Unit 3:
Database Management Systems (DBMS)

Unit 3. Database Management Systems

1.1. Schemas

Original proposal

Proposal of a DBMS architecture by the working group ANSI/SPARC (1977). They proposed the database definition with 3 levels of abstraction:

- **Internal level** → Internal schema

  Description of the DB in terms of its physical representation

- **Conceptual level** → Conceptual schema

  Description of the DB independently of the DBMS. It is usually a graphical representation.

- **External level** → External schemas

  Description of the different users' partial views

Refined proposal

Since there was no generalized conceptual model for different kinds of DBMS (it is difficult to obtain the physical data structures from a conceptual graphical representation) the “logical” level was added:

– **Internal (physical) level** → Internal schema *(not in this course)*
DB description in terms of its physical representation. Describe how the database is store in secondary memory.

– **Logical level** → Logical schema (Unit 1)
DB description in terms of the DBMS data model. It does not include any details of the physical representation.

– **Conceptual level** → Conceptual schema (Unit 4)
Description of the information system from the organizational point of view. It is independent of the DBMS. We will use a UML class diagram.

– **External level** → External schemas (authorizations and views)
Description of the partial views which the different users have on the DB.

Example: Logical schema

Department (cod_dep: char(4), nombre: char(50), teléfono: char(8), director: char(9))

PK:{cod_dep} **NNV**:{nombre}

FK:{director} ->Lecturer(dni) On delete set nulls. On update cascade

Subject (cod_asg: char(5), nombre: char(50), semestre: char(2), cod_dep: char(4), teoría: real, prácticas: real)

PK:{cod_asg} **NNV**:{nombre, semestre, cod_dep, teoría, prácticas}

Uni:{nombre} **FK**:{cod_dep} -> Department(cod_dep)

On delete restrict. On update cascade

IC$_1$:(teoría <= prácticas) **IC$_2$**:(semestre IN {'1A','1B','2A','2B','3A','3B','4A''4B'})

Lecturer (dni: char(9), nombre: char(80), teléfono: char(8), cod_dep: char(4), provincia: char(25), edad: entero)

PK :{dni} **NNV** :{nombre, cod_dep}

FK :{cod_dep} -> Department(cod_dep)

On delete restrict. On update cascade

Teaching (dni: char(9), cod_asg: char(5), gteo: entero, gpra: entero)

PK :{dni,cod_asg} **NNV** :{gteo,gpra}

FK :{dni} -> Lecturer(dni) On delete cascade. On update cascade

FK :{cod_asg} -> Subject(cod_asg) On delete restrict. On update cascade

General constraint: GC1: "All teacher must lecture at least one subject".

Example: Internal schema

Depends on the DBMS.

***Subject*:**

Hash file by cod_dep

B+ index over (semestre + cod_dep)

Lecturer:

Hash file by nombre

Department:

Hash file by cod_dep

B+ index over nombre

Teaching:

Disordered file

Example: External schema

External schema for the **Maths Department**

```
CREATE VIEW Maths-Lecturer  AS
        SELECT dni, nombre, teléfono, categoría,edad
        FROM Lecturer
        WHERE  dptocod_dep = ’DMA’;

CREATE VIEW Maths-Subject  AS
        SELECT cod_asg, nombre, semestre, teoría, prácticas
        FROM Subject
        WHERE cod_dep = ’DMA’;

CREATE VIEW Maths-Teaching  AS
        SELECT T.dni, T.cod_asg, T.gteo, T.gpra
        FROM Lecturer L, Teaching T, Subject S
        WHERE   L.cod_dep = ’DMA’
            AND S.cod_dep = ’DMA’
            AND L.dni = T.dni
            AND S.cod_asg = T.cod_asg;
```

A DBMS that supports the 3-level architecture must:

- Allow the definition of the different schemas for the database (except the conceptual schema),
- Establish the correspondence between schemas,
- Isolate the schemas: changes in one schema should not affect neither the schemas at upper levels nor the application programs.

