Cryptography and Network Security II

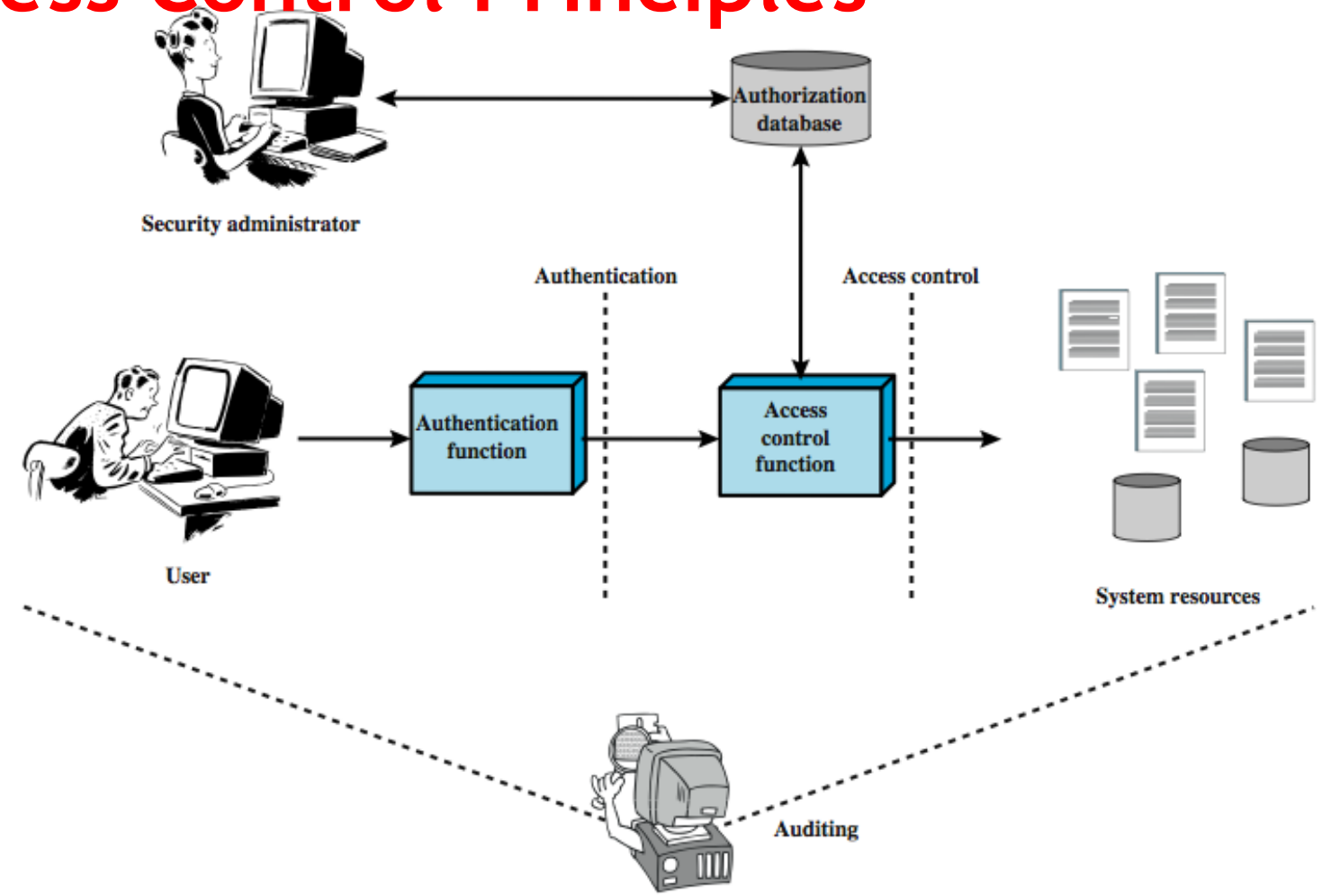
Second Course

Lecture 5: Access Control

Access Control

- "The prevention of unauthorized use of a resource, including the prevention of use of a resource in an unauthorized manner"
- Central element of computer security
- Assume have users and groups
  - authenticate to system
  - assigned access rights to certain resources on system

Access Control Principles



Access control policies

- **Discretionary** access control (DAC): based on the identity of the requestor and access rules
- **Mandatory** access control (MAC): based on comparing security labels with security clearances (mandatory: one with access to a resource cannot pass to others)
- **Role-based** access control (RBAC): based on user roles
- **Attribute-based** access control: based on the attributes of the user, the resources and the current environment

Access Control Requirements

- Reliable input: a mechanism to authenticate
- Fine and coarse specifications: regulate access at varying levels (e.g., an attribute or entire DB)
- Least privilege: min authorization to do its work
- Separation of duty: divide steps among different individuals
- Open and closed policies: accesses specifically authorized or all accesses except those prohibited
- Administrative policies: who can add, delete, modify rules

Access Control Elements

- Subject: entity that can access objects
  - a process representing user/application
  - often have 3 classes: **owner**, **group**, **world**
- Object: access controlled resource
  - e.g. files, directories, records, programs etc
  - number/type depend on environment
- Access right: way in which subject accesses an object
  - e.g. read, write, execute, delete, create, search

Discretionary Access Control

- Often provided using an access matrix
  - lists subjects in one dimension (rows)
  - lists objects in the other dimension (columns)
  - each entry specifies access rights of the specified subject to that object
- Access matrix is often sparse
- Can decompose by either row or column

Access Control Structures

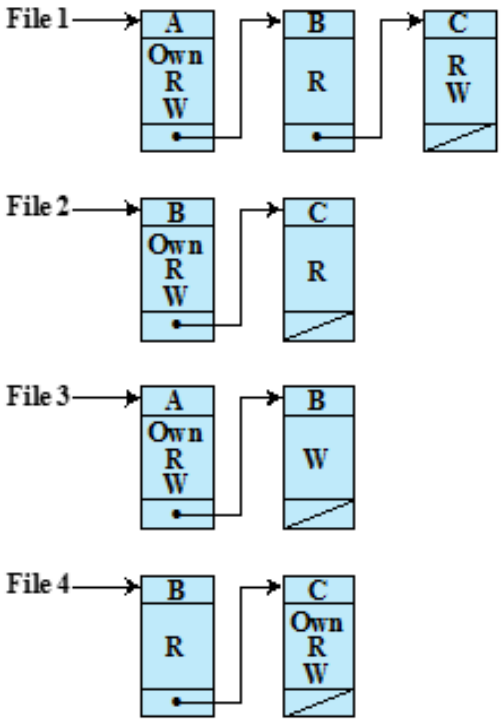
- Access control lists (decomposed by column)
- Capability tickets (decomposed by row)
- See page 119
- Also see alternative table representation on page 120 (tabular but not sparse)

