Access Control

Access Control

- **Access control**: ensures that all *direct accesses* to object are authorized – a scheme for mapping users to allowed actions
  - **Protection objects**: system resources for which protection is desirable, e.g., memory, file, directory, hardware resource, software resources, **tables, tuples**, …
  - **Subjects**: active entities requesting accesses to resources, e.g., user, owner, program, etc.
  - **Access mode**: type of access, e.g., read/select, write/update, execute
- 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 from modification

Access Control

- Access control requirement:
  - Cannot be bypassed
  - Enforce least-privilege and need-to-know restrictions
  - Enforce organizational policy
- 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

Authorization Management

Who can grant and revoke access rights?

- *Centralized* administration: security officer
- *Decentralized* administration: locally autonomous systems
- *Hierarchical decentralization*: security officer > departmental system administrator > Windows NT administrator
- *Ownership based*: owner of data may grant access to other to his/her data (possibly with grant option)
- *Cooperative authorization*: concurrence of several authorizers

Access Control

- Discretionary access control (DAC)
  - An individual user can set the policy
- Mandatory access control (MAC)
  - The policy is built into the system
  - The user cannot modify it
- Role-based access control (RBAC)

Discretionary Access Control

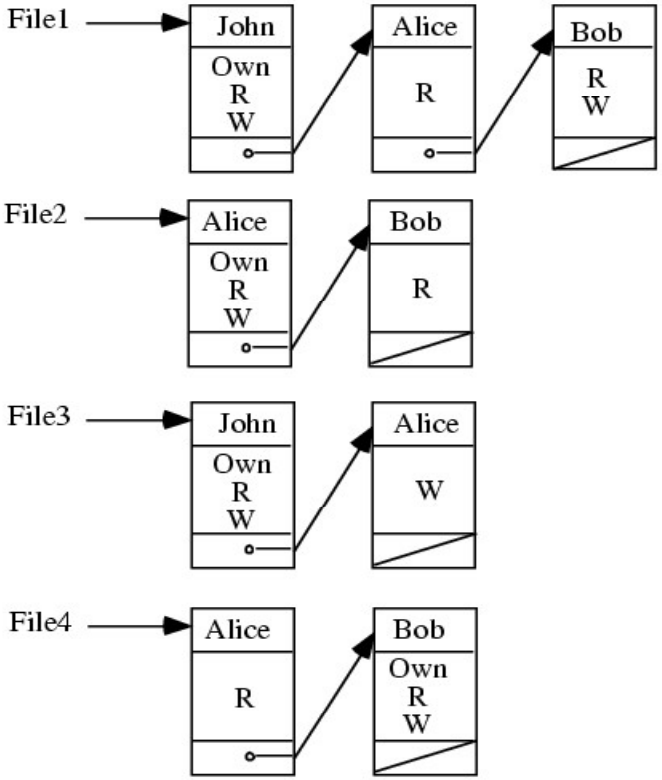
Discretionary Access Control

- 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

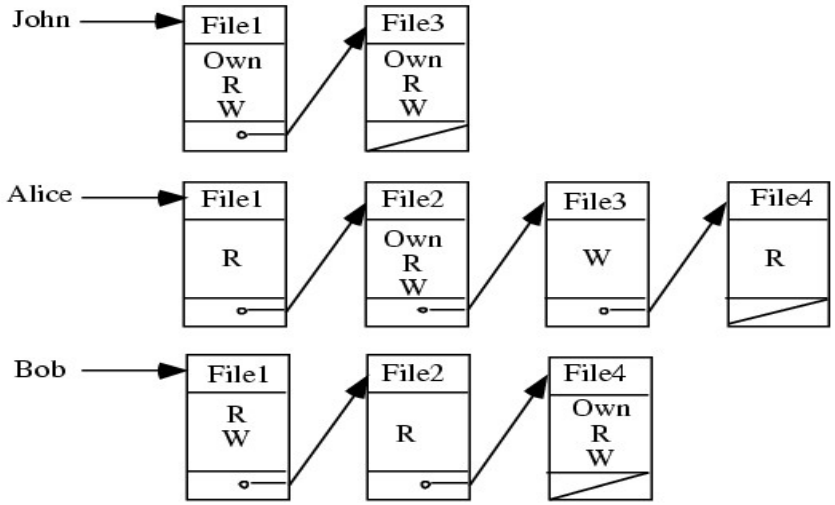
DAC – Access Matrix

	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

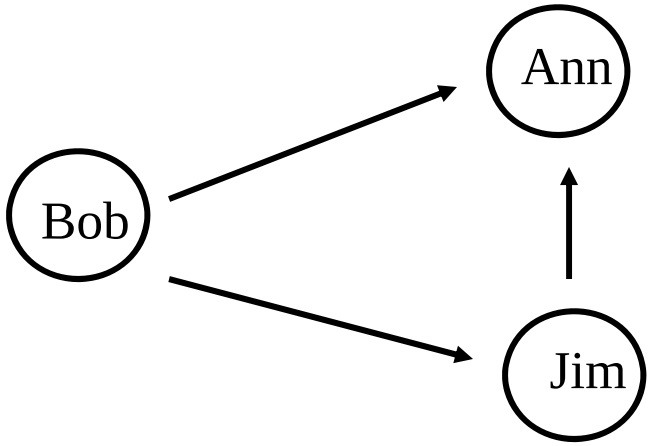
REVOKE *PrivilegeList* | ALL[PRIVILEGES]
ON *Relation* | *View*
FROM *UserList* | PUBLIC

- 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)

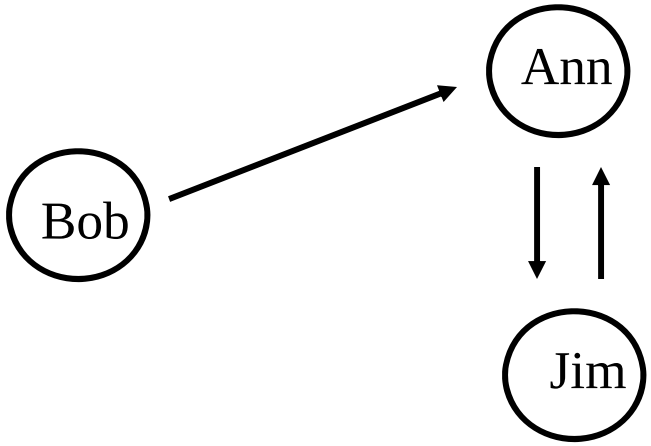


Assume all grant statements are with grant option

But …

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

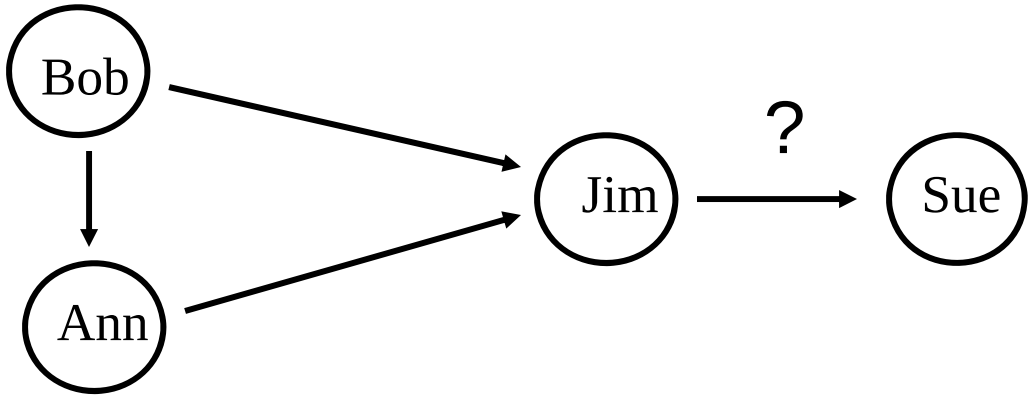
- grant(Bob, Ann)
- grant(Ann, Jim)
- grant(Jim,Ann)
- revoke(Bob, Ann)



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)



