Database Security

Access control

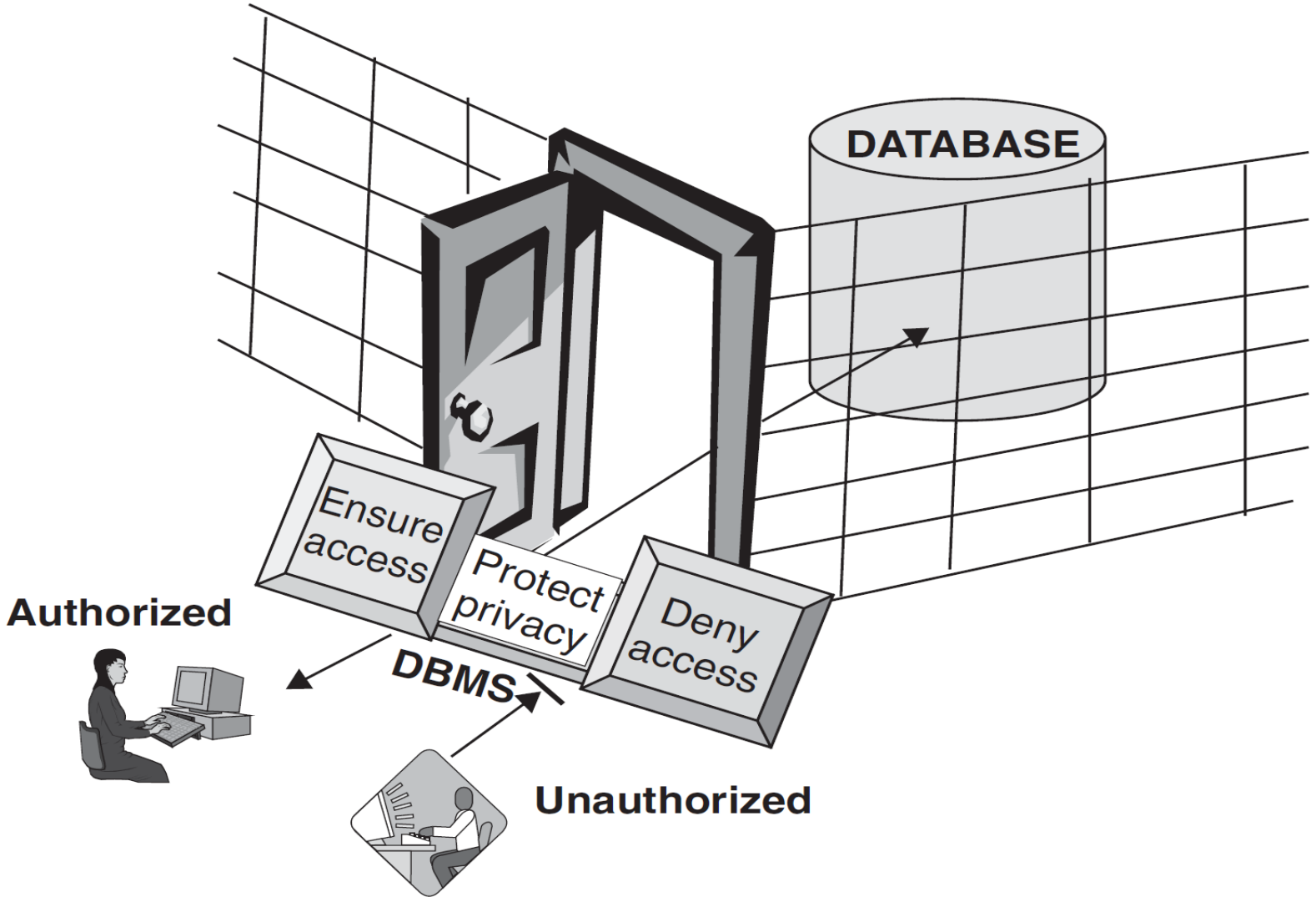
Notes_Set2

Authentication and DAC

Security Objectives

- **Confidentiality**
  - Confidentiality is the **concealment of information** or resources
  - **Resource hiding** is another important aspect of confidentiality
  - Prevent/detect/deter improper disclosure of information
- **Integrity**
  - Integrity refers to preventing **improper or unauthorized change**.
  - Prevent/detect/deter improper modification of information (includes both the correctness and the trustworthiness of the data)
  - Integrity includes data integrity (the **content** of the information) and origin integrity (the **source** of the data).
- **Availability**
  - Availability refers to the ability to use the information or resource desired.
  - Prevent/detect/deter **improper denial of access** to services
  - Attempts to block availability, called ***denial of service attacks***
- **Database privacy concerns the protection of information about individuals that is stored in a database**

Database Security System



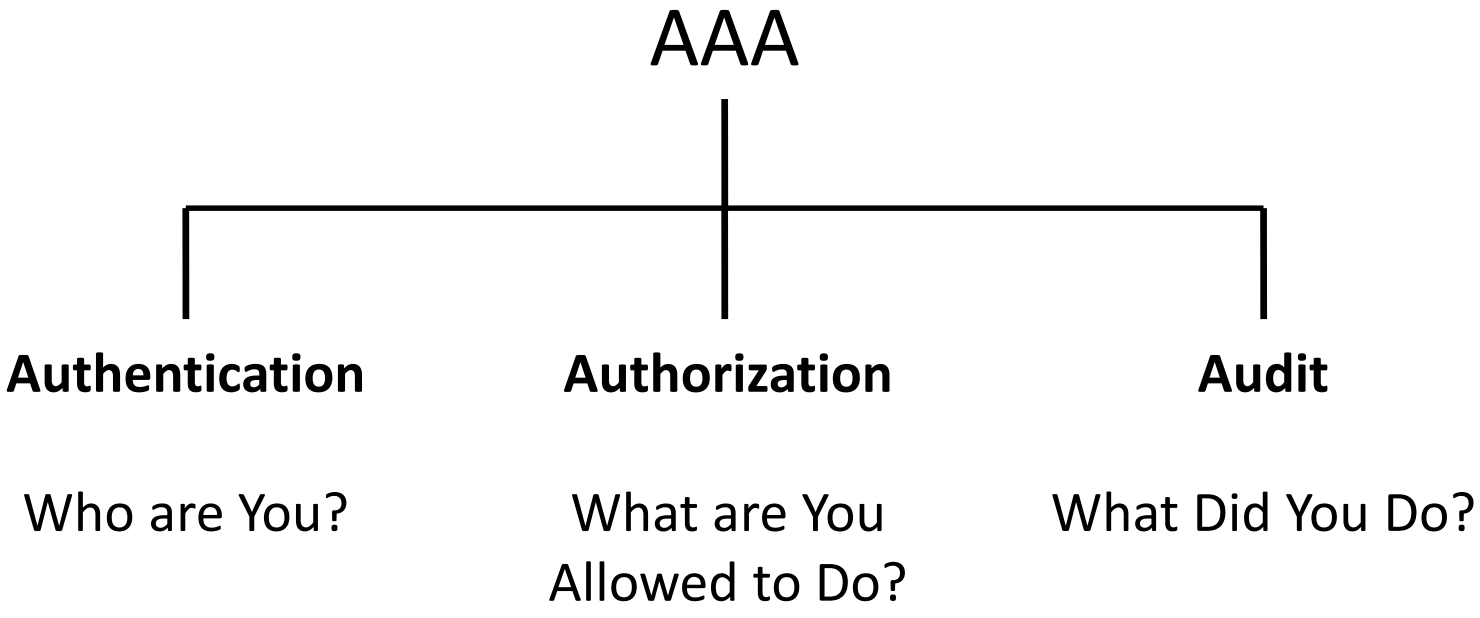
Database Security –How?

- Confidentiality is enforced by the **Access Control mechanism**
- Integrity is enforced by the **Access Control mechanism** and by the **Semantic Integrity Constraints** specified during schema definition
- Availability is enforced by the **Recovery and Concurrency Control** mechanisms
- Additional mechanisms:
  - **User authentication-** To verify the identity of subjects wishing to access the data
  - **Data authentication** - To ensure that data authenticity - it is supported by data signature mechanisms
  - **Cryptography** - To ensure that data privacy - To protect data when being transmitted across systems and when being stored on secondary storage

Data Security: Life-cycle

- It consists of
  - First defining a ***security policy***
  - Then choosing some ***mechanism*** **to enforce the policy**
  - Finally **providing *assurance*** that both the mechanism and the policy are sound
- **Achieving Security**
  - Policy
    - **What to protect?**
  - Mechanism
    - **How to protect?**
  - Assurance
    - **How good is the protection?**

Authentication, Authorization, Audit(AAA)

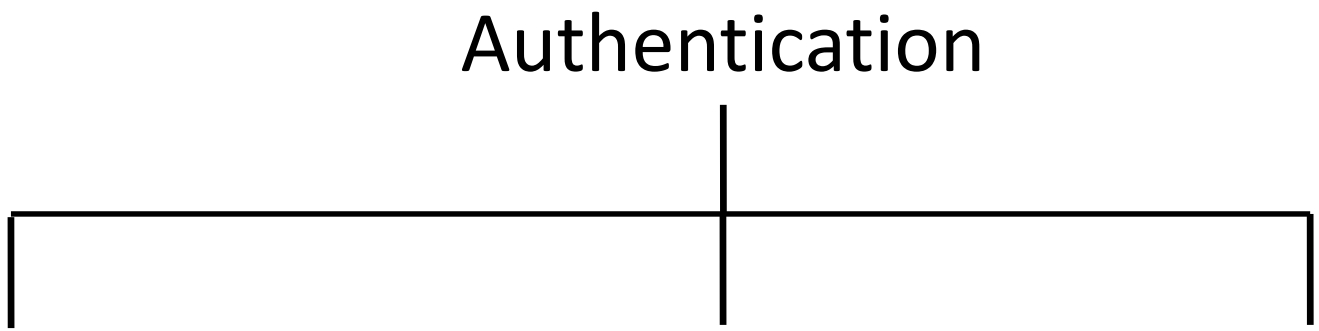


- **The two fundamental aspects of security are authentication and authorization**.
- After you enter your credentials to log in to your computer or sign in to an app or software, the device or application undertakes authentication to determine your level of authorization.
- Authorization may include what accounts you can use, what resources you have access to, and what functions you are permitted to carry out.

Authentication

- Allows an entity (a user or a system) to prove its identity within a context
- Typically, the entity whose identity is verified reveals knowledge of some secret S to the verifier
- Authentication generally follows **identification of the entity**
- **Strong authentication**: The entity reveals *knowledge of S* to the verifier *without* revealing S to the verifier
- **Authentication Information:** Must be securely maintained by the system
- **Authentication mechanism**: Verify the distinguishing characteristics
- **Access control mechanism:** Grant privileges upon successful authentication

Authentication Techniques



Something you know
(What the user knows)

- password
- Personal information
- “secret” questions

Something you have
(What the user possesses)

- smartphone
- registered device
- Physical key
- ticket
- Passport
- token
- smart card

Something you are
(What the user is (biometrics))

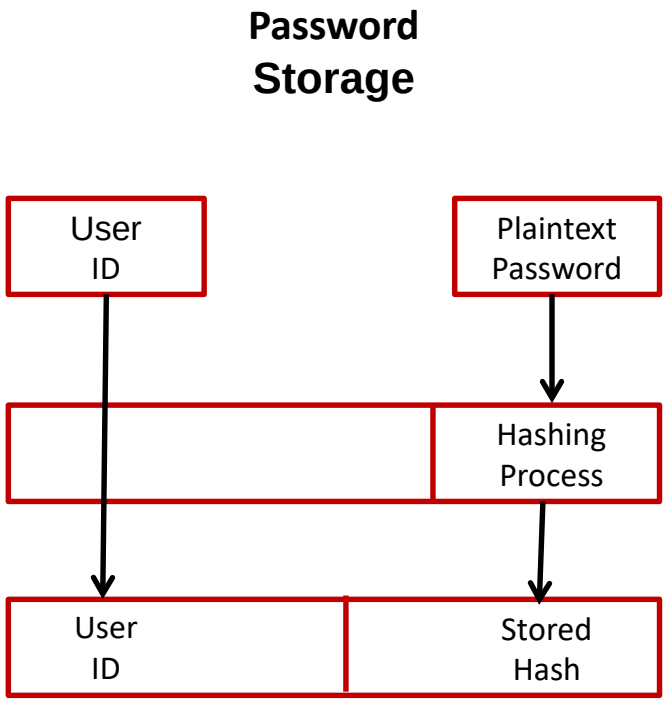
- Fingerprints, voiceprints
- Iris, Retina Scan
- keyboard dynamics
- signature dynamics
- Typing style

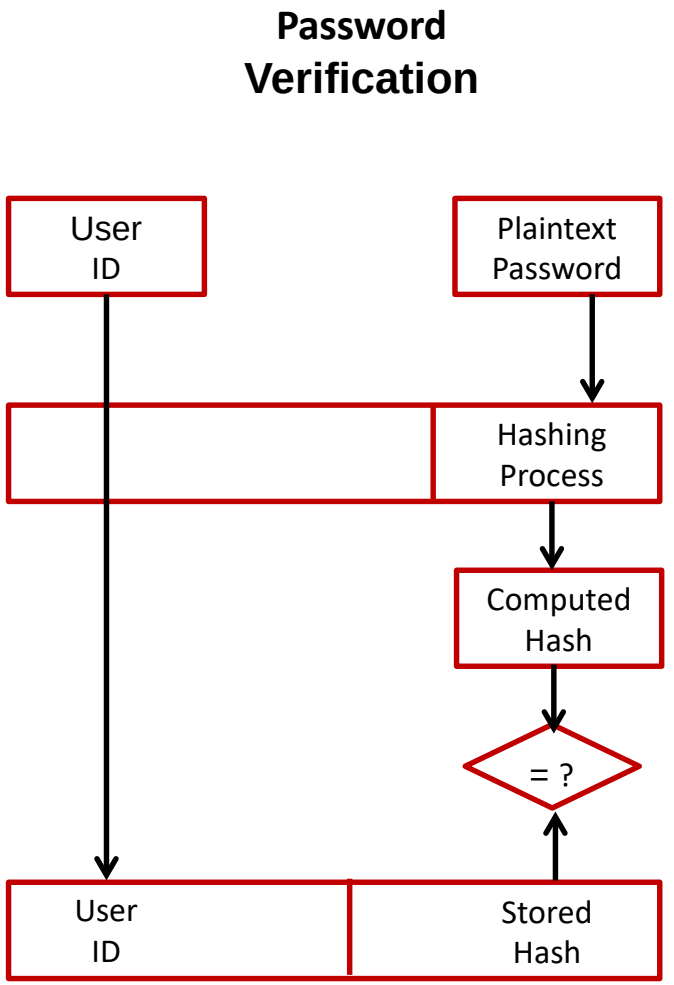
Single Factor → **Multi -Factor**

Passwords

- Commonly used method for authentication
- For each user, system stores (user name, F(password)), where F is some transformation (e.g., one-way hash) in a password file
- When user enters the password, system computes F(password); match provides proof of identity
- **Vulnerabilities of Passwords**
  - Easy to guess or snoop
  - No control on sharing
  - Visible if unencrypted in distributed and network environment
  - Susceptible for replay attacks if encrypted naively
- **Attacks on passwords**: Guessing attack/dictionary attack, Social Engineering, Sniffing, Trojan login
  - Guessing attack /dictionary attack : **Easy to detect (failed logins) and block**
- **Password advantage** : Easy to modify compromised password