An access matrix

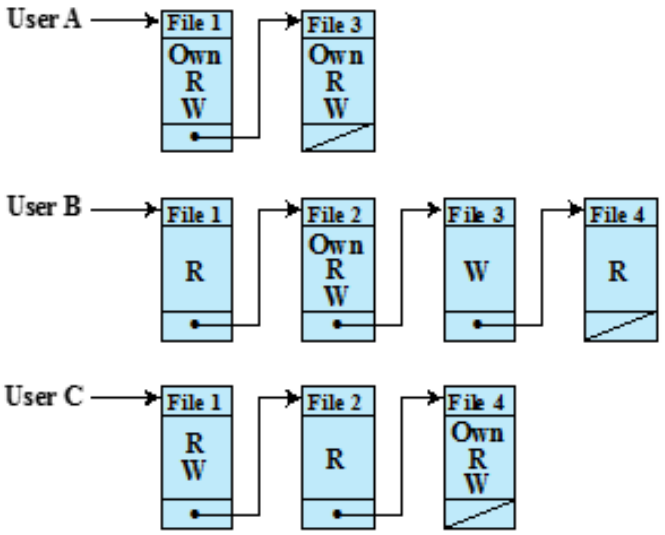
		OBJECTS			
		File 1	File 2	File 3	File 4
SUBJECTS	User A	Own Read Write		Own Read Write	
	User B	Read	Own Read Write	Write	Read
	User C	Read Write	Read		Own Read Write

(a) Access matrix

Access matrix data structures



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



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

Alternate authorization table

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

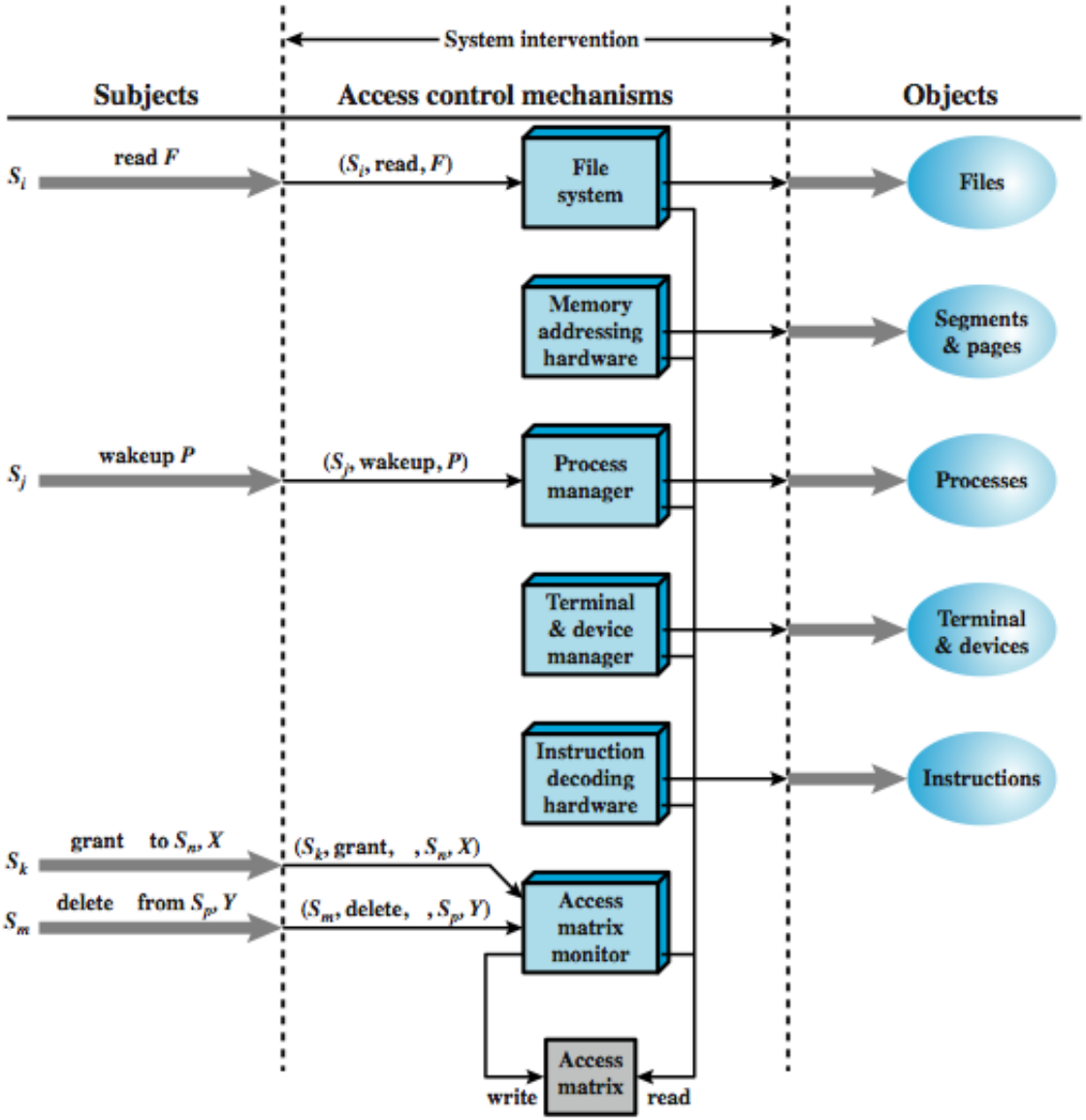
An Access Control Model

- Extend the universe of objects to include processes, devices, memory locations, subjects

		OBJECTS								
		subjects			files		processes		disk drives	
		S_1	S_2	S_3	F_1	F_1	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

