CS5322 Database Security

Access Control

- Purpose: Ensure that all *direct* accesses to objects are authorized
  - a scheme for mapping subjects to allowed actions
- Objects:
  - System resources to be protected – tables, tuples, etc.
- Subjects:
  - Entities requesting accesses to resources – users, programs, etc.
- Access mode:
  - Actions allowed: Read, write, etc.

Access Control Requirements

- Cannot be bypassed
- Enforce organizational policies
- Enforce least-privilege restrictions
  - Least-privilege
    - Every subject should be given the minimum set of access rights necessary to complete the job
    - Access rights should be added as needed and discarded after use
    - This limits the damage that can result from an accident or error
    - And it limits the number of privileged programs that could be compromised

Access Control

- Mandatory access control (MAC)
  - The system decides who can access what
  - Users cannot modify it
- Discretionary access control (DAC)
  - If a user has a certain access right, it is possible for him/her to grant it to another user
- Role-based access control (RBAC)
  - The system gives access rights to roles
  - Users are assigned to roles

Coming Next

- Discretionary Access Control

Discretionary Access Control (DAC)

- DAC policies govern the access of *subjects* to *objects*, based on
  - subjects' identifies,
  - objects' identifies, and
  - permissions
- Discretionary:
  - A subject may grant access rights to other subjects at his/her discretion

Discretionary Access Control in Commercial Databases

- All commercial database systems support DAC
- Current DAC models for relational databases 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): 242 – 255, 1976

System R Authorization Model

- Objects to protect:
  - Tables and views
- Privileges:
  - Select, update, insert, delete, drop, index (only for tables), alter (only for tables), …
- A user can grant privilege to another user and the latter can grant it to others
  - This is done via the *grant option*

GRANT Operation

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

- Some possible privileges:
  - select, insert, update, delete, references, trigger
- Privileges apply to all tuples in the relations or views
  - Not possible to specify a subset of tuples
- For the select, update, and references privileges, one can specify the columns to which they applies

GRANT Operation – Example

- Bob:
  - GRANT select, insert ON Employee
    TO Ann
    WITH GRANT OPTION
- Ann receives (from Bob) select and insert privileges on Employee
- She may grant these privileges to others
- Ann:
  - GRANT select (Name), insert ON Employee
    TO Jim
- Jim receives (from Ann) select Employee.Name and insert privileges, but cannot grant them to others

GRANT Operation – Example

- Bob:
  - GRANT select, insert ON Employee TO Ann WITH GRANT OPTION
- Ann:
  - GRANT select (Name), insert ON Employee TO Jim
- Bob:
  - GRANT select ON Employee TO Jim WITH GRANT OPTION
- Now Jim can grant the select privilege to others, but not the insert privilege

Checking GRANT Operation

- The system has an authorization catalog that keeps track of the privileges that each user can grant
- Whenever a user executes a GRANT operation, the system checks whether the privilege can indeed be granted
  - If not, then the corresponding privilege will not be granted

Checking GRANT Operation

- Consider the following command sequence, assuming that Bob owns the table Employee
  - 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;
- What are the results?

Checking GRANT Operation

- Consider the following command sequence, assuming that Bob owns the table Employee
  - 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;
- What are the results?
- The first three commands are fully executed
- The fourth command is not executed
- The last command is not executed

REVOKE Operation

REVOKE *PrivilegeList* | ALL PRIVILEGES

ON *Relation* | *View*

FROM *UserList* | PUBLIC

- Example:
  - Bob:
    - GRANT select, insert ON Employee TO Ann WITH GRANT OPTION
    - REVOKE select ON Employee FROM Ann
  - Result: Ann only retains her insert privilege

REVOKE Operation

REVOKE *PrivilegeList* | ALL PRIVILEGES

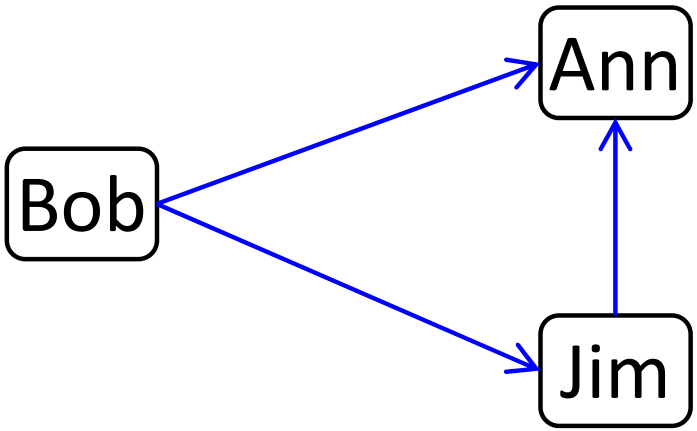
ON *Relation* | *View*

FROM *UserList* | PUBLIC

- In general, revocations can be tricky due to "WITH GRANT OPTION"
- Why?
  - Image that Bob grants Ann, and then Ann grants Jim, and then Bob revokes it from Ann…
- We will discuss this issue in the next few slides

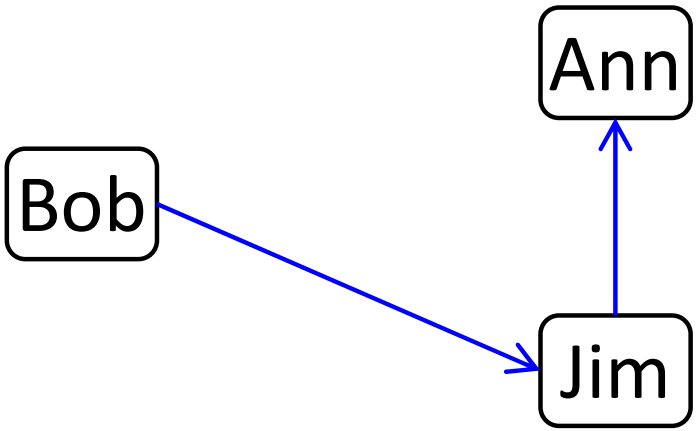
Grants from Multiple Sources

- Bob is the owner
- Bob grants Ann select WITH GRANT OPTION
- Bob grants Jim select WITH GRANT OPTION
- Jim grants Ann select WITH GRANT OPTION
- Bob revokes select from Ann



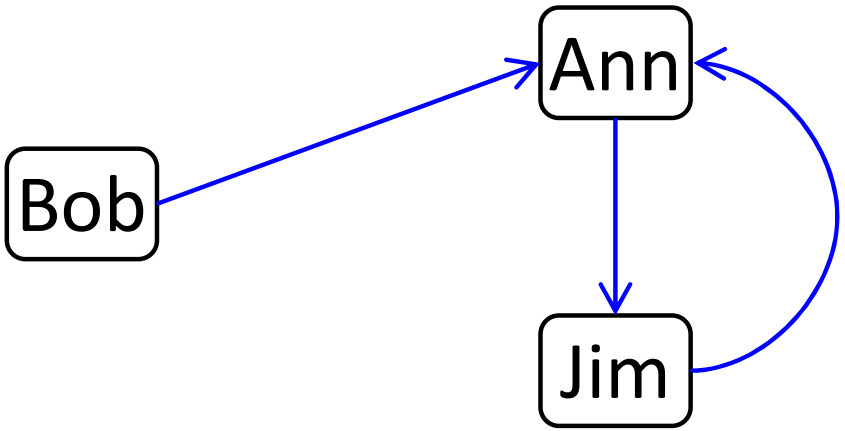
Grants from Multiple Sources

- Bob is the owner
- Bob grants Ann select WITH GRANT OPTION
- Bob grants Jim select WITH GRANT OPTION
- Jim grants Ann select WITH GRANT OPTION
- Bob revokes select from Ann
- What is the result?
- Ann still has select, due to Jim



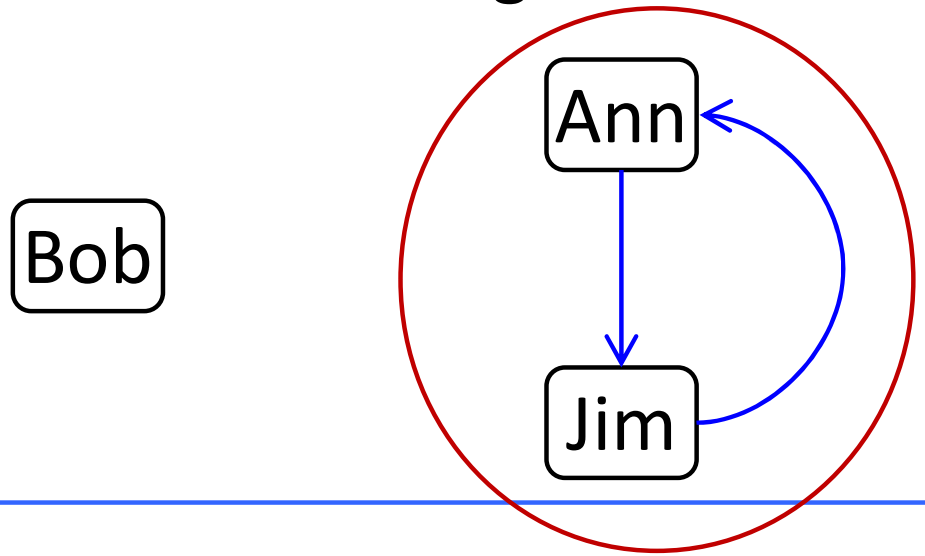
Grants from Multiple Sources

- Bob grants Ann select WITH GRANT OPTION
- Ann grants Jim select WITH GRANT OPTION
- Jim grants Ann select WITH GRANT OPTION
- Bob revokes select from Ann
- What is the result?
- We know for sure that Bob→Ann should be removed

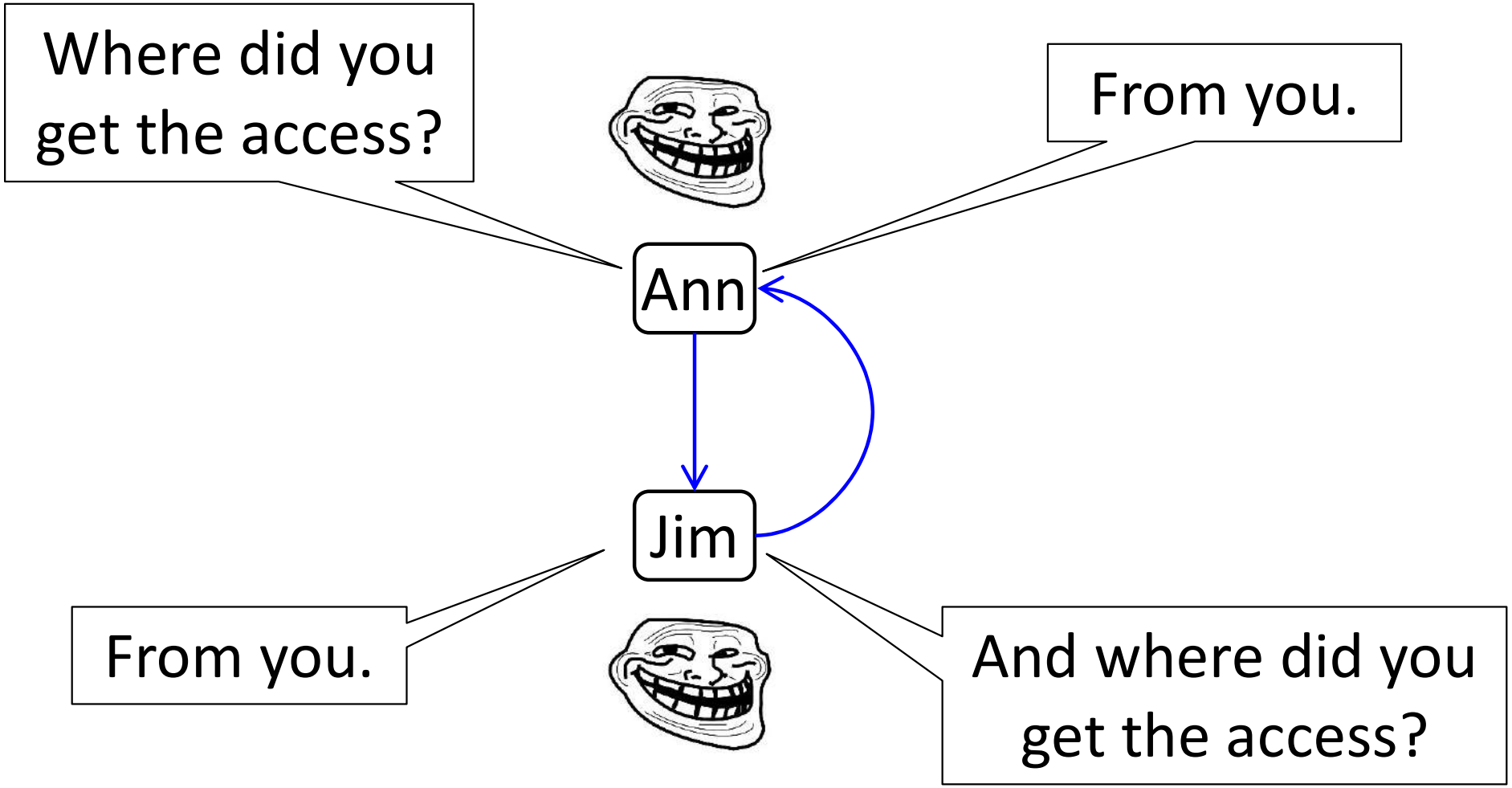


Grants from Multiple Sources

- Bob grants Ann select WITH GRANT OPTION
- Ann grants Jim select WITH GRANT OPTION
- Jim grants Ann select WITH GRANT OPTION
- Bob revokes select from Ann
- What is the result?
- We know for sure that Bob→Ann should be removed
- But what about the remaining ones?