Password Storage and Verification





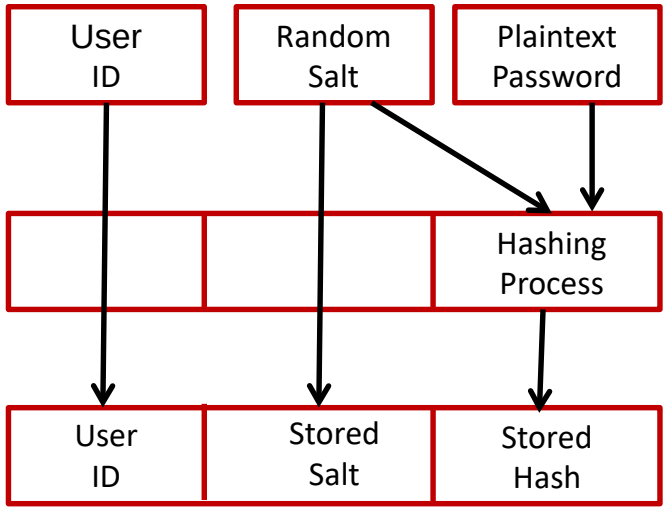
Loss of stored hashes = Attack by single dictionary

Password Storage and Verification

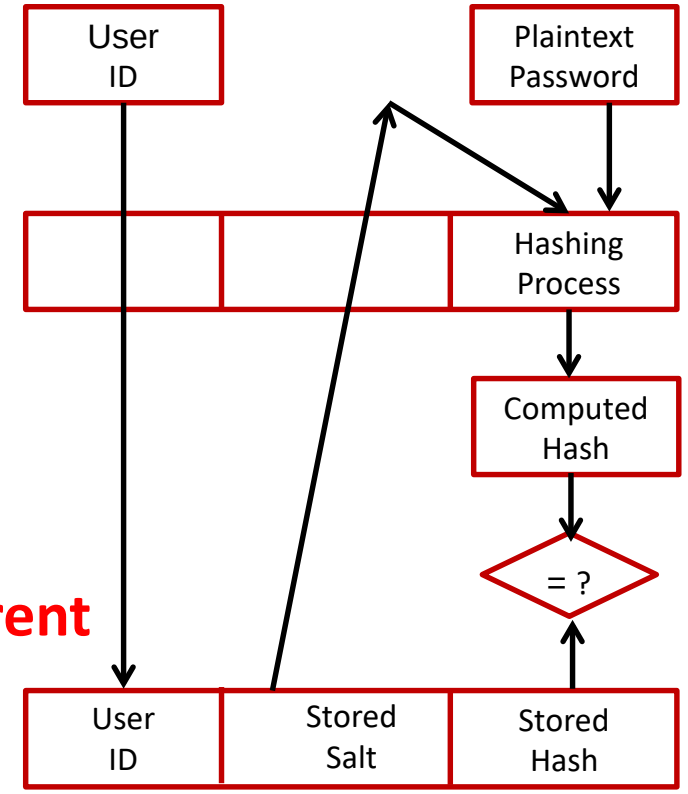
Password Salt

- Used to make dictionary attack more difficult
- Salt is a 12 bit number between 0 and 4095 and derived from the system clock and the process identifier
- Compute F(password+salt); both salt and F(password+salt) are stored in the password table
- User: gives password, system finds salt and computes F(password+salt) and check for match

Password Storage



Password Verification

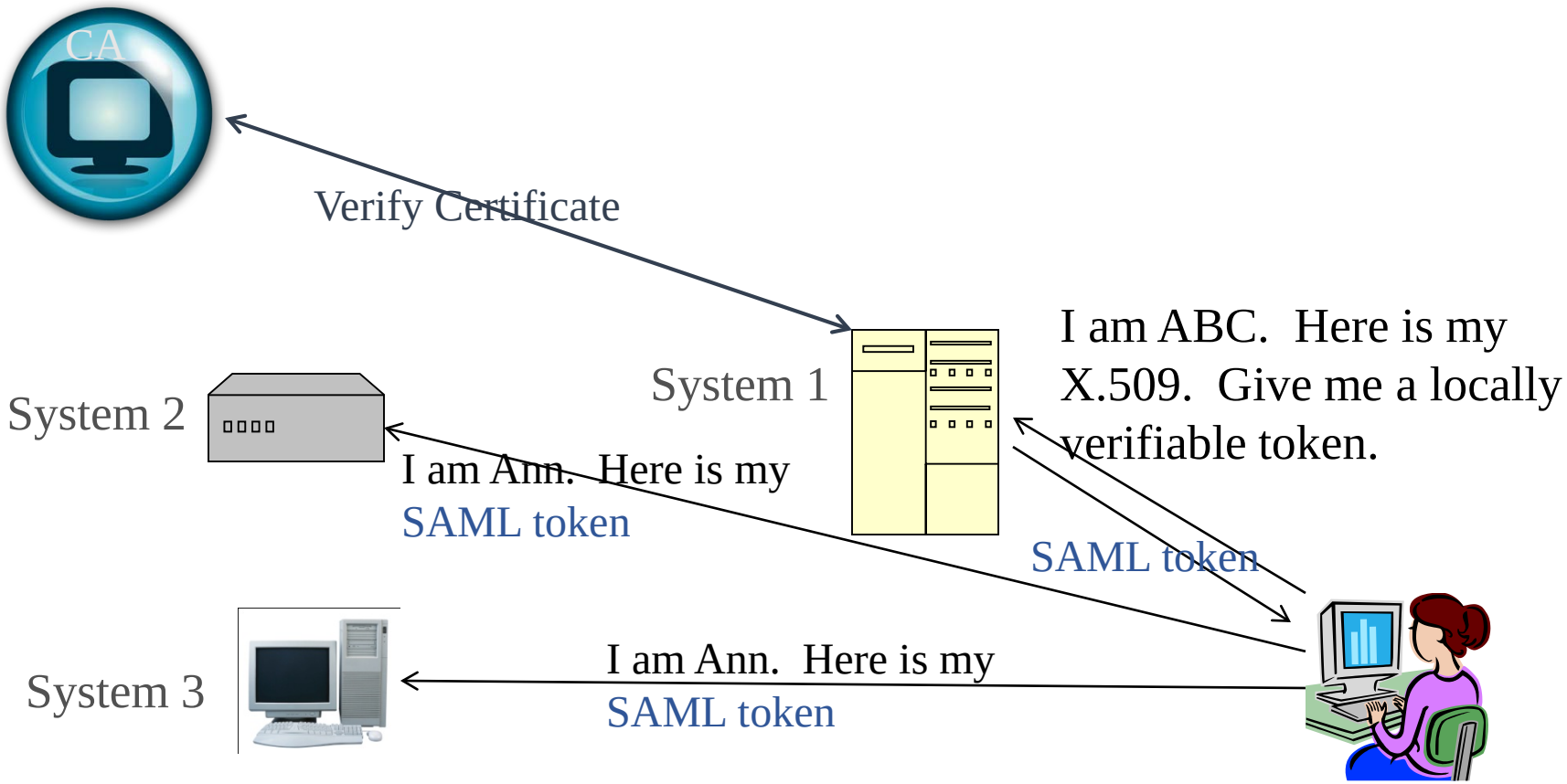


Loss of stored hashes = Attack by different dictionary for each salt value

Identity Management cont.

- Need verifiable proof of identity – without being authenticated during every single interaction
- Digital certificate: links identity and public key together
  - A user can prove his/her identity by signing the messages with his/her private key
- Most common digital certificate: X.509
- Rely on PKI and hierarchy of certificate authorities
- **Problem with X.509** : Large file, Long duration → needs validation of certificate for revocation

Single Sign On



Access Control

- **Access control System**
  - Ensures that all *direct accesses* to object are authorized
  - *Regulates the operations* that can be executed on data and resources to be protected
  - *Control operations executed by subjects* in order to prevent actions that could damage data and resources
  - *Protects against accidental and malicious threats* by regulating the *reading, writing and execution* of data and programs
- Need:
  - Proper *user identification* and *authentication*
  - Information specifying the *access rights is protected* form modification

Access Control

- Database security rests on controlling access to the database system.
  - **Controlling physical access**.
  - **Controlling access through the DBMS**.
- Two primary dimensions of access control.
  - **One dimension of access control deals with levels of data access**.
    - A single user or a category of users may be granted access privileges to database objects at various levels of detail.
  - **Another dimension of access control refers to the modes or types of access granted to a single user or to a category of users**.
    - How do you grant access privileges to a single user or user category?

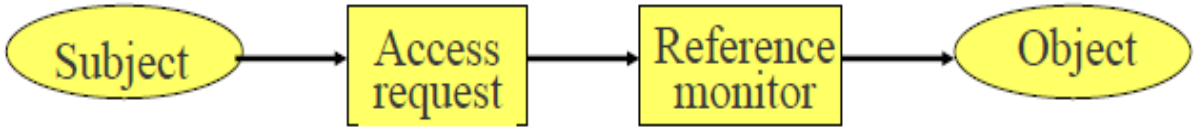
Access Control Models

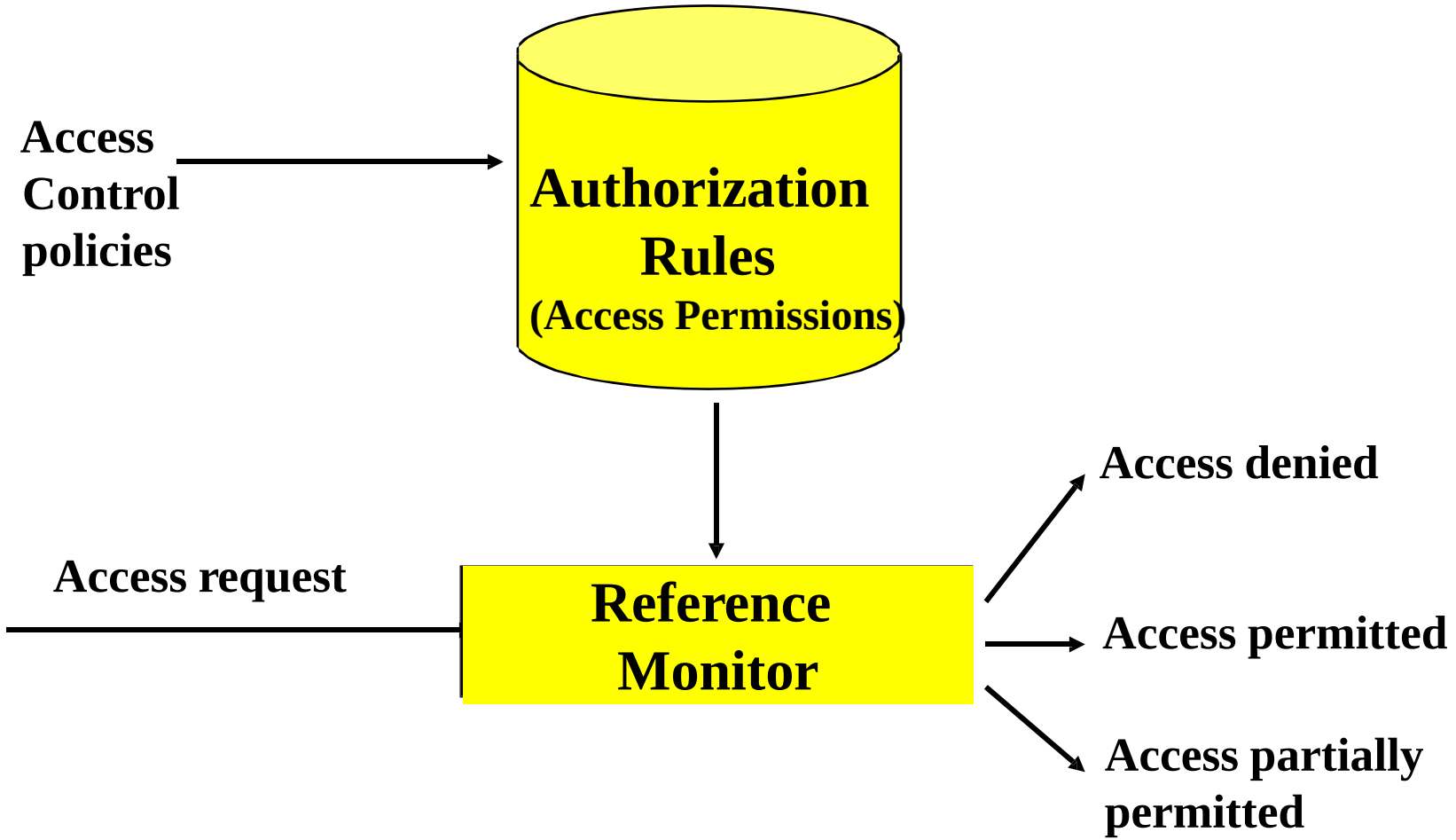
- Many Access Control Models: The granting of privileges or rights to individual users
  - **Discretionary Access control**
  - **Mandatory Access control**
  - **Role Based Access control**
  - **Attribute Based Access Control**
- **DAC** models enforce access control based on user identities, object ownership and permission delegation. The owner of an object may delegate the permission of the object to another user.
- **MAC** models govern access based on the sensitivity level of subjects and objects. A subject may read an object if the security level of the subject is higher than that of the object.
- **RBAC** models enforce access control based on User roles. Accessibility is determined by the permissions and users assigned to roles. **Depending on your role in the organization, you will have different access permissions.** RBAC controls broad access across an organization.
- ABAC Models enforce access control based on the attributes of the subject, resource, action, and environment. ABAC takes a fine-grain approach.