Access Control Function



Access control system commands

Rule	Command (by S_0)	Authorization	Operation
R1	**transfer** $\begin{Bmatrix}\alpha^* \\ \alpha\end{Bmatrix}$ **to** S, X	'α^*' in $A[S_0, X]$	store $\begin{Bmatrix}\alpha^* \\ \alpha\end{Bmatrix}$ in $A[S, X]$
R2	**grant** $\begin{Bmatrix}\alpha^* \\ \alpha\end{Bmatrix}$ **to** S, X	'owner' in $A[S_0, X]$	store $\begin{Bmatrix}\alpha^* \\ \alpha\end{Bmatrix}$ in $A[S, X]$
R3	**delete** α **from** S, X	'control' in $A[S_0, S]$ or 'owner' in $A[S_0, X]$	delete α from $A[S, X]$
R4	$w \leftarrow$ **read** S, X	'control' in $A[S_0, S]$ or 'owner' in $A[S_0, X]$	copy $A[S, X]$ into w
R5	**create object** X	None	add column for X to A; store 'owner' in $A[S_0, X]$
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; execute **create object** S; store 'control' in $A[S, S]$
R8	**destroy subject** S	'owner' in $A[S_0, S]$	delete row for S from A; execute **destroy object** S

Protection Domains: More Useful

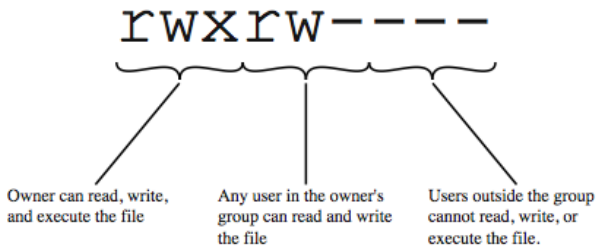
- Set of objects together with access rights to those objects
- More flexibility when associating capabilities with protection domains
- In terms of the access matrix, a row defines a protection domain
- User can spawn processes with a subset of the access rights of the user
- Association between a process and a domain can be static or dynamic
- 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

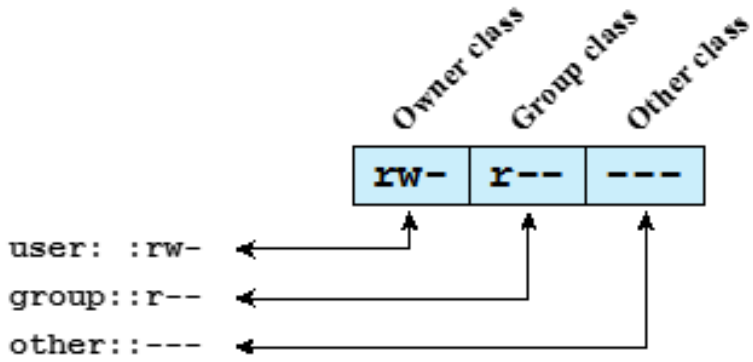
UNIX File Concepts

- UNIX files administered using inodes (index nodes)
- An inode:
  - control structure with key info on file (attributes, permissions, …)
  - on a disk: an inode table for all files
  - when a file is opened, its inode is brought to RAM
- Directories form a hierarchical tree
  - may contain files or other directories
  - are a file of names and inode numbers

UNIX File Access Control

- Unique user identification number (user ID)
- Member of a primary group identified by a group ID
- 12 protection bits
  - 9 specify read, write, and execute permission for the owner of the file, members of the group and all other users
  - 2 speficiy SetID, SetGID
  - 1 is the sticky bit (only owner can remove, delete, …, a directory)
- The owner ID, group ID, and protection bits are part of the file's inode





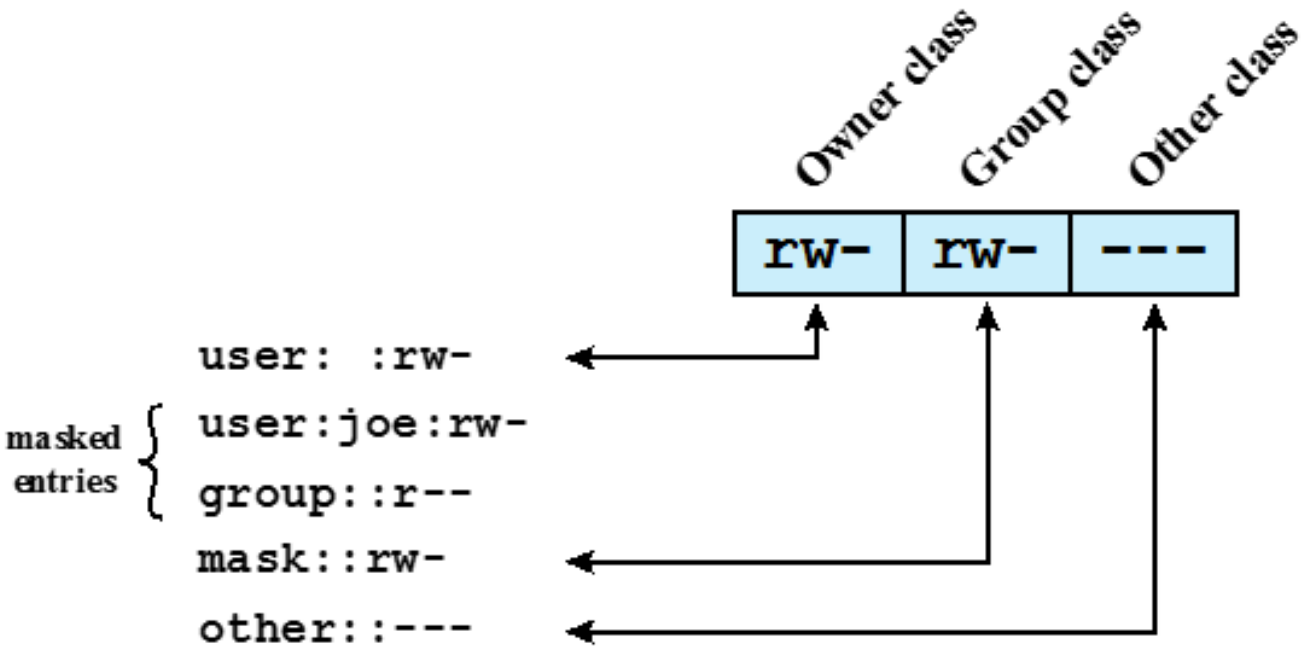
UNIX File Access Control

- “set user ID”(SetUID) or “set group ID”(SetGID)
  - system temporarily uses rights of the file owner/group in addition to the real user’s rights when making access control decisions
  - enables privileged programs to access files/resources not generally accessible
- Sticky bit
  - on directory limits rename/move/delete to owner
- Superuser
  - is exempt from usual access control restrictions