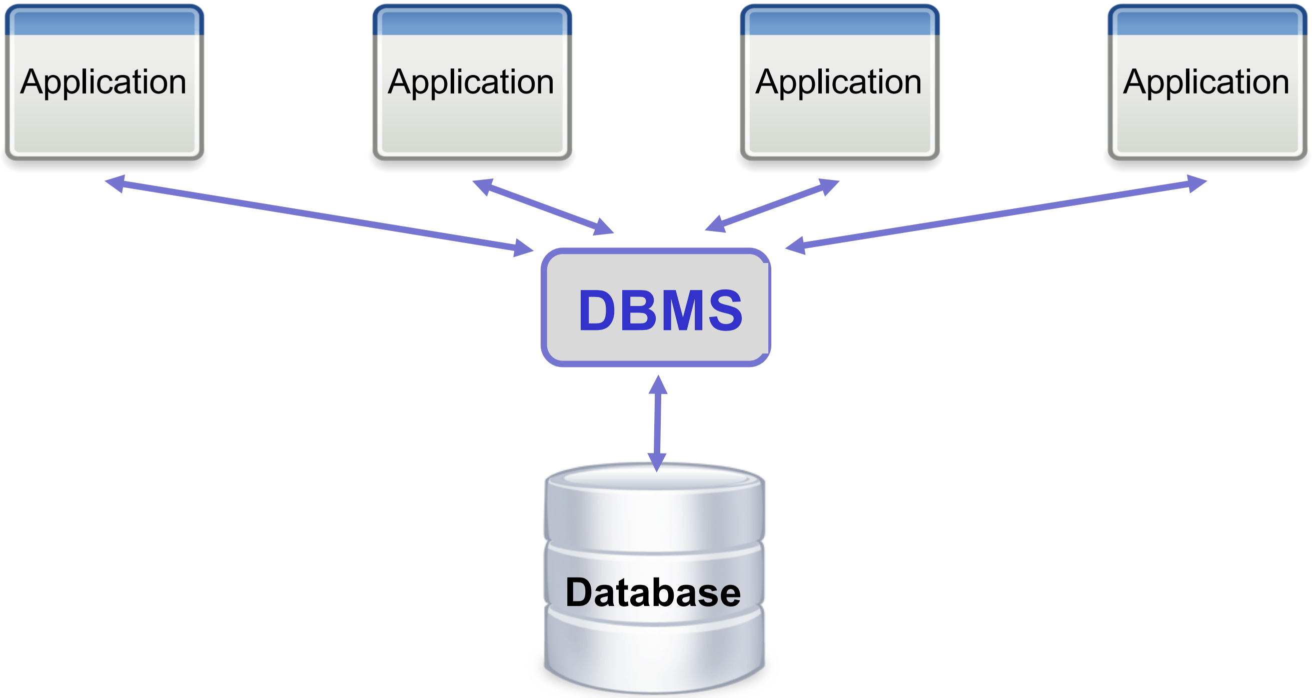
↓

DATA INDEPENDENCE

1.2. DBMS Fundamentals

Functions of a DBMS

DBMS: Software which allows the creation and manipulation of databases (DB).



A DBMS must maintain the independence, integrity and security of data.

Functions of a DBMS

Objectives of DB techniques

- Unified and independent **data description**
- **Application** independence
- Partial **view** definition

DBMS Functions

Data definition at several levels

- Logical schema
- Internal schema
- External schema

DMBS Components

Schema definition languages and their associated translators

Functions of a DBMS

Objectives of DB techniques

- Information **management**

DBMS Functions

Data manipulation

- Query
- Update

Management and administration of the database

DMBS Components

Manipulation languages and their associated translators

Tools for:

- Restructuring
- Simulation
- Statistics
- Printing and reporting

Functions of a DBMS

Objectives of DB techniques

- Data **integrity** and **security**

DBMS Functions

Control of:

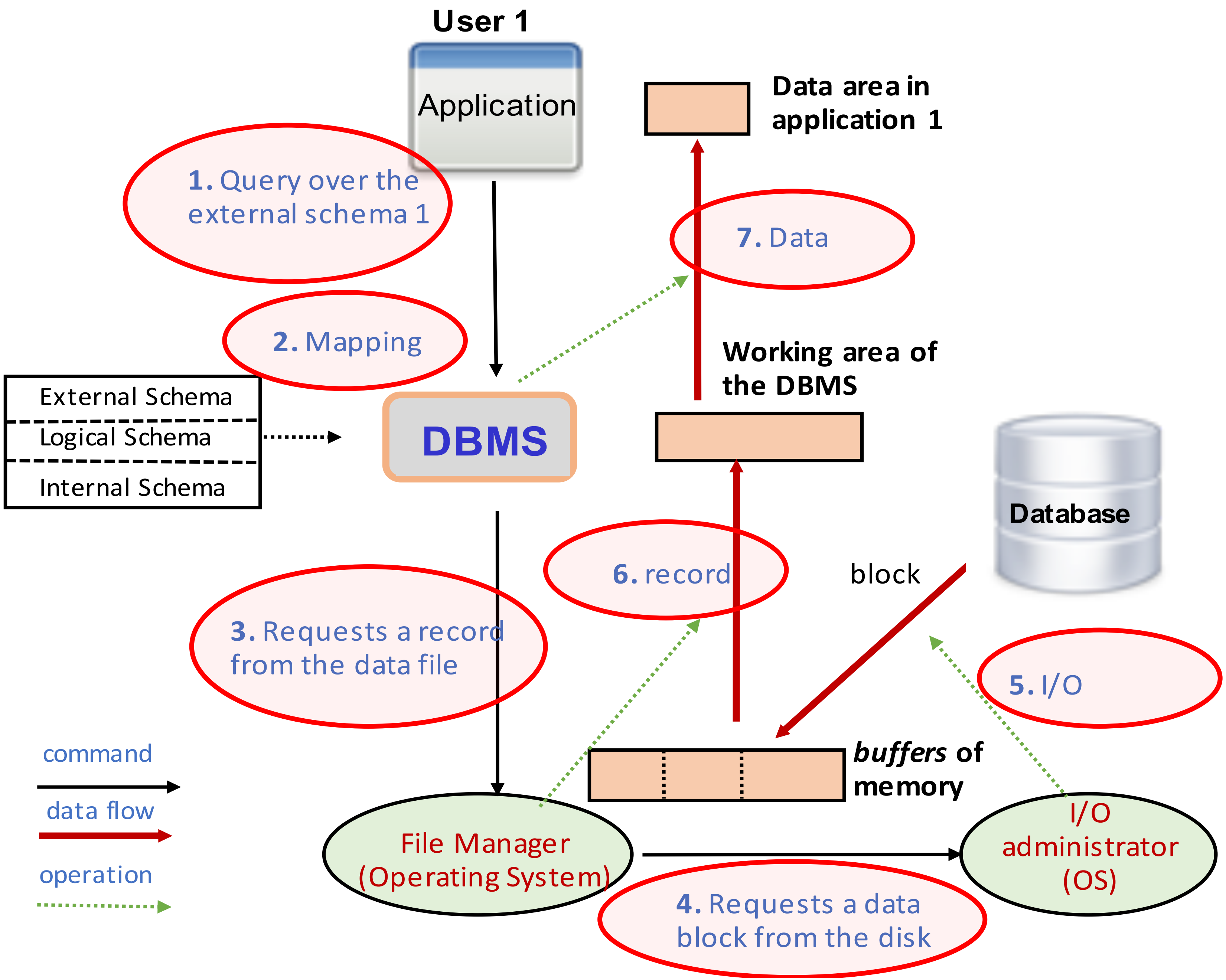
- Semantic integrity
- Concurrent access
- Recovery in case of failure
- Security (privacy)

DMBS Components

Tools for:

- Integrity control
- Reconstruction
- Security control

Accessing the data



1.3. Data Independence

Data independence

Property which ensures that the application programs are independent of

- the changes which are performed on data which they do not used

or

- the physical representation details of the accessed data

Logical independence

Logical independence between the logical schema and the external schemas:

The external schemas and the application programs cannot be affected by the modifications in the logical schema of data which are not used by these programs

EXAMPLE:

If we add new attributes to the "*Department*" table, such as the date in which the department was created, the building,… the external schema of the "*Maths-department"* does not need to be modified