Security Policies and Authorizations

- Policies deal with **defining what is authorized and who can grant authorizations**
- Policies are used **like requirements**; they are the starting point in the development of any system that has security features
- Adopted security policies mainly depend from organizational requirements, such as legal requirements, regulatory requirements, user requirements
- **Two main categories**:
  - Access control policies : Most access control policies are formulated in terms of *subjects*, *objects*, *and privileges*
  - Administration policies : Anything that holds data, such as relations, directories, inter-process messages, network packets, I/O devices, or physical media

Access Control



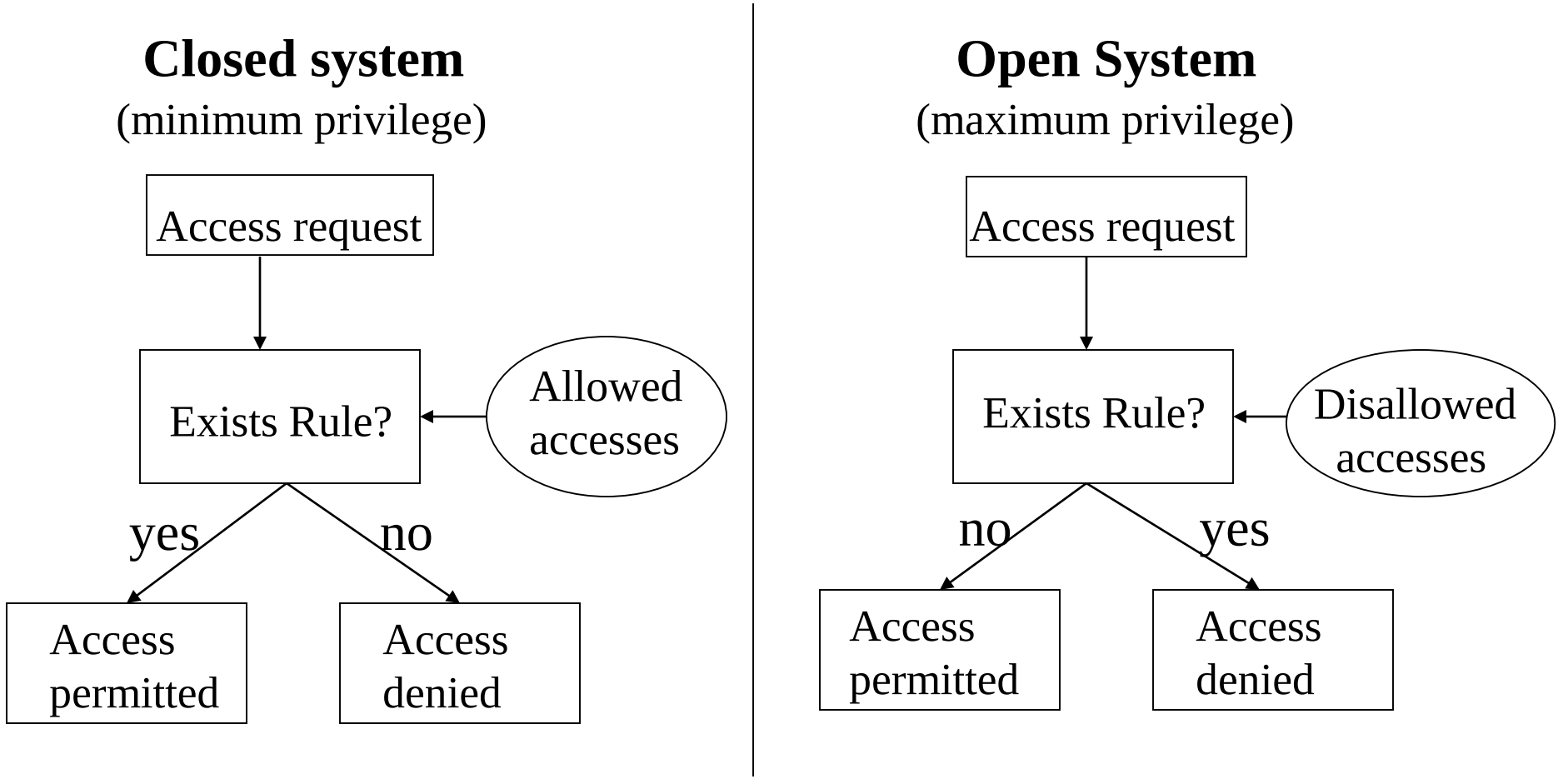


- The **Security Policies** are implemented by mapping them into a set of authorizations
- **Authorizations** thus establish the operations and rights that subjects can exercise on the protected objects
- The **Reference Monitor** is a control mechanism - it has the task of determining whether a given subject is authorized to access the data

Access Control

- **Access control components**:
  - *Access control policy*: specifies the authorized accesses of a system
  - *Access control mechanism*: implements and enforces the policy
- **Separation of components allows to:**
  - Define access requirements independently from implementation
  - Compare different policies
  - Implement mechanisms that can enforce a wide range of policies

Closed VS Open Systems



AC Models -DBMS vs OS

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

- **SQL system tables**
  - The DBMS will maintain tables to record all security information.
  - An SQL database is created and managed by the use of system tables.
  - These comprise a relational database using the same structure and access mechanism as the main database.
  - The examples are from Oracle database.

Command	Description
ALL_ARGUMENTS	Arguments in object accessible to the user
ALL_CATALOG	All tables, views, synonyms, sequences accessible to the user
ALL_COL_COMMENTS	Comments on columns of accessible tables and views
ALL_CONSTRAINTS	Constraint definitions on accessible tables
ALL_CONS_COLUMNS	Information about accessible columns in constraint definitions
ALL_DB_LINKS	Database links accessible to the user
ALL_ERRORS	Current errors on stored objects that user is allowed to create
ALL_INDEXES	Descriptions of indexes on tables accessible to the user
ALL_IND_COLUMNS	COLUMNs comprising INDEXes on accessible TABLES
ALL_LOBS	Description of LOBs contained in tables accessible to the user
ALL_OBJECTS	Objects accessible to the user
ALL_OBJECT_TABLES	Description of all object tables accessible to the user
ALL_SEQUENCES	Description of SEQUENCEs accessible to the user
ALL_SNAPSHOTS	Snapshots the user can access
ALL_SOURCE	Current source on stored objects that user is allowed to create
ALL_SYNONYMS	All synonyms accessible to the user
ALL_TABLES	Description of relational tables accessible to the user
ALL_TAB_COLUMNS	Columns of user's tables, views and clusters
ALL_TAB_COL_STATISTICS	Columns of user's tables, views and clusters
ALL_TAB_COMMENTS	Comments on tables and views accessible to the user
ALL_TRIGGERS	Triggers accessible to the current user
ALL_TRIGGER_COLS	Column usage in user's triggers or in triggers on user's tables
ALL_TYPES	Description of types accessible to the user
ALL_UPDATABLE_COLUMNS	Description of all updatable columns
ALL_USERS	Information about all users of the database
ALL_VIEWS	Description of views accessible to the user
DATABASE_COMPATIBLE_LEVEL	Database compatible parameter set via init.ora
DBA_DB_LINKS	All database links in the database
DBA_ERRORS	Current errors on all stored objects in the database
DBA_OBJECTS	All objects in the database
DBA_ROLES	All Roles which exist in the database
DBA_ROLE_PRIVS	Roles granted to users and roles
DBA_SOURCE	Source of all stored objects in the database
DBA_TABLESPACES	Description of all tablespaces
DBA_TAB_PRIVS	All grants on objects in the database
DBA_TRIGGERS	All triggers in the database
DBA_TS_QUOTAS	Tablespace quotas for all users
DBA_USERS	Information about all users of the database
DBA_VIEWS	Description of all views in the database
DICTIONARY	Description of data dictionary tables and views
DICT_COLUMNS	Description of columns in data dictionary tables and views
GLOBAL_NAME	global database name
NLS_DATABASE_PARAMETERS	Permanent NLS parameters of the database
NLS_INSTANCE_PARAMETERS	NLS parameters of the instance
NLS_SESSION_PARAMETERS	NLS parameters of the user session
PRODUCT_COMPONENT_VERSION	version and status information for component products
ROLE_TAB_PRIVS	Table privileges granted to roles
SESSION_PRIVS	Privileges which the user currently has set
SESSION_ROLES	Roles which the user currently has enabled.
SYSTEM_PRIVILEGE_MAP	Description table for privilege type codes. Maps privilege type numbers to type names
TABLE_PRIVILEGES	Grants on objects for which the user is the grantor, grantee, owner, or an enabled role or PUBLIC is the grantee
TABLE_PRIVILEGE_MAP	Description table for privilege (auditing option) type codes. Maps privilege (auditing option) type numbers to type names

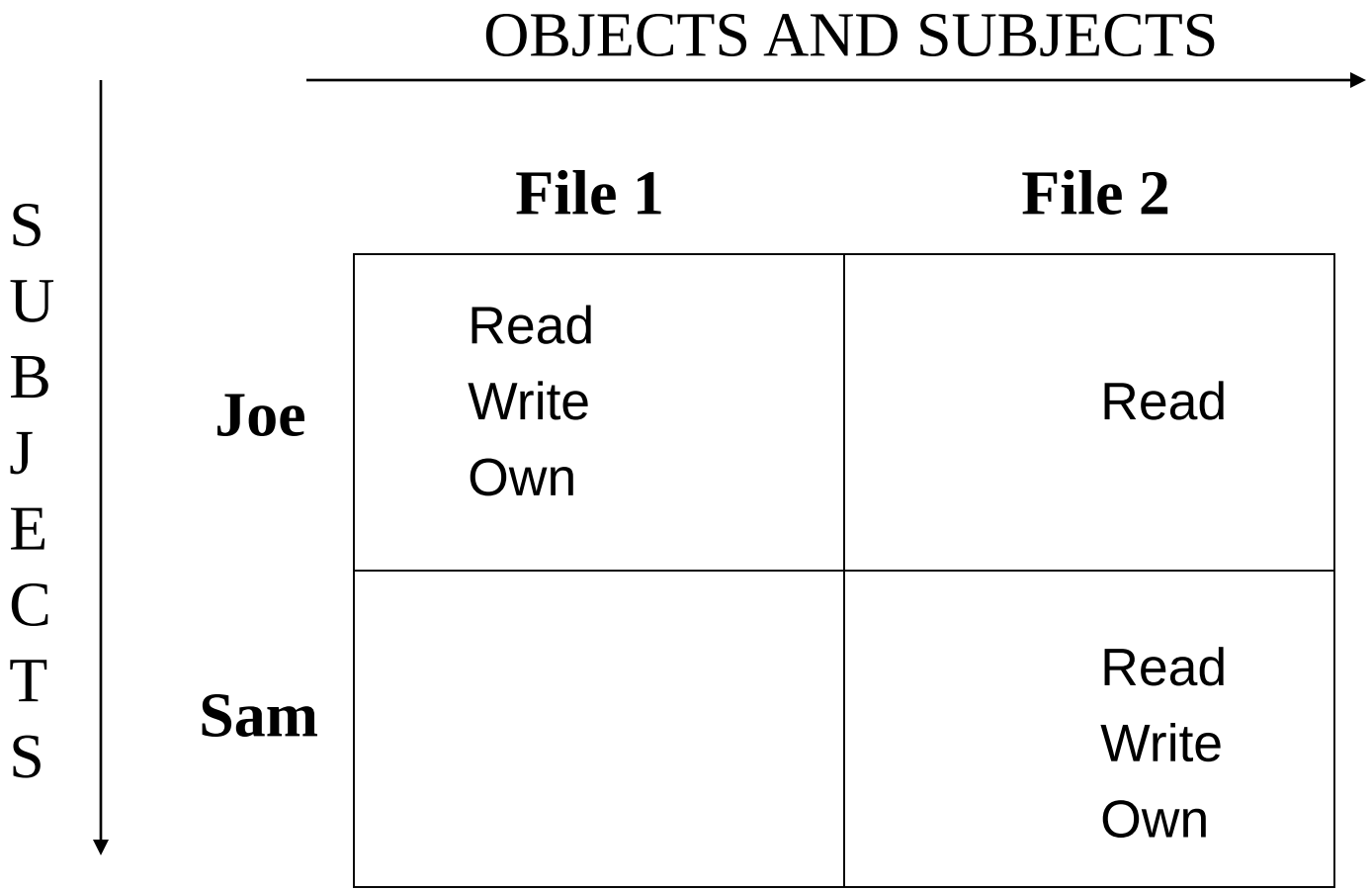
Discretionary Access Control (DAC)

- Access control is based on
  - User's identity and
  - Access control rules
- Most common administration: owner based
  - Users can protect what they own
  - Owner may grant access to others
  - Owner may define the type of access given to others
- A user who created a database object automatically derives all privileges to access the object including the passing on of privileges to other users with regard to that object
- ***Basic Levels :*** There are two basic components or levels for granting or revoking access privileges:
  - **Database Objects** Data item or data element, generally a base table or view
  - **Users** A single user or a group of users identifiable with some authorization identifier
- With these two components, access privileges may be granted as shown in the following general command:
  - **GRANT privileges ON database object TO users**

Authorization Matrix

- We can think of the two levels of *users* and *database objects* forming a matrix for the purpose of granting access privileges.
- Set the users as columns and the database objects as rows.
- Then in the cells formed by the intersection of these columns and rows we can specify the type of privilege granted
- This type of presentation makes it easy to review the access privileges in a database environment.

Access Matrix Model



Authorization Matrix

SUBJECT or USER	DATABASE OBJECT	PRIVILEGE	CONSTRAINT
Rogers	Department record	Modify	None
Miller	Department record	Create	DeptNo NOT EQUAL 101
Chen	Employee record	Select	None
Goldstein	Employee record	Create	None
Jenkins	Department record	Modify	Address ONLY
Gonzales	Employee record	Drop	EmployeePosition = 'Staff'

- Here is an example of a cycle of privileges passed along from Rogers, who is the owner of table EMPLOYEE:
  - *By Rogers* GRANT ALL PRIVILEGES ON EMPLOYEE TO Miller WITH GRANT OPTION
  - *By Miller* GRANT ALL PRIVILEGES ON EMPLOYEE TO Chen WITH GRANT OPTION
  - *By Chen* GRANT ALL PRIVILEGES ON EMPLOYEE TO Williams WITH GRANT OPTION
  - *By Rogers* GRANT SELECT ON EMPLOYEE TO Goldstein WITH GRANT OPTION
  - *By Goldstein* GRANT SELECT ON EMPLOYEE TO Rodriguez WITH GRANT OPTION
  - *By Rogers* REVOKE ALL PRIVILEGES ON EMPLOYEE FROM Miller CASCADE

Access Control

- The access matrix model can be implemented through different mechanisms.
- The straightforward solution exploiting a two-dimensional array is not viable, since A is usually sparse.
- The mechanisms typically adopted are:
  - **Authorization table**: The non empty entries of A are stored in a table with three attributes: user, action, and object.
  - **Access control list (ACL):** The access matrix is stored by column, that is, each object is associated with a list of subjects together with a set of actions they can perform on the object.
  - **Capability**: The access matrix is stored by row, that is, each subject is associated with a list indicating, for each object, the set of actions the subject can perform on it.