If we retain Ann and Jim's privileges…



What was the problem?

- Command sequence
  - Bob is the owner
  - Bob grants Ann select WITH GRANT OPTION
  - Ann grants Jim select WITH GRANT OPTION
  - Jim grants Ann select WITH GRANT OPTION
  - Bob revokes select from Ann
- Observe that Jim's privilege was "derived" from Ann's privilege
- When Bob revokes Ann's privilege, Jim's privilege should also be revoked
- This is referred to as *recursive revocation*

Formalization of Recursive Revocation

- Let $G_1, \ldots, G_n$ be a sequence of GRANT operations.
- Let R_i be the revoke operation for the privileges granted by operation G_i.
- Then, the effect of

  $G_1, \ldots, G_n, R_i$

  should be identical to

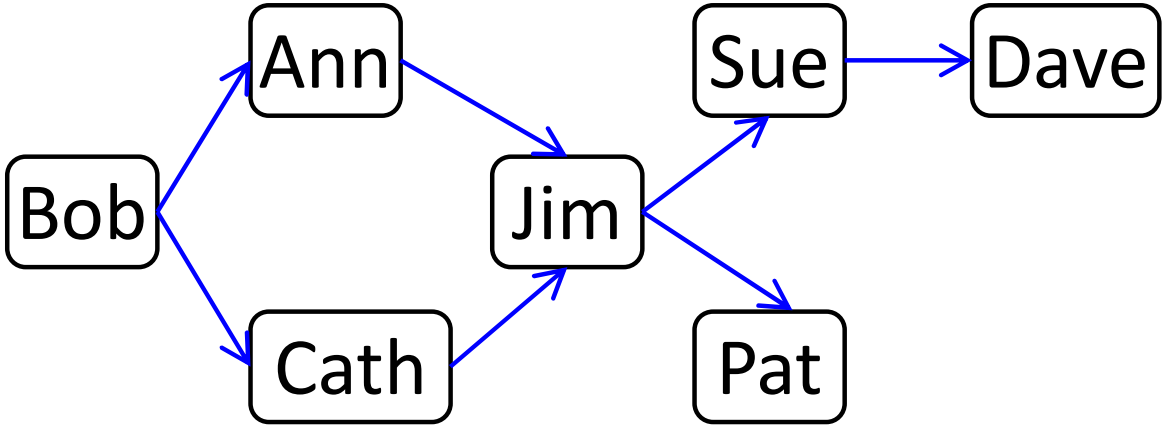
  $G_1, \ldots, G_{i-1}, G_{i+1}, \ldots, G_n$
- In other words, R_i should "erase" G_i from the history

Formalization of Recursive Revocation – Example

- Previous example
  - Bob is the owner
  - Bob grants Ann select WITH GRANT OPTION
  - Ann grants Jim select WITH GRANT OPTION
  - Jim grants Ann select WITH GRANT OPTION
  - Bob revokes select from Ann
- Recursive revocation requires that the effect of the above command sequence should be identical to
  - Bob is the owner
  - Ann grants Jim select WITH GRANT OPTION
  - Jim grants Ann select WITH GRANT OPTION

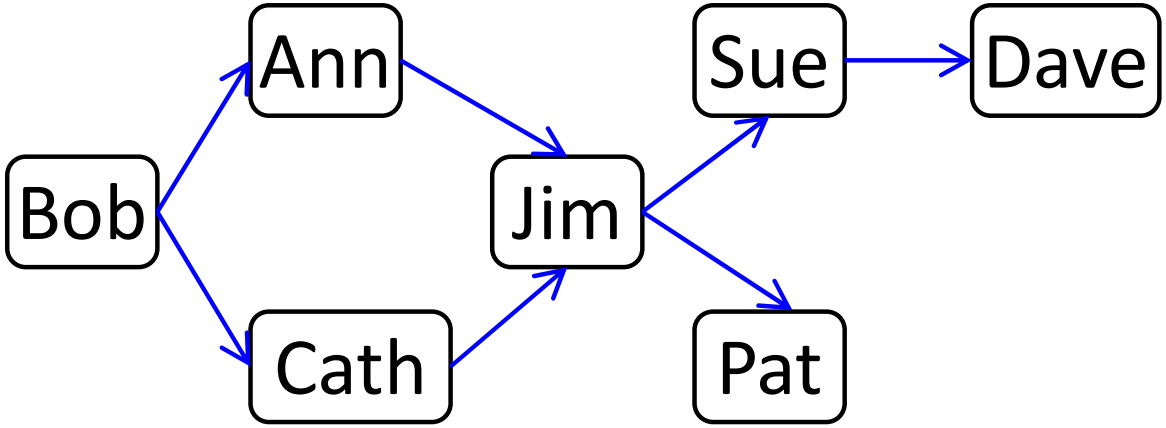
Another Example

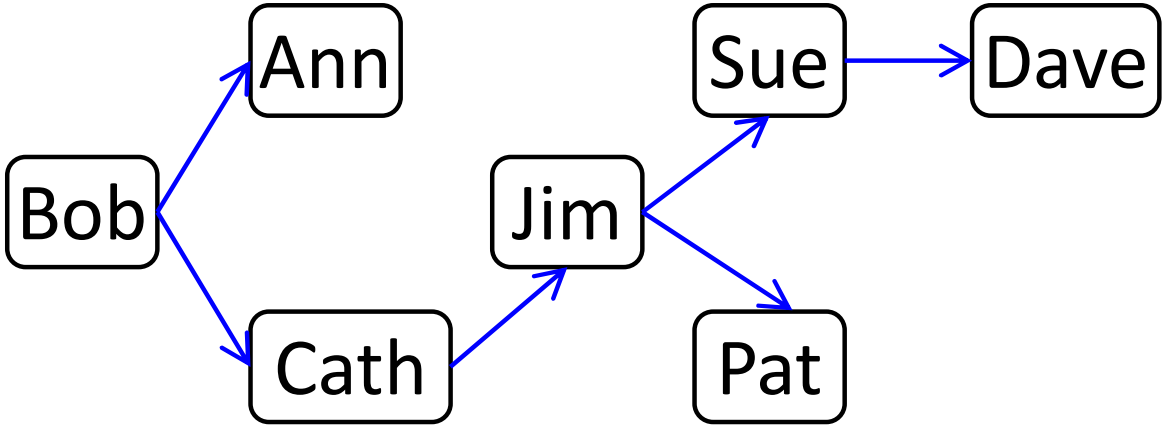
- Bob is the owner
- Bob grants Ann
- Bob grants Cath
- Ann grants Jim
- Jim grants Sue
- Cath grants Jim
- Jim grants Pat
- Sue grants Dave
- Ann revokes Jim
- Results?



Another Example

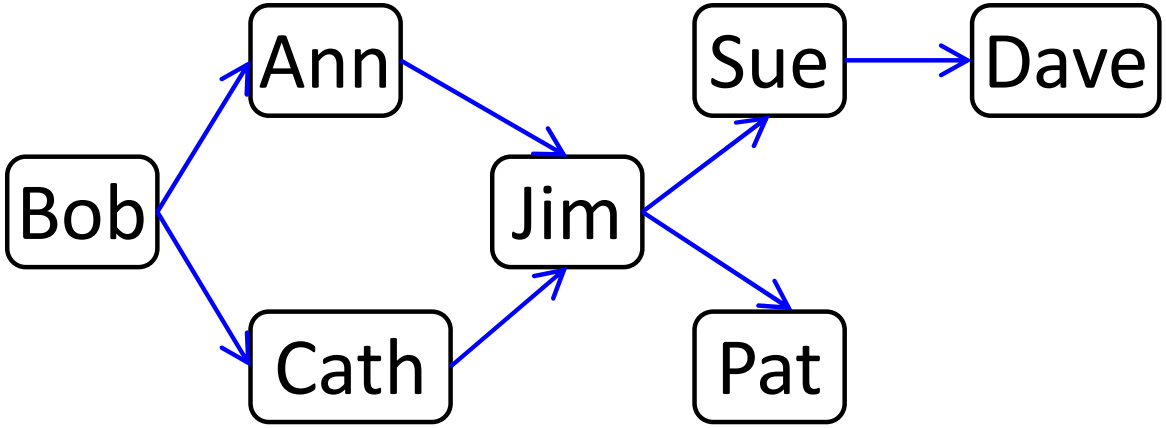
- Bob is the owner
- Bob grants Ann
- Bob grants Cath
- Ann grants Jim
- Jim grants Sue
- Cath grants Jim
- Jim grants Pat
- Sue grants Dave
- Ann revokes Jim
- Results?

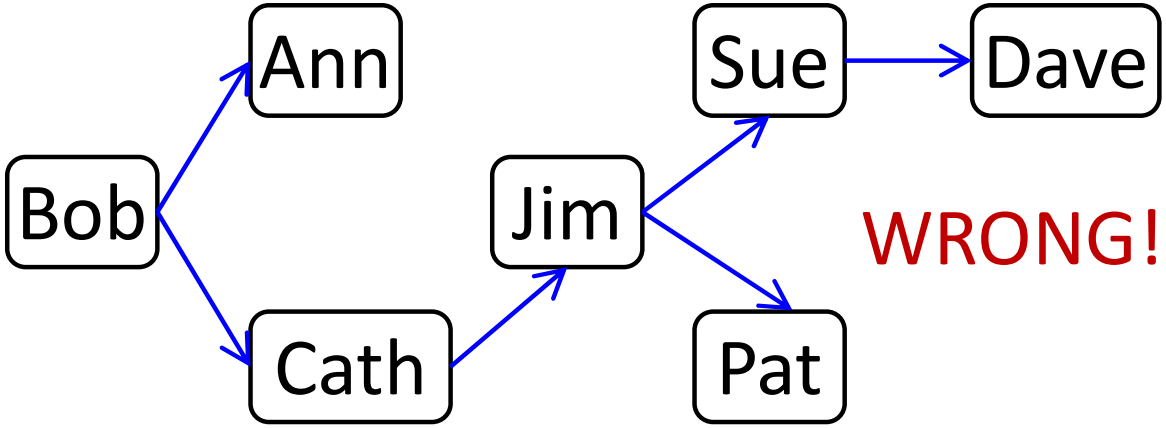




Another Example

- Bob is the owner
- Bob grants Ann
- Bob grants Cath
- Ann grants Jim
- Jim grants Sue
- Cath grants Jim
- Jim grants Pat
- Sue grants Dave
- Ann revokes Jim
- Results?