Physical independence

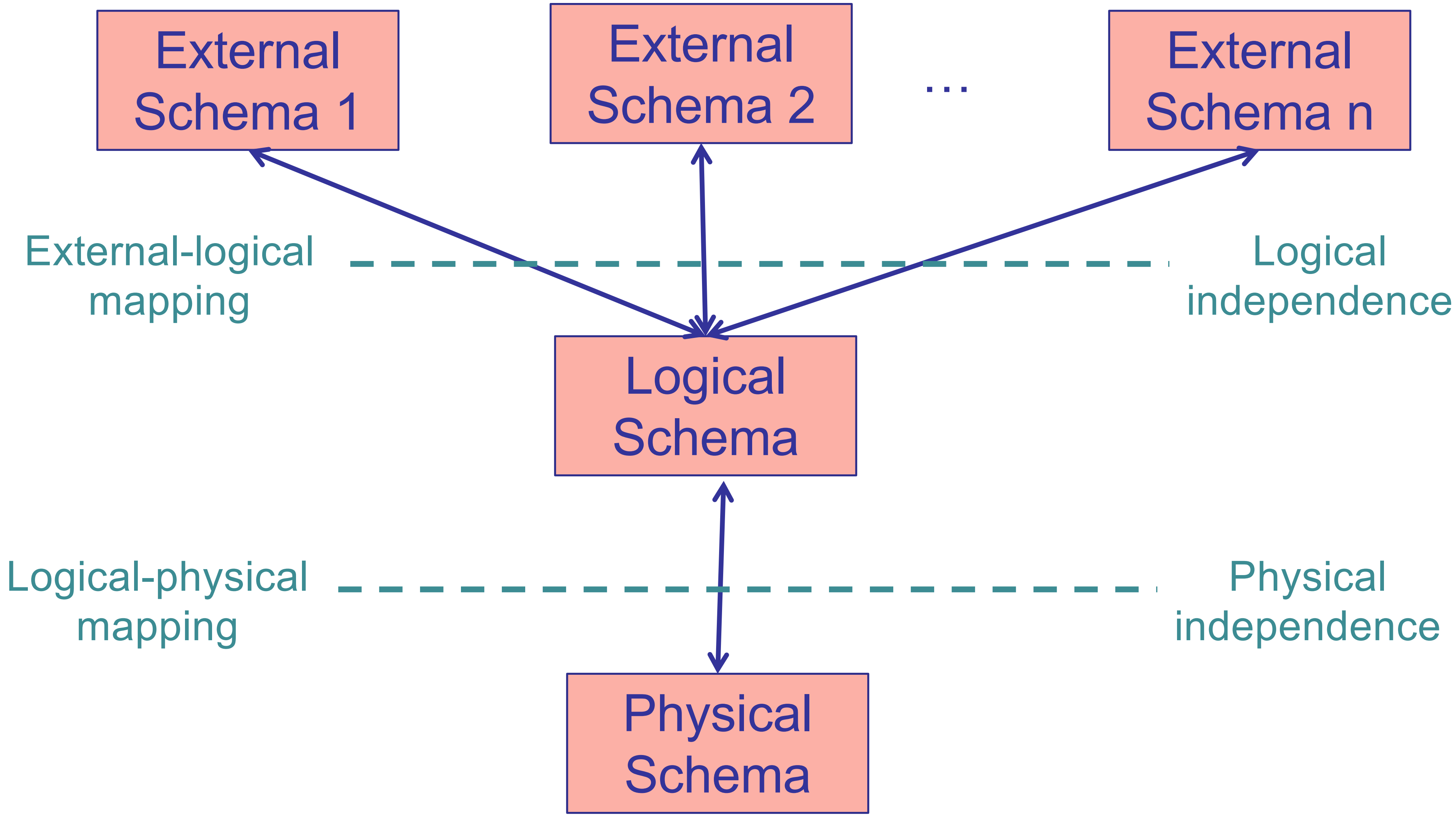
Physical independence between the internal schema and the logical schema:

> The logical schema cannot be affected by changes in the internal schema which refer to the implementation of the data structures, access modes, page size, search path, etc.

EXAMPLE:

If the data structures used in the implementation of the "*Subject*" table are changed, the logical schema does not need to be modified

Data independence



Binding

Binding (ligadura):

Transformation of the external schema into the internal schema.

Types

- Logical binding (steps 2 and 7).
- Physical binding (steps 3 and 6).