Implementation

Access Control List (column) (ACL)

File 1	**File 2**
Joe:Read	Joe:Read
Joe:Write	Sam:Read
Joe:Own	Sam:Write
	Sam:Own

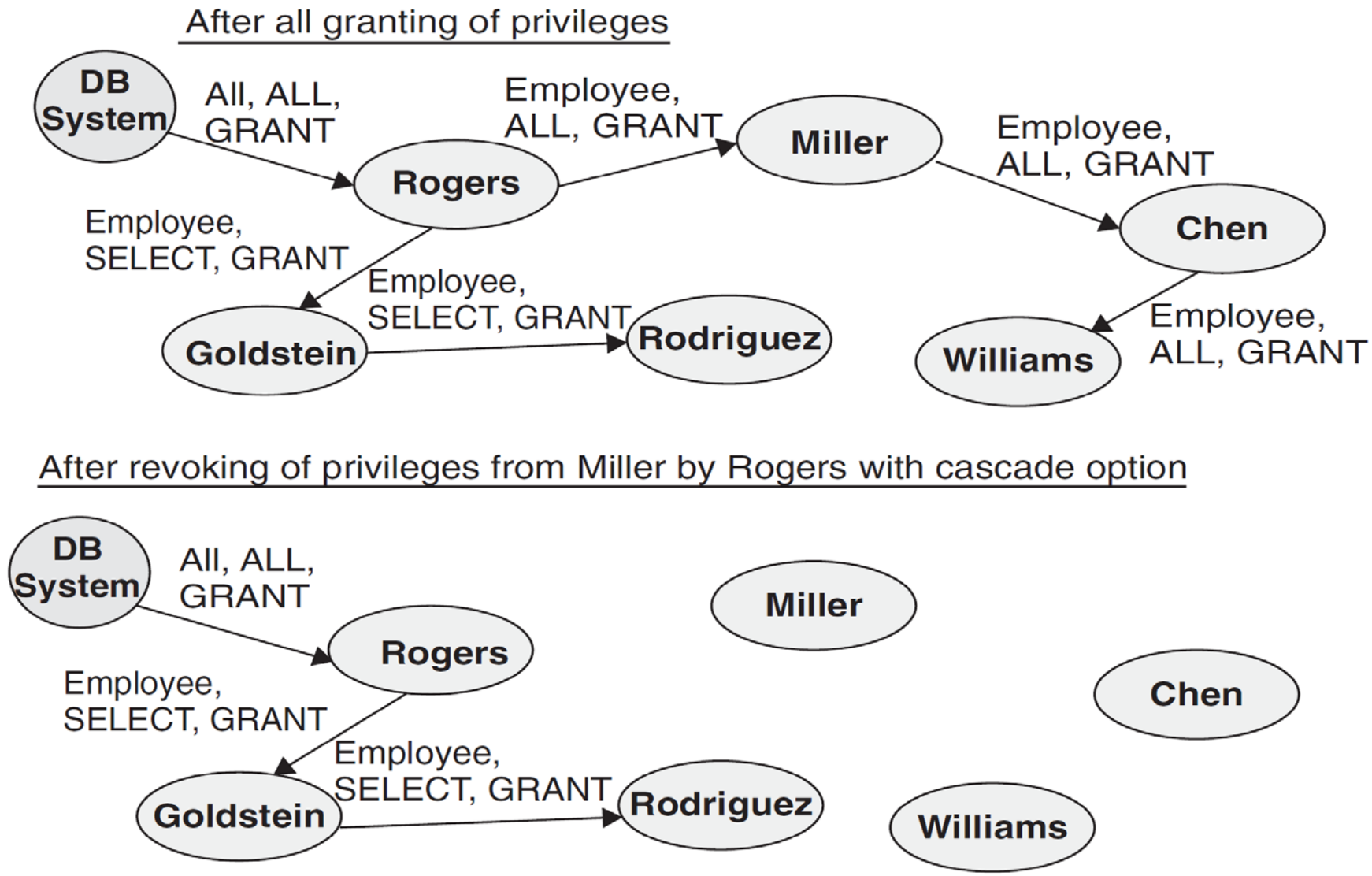
Capability List (row)

Joe: File 1/Read, File 1/Write, File 1/Own, File 2/Read
Sam: File 2/Read, File 2/Write, File 2/Own

Access Control Triples

Subject	Access	Object
Joe	Read	File 1
Joe	Write	File 1
Joe	Own	File 1
Joe	Read	File 2
Sam	Read	File 2
Sam	Write	File 2
Sam	Own	File 2

Authorization Graph



Authorization Graph

- ***REFERENCES Option*** The REFERENCES privilege is not the same as the SELECT privilege.
- Let us take an example. Suppose Nash is the owner of the DEPARTMENT table as indicated below:
  - DEPARTMENT (DeptNo, DeptName, DeptLocation)
- Nash can authorize Miller to create another table EMPLOYEE with a foreign key in that table to refer to the DeptNo column in the DEPARTMENT table.
- Nash can do this by granting Miller the REFERENCES privilege with respect to the DeptNo column.
- Note the EMPLOYEE table shown below:
  - EMPLOYEE (EmployeeNo, FirstName, LastName, Address, Phone, Employee Position, Salary, EmployeeCode, DeptNo)
  - Foreign Key: DeptNo REFERENCES DEPARTMENT
- If Miller loses the REFERENCES privilege with respect to the DeptNo column in the DEPARTMENT table, the foreign key constraint in the EMPLOYEE table will be dropped. The EMPLOYEE table itself, however, will not be dropped.
- Now suppose Miller has the SELECT privilege on the DeptNo column of the DEPARTMENT table, not the REFERENCES privilege. In this case, Miller will not be allowed to create the EMPLOYEE table with a foreign key column referring to DeptNo in the DEPARTMENT table.

Authorization Graph

- Why not grant Miller the SELECT privilege and allow him to create the EMPLOYEE table with a foreign key column referring to the DeptNo column in the DEPARTMENT table? If this is done, assume that Miller creates the table with a foreign key constraint as follows:
  - EMPLOYEE (EmployeeNo, FirstName, LastName, Address, Phone, Employee Position, Salary, EmployeeCode, DeptNo)
- Foreign Key: DeptNo REFERENCES DEPARTMENT ON DELETE NO ACTION
- With the NO ACTION option in the foreign key specification, Nash is prevented from deleting rows from the DEPARTMENT table even though he is the owner.
- For this reason, whenever such a restrictive privilege needs to be authorized, the more stringent privilege REFERENCES is applied.The SELECT privilege is therefore intended as permission just to read the values.

Types of Discretionary Privileges

- **The account level:**
  - At this level, the DBA specifies the particular privileges that each account holds independently of the relations in the database.
  - **At the level of *users,* access privileges include the following**:
    - CREATE privilege To create a database schema, a table or relation, or a user view
    - ALTER privilege To alter and make changes to schema such as adding or eliminating columns
    - DROP privilege To delete a table or view
    - SELECT privilege To retrieve data
    - MODIFY privilege To add or insert, update, or delete data
    - REFERENCES privilege To define foreign keys and reference columns in another table
  - **At the level of *database objects***, the access privileges listed above apply to the following:
    - Base table All data in the table or relation
    - View All data defined in the virtual relation or view
    - Column Data in the specified column only

Types of Discretionary Privileges

- **The relation level (or table level):**
  - At this level, the DBA can control the privilege to access each individual relation or view in the database.
  - This includes **base relations** and virtual (**view**) relations
  - **The granting and revoking of privileges generally follow an authorization model for discretionary privileges known as the access matrix model where**
    - The **rows** of a matrix M represents **subjects** (users, accounts, programs)
    - The **columns** represent **objects** (relations, records, columns, views, operations).
    - Each position **M(i,j)** in the matrix represents the types of privileges (read, write, update) that **subject i** holds on **object j**.

Types of Discretionary Privileges

- ***Owner Account***
  - Each database table or relation has an owner.
  - This user account that created the table possesses all access privileges on that table.
  - The DBA can assign an owner to an entire schema and grant the appropriate access privileges.
  - The owner of a database object can grant privileges to another user.
  - This second user can then pass along the privileges to a third user and so on.
  - The DBMS keeps track of the cycle of granting of privileges.
- **To control the granting and revoking of relation privileges:**
  - **Each relation R in a database is assigned and owner account (creator)**
  - The owner of a relation is given all privileges on that relation.
  - DBA can assign and owner to a whole schema by creating the schema and associating the appropriate authorization identifier with that schema, using the **CREATE SCHEMA** command.
  - The owner account holder can **pass privileges** on any of the owned relation to other users by **granting** privileges to their accounts.

Types of Discretionary Privileges

- **In SQL the following types of privileges can be granted on each individual relation R:**
- **SELECT** (retrieval or read) privilege on R:
  - Gives the account retrieval privilege.
  - In SQL this gives the account the privilege to use the **SELECT** statement to retrieve tuples from R.
- **MODIFY** privileges on R:
  - This gives the account the capability to modify tuples of R.
  - In SQL this privilege is further divided into **UPDATE**, **DELETE**, and **INSERT** privileges to apply the corresponding SQL command to R.
  - In addition, both the **INSERT** and **UPDATE** privileges can specify that only certain attributes can be updated by the account.
- **In SQL the following types of privileges can be granted on each individual relation R (contd.):**
  - **REFERENCES** privilege on R:
  - This gives the account the capability to **reference** relation R when specifying integrity constraints.
  - The privilege can also be **restricted** to specific attributes of R.
  - **Notice that to create a view, the account must have SELECT privilege on all relations involved in the view definition.**

Access Control Conditions

- **Data-dependent conditions**: Access constraints based on the value of the accessed data
- **Time-dependent**: Access constraints based on the time of the data access
- **Context-dependent**: Access constraints based on combinations on data which can be accessed
- **History-dependent**: Access constraints based on previously accessed data
- **Content-based a**ccess control conditions the access to a given object to its content
  - As an example, in a RDBMS supporting content-based access control it is possible to authorize a subject to access information only of those employees whose salary is not greater than 30K
  - Two are the most common approaches to enforce content-based access control in a DBMS:
    - by associating a predicate (or a Boolean combination of predicates) with the authorization
    - by defining a *view* which selects the objects whose content satisfies a given condition, and then granting the authorization on the view instead of on the basic objects

Use of Views

- A user view is like a personalized model of the database tailored for individual groups of users.
- If a user group, say, in the marketing department, needs to access only some columns of the DEPARTMENTand EMPLOYEE tables, then you can satisfy their information requirements by creating a view comprising just those columns.
- This view hides the unnecessary parts of the database from the marketing group and shows them only those columns they require.
- Views are not like tables in the sense that they do not store actual data. You know that views are just like windows into the database tables that store the data.
- Views are virtual tables. When a user accesses data through a view, he or she is getting the data from the base tables, but only from the columns defined in the view.
- Views are intended to present to the user exactly what is needed from the database and to make the rest of the data content transparent to the user.
- However, views offer a flexible and simple method for granting access privileges in a personalized manner.
- Views are powerful security tools.When you grant access privileges to a user for a specific view, the privileges apply only to those data items defined in the views and not to the complete base tables themselves.

Use of Views

- An example of a view and see how it may be used to grant access privileges.
- For a user to create a view from multiple tables, the user must have access privileges on those base tables. The view is dropped automatically if the access privileges are dropped.
- Note the following example granting access privilege to Miller for reading EmployeeNo, FirstName, LastName, Address, and Phone information of employees in the department where Miller works.
- CREATE VIEW MILLER AS
  - SELECT EmployeeNo, FirstName, LastName, Address, Phone
  - FROM EMPLOYEE
  - WHERE DeptNo =
    - (SELECT DEPARTMENT.DeptNo WHERE DEPARTMENT.DeptNo = EMPLOYEE.DeptNo AND (EMPLOYEE.LastName = ‘Miller’)) ;
  - GRANT SELECT ON MILLER TO Miller;

Use of Views

- Remember that a VIEW is created by an SQL SELECT, and that a view is only a virtual table. Although not part of the base tables, it is processed and appears to be maintained by the DBMS as if it were.
- To provide privileges at the level of the row, the column or by values, it is necessary to grant rights to a view. This means a certain amount of effort but gives a considerable range of control. First create the view: 'the first statement creates the view'

```
CREATE VIEW VIEW1
AS SELECT A1, A2, A3
FROM TABLE1
WHERE A1 < 20000;
```

- 'and the privilege is now assigned'

```
GRANT SELECT ON VIEW1 TO U1
WITH GRANT OPTION;
```

- The optional "with grant option" allows the user to assign privileges to other users. This might seem like a security weakness and is a loss of DBA control.
- On the other hand, the need for temporary privileges can be very frequent and it may be better that a user assign temporary privileges to cover for an office absence, than divulge a confidential password and user-id with a much higher level of privilege.
- The rights to change data are granted separately:
  - GRANT INSERT ON TABLE1 TO U2, U3;
  - GRANT DELETE ON TABLE1 TO U2, U3;
  - GRANT UPDATE ON TABLE1(salary) TO U5;
  - GRANT INSERT, DELETE ON TABLE1 TO U2, U3;
- To provide general access:
  - GRANT ALL TO PUBLIC;

Specifying Privileges Using Views

- **The mechanism of views is an important discretionary authorization mechanism in its own right.**
- **For example,**
  - If the owner A of a relation R wants another account B to be able to retrieve only some fields of R, then A can create a view V of R that includes only those attributes and then grant SELECT on V to B.
  - The same applies to limiting B to retrieving only certain tuples of R; a view V' can be created by defining the view by means of a query that selects only those tuples from R that A wants to allow B to access.

```
CREATE VIEW Fall-Student
        AS SELECT NAME, COURSE
        FROM Student
        WHERE SEMESTER="Fall 2000"
```

Fall-Student

NAME	COURSE
White	CSCE 122
Black	CSCE 313
Green	CSCE 850
Blue	CSCE 122

SQL Examples

- Use the DEPARTMENT and EMPLOYEE tables shown above.
- **DBA gives privileges to Miller to create the schema:**
  - GRANT CREATETAB TO Miller;
- **Miller defines schema, beginning with create schema statement:**
  - CREATE SCHEMA EmployeeDB AUTHORIZATION Miller; (other DDL statements follow to define DEPARTMENT and EMPLOYEE tables)
- **Miller gives privileges to Rodriguez for inserting data in both tables:**
  - GRANT INSERT ON DEPARTMENT, EMPLOYEE TO Rodriguez;
- **Miller gives Goldstein privileges for inserting and deleting rows in both tables, allowing permission to propagate these privileges:**
  - GRANT INSERT, DELETE ON DEPARTMENT, EMPLOYEE TO Goldstein WITH GRANT OPTION;
- **Goldstein passes on the privileges for inserting and deleting rows in the DEPARTMENT table to Rogers:**
  - GRANT INSERT, DELETE ON DEPARTMENT TO Rogers;
- **Miller gives Williams the privilege to update only the salary and position columns in the EMPLOYEE table:**
  - GRANT UPDATE ON EMPLOYEE (Salary, EmployeePosition) TO Williams
- **DBA gives Shady privilege to create tables:**
  - GRANT CREATETAB TO Shady;
- **Shady creates table MYTABLE and gives Miller privilege to insert rows into MYTABLE:**
  - GRANT INSERT ON MYTABLE TO Miller;

Issues

- Note the actions of user Shady indicated in the last few statements of the previous subsection.
- Shady has created a private table MYTABLE of which he is the owner. He has all privileges on this table. All he has to do is somehow get sensitive data into MYTABLE.
- Being a clever professional, Shady may temporarily alter one of Miller's programs to take data from the EMPLOYEE data and move the data into MYTABLE.
- For this purpose, Shady has already given privileges to Miller for inserting rows into the MYTABLE table.
- This scenario appears as too unlikely and contrived. Nevertheless, it makes the statement that discretionary access control has its limitations.