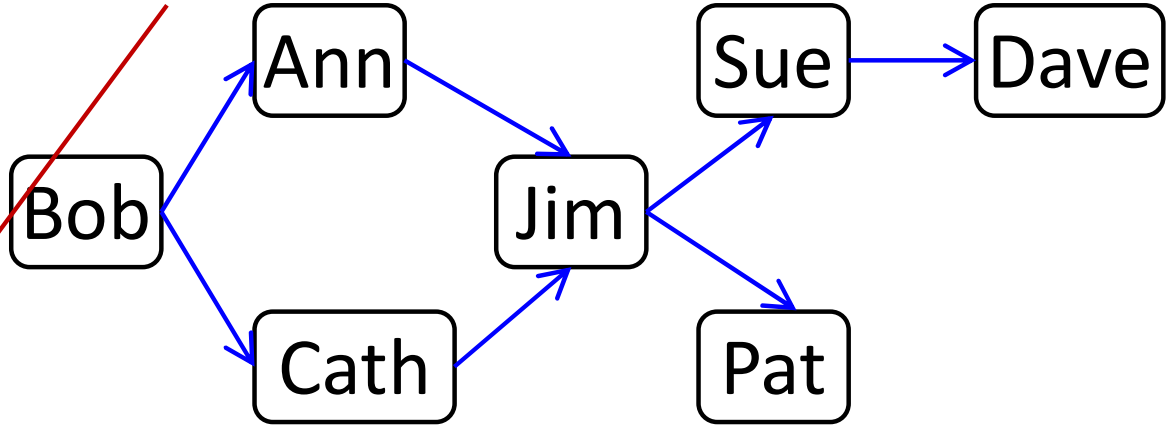


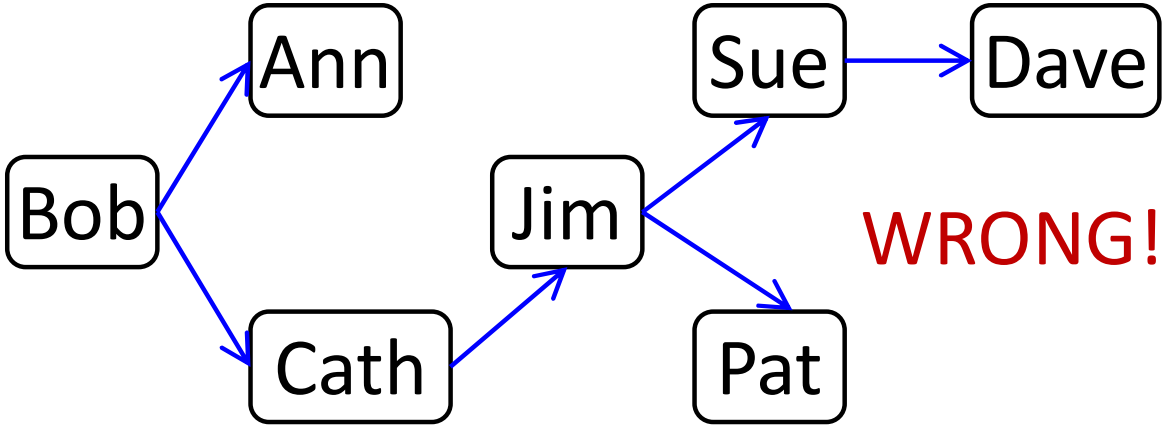
Why?

This command should be erased from history

In that case, how could Jim grant Sue?

- Bob is the owner
- Bob grants Ann
- Bob grants Cath
- Ann grants Jim
- Jim grants Sue
- Cath grants Jim
- Jim grants Pat
- Sue grants Dave
- Ann revokes Jim
- Results?





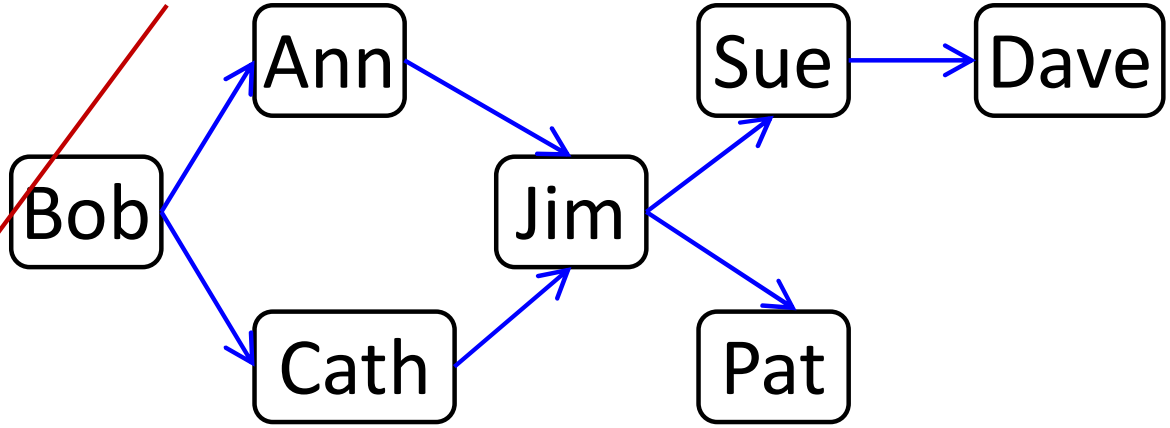
WRONG!

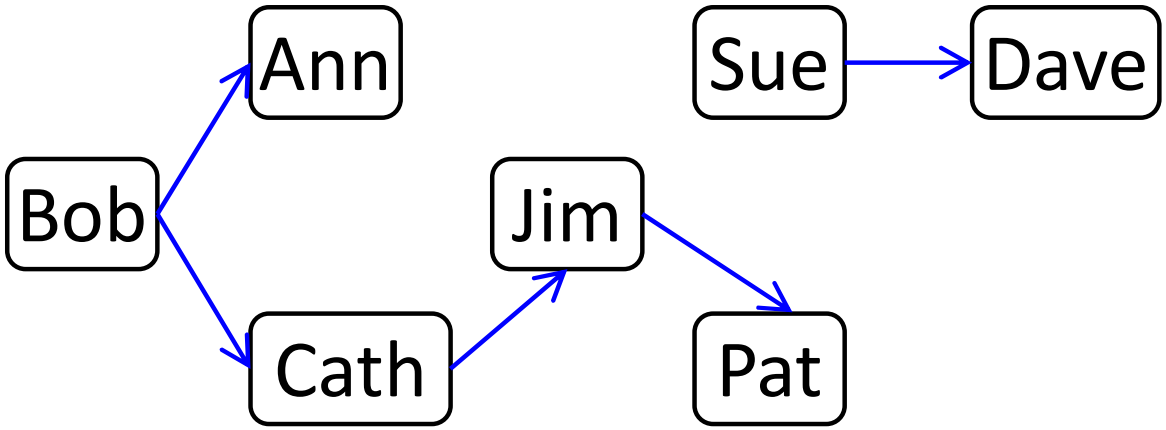
Why?

This command should be erased from history

In that case, how could Jim grant Sue?

- Bob is the owner
- Bob grants Ann
- Bob grants Cath
- Ann grants Jim
- Jim grants Sue
- Cath grants Jim
- Jim grants Pat
- Sue grants Dave
- Ann revokes Jim
- Results?



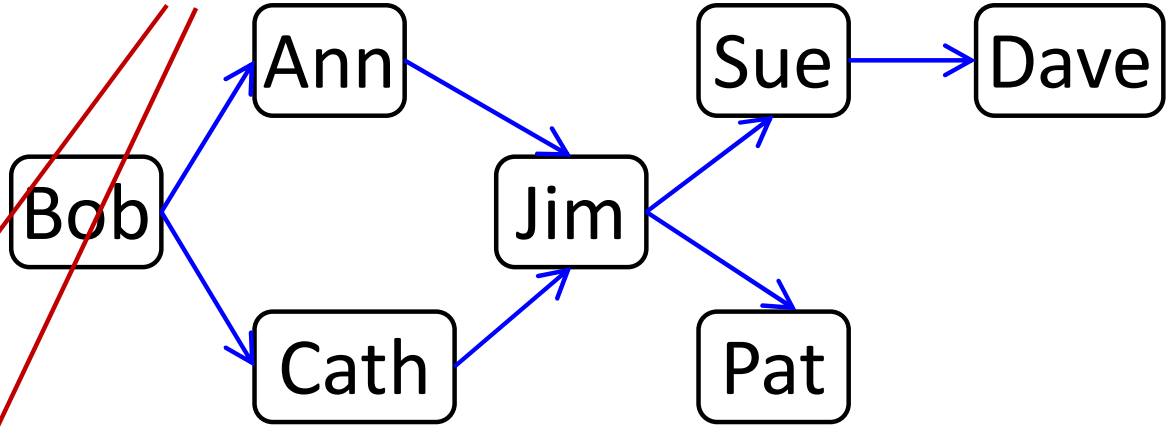


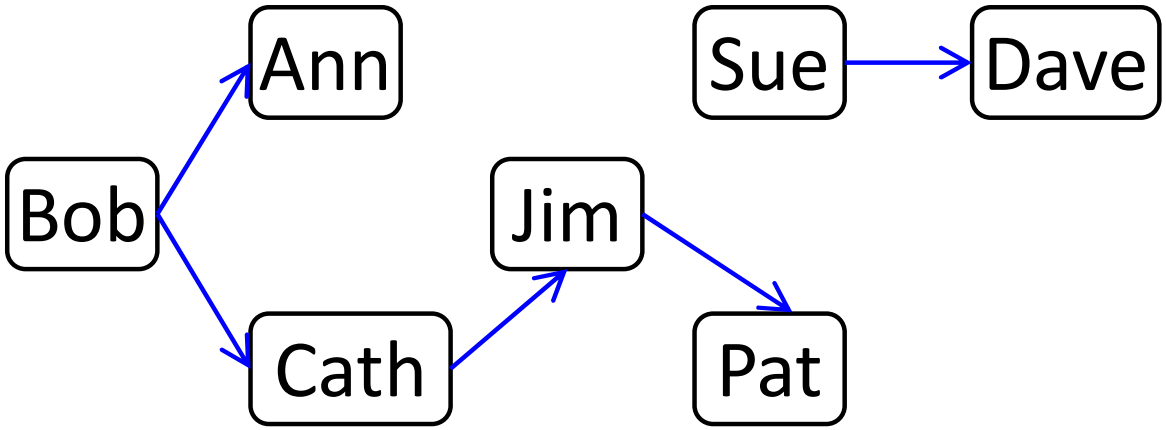
Why?

This command should be erased from history

In that case, how could Jim grant Sue?

- Bob is the owner
- Bob grants Ann
- Bob grants Cath
- Ann grants Jim
- Jim grants Sue
- Cath grants Jim
- Jim grants Pat
- Sue grants Dave
- Ann revokes Jim
- Results?



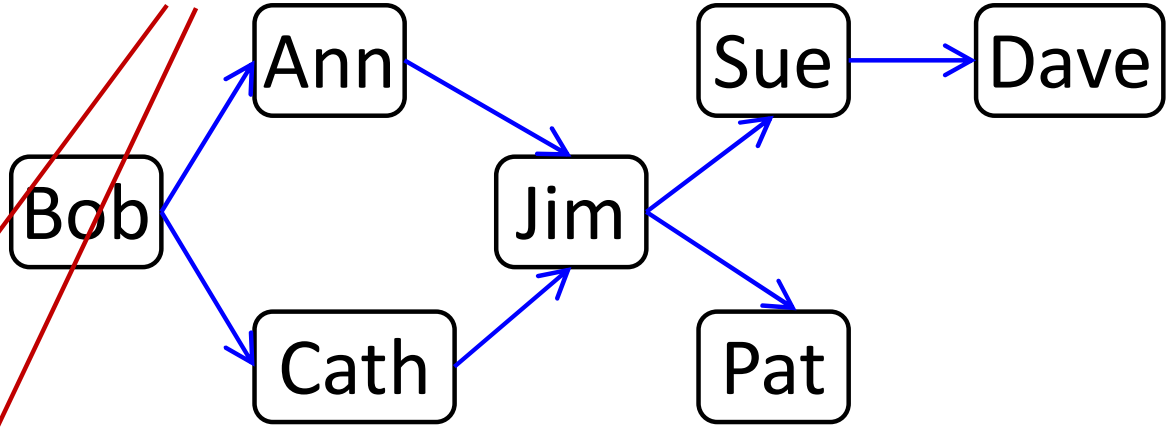


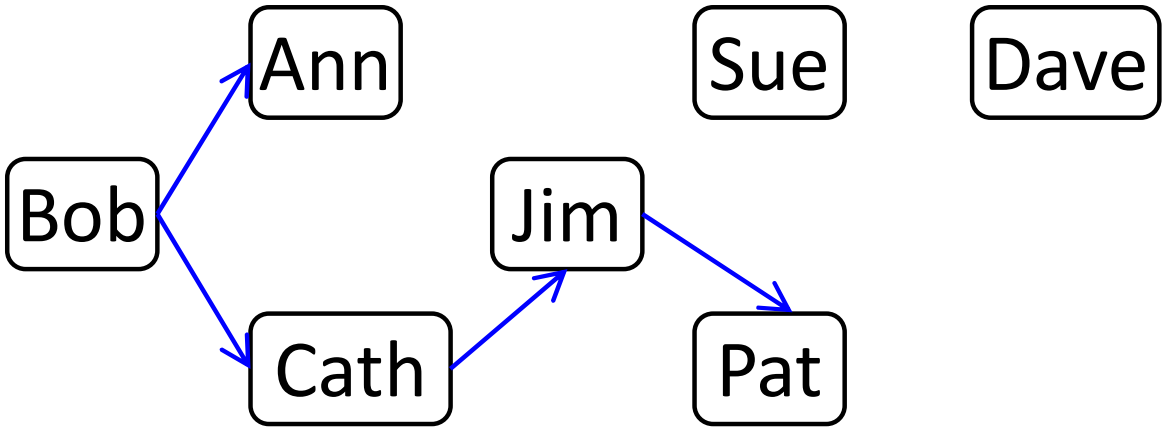
Why?