UNIX Access Control Lists

- Modern UNIX systems support ACLs
- Can specify any number of additional users/groups and associated rwx permissions
- When access is required
  - select most appropriate ACL
    - owner, named users, owning/named groups, others
  - check if have sufficient permissions for access

UNIX extended access control list



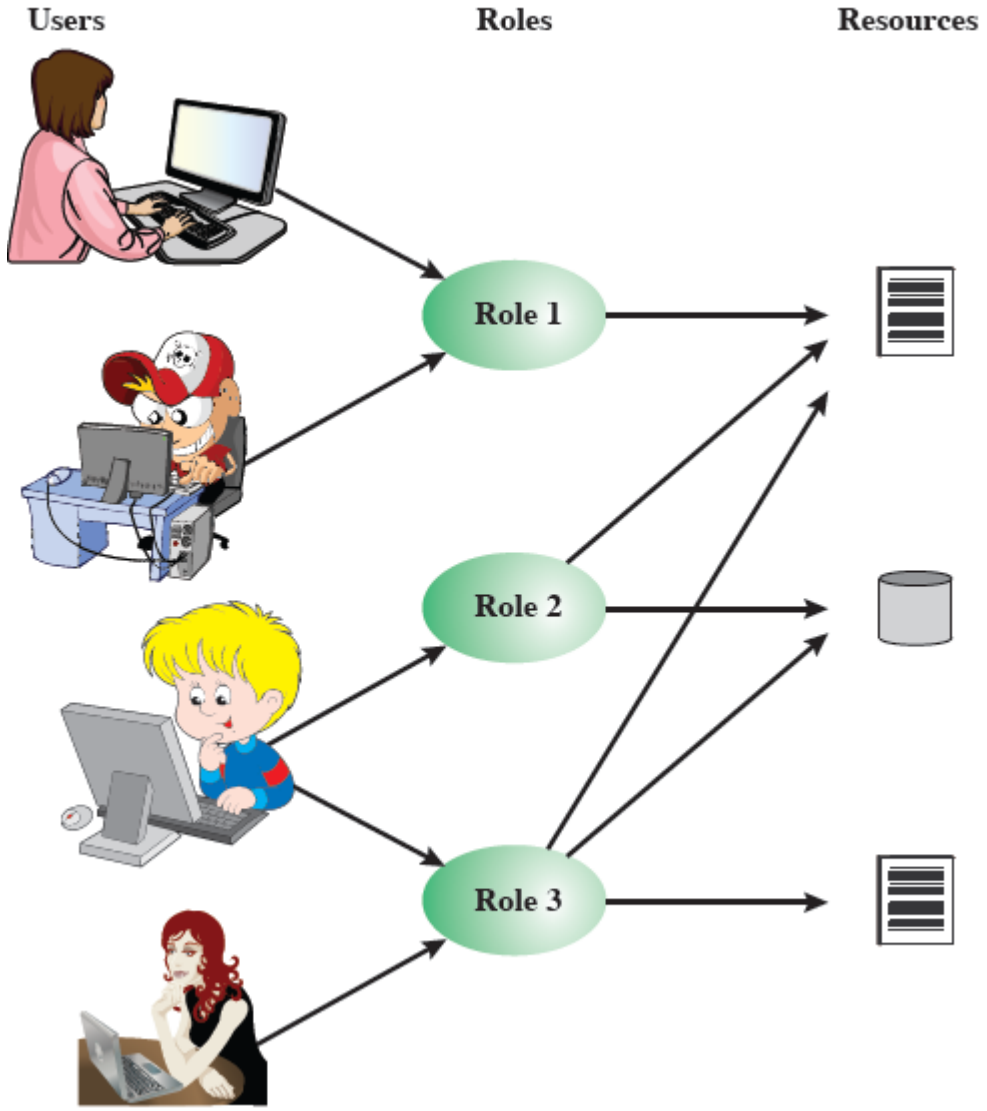
(b) Extended access control list

Role-Based Access Control

Access based on ‘role’, not identity

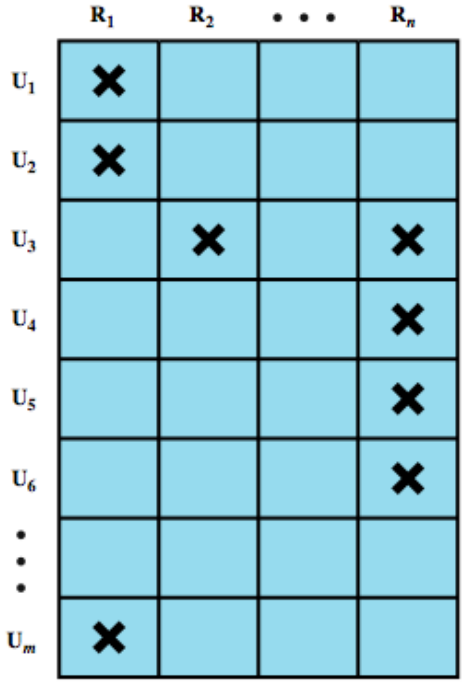
Many-to-many relationship between users and roles

Roles often static



Role-Based Access Control

Role-users and roles-object access matrix



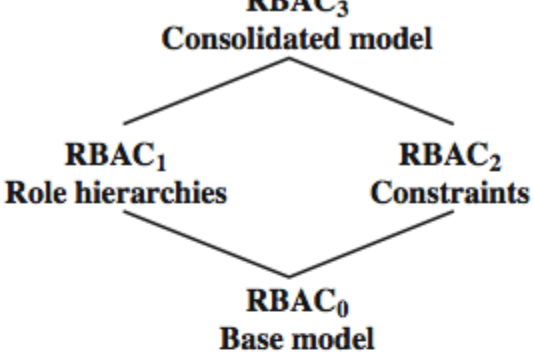
OBJECTS

ROLES	R1	R2	Rn	F1	F1	P1	P2	D1	D2
R1	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
R2		control		write *	execute			owner	seek *
• • •									
Rn			control		write	stop			