When the binding is performed, independence disappears

↓

It is important to determine the binding moment

Currently, most DBMS do the binding for each query.

Physical independence is found in most DBMS, while logical independence is more difficult to find.

Unit 3. Database Management Systems

1. The ANSI/SPARC Architecture
   1.1. Schemas
   1.2. DBMS fundamentals
   1.3. Data independence
2. **Transactions, Integrity, and Concurrency**
   2.1. Transactions
   2.2. Semantic integrity
   2.3. Concurrent access control
3. Recovery and Security
   3.1. DB Recovery
   3.2. Security

2. Transactions, Integrity, and Concurrency

Objective of DB technology

↓

Information quality

"Data must be structured in such a way to adequately reflect the objects, relations, and constraints which exist in the part of the real world modeled by the database model."

When reality changes → User updates the database

The information contained in the DB must preserve the schema definition.

2. Transactions, Integrity, and Concurrency

Information quality (integrity perspective) means that:

- The DBMS must ensure that the data are correctly stored
- The DBMS must ensure that user updates over the DB are correctly executed and become permanent.

2. Transactions, Integrity, and Concurrency

DBMS Tools oriented towards integrity:

- Check (when an update is performed) the integrity constraints defined in the schema.
- Control the correct execution of the updates in a concurrent environment.
- Recover (reconstruct) the DB in case of loss or accident

2.1. Transactions

2.1. Transactions

The operations in a DB are organized in transactions.

Transaction:

Sequence of access operations to the DB which constitute a logical execution unit.

Example

Emp (id, name, address, dept)
PK: {id}
FK: {dept} $\rightarrow$ Dep(code)

Dep (code, name, location)
PK: {code}

IC_1: All departments have at least one employee

Insert a new department:
<"d2", "Human Resources", "2nd floor">
whose first employee is the id 20

Example

1st Idea

1) Insert in *Dep*: <d2, “Human Resources”, “2nd floor”>

ERROR: IC_1 *is violated*

2) Modification of *Emp* on the tuple with *id* 20

2nd Idea

1) Modification of *Emp* on the tuple with *id* 20

ERROR: the FK over dept in Emp is violated

2) Insertion in *Dep*: <d2, “Human Resources”, “2nd floor>

Defining transactions

Actions which change transactions states:

Begin:

Indicates the beginning of the execution of the transaction

Cancelation (user rollback):

The user aborts the transaction.

Confirmation *(user commit)*:

The user considers the transaction as ended. Then the DBMS performs some checking to determine how the transaction will end:

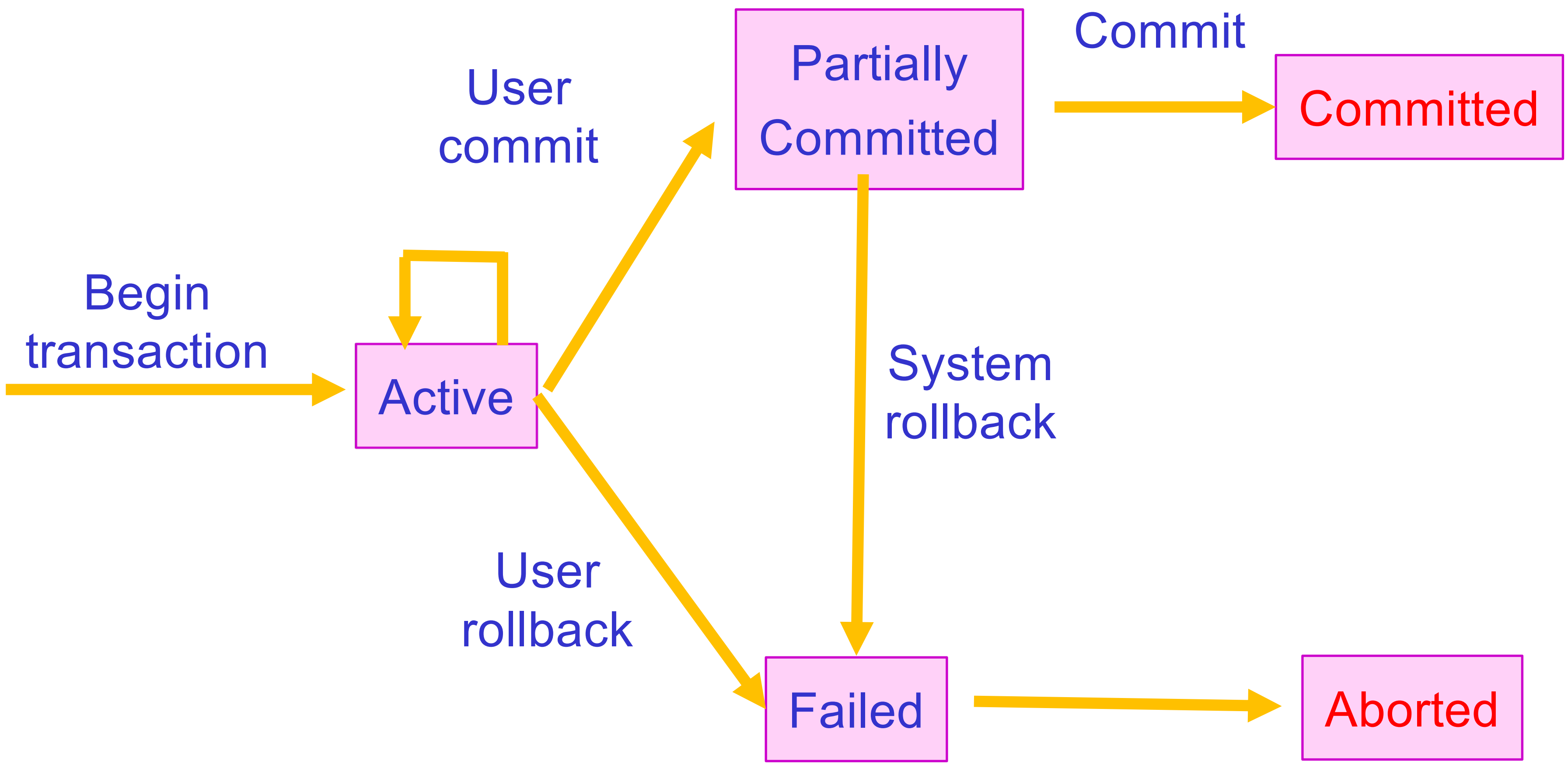
- Success (system commit):

  Indicates the success of the transaction, making the DBMS store the changes performed on the DB.

- Failure (system rollback):

  Indicates the failure of the transaction, or that the transaction hasn’t passed the checking. The DBMS undoes all the possible changes performed by the transaction.

States of a transaction



Properties of Transactions (ACID)

Atomicity:

A transaction is an indivisible unit that is either performed in its entirety or is not performed at all ("All or nothing").

Consistency:

The transaction must transform the DB from one consistent state to another consistent state (all integrity constraints must be met)

Isolation:

Concurrent transactions execute independently: All the partial effects of incomplete transactions should not be visible to other transactions

Durability:

The effects of a successfully completed (committed) transaction are permanently recorded in the DB and must not be lost because of a subsequent system or other transaction failure

2.2. Semantic Integrity

2.2. Semantic Integrity

Integrity constraint:

Property of the real world which is modelled by the DB

- Constraints are defined in the logical schema and the DBMS must ensure that they are met.
- Checking is performed whenever the DB changes (when any updating operation is executed)
- Constraints not included in the DB schema must be maintained by the application programs

This situation is, in general, inappropriate if the constraints are common to more than one application, since the responsibility to check them is dispersed

Types of integrity constraints

Static: They must be met in each state of the DB (they can be represented by logical expressions)

Example:
The *lab credits* in a *subject* cannot be greater than the *lectures credits*.

Dynamic (Transition): They must be met regarding two consecutive states.

They are not usually implemented by commercial DBMS

Example:
The *credits* of a *subject* cannot decrease.

Expressing static constraints in SQL

- Constraints over possible data values. e.g. Domains
- Constraints over attributes. e.g. NNV.
- Constraints over relations. e.g. PK, FK
- General constraints over the DB. e.g: "*All subject must be lectured by at least one teacher.*"

When are checked:

- After every command (IMMEDIATE)
- At the end of the transaction(DEFERRED)

Expressing transition constraints in SQL

Triggers ("*disparadores*")

- Using triggers, the designer can program the system response when some events are produced.
- This allows to incorporate complex constraints into the DB.