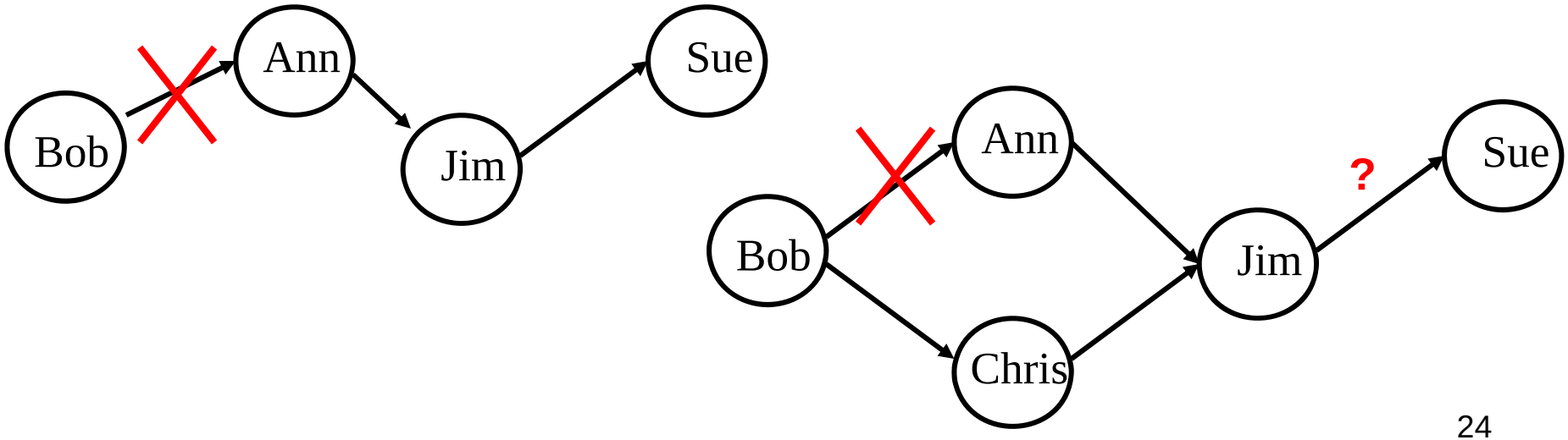
Revoke operation

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

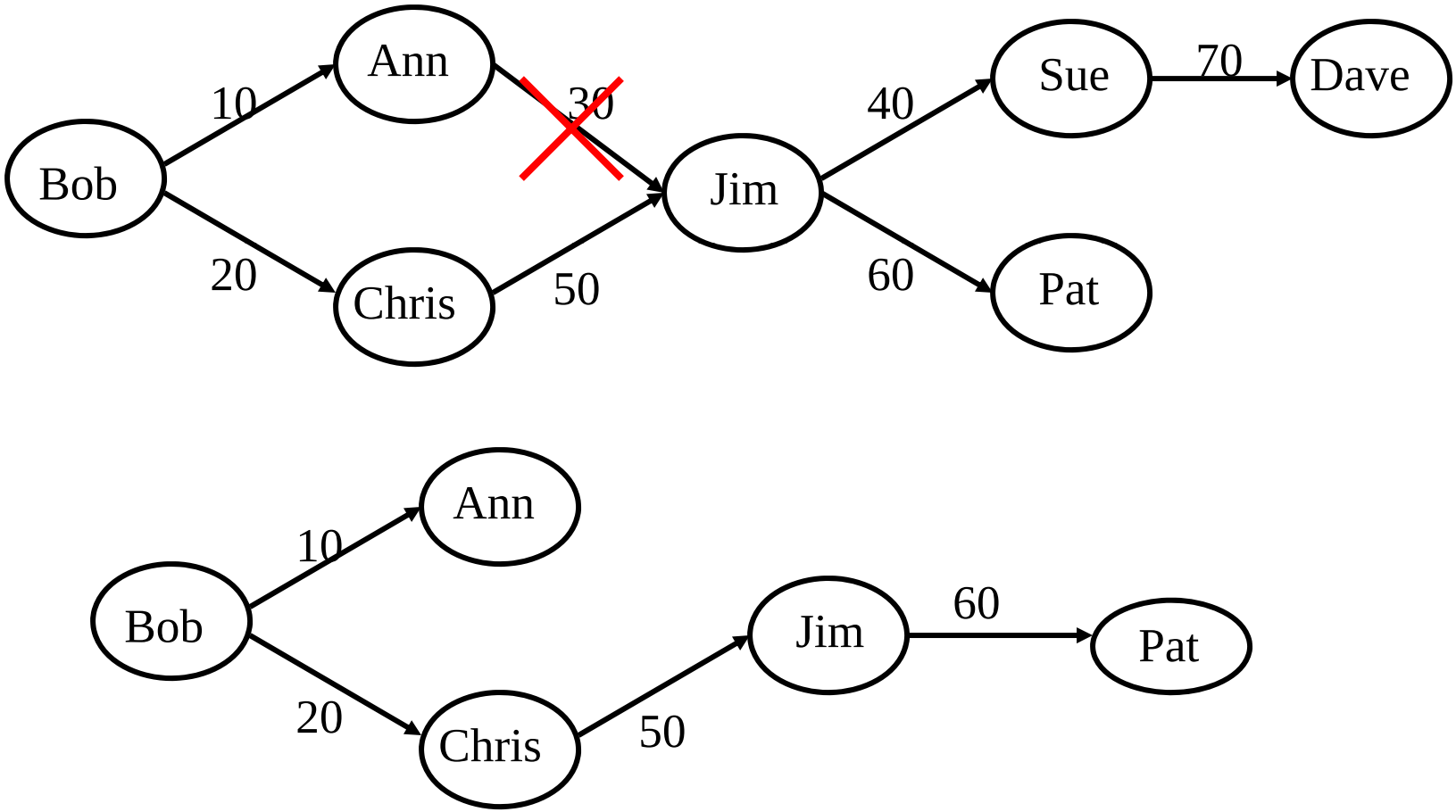
- Let G_1, …., G_n be a sequence of grant operations with a single privilege on the same relations, such that i,k = 1,…., n, if i<k, then G_i is executed before G_k. Let R_i be the revoke operation for the privilege granted with operation G_i.
- The semantics of the recursive revoke requires that the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