General RBAC, Variations

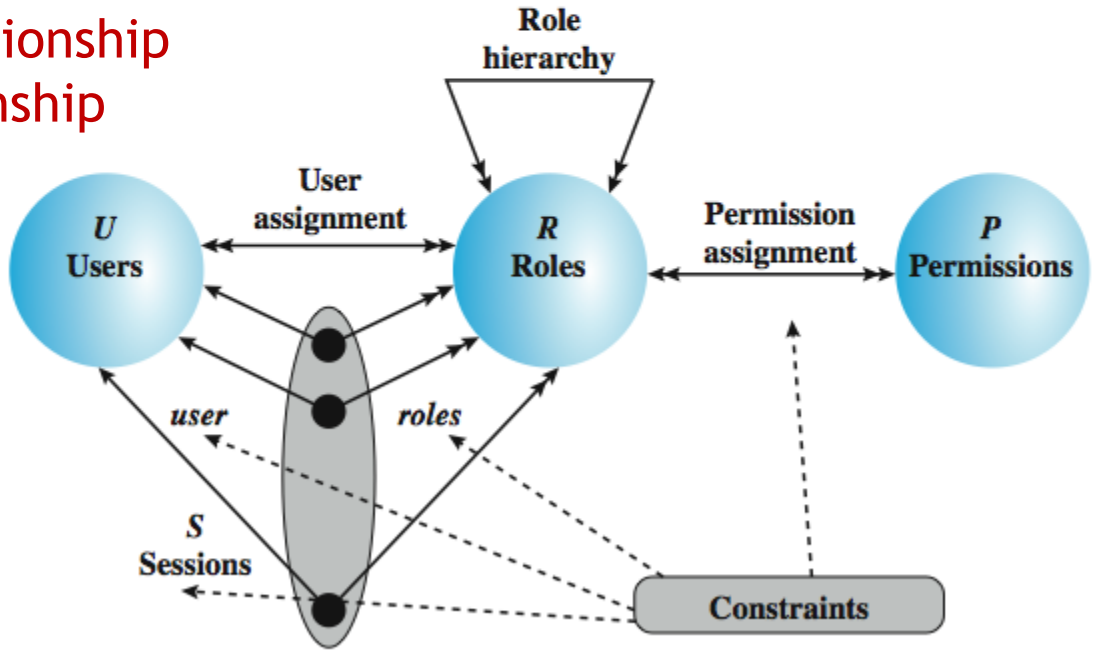
- A family of RBAC with four models
  1. RBAC0: min functionality
  2. RBAC1: RBAC0 plus role (permission) inheritance
  3. RBAC2: RBAC0 plus constraints (restrictions)
  4. RBAC3: RBAC0 plus all of the above
- RBAC0 entities
  - User: an individual (with UID) with access to system
  - Role: a named job function (tells authority level)
  - Permission: equivalent to access rights
  - Session: a mapping between a user and set of roles to which a user is assigned

Role-Based Access Control



(a) Relationship among RBAC models

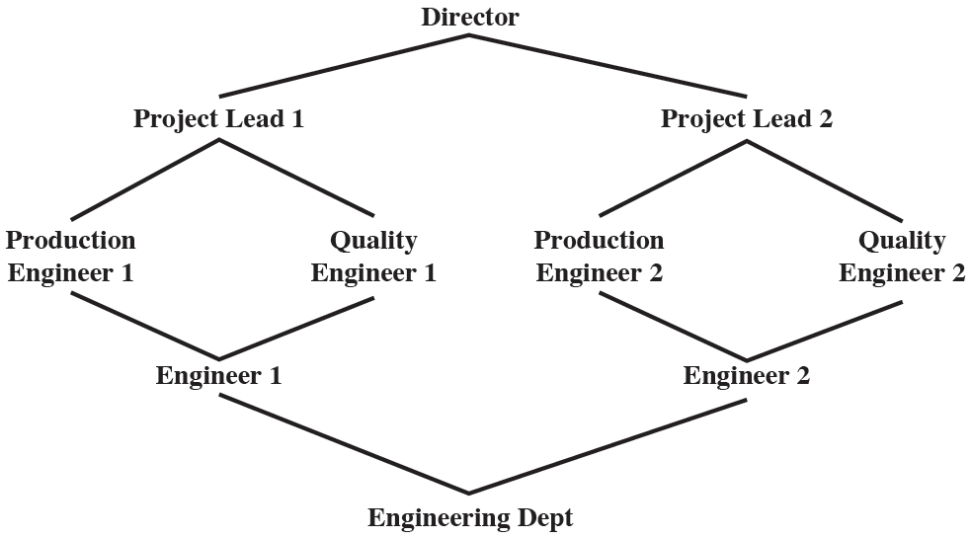
Double arrow: 'many' relationship
Single arrow: 'one' relationship



(b) RBAC models

Example of role hierarchy

- Director has most privileges
- Each role inherits all privileges from lower roles
- A role can inherit from multiple roles
- Additional privileges can be assigned to a role



Constraints

- A condition (restriction) on a role or between roles
  - **Mutually exclusive**
    - role sets such that a user can be assigned to only one of the role in the set
    - Any permission can be granted to only one role in the set
  - **Cardinality**: set a maximum number (of users) wrt a role (e.g., a department chair role)
  - **Prerequisite role**: a user can be assigned a role only if that user already has been assigned to some other role

Attribute-based access control

- Fairly recent
- Define authorizations that express conditions on properties of both the resource and the subject
  - Each resource has an attribute (e.g., the subject that created it)
  - A single rule states ownership privileges for the creators
- Strength: its flexibility and expressive power
- Considerable interest in applying the model to cloud services

Types of attributes

- Subject attributes
- Object attributes
- Environment 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
  - Name
  - Organization
  - Job title