Stored Procedures

- Assign rights to execute compiled programs

  GRANT RUN ON <program> TO <user>

Problem: Programs may access resources for which the user who runs the program does not have permission.

Grant and Revoke

REVOKE <privileges> [ON <relation>]

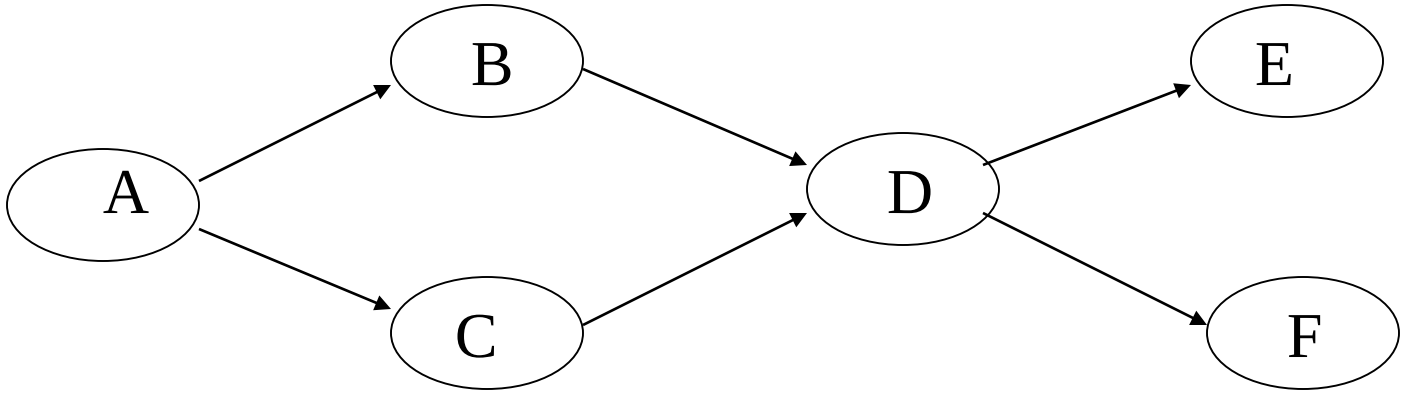
FROM <user>

- REVOKE SELECT* ON *Student* FROM Blue
- REVOKE UPDATE ON *Student* FROM Black
- REVOKE SELECT(NAME) ON *Student* FROM Brown

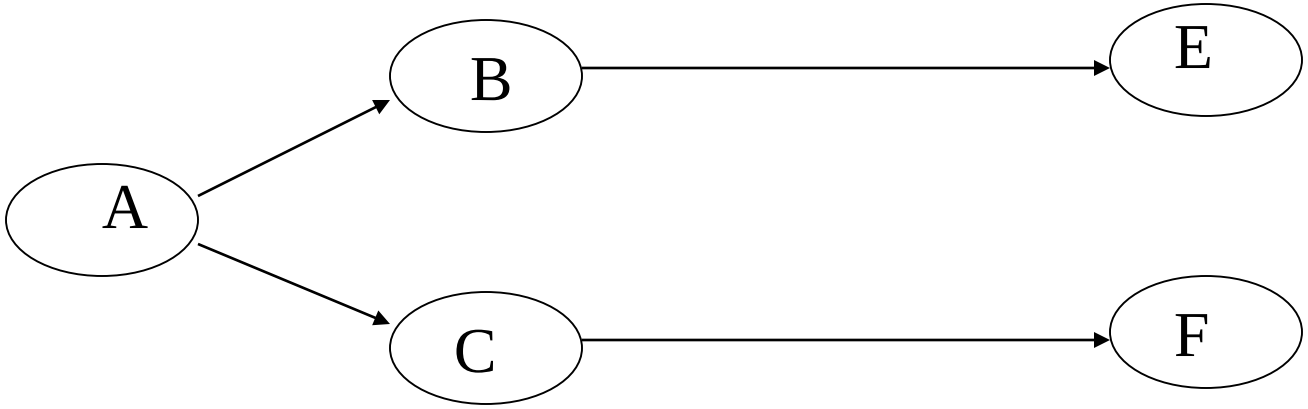
Revoking Privileges

- **In some cases it is desirable to grant a privilege to a user temporarily. For example,**
  - The owner of a relation may want to grant the **SELECT** privilege to a user for a specific task and then revoke that privilege once the task is completed.
  - Hence, a mechanism for **revoking** privileges is needed. In SQL, a **REVOKE** command is included for the purpose of **canceling privileges**.
- Propagation of Privileges using the GRANT OPTION
  - **Whenever the owner A of a relation R grants a privilege on R to another account B, privilege can be given to B with or without the GRANT OPTION.**
  - **If the GRANT OPTION is given, this means that B can also grant that privilege on R to other accounts.**
    - Suppose that B is given the **GRANT OPTION** by A and that B then grants the privilege on R to a third account C, also with **GRANT OPTION**. In this way, privileges on R can **propagate** to other accounts without the knowledge of the owner of R.
    - If the owner account A now revokes the privilege granted to B, all the privileges that B propagated based on that privilege should automatically be revoked by the system.

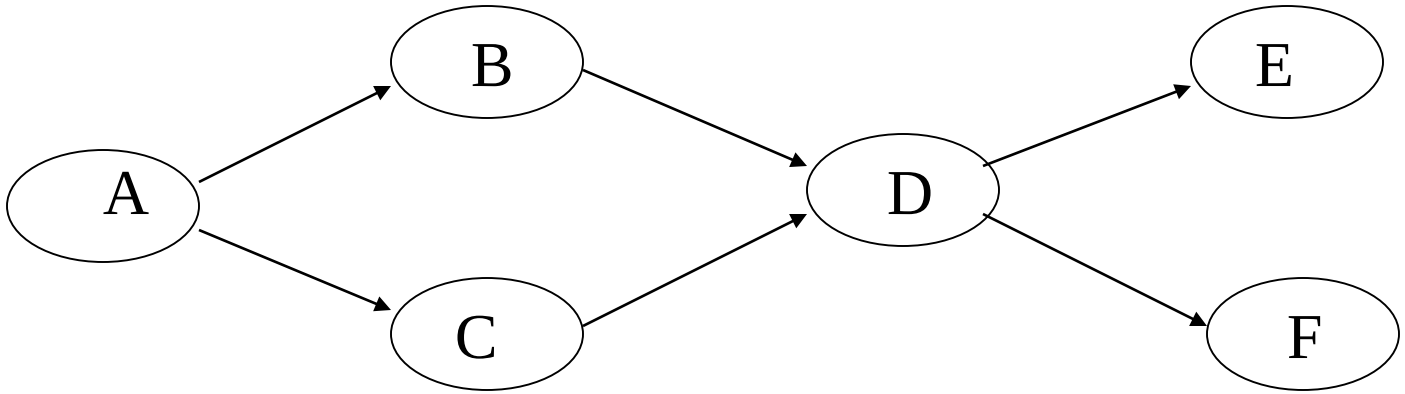
Non-cascading Revoke



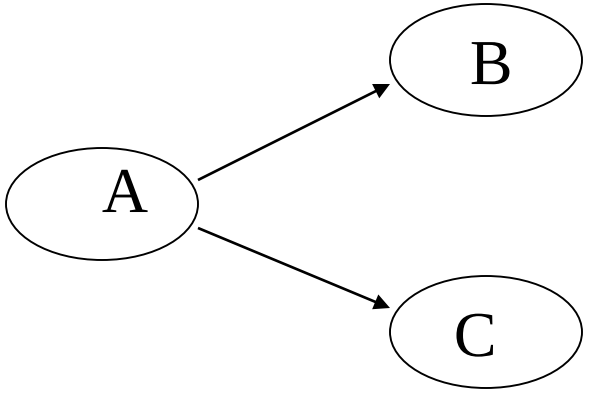
A revokes D’s privileges



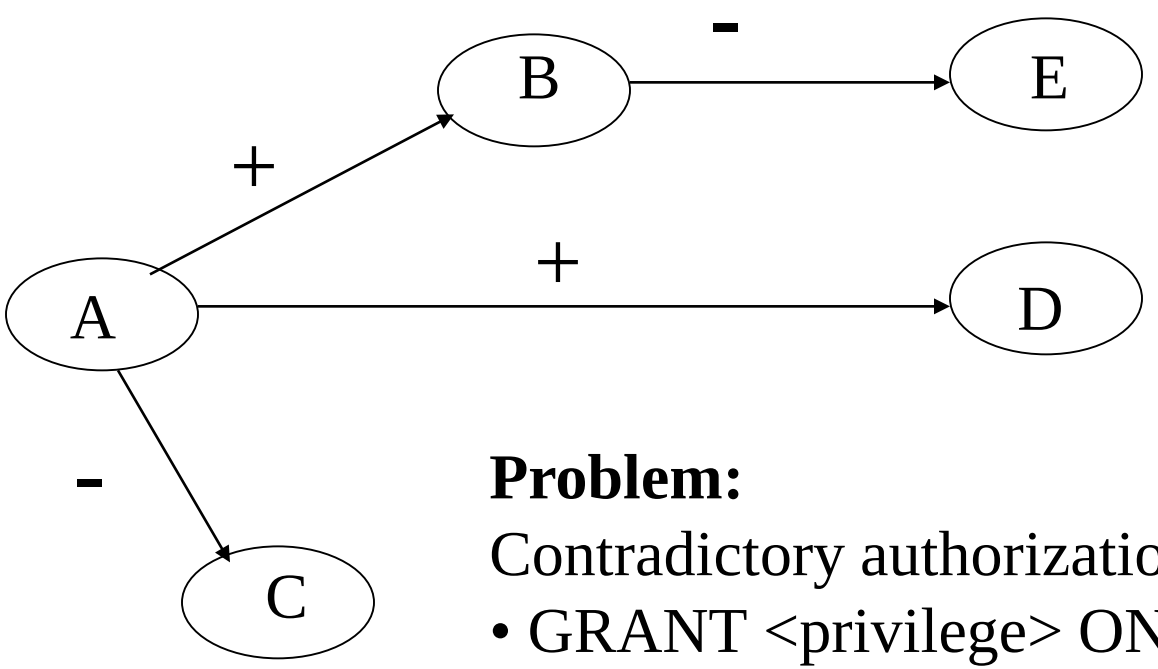
Cascading Revoke



A revokes D’s privileges



Positive and Negative Authorization

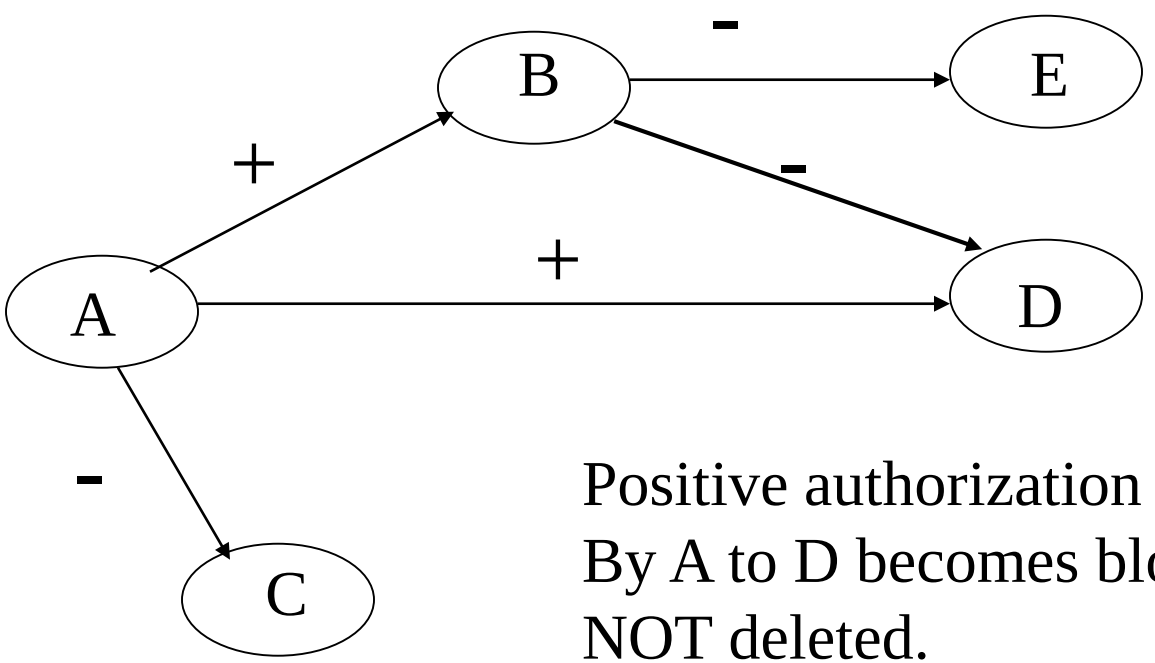


Problem:

Contradictory authorizations

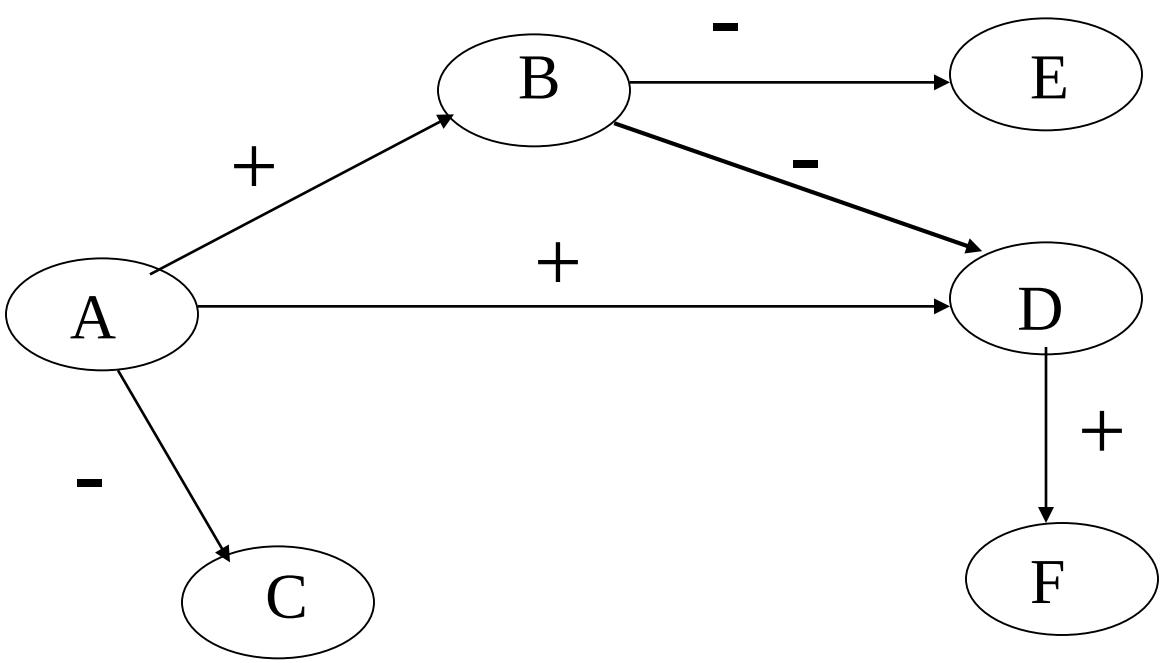
- GRANT <privilege> ON *X* TO <user>
- DENY <privilege> ON *X* TO <user>

Negative Authorization



Positive authorization granted
By A to D becomes blocked but
NOT deleted.