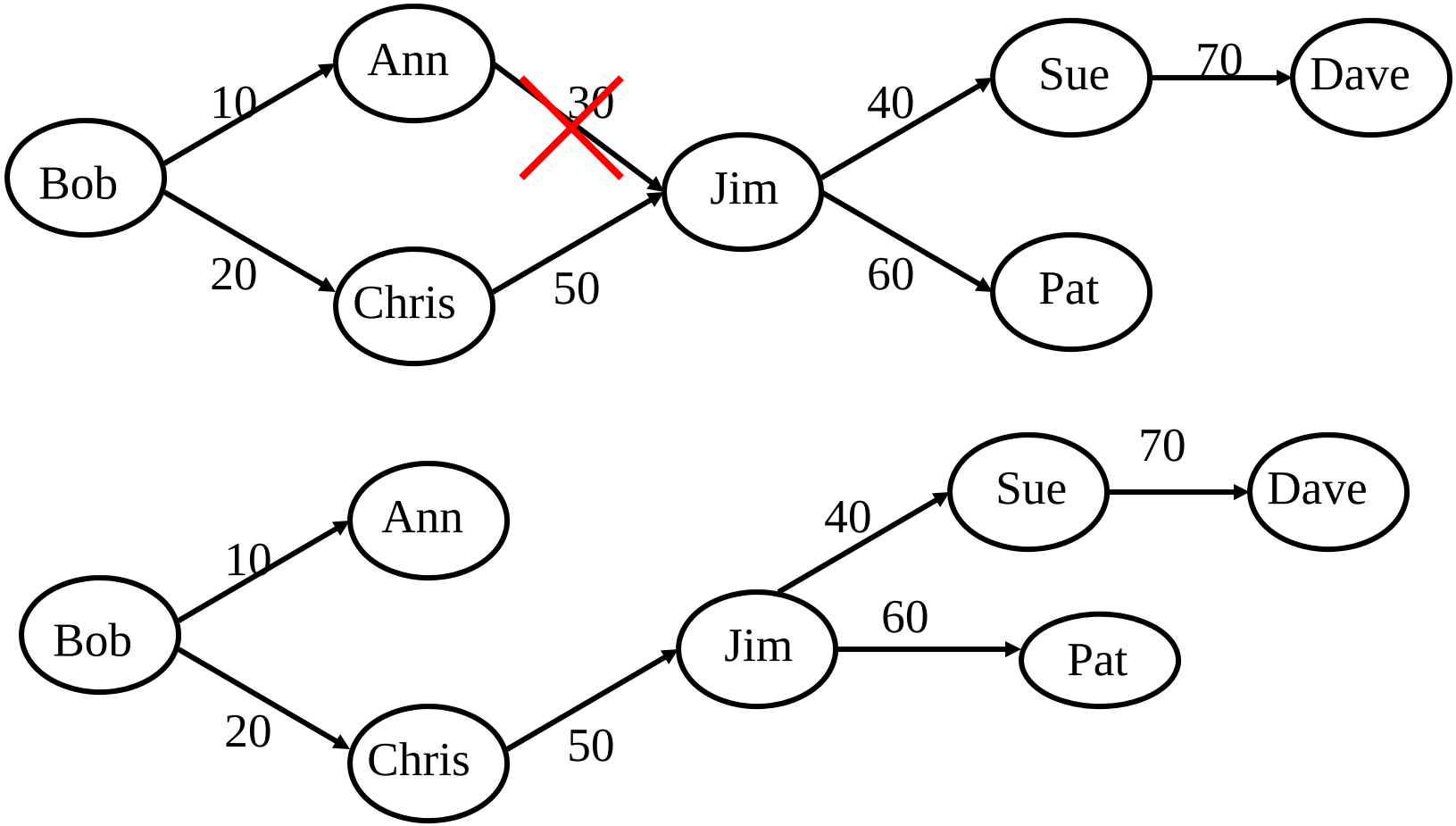
Recursive Revocation



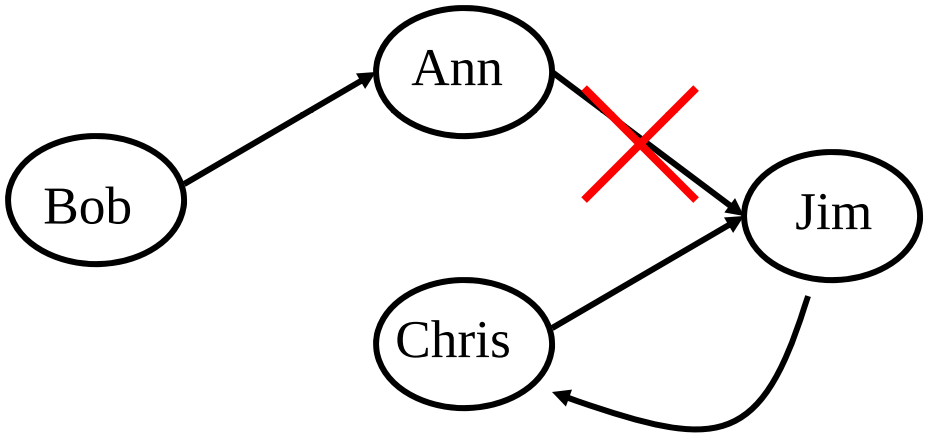
Recursive revocation

- 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



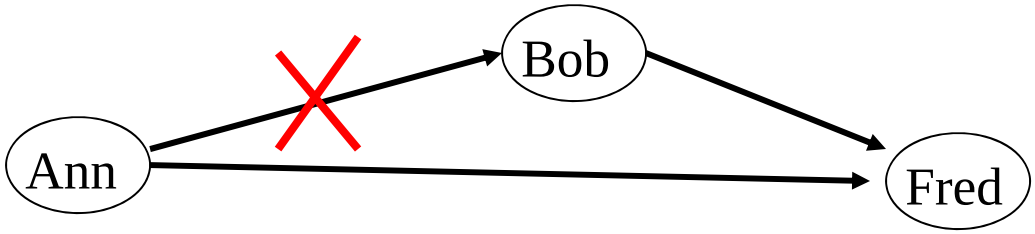
?????