Object attribute

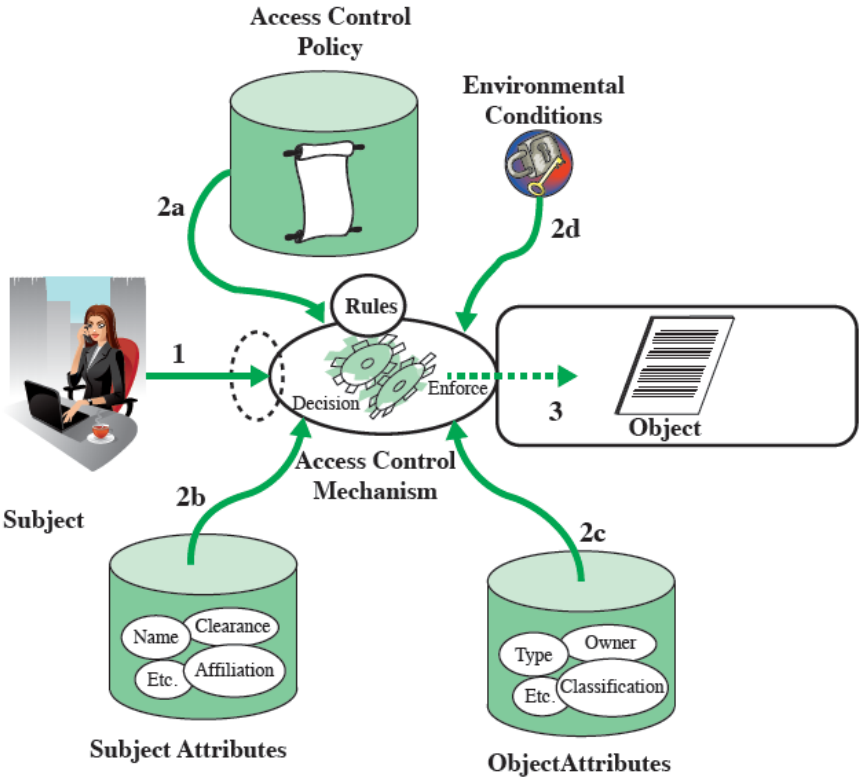
- An object (or resource) is a passive information system-related entity containing or receiving information
- Objects have attributes that can be leveraged to make access control decisions
  - Title
  - Author
  - Date

Environment attributes

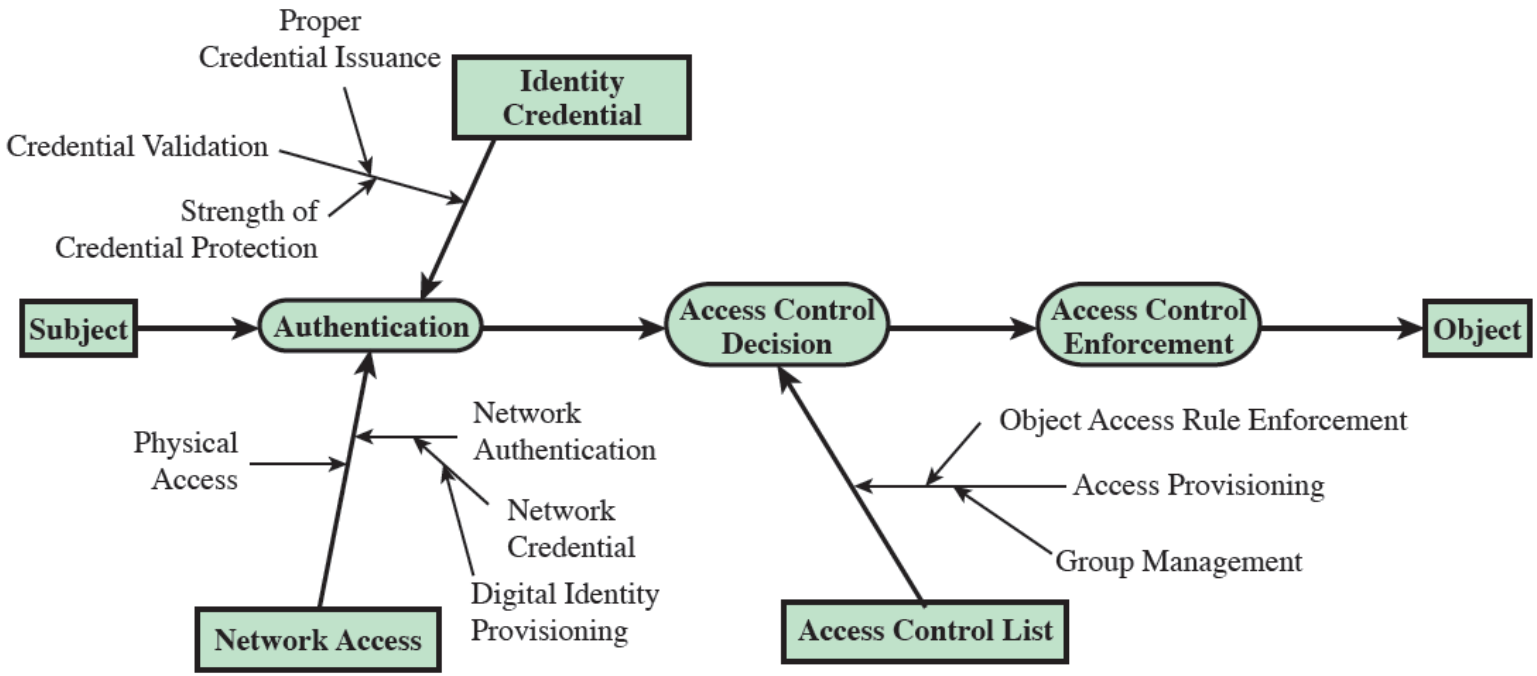
- Describe the operational, technical, and even situational environment or context in which the information access occurs
  - Current date
  - Current virus/hacker activities
  - Network security level
  - *Not associated with a resource or subject*
- These attributes have so far been largely ignored in most access control policies