Negative Authorization



What should happen with the privilege given by D To F? (Blocked but not deleted)

Query Modification

- GRANT SELECT(NAME) ON *Student* TO Blue WHERE COURSE="CSCE 580"
- **Blue's query:**

  SELECT *

  FROM *Student*
- **Modified query:**

  SELECT NAME

  FROM *Student*

  WHERE COURSE="CSCE 580"

Current Research

- Make cascading optional
- Permit negative authorization
- Flexible policies for resolving conflicts
- Extend to groups and views

Disadvantages of DAC

The DAC model has two primary disadvantages:

- Since it is left to the discretion of the owner of an object do decide about giving access permissions, the users may have a different perception/ concept of security which may or may not be in line with the enterprise policy.
  - Hence, it becomes difficult to enforce uniform security throughout the enterprise policy. Technically, the enterprise is the owner of all data.
- Revocations of access permissions granter earlier may not have the intended effect.
  - For example, assume a subject S1 has granted the Update access on object O to subject S2, which after a while grants subject S3 the access to object O. Later, if S1 has granted the Update access on object O to S3.
  - Now, even if one of the two subjects (S1 or S2) revoke their access permission on object O to S3, S3 can still access object O by virtue of the grant permission granted by the subject (S1 or S2) which has not yet revoked the same.

DAC and Trojan Horse

Brown

Read Employee

Black

REJECTED!
Black is not allowed
To access Employee

Brown: read, write

Employee

Black, Brown: read, write

Black’s Employee

DAC and Trojan Horse

Brown

Uses shared program

Word Processor

TH

Inserts Trojan Horse
Into shared program

Black

Brown: read, write

Employee

Reads
Employee

Black, Brown: read, write

Black's Employee

Copies
Employee
To Black's
Employee

DAC : Summary

- Advantages:
  - Intuitive - Flexibility in terms of policy specification
  - Easy to implement
  - Most of the common commercial DBMSs support it
- Disadvantages:
  - Inherent vulnerability -No information flow control (Trojan Horses attacks)
  - Maintenance of ACL or Capability lists
  - Maintenance of Grant/Revoke
  - Limited power of negative authorization
- Flexibility is enhanced by supporting different kinds of authorizations:
  - Positive vs. negative
  - Strong vs. weak
  - Implicit vs. explicit
  - Content-based

LIMITATIONS OF DISCRETIONARY CONTROLS

- The standard access controls of SQL are said to be discretionary, because the granting of access is under user control.
- Discretionary controls have a fundamental weakness.
- Even if access to a relation is strictly controlled, it is possible for a user with SELECT access to create a copy of the relation thereby circumventing these controls.
- Furthermore, even if users are trusted not to deliberately engage in such mischief it is possible for Trojan Horse infected programs to do so.
- To illustrate the basic limitation of discretionary access controls, consider the following grant operation.
  - TOM: GRANT SELECT ON EMPLOYEE TO DICK
- Tom's intention is that Dick should not be allowed to further grant SELECT access on EMPLOYEE to other users.
- However, this intent is easily subverted as follows. Dick creates a new relation, call it COPY-OF-EMPLOYEE, into which he copies all the rows of EMPLOYEE. As the creator of COPY-OF-EMPLOYEE, Dick has the authority to grant any privileges for it to any user. Dick can therefore grant Harry access to COPY-OF-EMPLOYEE as follows.
  - DICK: GRANT SELECT ON COPY-OF-EMPLOYEE TO HARRY
- At this point Harry has access to all the information in the original EMPLOYEE relation. For all practical purposes Harry has SELECT access to EMPLOYEE, so long as Dick keeps COPY-OF-EMPLOYEE reasonably up to date with respect to EMPLOYEE.
- The situation is actually worse than the above scenario indicates. So far, we have portrayed Dick as a cooperative participant in this process. Now suppose that Dick is a trusted confidant of Tom and would not deliberately subvert Tom's intentions regarding the EMPLOYEE relation. However, Dick uses a fancy text editor supplied to him by Harry. This editor provides all the editing services that Dick needs. In addition Harry has also programmed it to create the COPY-OF-EMPLOYEE relation and execute the above grant operation. **Such software is said to be a Trojan Horse**, because in addition to the normal functions expected by its user it also engages in surreptitious actions to subvert security. Note that a Trojan Horse executed by Tom could actually grant Harry the privilege to SELECT on EMPLOYEE.
- **The solution is to impose mandatory controls which cannot be violated, even by Trojan Horses.**

Discretionary Access Control

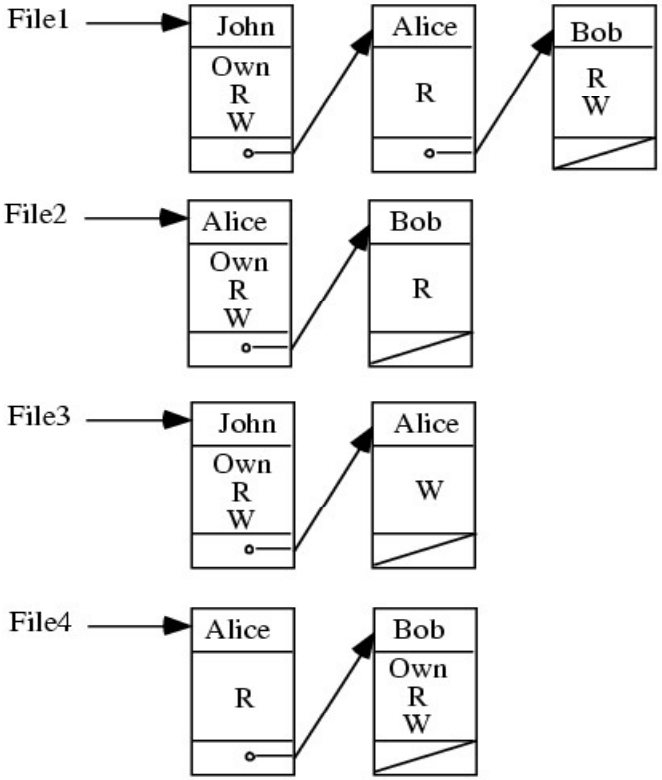
Discretionary Access Control

- DAC policies govern the access of subjects to objects on the basis of subjects' identity, objects' identity and permissions
- When an access request is submitted to the system, the access control mechanism verifies whether there is a permission authorizing the access
- Such mechanisms are discretionary in that *they allow subjects to grant other subjects authorization to access their objects at their discretion*
- Most common administration: owner based
  - Users can protect what they own
  - Owner may grant access to others
  - Owner may define the type of access given to others

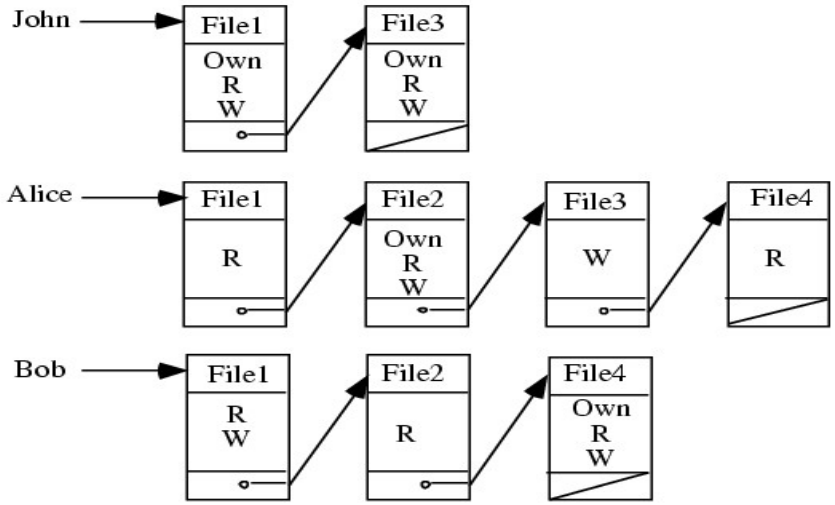
DAC – Access Matrix

	File 1	File 2	File 3	File 4	Account 1	Account 2
John	Own R W		Own R W		Inquiry Credit	
Alice	R	Own R W	W	R	Inquiry Debit	Inquiry Credit
Bob	R W	R		Own R W		Inquiry Debit

DAC – Implementation



Access control lists



Capability lists: What can this User do?

Authorization Relation

Subject	Access mode	Object
John	Own	File 1
John	R	File 1
John	W	File 1
John	Own	File 3
John	R	File 3
John	W	File 3
Alice	R	File 1
Alice	Own	File 2
Alice	R	File 2
Alice	W	File 2
Alice	W	File 3
Alice	R	File 4
Bob	R	File 1
Bob	W	File 1
Bob	R	File 2
Bob	Own	File 4
Bob	R	File 4
Bob	W	File 4

Access Control Conditions

- **Data-dependent conditions**: access constraints based on the value of the accessed data
- **Time-dependent**: access constraints based on the time of the data access
- **Context-dependent**: access constraints based on collection of information (rather than sensitivity of data) which can be accessed
- **History-dependent**: access constraints based on previously accessed data

OS vs DBMS

- Data model is richer than that provided by OS – files vs different levels of abstractions (physical, logical, view).
- Different abstractions are used to represent data at logical level (e..g, relations, XML) and require different ways of protection.
- DBMS usually requires a variety of granularity levels for access control, e.g., relation and view, and finer granularity like attributes.
- Logical level introduces complexity
  - objects are usually related by different semantic relations, and these relations must be carefully protected, e.g., data in different tables are linked through foreign keys.
  - several logical objects (e.g., different views) may also correspond to the same logical/physical objects (same file) or same logical object (views) may correspond to different physical/logical objects (different files/relations the views have been built)
- Data accessed by a wider variety of access modes (update, based on SQL statements).

Access Control in Commercial DBMSs

- All commercial systems adopt DAC
- Current discretionary authorization models for relational DBMS are based on the System R authorization model
  - P. P. Griffiths and B. W. Wade. **An Authorization Mechanism for a Relational Database System.** ACM Trans. Database Syst. 1, 3 (Sep. 1976), Pages 242 - 255.
- It is based on ownership administration with administration delegation

The System R Authorization Model

- Objects to be protected are tables and views
- Privileges include: *select*, *update*, *insert*, *delete*, *drop*, *index* (only for tables), *alter* (only for tables)
- Groups are supported, whereas roles are not
- Privilege *delegation* is supported through the *grant option*:
  - if a privilege is granted with the grant option, the user receiving it can *exercise the privilege* AND *grant it to other users*
  - a user can only grant a privilege on a given relation if he/she is the table owner or if he/she has received the privilege with grant option

Grant operation

GRANT *PrivilegeList* | ALL[PRIVILEGES]
ON *Relation* | *View*
TO *UserList* | PUBLIC
[WITH GRANT OPTION]

- it is possible to grant privileges on both relations and views
- privileges apply to entire relations (or views)
- for the update privilege, one needs to specify the columns to which it applies

Grant operation - example

Bob: GRANT select, insert ON Employee TO Ann
WITH GRANT OPTION;
Bob: GRANT select ON Employee TO Jim
WITH GRANT OPTION;
Ann: GRANT select, insert ON Employee TO Jim;

- Jim has the select privilege (received from both Bob and Ann) and the insert privilege (received from Ann)
- Jim can grant to other users the select privilege (because it has received it with grant option); however, he cannot grant the insert privilege

Grant operation

- The authorization catalog keeps track of the privileges that each user can delegate
- Whenever a user u executes a Grant operation, the system intersects the delegable privileges of u with the set of privileges specified in the command
- If the intersection is empty, the command is not executed

Grant operation - example

Bob: GRANT select, insert ON Employee TO Jim WITH GRANT OPTION;

Bob: GRANT select ON Employee TO Ann WITH GRANT OPTION;

Bob: GRANT insert ON Employee TO Ann;

Jim: GRANT update ON Employee TO Tim WITH GRANT OPTION;

Ann: GRANT select, insert ON Employee TO Tim;

- The first three GRANT commands are fully executed (Bob is the owner of the table)
- The fourth command is not executed, because Jim does not have the update privilege on the table
- The fifth command is partially executed; Ann has the select and insert but she does not have the grant option for the insert; so Tim only receives the select privilege

Revoke operation

REVOKE *PrivilegeList* | ALL[PRIVILEGES]
ON *Relation* | *View*
FROM *UserList* | PUBLIC

- When a privilege is revoked, the access privileges of the revokee should be indistinguishable from a sequence in which the grant never occurred.

Revoke operation

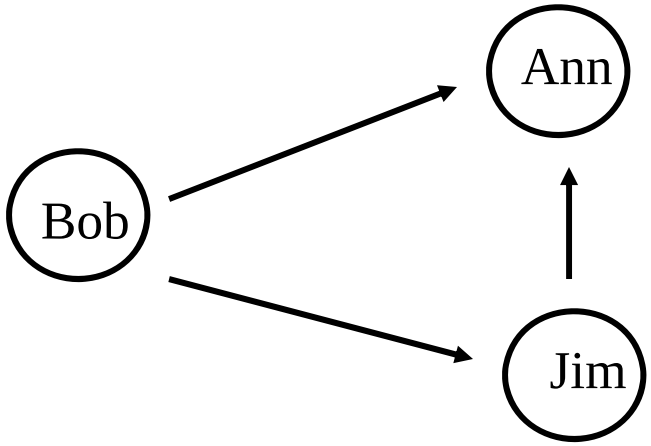
REVOKE *PrivilegeList* | ALL[PRIVILEGES]
ON *Relation* | *View*
FROM *UserList* | PUBLIC

- What happens when a “with grant option” privilege is revoked?
- What happens when a user is granted access from two different sources, and one is revoked?