This command should be erased from history

In that case, how could Jim grant Sue?

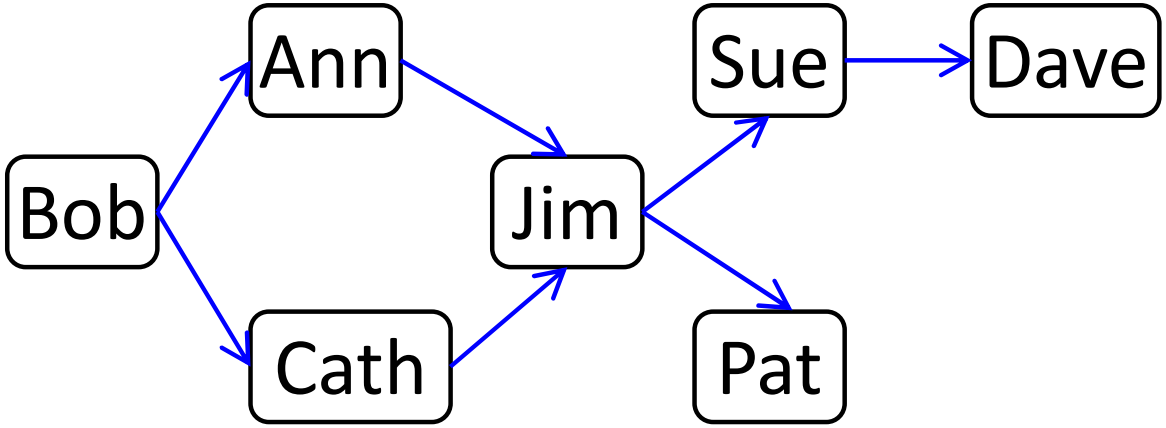
- Bob is the owner
- Bob grants Ann
- Bob grants Cath
- Ann grants Jim
- Jim grants Sue
- Cath grants Jim
- Jim grants Pat
- Sue grants Dave
- Ann revokes Jim
- Results?

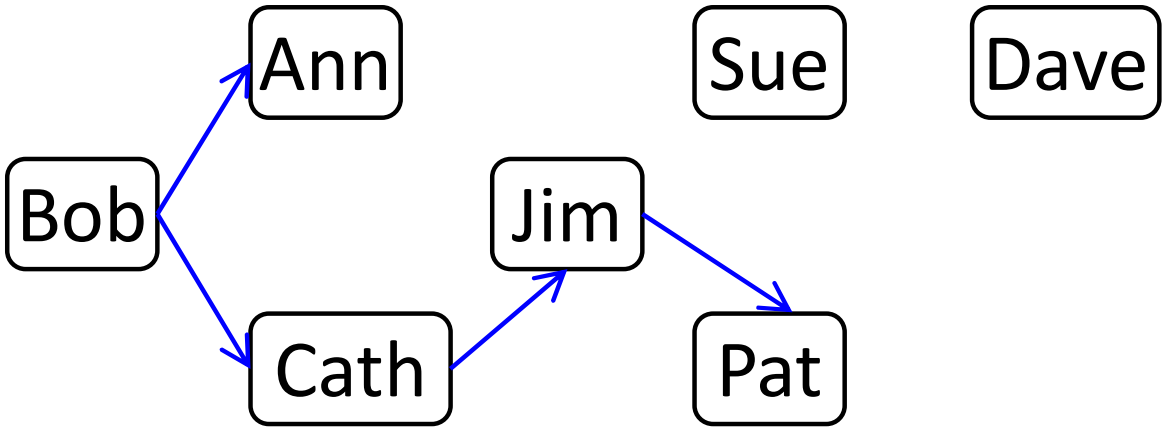




Why?

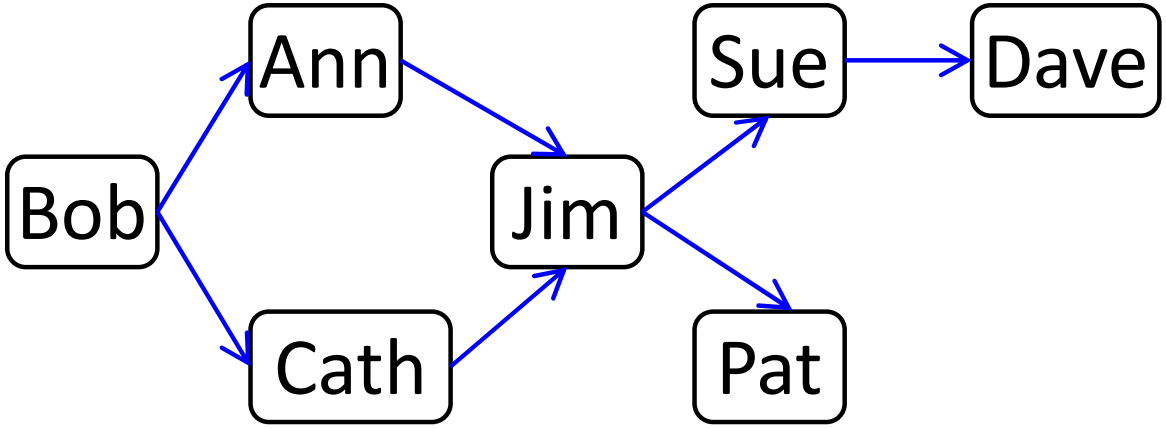
- Bob is the owner
- Bob grants Ann
- Bob grants Cath
- ~~Ann grants Jim~~
- ~~Jim grants Sue~~
- Cath grants Jim
- Jim grants Pat
- ~~Sue grants Dave~~
- ~~Ann revokes Jim~~
- Results?

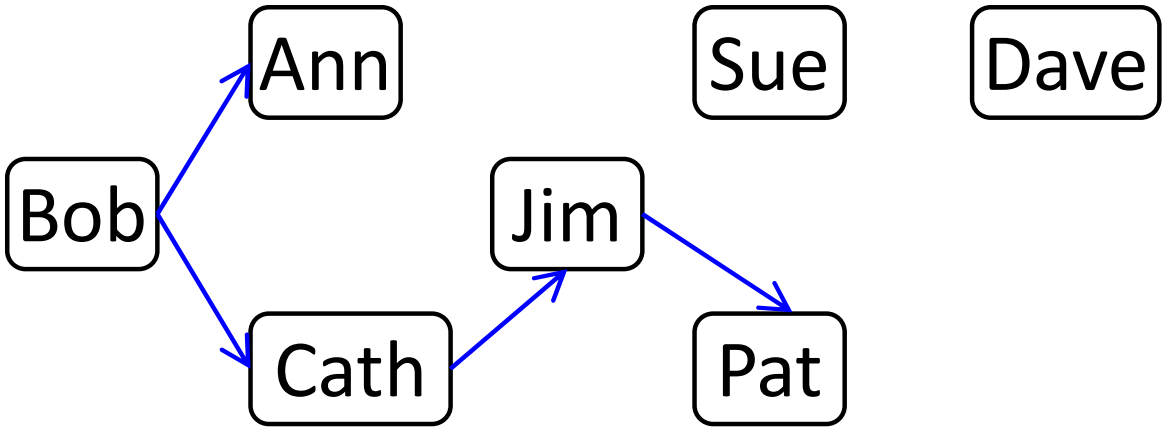




Why?

- Bob is the owner
- Bob grants Ann
- Bob grants Cath
- ~~Ann grants Jim~~
- ~~Jim grants Sue~~
- Cath grants Jim
- Jim grants Pat ?
- ~~Sue grants Dave~~
- ~~Ann revokes Jim~~
- Results?

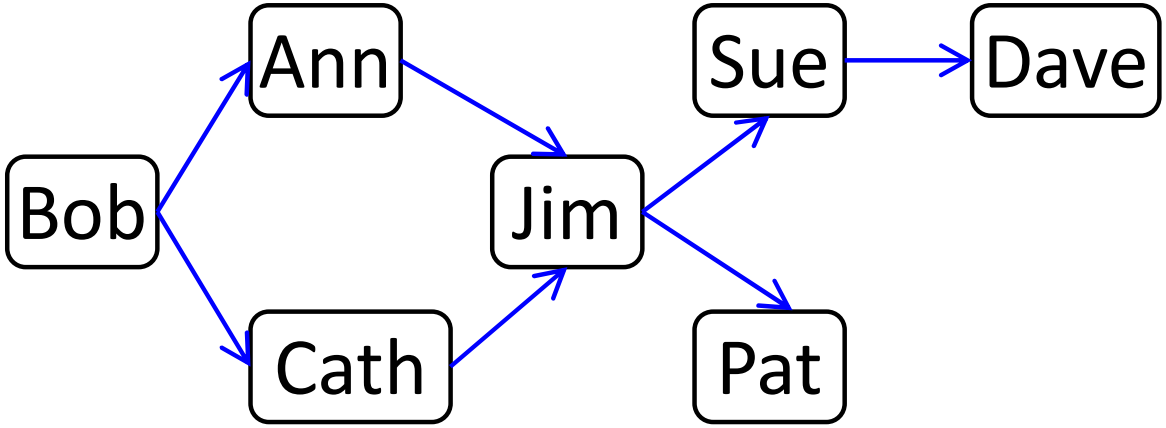


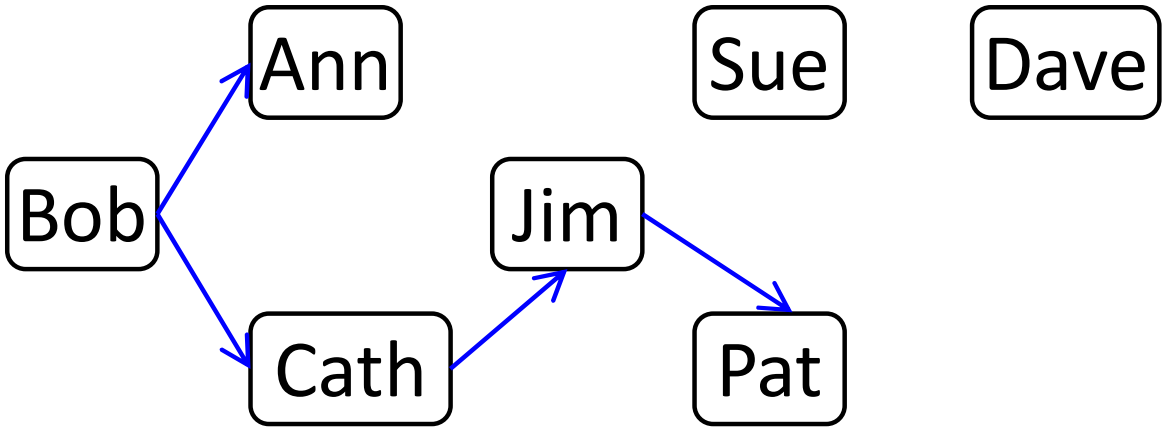


Why?

It is OK because Cath has granted Jim at this point of time

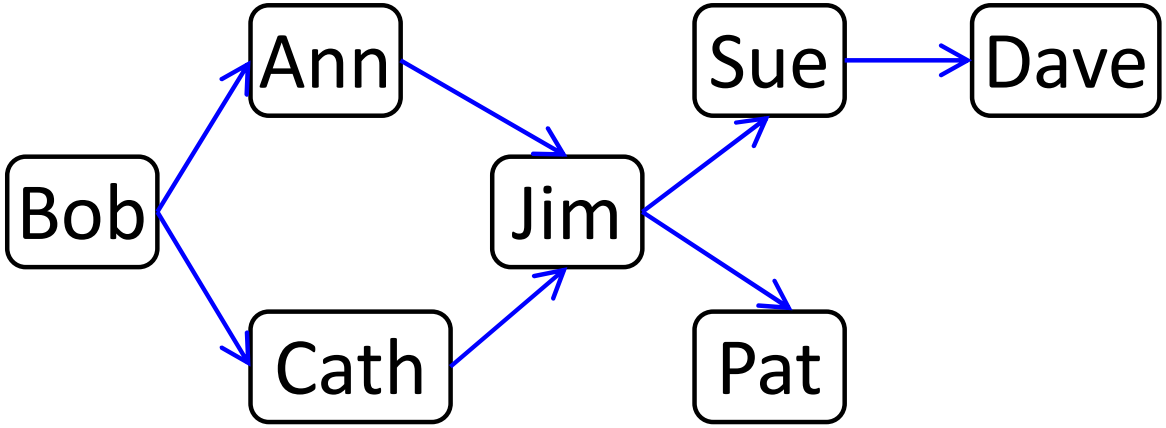
- Bob is the owner
- Bob grants Ann
- Bob grants Cath
- ~~Ann grants Jim~~
- ~~Jim grants Sue~~
- Cath grants Jim
- Jim grants Pat ?
- ~~Sue grants Dave~~
- ~~Ann revokes Jim~~
- Results?