Sample ABAC scenario

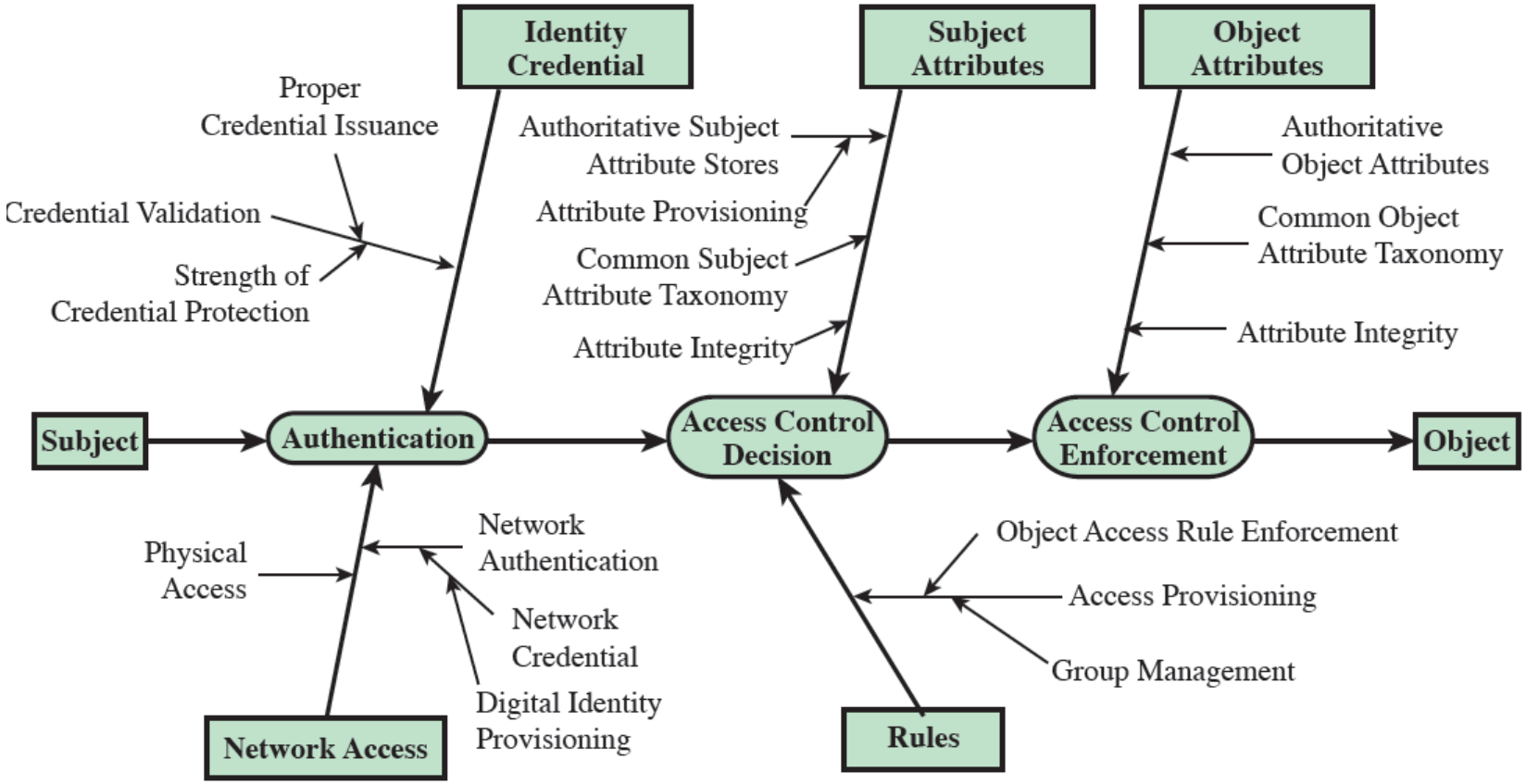
1. A subject requests access to an object
2. AC is governed by a set of rules (2a): assesses the attr of subject (2b), object (2c) and env (2d)
3. AC grants subject access to object if authorized



ACL vs ABAC trust relationships



ACL vs ABAC trust relationships



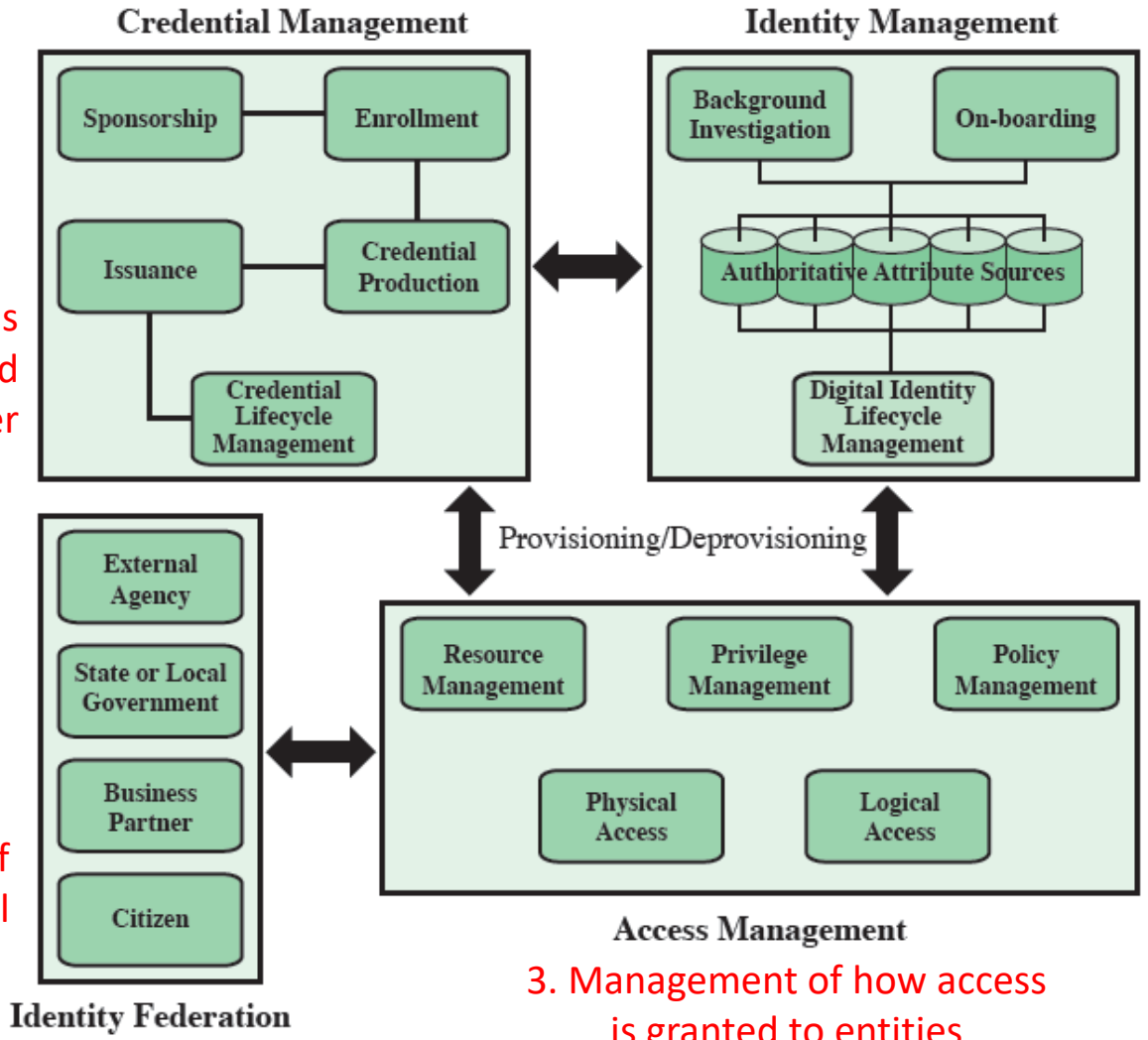
Identity, Credential, and Access Management (ICAM)