Grants from multiple sources

- grant(Bob, Ann)
- grant(Bob, Jim)
- grant(Jim,Ann)
- revoke(Bob, Ann)

- grant(Bob, Ann)
- grant(Bob, Jim)
- grant(Jim,Ann)
- revoke(Bob, Ann)

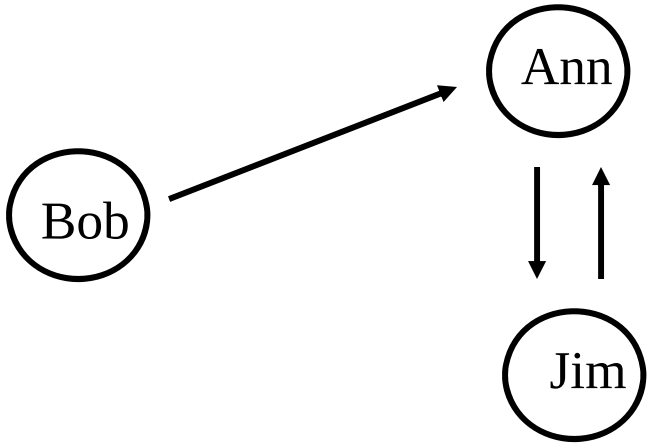


Assume all grant statements are with grant option

But …

- grant(Bob, Ann)
- grant(Ann, Jim)
- grant(Jim,Ann)
- revoke(Bob, Ann)

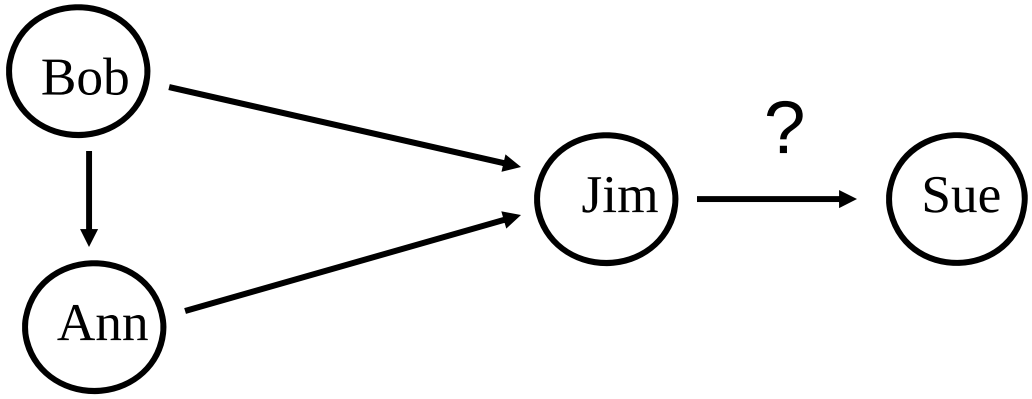
- grant(Bob, Ann)
- grant(Ann, Jim)
- grant(Jim,Ann)
- revoke(Bob, Ann)



Recursive revocation …

- grant(Bob, Ann)
- grant(Bob, Jim)
- grant(Jim,Sue)
- grant (Ann, Jim)
- revoke(Bob, Jim)

- grant(Bob, Ann)
- grant(Bob, Jim)
- grant(Jim,Sue)
- grant (Ann, Jim)
- revoke(Bob, Jim)



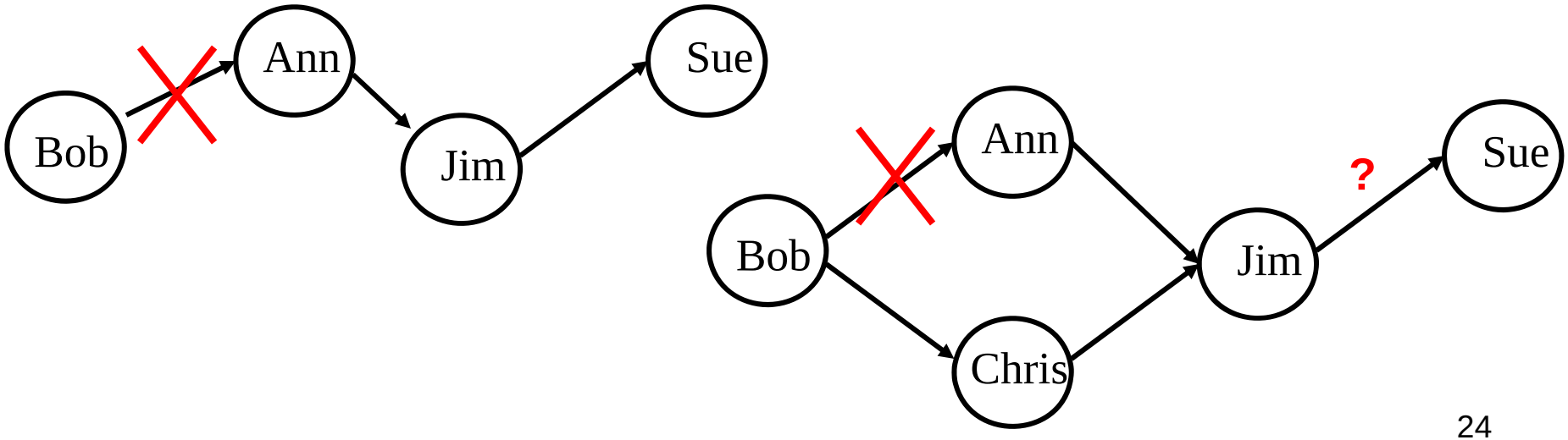
Revoke operation

REVOKE *PrivilegeList* | ALL[PRIVILEGES]
ON *Relation* | *View*
FROM *UserList* | PUBLIC

- a user can only revoke the privileges he/she has granted; it is not possible (?) to only revoke the grant option
- upon execution of a revoke operation, the user from whom the privileges have been revoked looses these privileges, unless the user has them from some source *independent* from that that has executed the revoke

Revoke operations

- Recursive revocation
  - whenever a user revokes an authorization on a table from another user, all the authorizations that the revokee had granted because of the revoked authorization are removed
  - The revocation is iteratively applied to all the subjects that received the access authorization from the revokee



Recursive revocation

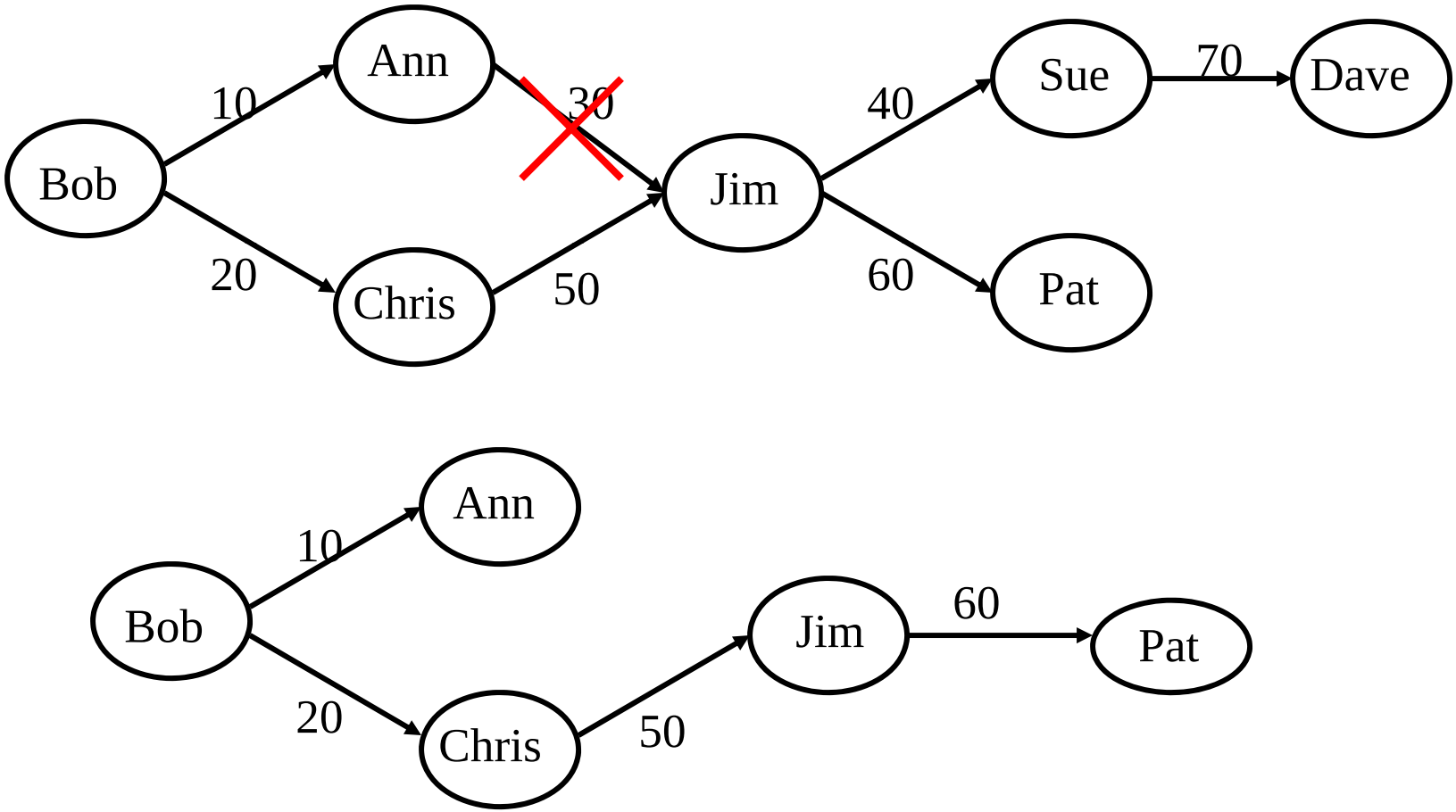
- Let G_1, …., G_n be a sequence of grant operations with a single privilege on the same relations, such that i,k = 1,…., n, if i<k, then G_i is executed before G_k. Let R_i be the revoke operation for the privilege granted with operation G_i.
- The semantics of the recursive revoke requires that the state of the authorization system after the execution of the sequence

  G_1, …., G_n , R_i

  be identical to the state that one would have after the execution of the sequence

  G_1, …., G_{i-1}, G_{i+1} , …., G_n

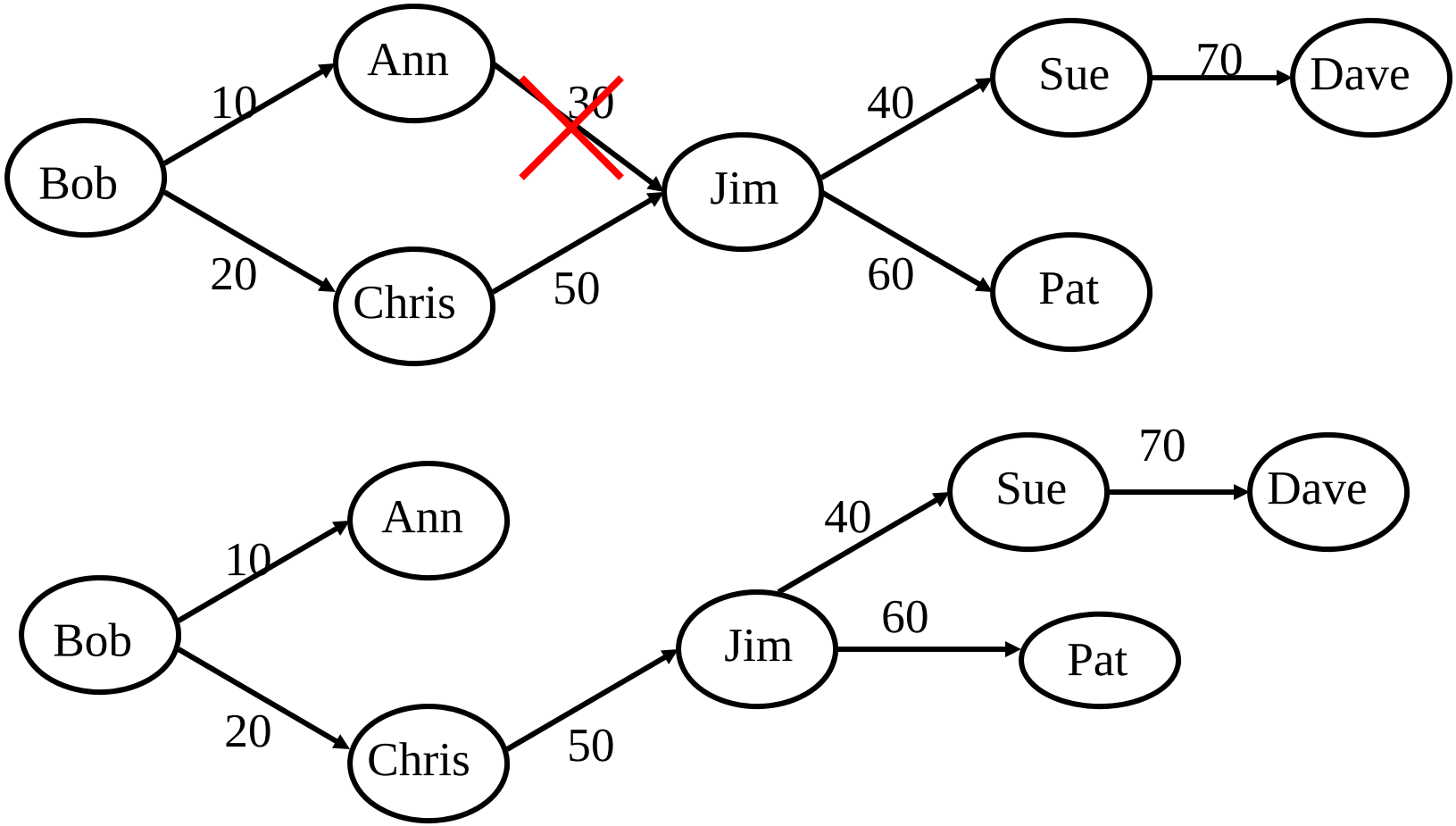
Recursive Revocation



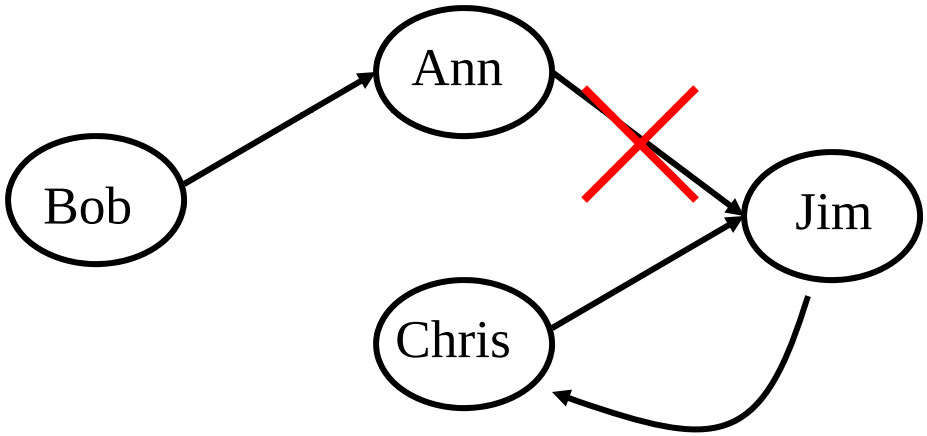
Recursive revocation

- Recursive revocation in the System R takes into account the timestamps denoting when each authorization has been granted
- Variations to this approach have been proposed that do not take into account the timestamps
  - Why?
    - To avoid cascades of revoke
    - The authorizations granted by the revokee are kept as long as the revokee has other authorizations for the same privilege (even if these authorizations have a larger timestamps with respect to the timestamps of the grant operations performed by the revokee)