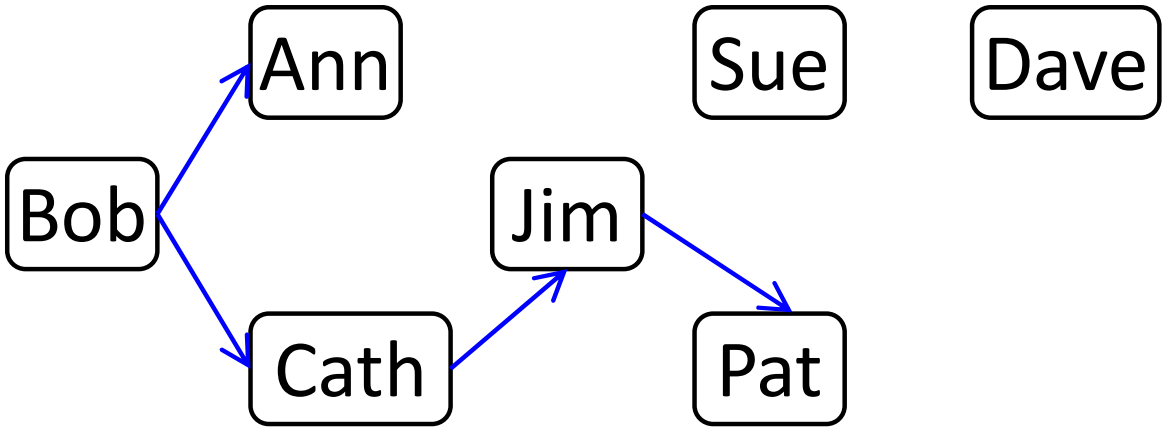




How can we implement such delicate controls?

- Bob is the owner
- Bob grants Ann
- Bob grants Cath
- ~~Ann grants Jim~~
- ~~Jim grants Sue~~
- Cath grants Jim
- Jim grants Pat
- ~~Sue grants Dave~~
- ~~Ann revokes Jim~~
- Results?



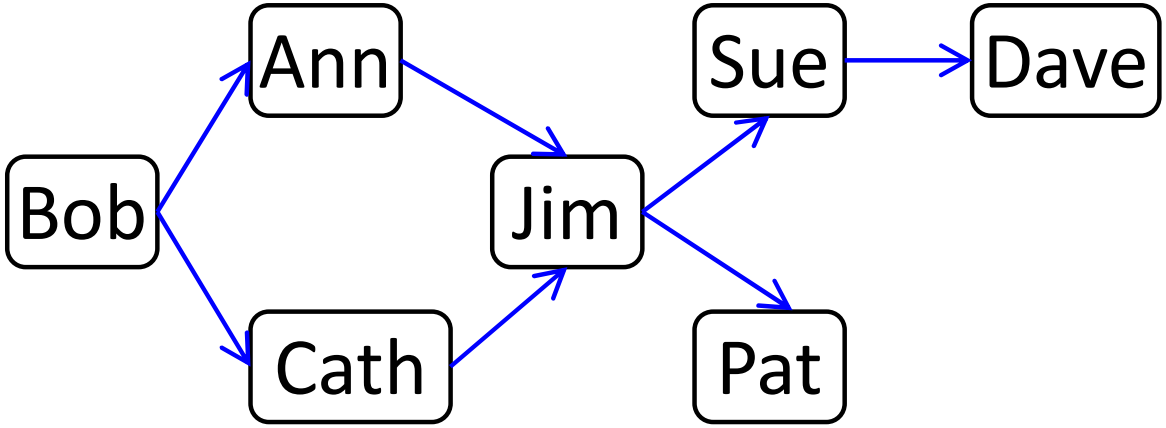


System R's Implementation

- Record that *timestamp* that each privilege is granted
- When a privilege is revoked, use the timestamps to decide whether recursive revocation is needed

Previous Example

- Bob is the owner
- Bob grants Ann
- Bob grants Cath
- Ann grants Jim
- Jim grants Sue
- Cath grants Jim
- Jim grants Pat
- Sue grants Dave

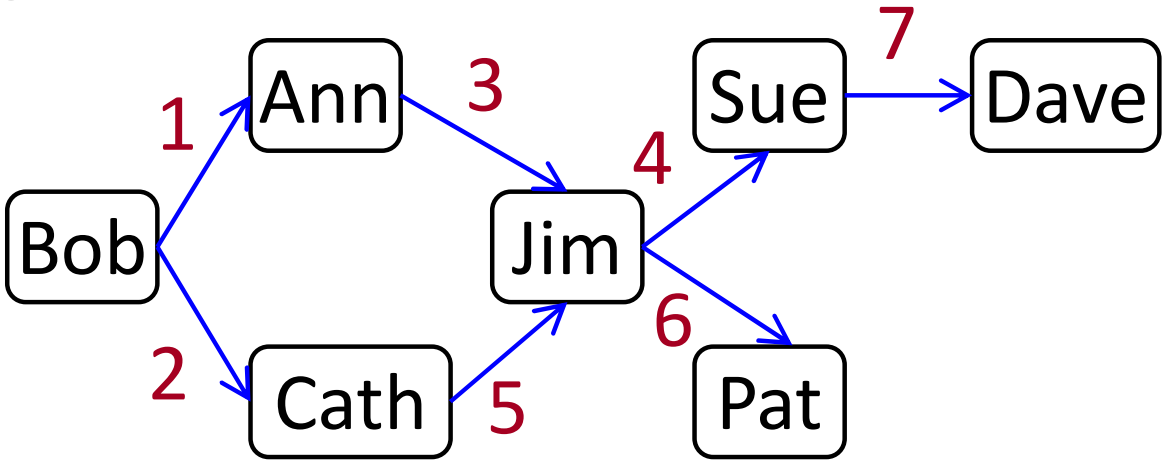


Previous Example

- Bob is the owner

1. Bob grants Ann
2. Bob grants Cath
3. Ann grants Jim
4. Jim grants Sue
5. Cath grants Jim
6. Jim grants Pat
7. Sue grants Dave

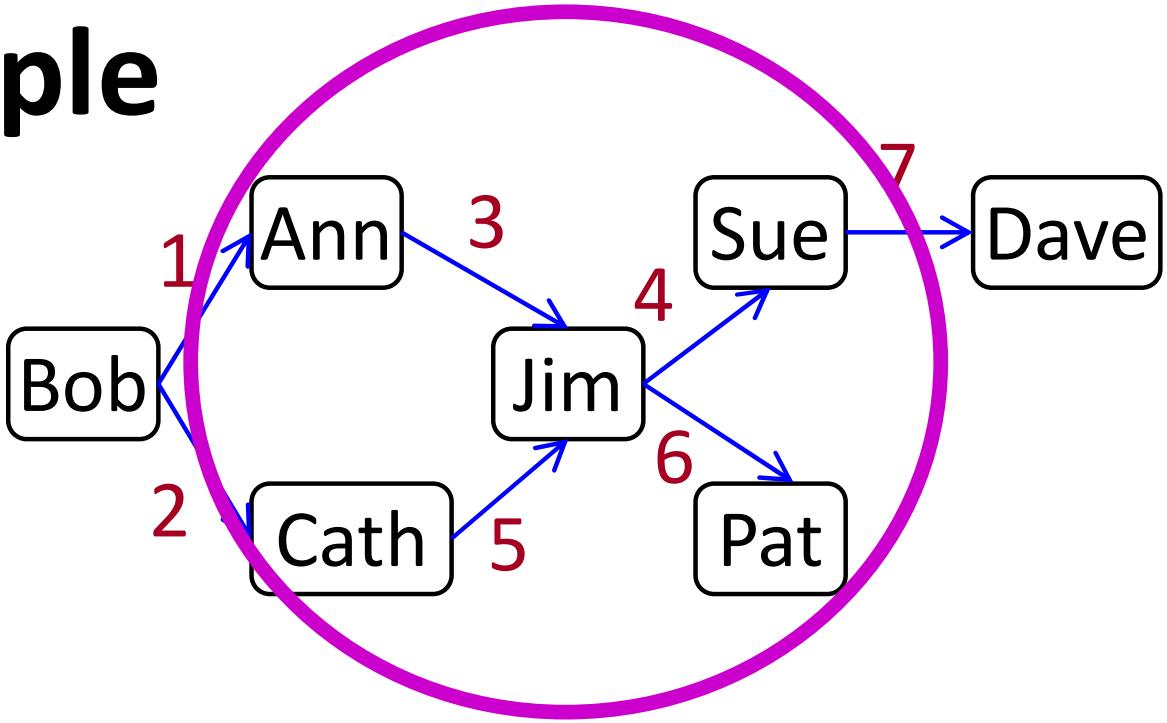
timestamps



Previous Example

- Bob is the owner

1. Bob grants Ann
2. Bob grants Cath
3. Ann grants Jim
4. Jim grants Sue
5. Cath grants Jim
6. Jim grants Pat
7. Sue grants Dave
8. Ann revokes Jim

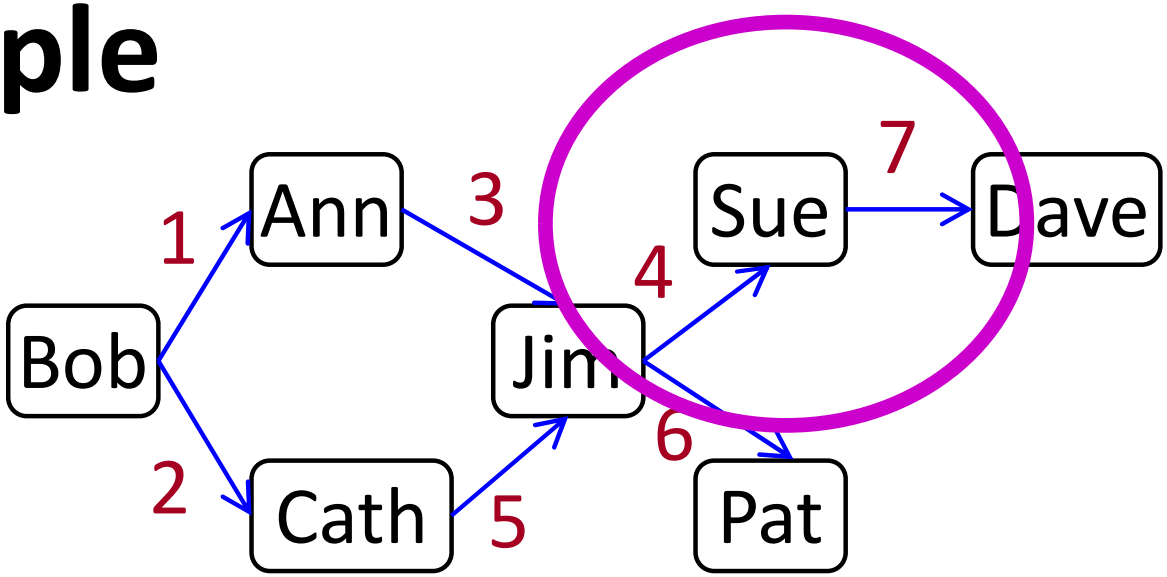


- Jim receives privileges only from Ann and Cath, at timestamps 3 and 5, respectively
- Since Ann revokes, the privileges that Jim grants in [3, 5) must be revoked
- Hence, Jim→Sue should be removed

Previous Example

- Bob is the owner

1. Bob grants Ann
2. Bob grants Cath
3. Ann grants Jim
4. Jim grants Sue
5. Cath grants Jim
6. Jim grants Pat
7. Sue grants Dave
8. Ann revokes Jim



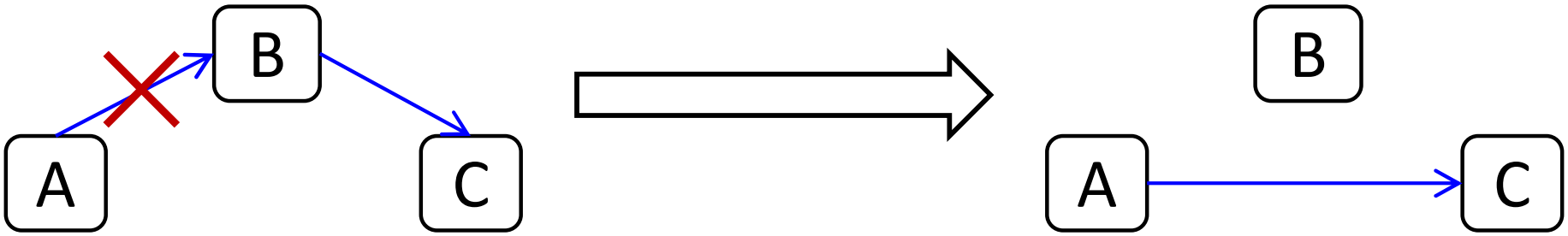
- Sue receives privilege only from Jim
- Since Jim→Sue is removed, Sue→Dave should also be removed