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

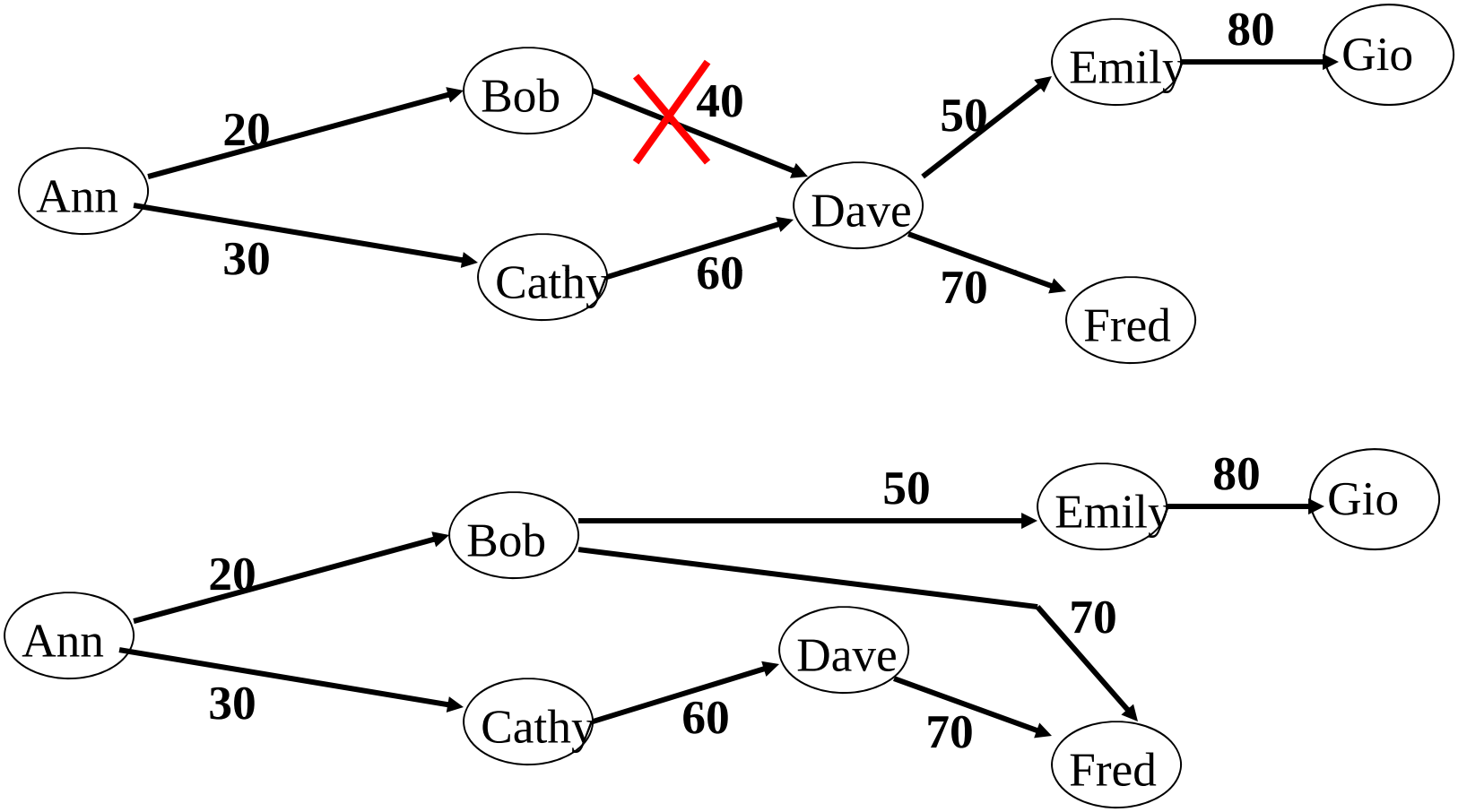
- A user can revoke a privilege on a table from another user without entailing automatic revocation of the authorizations for the privilege on the table the latter may have granted
- Instead of deleting the authorizations the revokee may have granted by using the privilege received by the revoker, all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