Recursive revocation without timestamp



Recursive revocation without timestamp



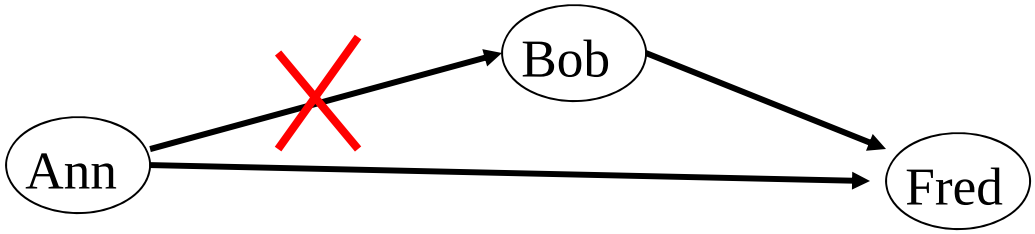
?????

Noncascading Revocation

- Recursive revocation can be a very disruptive operation
- A recursive revoke entails:
  - Revoking all authorizations the revokee granted, for which no other supporting authorizations exist and, recursively, revoking all authorizations granted through them
  - Invalidating application programs and views

Noncascading Revoke

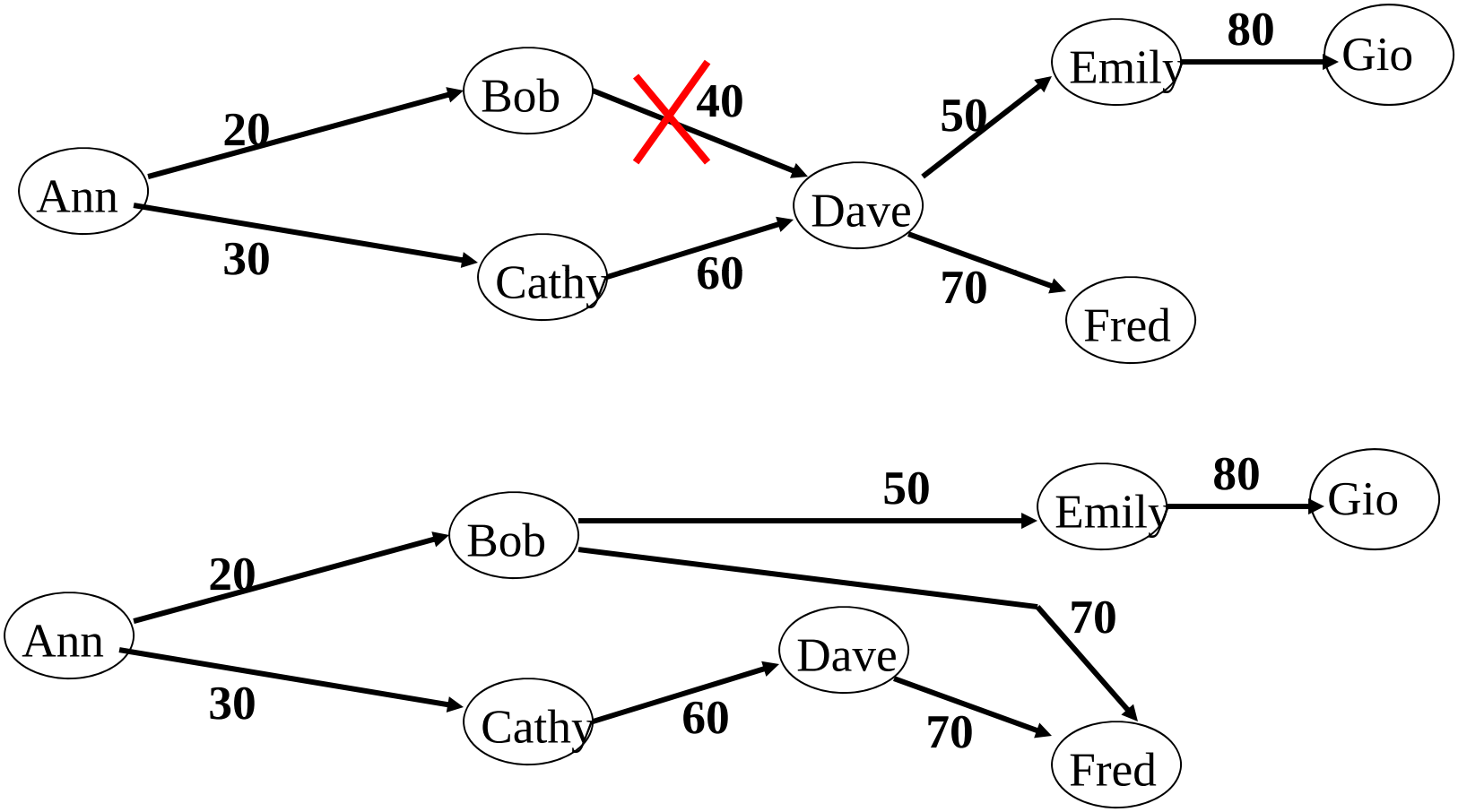
- A user can revoke a privilege on a table from another user without entailing automatic revocation of the authorizations for the privilege on the table the latter may have granted
- Instead of deleting the authorizations the revokee may have granted by using the privilege received by the revoker, all these authorizations are restated as if they had been granted by the revoker



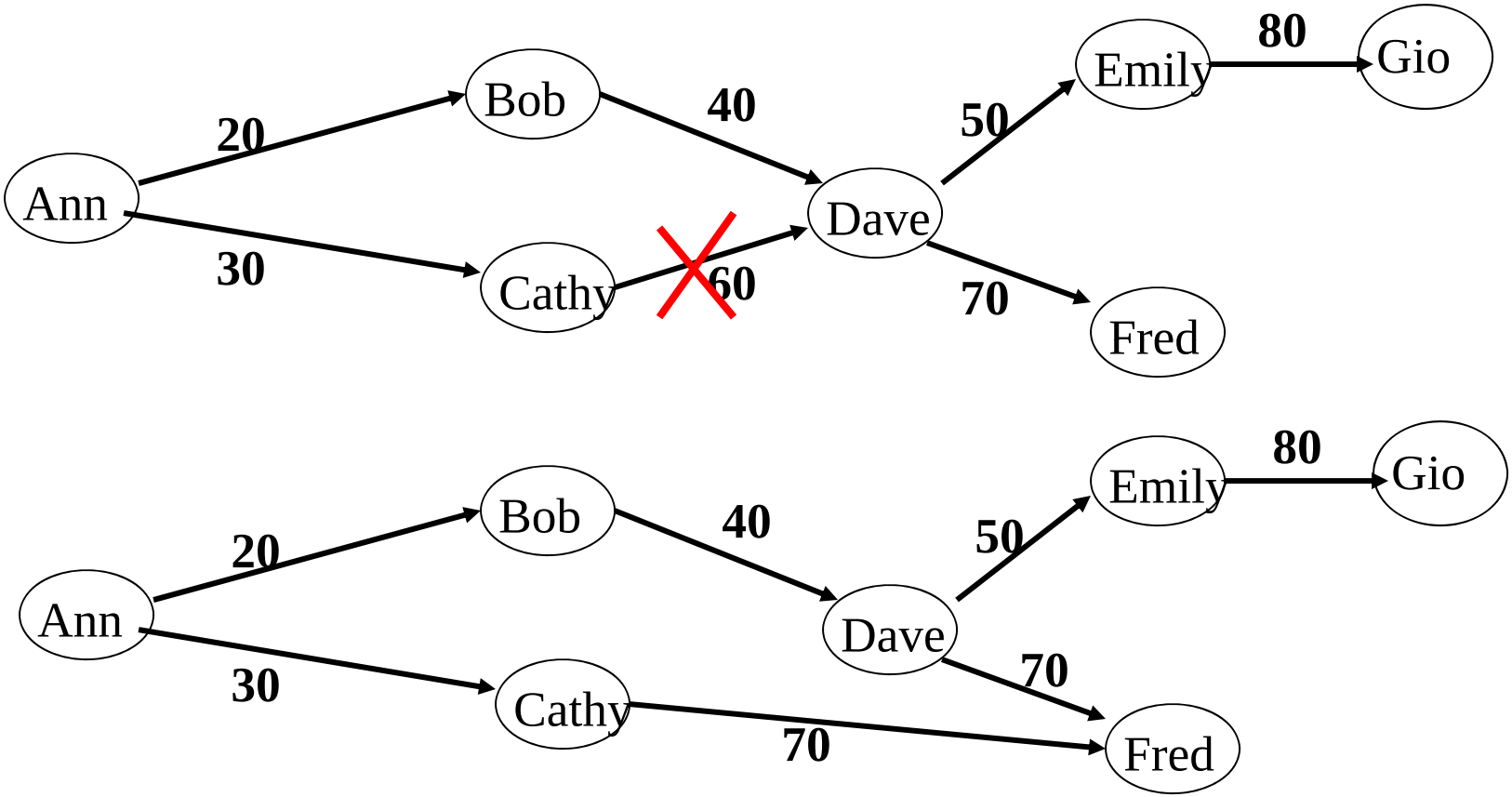
Noncascading Revoke

- The semantics of the revocation without cascade of privilege *p* on table *t* from user *y* by user *x* is:
  - To restate with *x* as grantor all authorizations that *y* granted by using the authorization being revoked
- Since *y* may have received the grant option for the privilege on the table from some other users different from *x*, not all authorizations he/she granted will be given to *x*
  - *x* will be considered as grantor only of the authorizations *y* granted after receiving the privilege with the grant option from *x*;
  - *y* will still be considered as grantor of all authorizations he/she granted that are supported by other authorizations not granted by *x*

Noncascading Revoke



Noncascading Revoke



- Note that the authorization granted by Dave to Emily has not been specified with Cathy as grantor. Why?
- Because it was granted before Dave received the privilege from Cathy

Views and content-based authorization

- *Views* are commonly used to support content-based access control in RDBMS
- Content-based access authorizations should be specified in terms of predicates
  - Views can also be used to grant privileges on simple statistics calculated on data (such as AVG, SUM,..)
- Only the tuples of a relation verifying a given predicate are considered as the protected objects of the authorization
- The approach to support content-based access control in RDBMS can be summarized as follows:
  - Define a view containing the predicates to select the tuples to be returned to a given subject S
  - Grant S the select/insert/update privileges on the view, and not on the underlying table

Views and content-based authorization

- Queries against views are transformed through *view composition* into queries against base tables
- The view composition operation combines the predicates specified in the query on the view with the predicates which are part of the view definition

- Example: suppose we want to authorize user Ann to access only employees whose salary is lower than 20000:

```
CREATE VIEW Vemp AS
SELECT * FROM Employee
WHERE Salary < 20000;

GRANT Select ON Vemp TO Ann;
```

```
Ann:  SELECT * FROM Vemp
      WHERE Job = 'Programmer';
```

Query after view composition:

```
SELECT * FROM Employee
WHERE Salary < 20000 AND
      Job = 'Programmer';
```

Steps in Query Processing

- Parsing
- Catalog lookup
- Authorization checking
- View Composition
- Query optimization

  Note that authorization is performed before view composition; therefore, authorization checking is against the views used in the query and not against the base tables used in these views

Authorizations on views

- The user creating a view is called the *view definer*
- The privileges that the view definer gets on the view depend on:
  - The view semantics, that is, its definition in terms of the base relation(s)
  - The authorizations that the definers has on the base table(s)
- The view definer does not receive privileges corresponding to operations that cannot be executed on the view e.g., `alter` and `index` do not apply to views
- To determine the privileges that the view definer has on the view, the system needs to intersect the set of privileges that the view definer has on the base tables with the set of privileges corresponding to the operations that can be performed on the view

Authorizations on views

- Consider the following view

  Bob: CREATE VIEW V1 (Emp#, Total_Sal)
  AS SELECT Emp#, Salary + Bonus
  FROM Employee WHERE
  Job ='Programmer';

  The update operation is not defined on column Total_Sal of the view; therefore, Bob will not receive the update authorization on such column

Authorizations on views - example

- Consider relation Employee and assume Bob is the creator of Employee
- Consider the following sequence of commands:
  - Bob: GRANT Select, Insert, Update ON Employee to Tim;
  - Tim: CREATE VIEW V1 AS SELECT Emp#, Salary FROM Employee;
  - Tim: CREATE VIEW V2 (Emp#, Annual_Salary) AS SELECT Emp#, Salary*12 FROM Employee;
- Tim can exercise on V1 all privileges he has on relation Employee, that is, Select, Insert, Update
- However, Tim can exercise on V2 only the privileges of Select and Update on column Emp#; update operation is not defined on column Annual_Sal of V2

Authorizations on views

- It is possible to grant authorizations on a view: the privileges that a user can grant are those that he/she owns with grant option on the base tables
  - Tim cannot grant any authorization on views V1 and V2 he has defined, because he does not have the authorizations with grant option on the base table
- Consider the following sequence of commands:
  - Bob: GRANT Select ON Employee TO Tim WITH GRANT OPTION;
  - Bob: GRANT Update, Insert ON Employee TO Tim;
  - Tim: CREATE VIEW V4 AS SELECT Emp#, Salary FROM Employee;

  Authorizations of Tim on V4:

  - Select with Grant Option;
  - Update, Insert without Grant Option;

DAC Summary

- Advantages:
  - Intuitive
  - Easy to implement
- Disadvantages:
  - Maintenance of ACL or Capability lists
  - Maintenance of Grant/Revoke