Drawbacks of Recursive Revocation

- It could be rather disruptive
  - The system needs to carefully revoke a potentially long list of privileges, which could incur considerable overheads
  - The revocations may invalidate a large number of applications, disrupting the services replying upon those applications.
- This motivates the *non-cascading revocation* approach

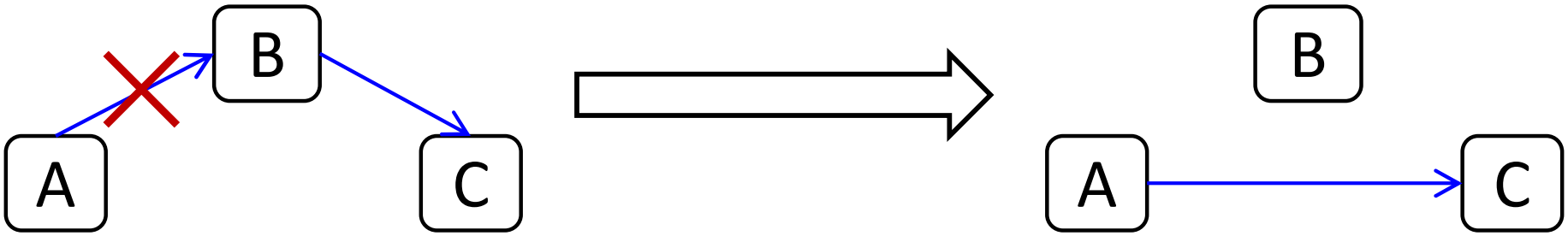
Non-Cascading Revocation

- When a user A revokes a privilege from another user B, the relevant privileges that B grants to others are restated as if they are granted by user A
- Example:



Non-Cascading Revocation

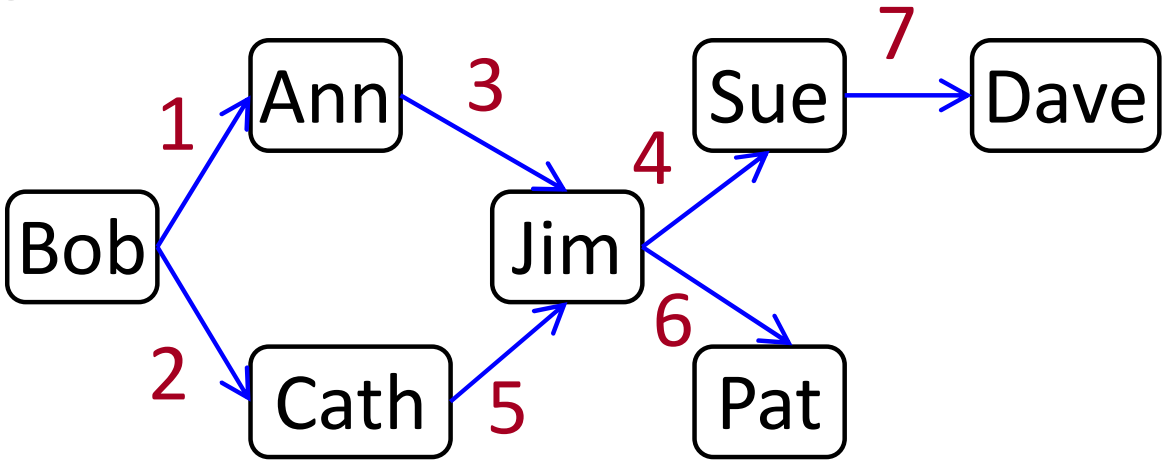
- How to implement non-cascading revocation?
- We can still use timestamps.

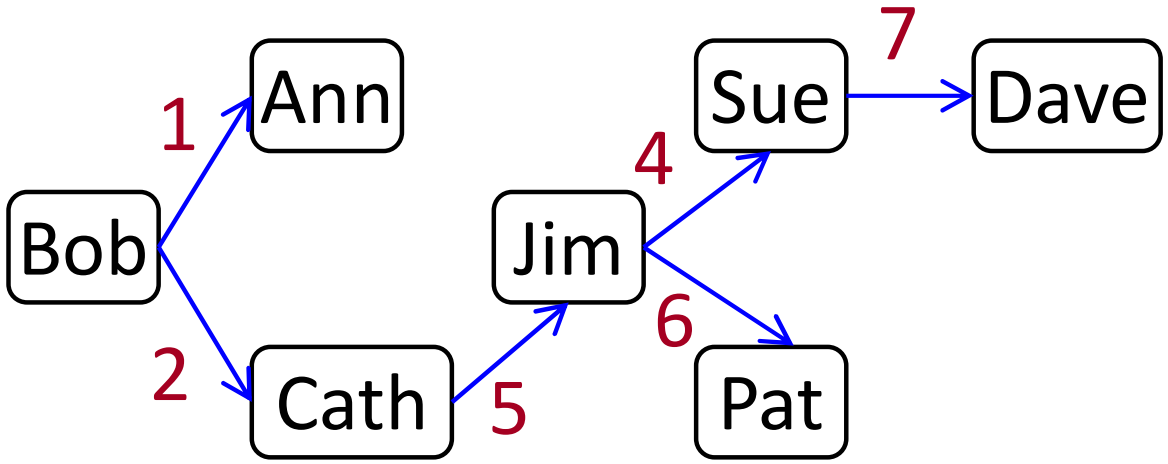


Previous Example

- Bob is the owner

1. Bob grants Ann
2. Bob grants Cath
3. Ann grants Jim
4. Jim grants Sue
5. Cath grants Jim
6. Jim grants Pat
7. Sue grants Dave
8. Ann revokes Jim

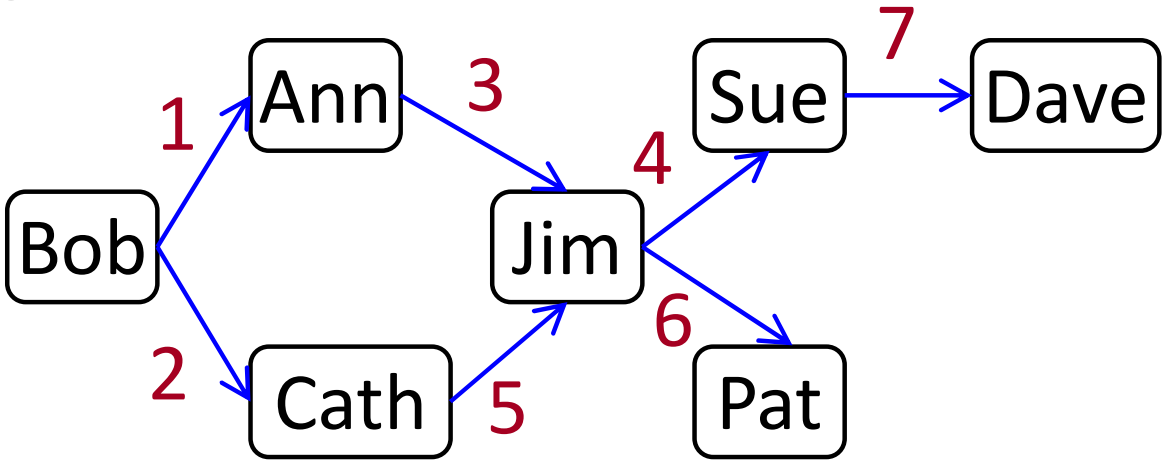


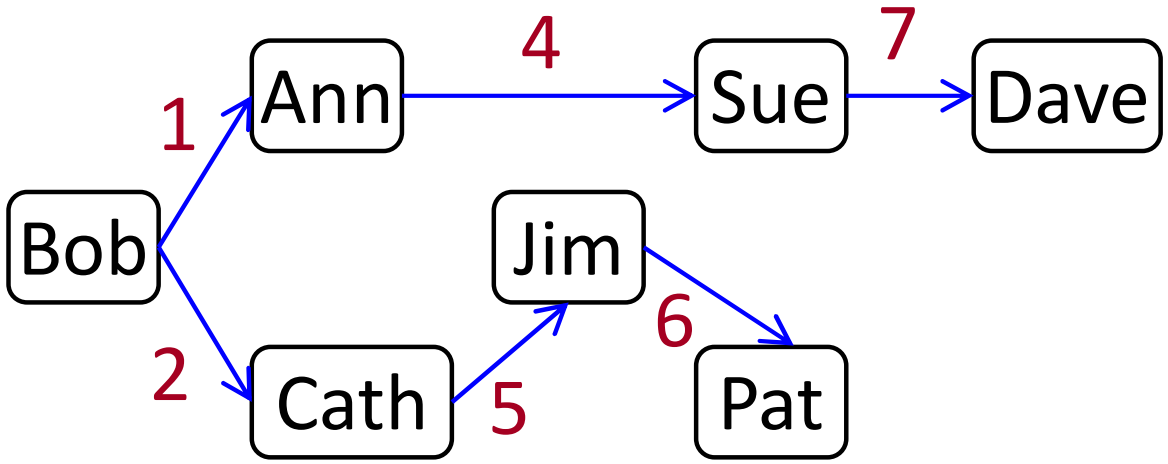


Previous Example

- Bob is the owner

1. Bob grants Ann
2. Bob grants Cath
3. Ann grants Jim
4. Jim grants Sue
5. Cath grants Jim
6. Jim grants Pat
7. Sue grants Dave
8. Ann revokes Jim

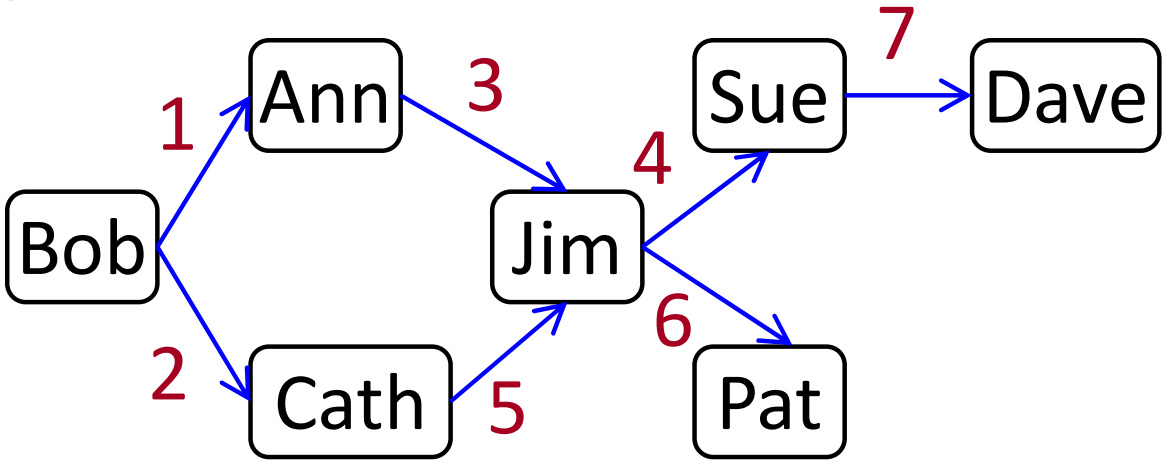


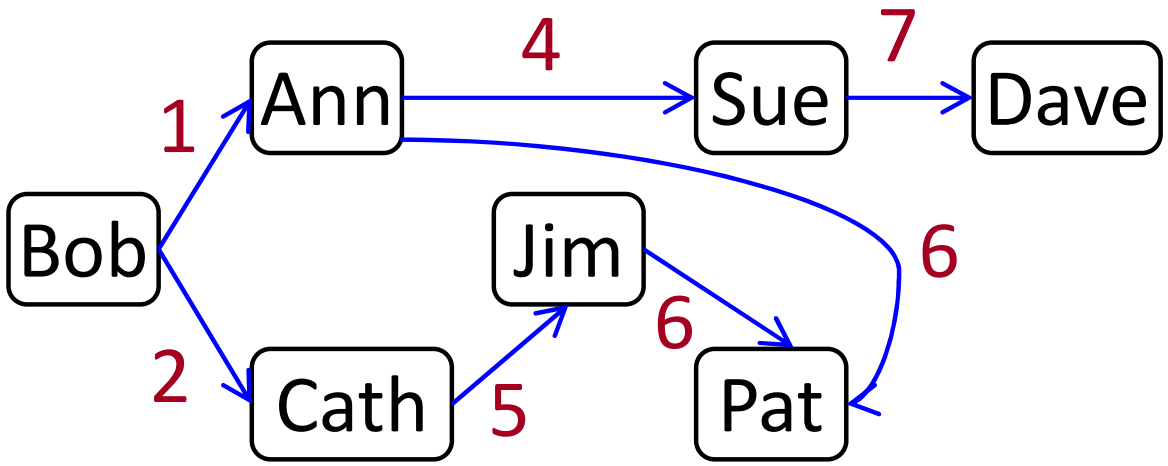


Previous Example

- Bob is the owner

1. Bob grants Ann
2. Bob grants Cath
3. Ann grants Jim
4. Jim grants Sue
5. Cath grants Jim
6. Jim grants Pat
7. Sue grants Dave
8. Ann revokes Jim

