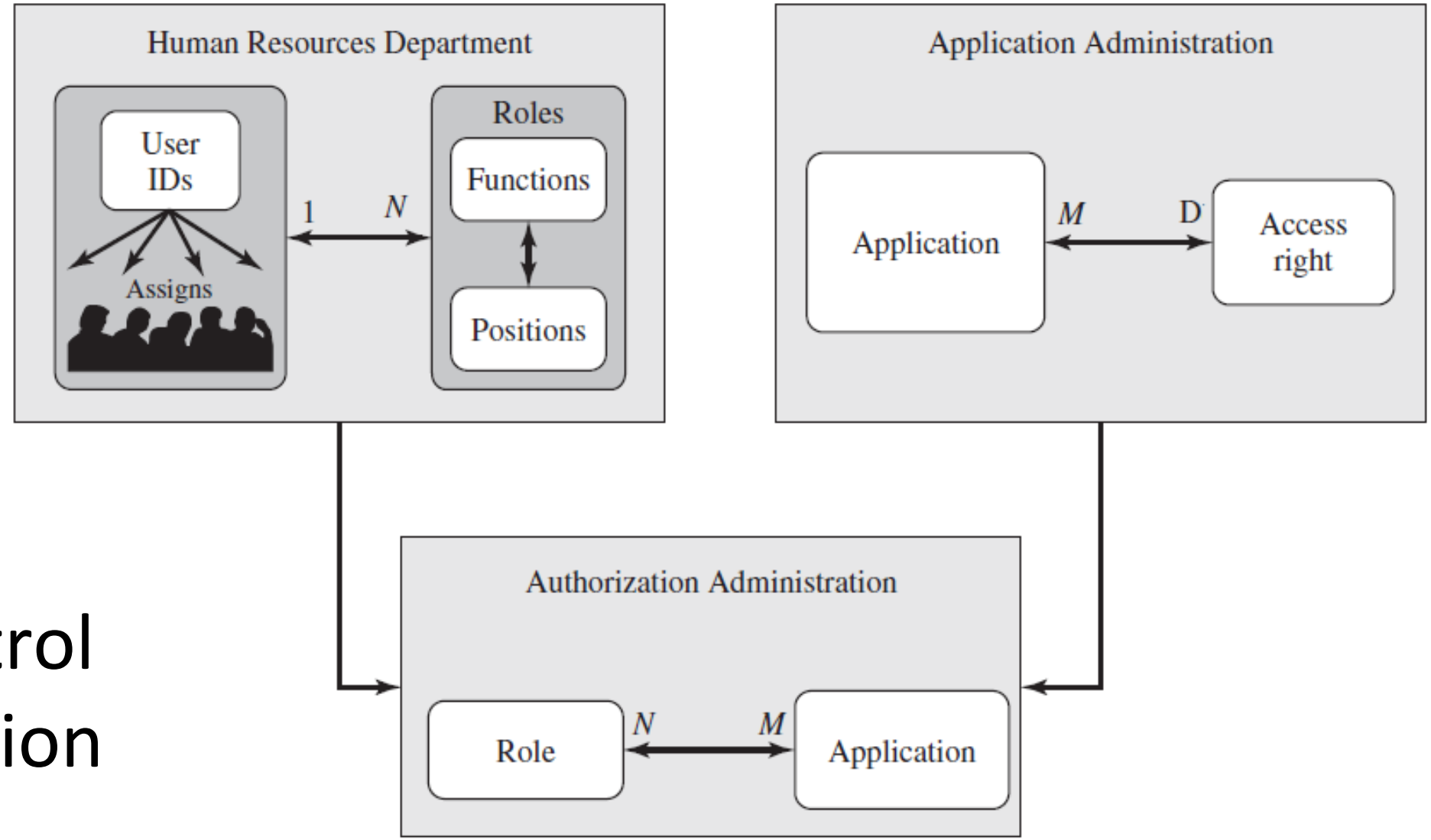
Permission Assignments

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
• • •	• • •	• • •

Permission Assignments 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
• • •	• • •	• • •

Case Study: Access Control Administration



Questions?