- A comprehensive approach to managing and implementing digital identities, credentials, and access control
- Developed by the U.S. government
- Designed to create trusted digital identity representations of individuals and nonperson entities (NPEs)
- A credential is an object or data structure that authoritatively binds an identity to a token possessed and controlled by a subscriber
- Use the credentials to provide authorized access to an agency's resources

ICAM

1. Connects digital identity to individuals

2. Data structures that binds a token possessed by a subscriber



4. Identity verification of individuals from external organizations

3. Management of how access is granted to entities

Trust frameworks

- Skip

Case study: RBAC system for a bank

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: RBAC system for a bank

- b has more access than A (strict ordering)
- Inheritance makes tables simpler

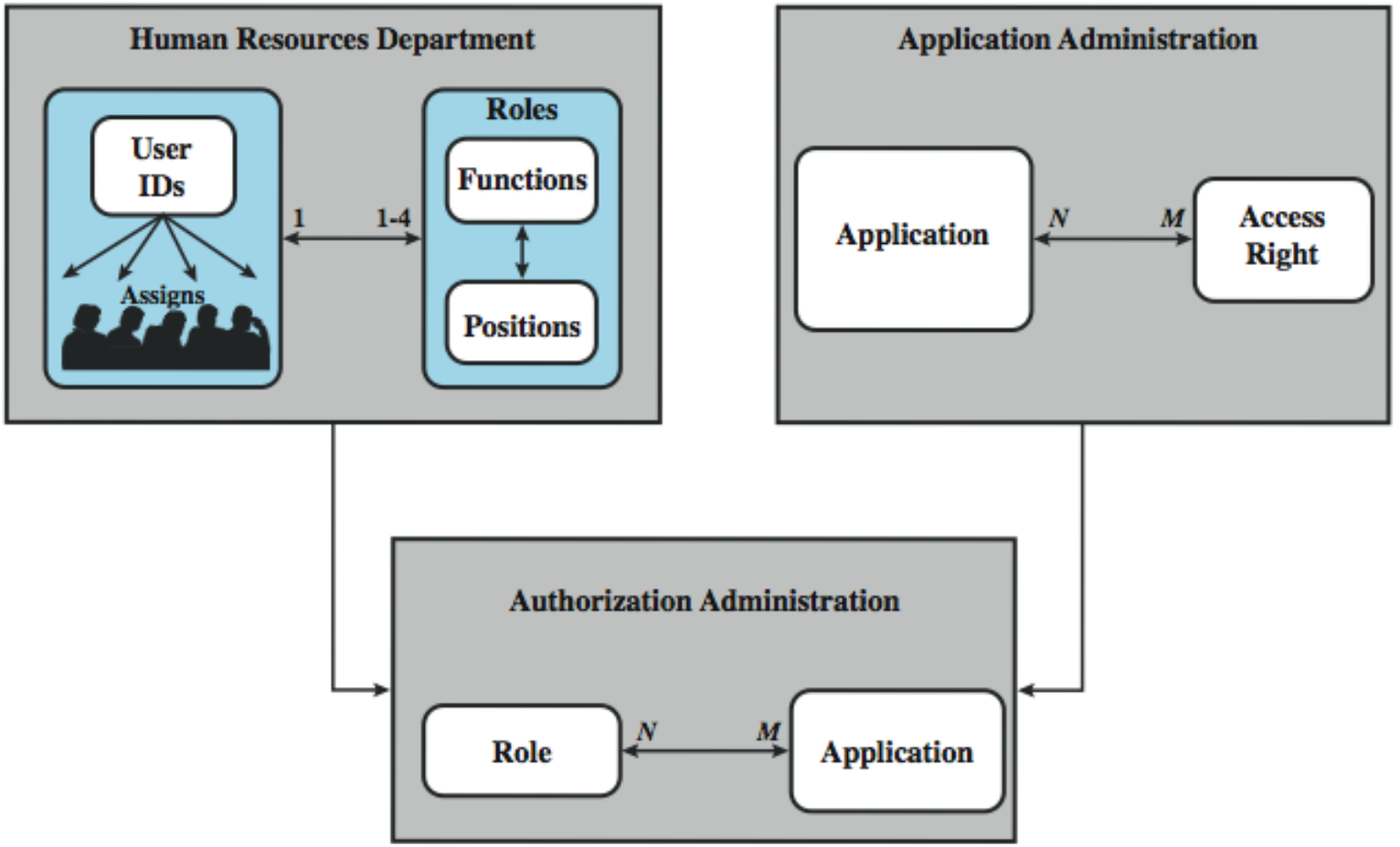
(b) Permission Assignments

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
•••	•••	•••

(c) 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
•••	•••	•••

Case study: RBAC system for a bank



Summary

- introduced access control principles
  - subjects, objects, access rights
- discretionary access controls
  - access matrix, access control lists (ACLs), capability tickets
  - UNIX traditional and ACL mechanisms
- role-based access control
- case study