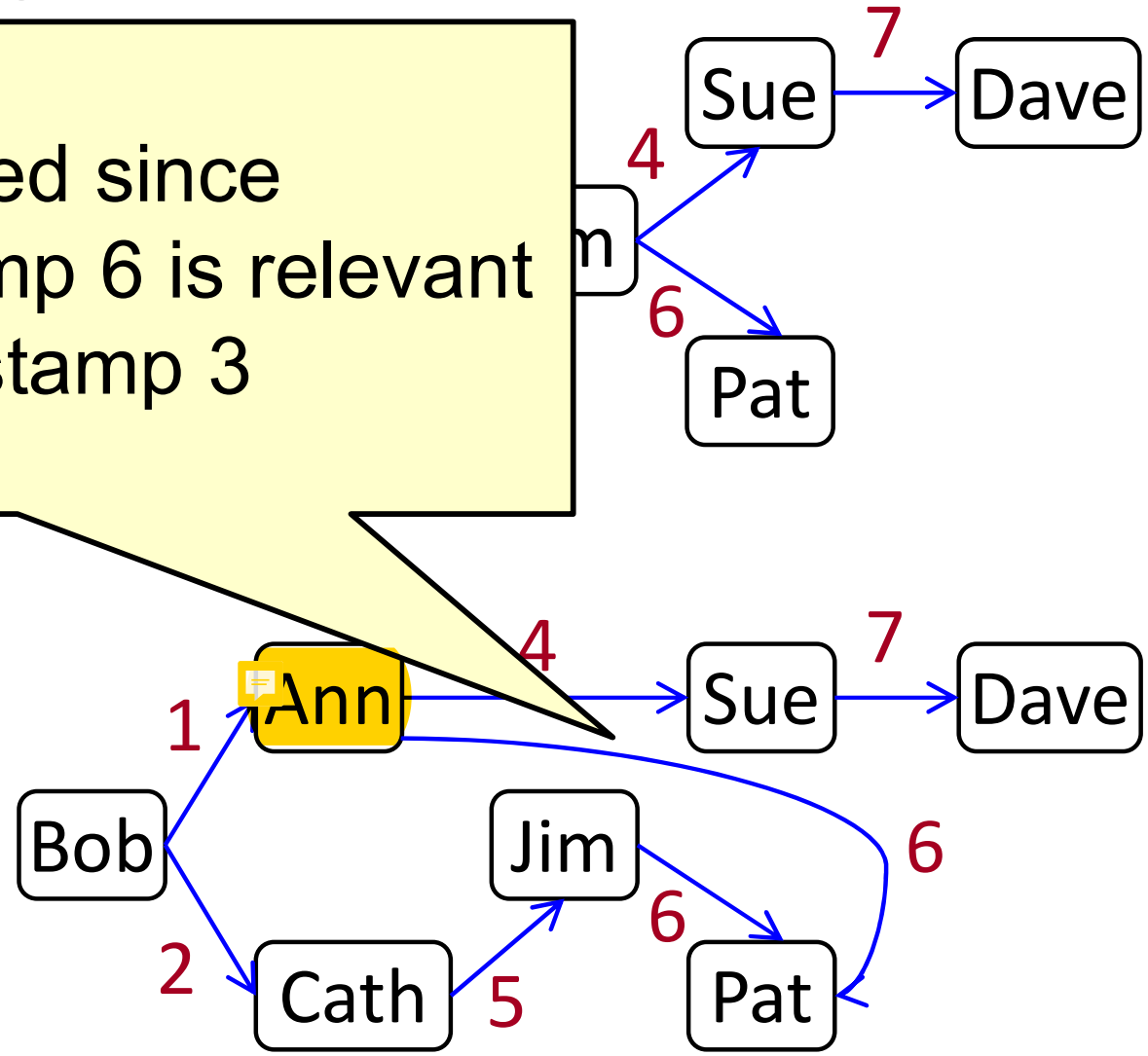




Previous Example

This privilege is added since Jim→Pat at timestamp 6 is relevant to Ann→Jim at timestamp 3

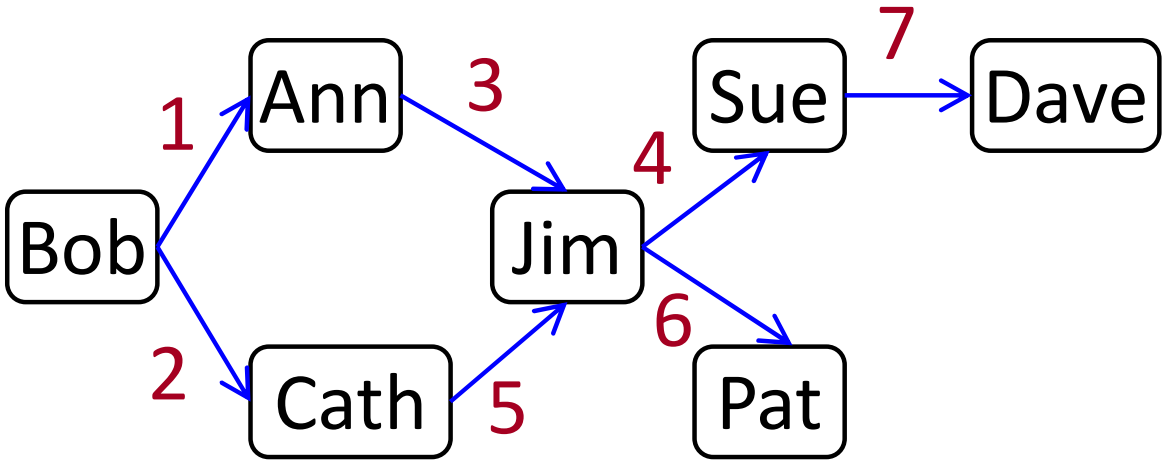
4. Jim grants Sue
5. Cath grants Jim
6. Jim grants Pat
7. Sue grants Dave
8. Ann revokes Jim

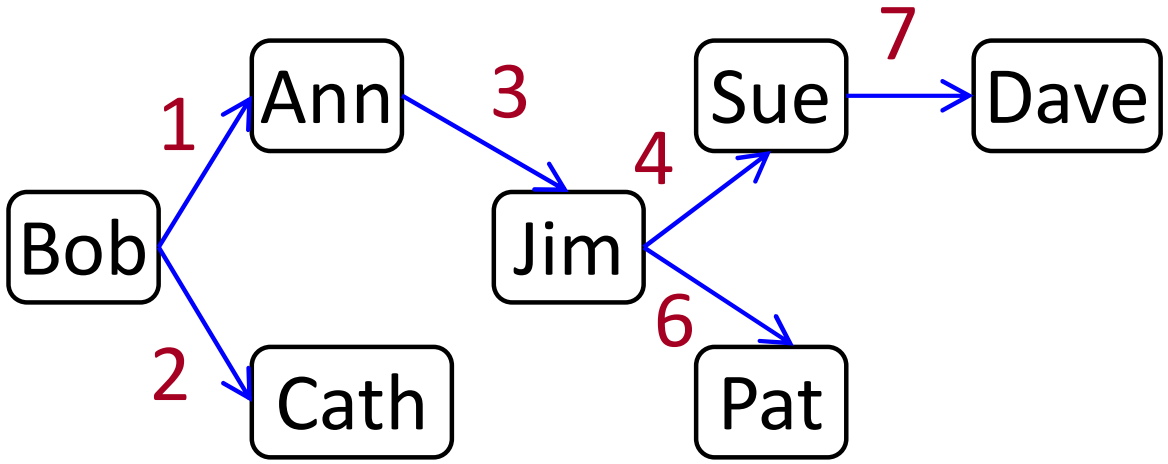


Another Example

- Bob is the owner

1. Bob grants Ann
2. Bob grants Cath
3. Ann grants Jim
4. Jim grants Sue
5. Cath grants Jim
6. Jim grants Pat
7. Sue grants Dave
8. Cath revokes Jim

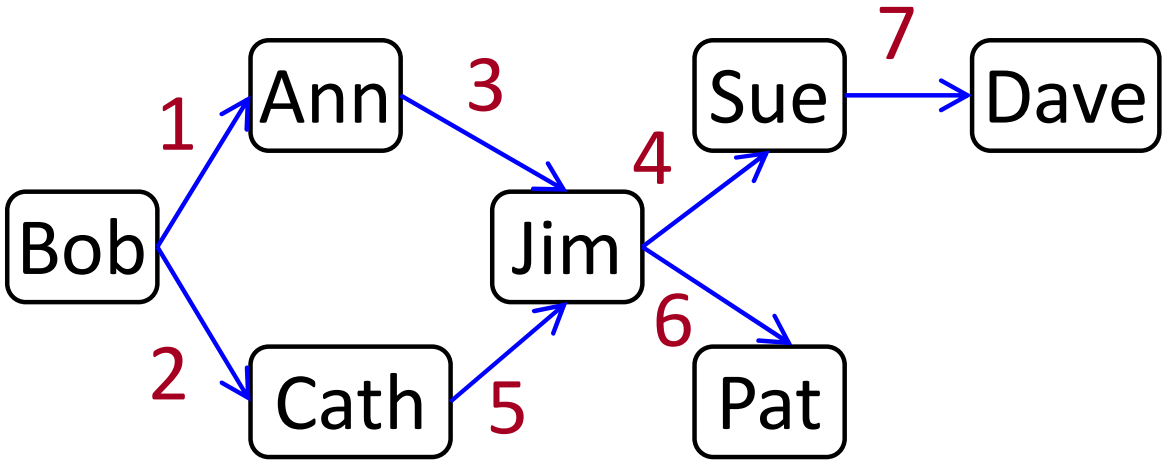


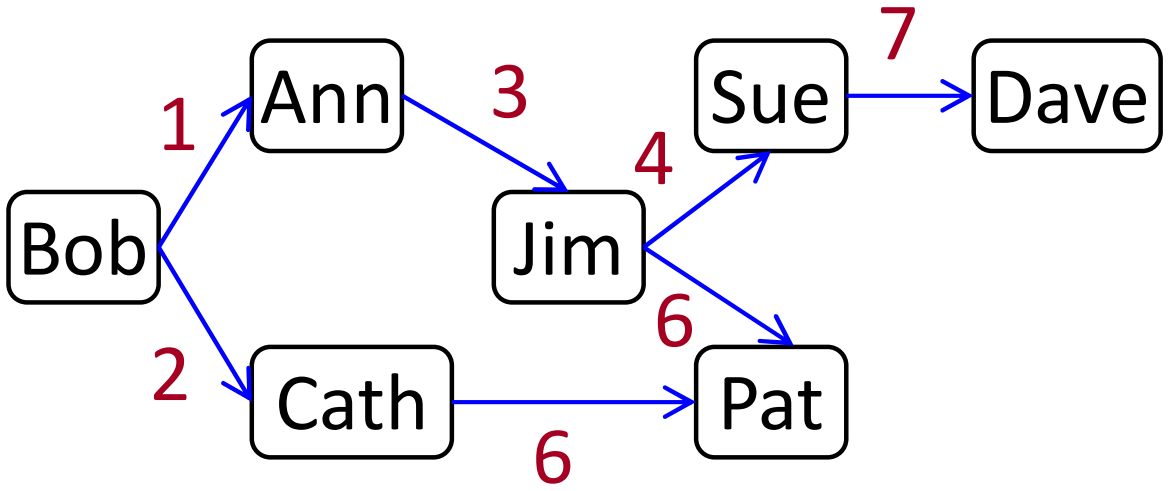


Another Example

- Bob is the owner

1. Bob grants Ann
2. Bob grants Cath
3. Ann grants Jim
4. Jim grants Sue
5. Cath grants Jim
6. Jim grants Pat
7. Sue grants Dave
8. Cath revokes Jim