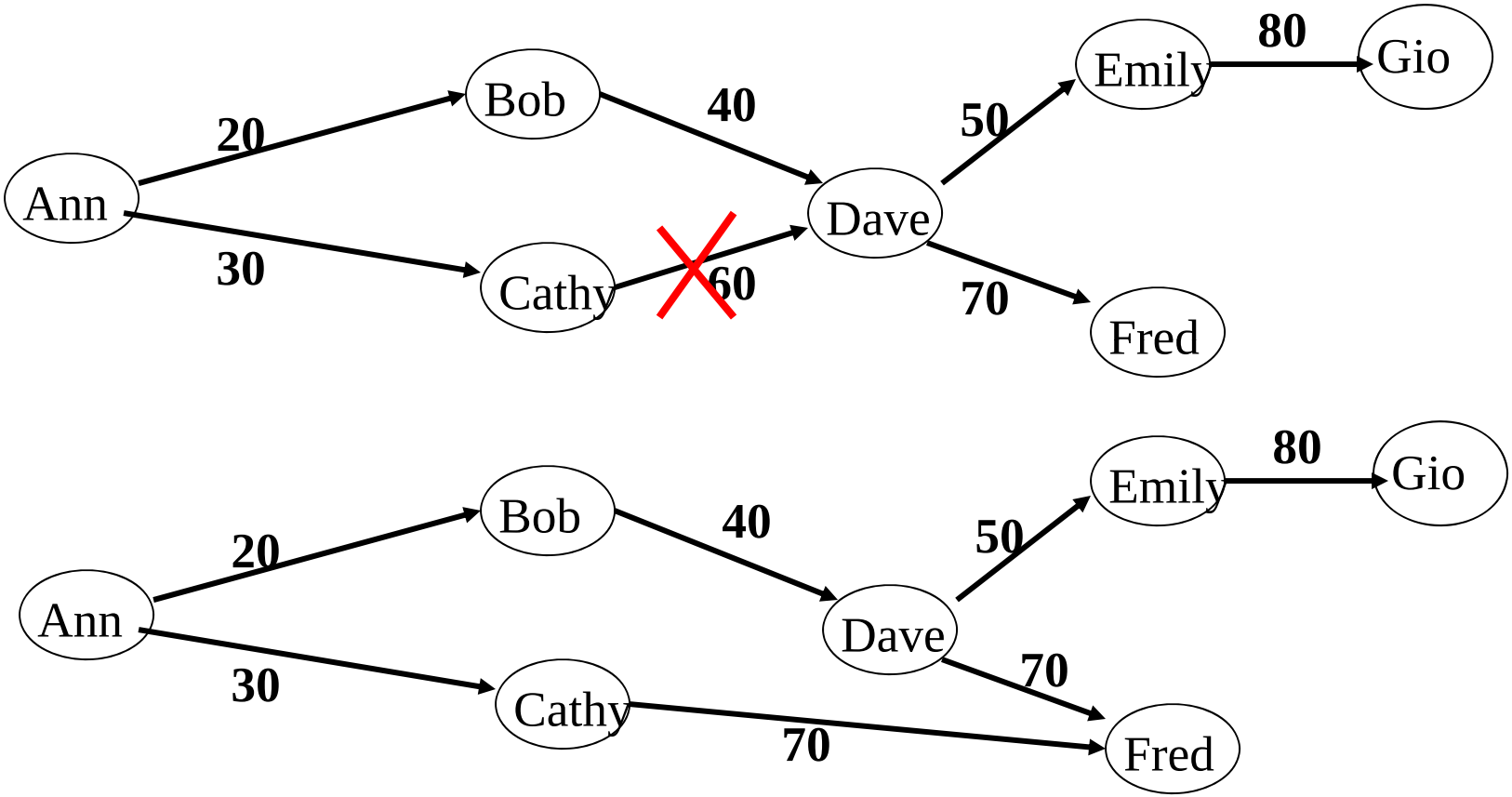
Noncascading Revoke

- 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