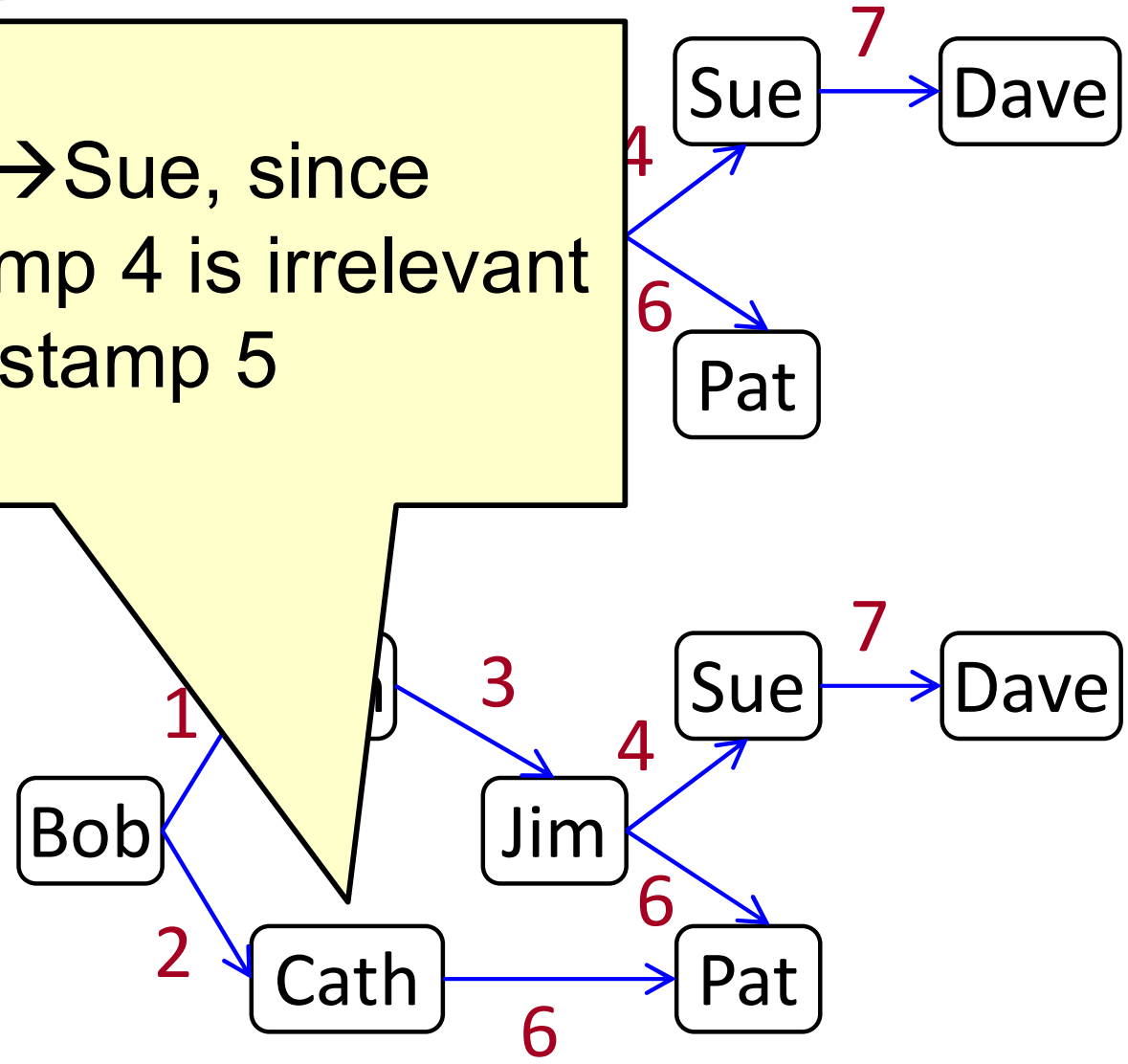




Another Example

We do not add Cath→Sue, since Jim→Sue at timestamp 4 is irrelevant to Cath→Jim at timestamp 5

4. Jim grants Sue
5. Cath grants Jim
6. Jim grants Pat
7. Sue grants Dave
8. Cath revokes Jim



Views / Content-Based Authorization

- Previous discussions focus on authorizations on tables
- Coming next: authorizations on views

Views / Content-Based Authorization

- Views are commonly used to support content-based access control in RDBMS
  - Define a view containing the predicates to select the tuples (or statistics) that a subject S is allowed to access
  - Grant S the select/insert/update privileges on the view, and not on the underlying table

Example

- CREATE VIEW Vemp AS
  SELECT * FROM Employee
  WHERE Salary < 20000;
- GRANT Select ON Vemp TO Ann;

- Ann can only see employees whose salary is lower than 20000

Views / Content-Based Authorization

- Queries on views are transformed through view composition into queries on base tables

- 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

- Query 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 authorizations that the definers has on the base table(s)
  - The view semantics, that is, its definition in terms of the base relation(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

Authorizations on Views – Example

- Suppose that Bob is the owner of Employee
- Consider the following view
  - Bob:

```
CREATE VIEW V1 (Emp#, Sal)
AS SELECT Emp#, Salary
FROM Employee WHERE
Job ='Programmer';
```

- Bob has rights to update both Emp# and Sal on V1, since he has rights to update Emp# and Salary on Employee

Authorizations on Views – Example

- Suppose that Bob is the owner of Employee
- 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 have the update authorization on the Total_Sal column

Authorizations on Views – Example

- Suppose that Bob is the owner of Employee
- Consider the following sequence of commands:
  - Bob: GRANT Select, 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, i.e., Select and Update
- The same goes for V2

Authorizations on Views

- Suppose that Bob is the owner of Employee
- Consider the following sequence of commands:
  - Bob: GRANT Select, 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;

- 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

Authorizations on Views

- 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 access control lists
  - Maintenance of Grant/Revoke

DAC Exercise

- Consider the following two GRANT statements on a table T with 4 attributes A, B, C, D. Assume that Bob owns T.
  - Bob: GRANT update ON T TO Cath
  - Bob: GRANT update (A, B, C, D) ON T TO Cath
- Is there any difference between these two statements?

DAC Exercise

- Consider the following two GRANT statements on a table T with 4 attributes A, B, C, D. Assume that Bob owns T.
  - Bob: GRANT update ON T TO Cath
  - Bob: GRANT update (A, B, C, D) ON T TO Cath
- Is there any difference between these two statements?
- Yes.
  - If Bob adds a new attribute E to T, then the first statement would allow Cath to update E, whereas the second one does not allow this

DAC Exercise

- Suppose that Bob owns a Boats table that contains an attribute bid, which represents the ids of boats.
- Bob: GRANT select ON Boats TO Cath
- Cath: CREATE TABLE Reserves(

```
sid INT, bid INT, day DATE,
PRIMARY KEY (bid, day),
FOREIGN KEY (bid) REFERENCES Boats(bid)
)
```

- Is this OK?

DAC Exercise

- Suppose that Bob owns a Boats table that contains an attribute bid, which represents the ids of boats.
- Bob: GRANT select ON Boats TO Cath
- Cath: CREATE TABLE Reserves(

```
sid INT, bid INT, day DATE,
PRIMARY KEY (bid, day),
FOREIGN KEY (bid) REFERENCES Boats(bid)
)
```

- Is this OK?
- No.
  - Because Cath does not have the right to reference Boats(bid)

DAC Exercise

- Suppose that Bob owns a Boats table that contains an attribute bid, which represents the ids of boats.
- Bob: GRANT select, references(bid) ON Boats TO Cath
- Cath: CREATE TABLE Reserves(

```
sid INT, bid INT, day DATE,
PRIMARY KEY (bid, day),
FOREIGN KEY (bid) REFERENCES Boats(bid)
)
```

- Is this OK?
- Now it is OK

DAC Exercise

- Suppose that Bob owns the following table:
  - Restaurant(rid, name, address, rating)
- Bob:
  - GRANT update(rating) ON Restaurant TO Cath
- Cath:
  - UPDATE Restaurant R
    SET R.rating = 3
    WHERE R.rid = 1
- Is this OK?

DAC Exercise

- Suppose that Bob owns the following table:
  - Restaurant(rid, name, address, rating)
- Bob:
  - GRANT update(rating) ON Restaurant TO Cath
- Cath:
  - UPDATE Restaurant R
    SET R.rating = 3
    WHERE R.rid = 1
- Is this OK?
- No.
  - Because Cath does not have select (i.e., read) right on Restaurant

DAC Exercise

- Suppose that Bob owns the following table:
  - Restaurant(rid, name, address, rating)
- Bob:
  - GRANT select, update(rating) ON Restaurant TO Cath
- Cath:
  - UPDATE Restaurant R
    SET R.rating = 3
    WHERE R.rid = 1
- Now it is OK

DAC Exercise

- Suppose that Bob owns the following table:
  - Restaurant(rid, name, address, rating)
- Bob:
  - GRANT update(rating) ON Restaurant TO Cath
- Cath:
  - UPDATE Restaurant R
    SET R.rating = 3
- Is this OK?

DAC Exercise

- Suppose that Bob owns the following table:
  - Restaurant(rid, name, address, rating)
- Bob:
  - GRANT update(rating) ON Restaurant TO Cath
- Cath:
  - UPDATE Restaurant R
    SET R.rating = 3
- Is this OK?
- Yes
  - Because the update does not read from the Restaurant table; it directly writes on it

DAC Exercise

- Suppose that Bob owns the following table:
  - Restaurant(rid, name, address, rating)
- Bob:
  - GRANT update(rating) ON Restaurant TO Cath
- Cath:
  - UPDATE Restaurant R
    SET R.rating = R.rating + 1
- Is this OK?

DAC Exercise

- Suppose that Bob owns the following table:
  - Restaurant(rid, name, address, rating)
- Bob:
  - GRANT update(rating) ON Restaurant TO Cath
- Cath:
  - UPDATE Restaurant R
    SET R.rating = R.rating + 1
- Is this OK?
- No
  - R.rating = R.rating + 1 cannot be done without reading R.rating

DAC Exercise

- Suppose that Bob owns the following table:
  - Restaurant(rid, name, address, rating)
- Bob:
  - GRANT select ON Restaurant TO Cath with GRANT OPTION
- Cath:
  - CREATE VIEW BadRestaurant(rid, name, address) AS
    SELECT R.rid, R.name, R.address FROM Restaurant
    WHERE R.rating < 3
  - GRANT select ON BadRestaurant TO Dave
- Dave:
  - CREATE VIEW BadLocalRestaurant(rid, name, address) AS
    SELECT R.rid, R.name, R.address FROM BadRestaurant
    WHERE R.address LIKE '%Singapore'
- What happens if Bob revokes the select privilege on Restaurant from Cath? (Assuming recursive revocation.)

DAC Exercise

- Suppose that Bob owns the following table:
  - Restaurant(rid, name, address, rating)
- Bob:
  - GRANT select ON Restaurant TO Cath with GRANT OPTION
- Cath:
  - CREATE VIEW BadRestaurant(rid, name, address) AS
    SELECT R.rid, R.name, R.address FROM Restaurant
    WHERE R.rating < 3
  - GRANT select ON BadRestaurant TO Dave
- Dave:
  - CREATE VIEW BadLocalRestaurant(rid, name, address) AS
    SELECT R.rid, R.name, R.address FROM BadRestaurant
    WHERE R.address LIKE '%Singapore'
- What happens if Bob revokes the select privilege on Restaurant from Cath? (Assuming recursive revocation.)
- BadRestaurant will be removed; same for BadLocalRestaurant

DAC Exercise

- Suppose that Bob owns the following table:
  - Restaurant(rid, name, address, rating)
- Bob:
  - GRANT select ON Restaurant TO Cath with GRANT OPTION
- Cath:
  - CREATE VIEW BadRestaurant(rid, name, address) AS
    SELECT R.rid, R.name, R.address FROM Restaurant
    WHERE R.rating < 3
  - GRANT select ON BadRestaurant TO Dave
- Bob:
  - GRANT update ON Restaurant TO Cath with GRANT OPTION
- What privileges on BadRestaurant do Cath and Dave have now?

DAC Exercise

- Suppose that Bob owns the following table:
  - Restaurant(rid, name, address, rating)
- Bob:
  - GRANT select ON Restaurant TO Cath with GRANT OPTION
- Cath:
  - CREATE VIEW BadRestaurant(rid, name, address) AS
    SELECT R.rid, R.name, R.address FROM Restaurant
    WHERE R.rating < 3
  - GRANT select ON BadRestaurant TO Dave
- Bob:
  - GRANT update ON Restaurant TO Cath with GRANT OPTION
- What privileges on BadRestaurant do Cath and Dave have now?
  - Cath has select and update privileges
  - Dave has only select privilege
- Why?
  - As the view creator, Cath's privileges on the view equal her privileges on the base table
  - Dave is not the view creator; his privilege is as granted by Cath