A trigger includes:

1. Events: Operations over the DB which trigger it
2. Conditions to determine if the actions must be executed or not.
3. Actions to be executed when an event happen and the conditions are met. They are usually written in a data-oriented high level programming language, that can include SQL commands.

Expressing transition constraints in SQL

Example:

A trigger to implement the integrity constraint: *“A lecturer can only teach a subject assigned to his/her department”*

1. Events: INSERTion of a tuple in the “Subject” table
2. Conditions: The lecturer and the subject are not in the same department.
3. Actions: Reject the operation (the insertion).

2.3. Concurrent access control

2.3. Concurrent access control

In order to keep the integrity of the database, the DBMS must control concurrent access to the database:

To avoid that the results of the execution of several processes (users or programs) lead to incorrect, incoherent or lost results because of the simultaneous execution of other program accessing the same data

Basic operations

Basic operations in a transaction which are relevant to the DBMS:

read(X):

Reading or access to a piece of data X in the DB over the program variable with the same name.

write(X):

Update (insertion, deletion, or modification) of a piece of data X in the DB by using the program variable with the same name

Reading steps

read(X):

1. Seek the address of the block which contains the datum X

2. Copy the block to a buffer into main memory

3. Copy the datum X from the buffer to the program variable X

Writing steps

write(X):

If not read before

1. Find the address of the block containing the datum X
2. Copy the block into a database buffer in main memory
3. Copy the datum X from the program variable into the the database buffer
4. Write the updated block from the database buffer to the disk

Possible problems

The DBMS must control the concurrent access by the applications.

Problems due to interference of concurrent accesses:

a) Loss of updates. An apparently successfully completed update operation by one user can be overridden by another user.

b) Inconsistent information corresponding to several valid database states. One transaction reads several values but a second transaction updates some of them during the execution of the first.

c) Access to updated data (but still not confirmed) that can still be cancelled. One transaction is allowed to use the intermediate results of another transaction before it has committed (*dirty read*).

A. The lost update problem

Time	Program 1	Program 2
t1	**read(11548, teoría)** teoría=4,5	
t2		**read(11548, teoría)** teoría=4,5
t3	teoría←teoría+1,5 teoría=6	
t4		teoría←teoría+2 teoría=6,5
t5	**write(11548, teoría)**	
t6		**write(11548, teoría)**

Two programs reading and updating the theory credits of subject 11548:

- *P1 adds 1,5 credits*
- *P2 adds 2 credits*

Subject

cod_asg	nombre	semestre	cod_dep	teoría	prácticas
11545	Análisis Matemático	1A	DMA	4,5	1,5
11546	Álgebra	1B	DMA	4,5	1,5
11547	Matemática Discreta	1A	DMA	4,5	1,5
11548	Bases de Datos y Sistemas de Información	3A	DSIC	6,5	1,5

4,5 + 1,5 + 2 = 7

B. The inconsistent analysis problem

Program 1: List for each lecturer his/her amount of credits

Teaching

dni	cod_asg	gteo	gpra
111	11545	2	3
123	11545	0	2
123	11547	1	1
564	11545	1	2

Subject

cod_asg	...	teoría	prácticas
11545	...	4,5	1,5
11546	...	4,5	1,5
11547	...	4,5	1,5
11548	...	4,5	1,5

Time	Program 1	Program 2
t1	Calculate credits for lecturer 111 **Credits 111= 9** (1×4,5 + 3×1,5)	
t2	Calculate credits for lecturer 123 **Credits 123= 9** (0×4,5 + 2×1,5 + 1×4,5 + 1× 1,5)	
t3		Change a theory group of subject 11545 from lecturer 564 to lecturer 111
t4	Calculate credits for lecturer 453 **Credits 453= 0**	
t5	Calculate credits for lecturer 564 **Credits 564= 7,5** (1×4,5 + 2×1,5)	

Result:

DNI	Credits
111	*9 credits*
123	*9 credits*
453	*0 credits*
564	*7,5 credits*

C. The uncommitted dependency problem

Subject

cod_asg	...	teoría	prácticas
11545	...	4,5	1,5
11546	...	4,5	1,5
11547	...	4,5	1,5
11548	...	6	1,5

Time	Program 1	Program 2
t1	**read(11548, teoría)**	
t2	teoría←teoría+1,5	
t3	**write(11548,teoría)**	
t4		**read(11548,teoría)** teoría=6
t5		*Use this value (teoría=6) in its instructions*
t6		**confirmation**
t7	**cancellation**	

Techniques

Reserving some data occurrences (locks)

- In examples **a)** and **c)** P1 must lock a record.
- In examples **b)** all the table must be locked.
- Need for controlling deadlocks

Other solutions (for the example c): Cascade cancellation or transaction isolation

Unit 3. Database Management Systems

1. The ANSI/SPARC Architecture
   1.1. Schemas
   1.2. DBMS fundamentals
   1.3. Data independence
2. Transactions, Integrity, and Concurrency
   2.1. Transactions
   2.2. Semantic integrity
   2.3. Concurrent access control
3. **Recovery and Security**
   3.1. DB Recovery
   3.2. Security

Recovery and Security

A database must guarantee:

- Recovery:

  (Part of integrity, but not from the point of view of consistency. Recovery is focus on durability and persistence)

  A database must always be recovered from any type of failure.

- Security:

  A database cannot allow non-authorized access.

3.1. DB Recovery

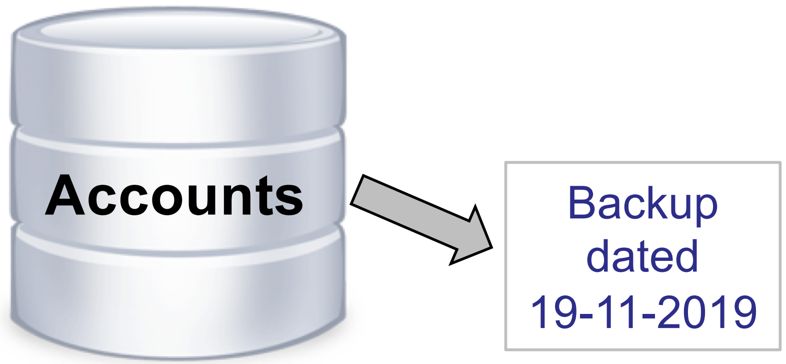
DB recovery

The transaction properties of atomicity and durability force a DBMS to ensure that:

- If confirmed, the changes performed are recorded in the DB to make them persistent.
- If cancelled, the changes performed over the DB are undone.

Database recovery: Example

Backups are not sufficient



20-11-2019: Update on accounts

- Transaction #1: Update ….
- Transaction #2: Update ….
- Transaction #3: Update ….
- …
- Transaction #51: System failure !

Recovery Procedure:

- Replace the file “Accounts” with its backup

Negative effect:

- The updates of 50 transactions are lost

DB Recovery

Backups alone are not the solution to the recovery problem.

- The increase of the backup frequency is not a feasible solution.

DB technology provides much more efficient and robust techniques for DB recovery.

Lost of confirmed data is inadmissible with current technology.

DB Recovery

Backups alone are not the solution to the recovery problem.

- The increase of the backup frequency is not a feasible solution.

DB technology provides much more efficient and robust techniques for DB recovery.

Lost of confirmed data is inadmissible with current technology.

Causes of transaction failure

1. **Local to the transaction** (normal system operation)
   - Transaction error (incorrect DB access, failed calculations, etc.)
   - Exceptions (integrity violation, security problems, etc.)
   - Concurrency control (locked state between two transactions)
   - Human decision (inside a program or explicitly).

2. **Extern to the transaction** (system error)
   A. System failures with loss of main memory.
   B. Failures in the storage system with loss of secondary memory (disk failure, human errors, virus infection,…)

A. Failures of system main memory

- The changes performed by a transaction are located in memory buffers (main memory).
- When the transaction is confirmed its changes must be recorded in secondary memory.
- If a failure with loss of main memory occurs between the transaction confirmation and flushing the buffers to secondary memory, the blocks in the buffers will be lost.

B. Failures of secondary memory

- The changes performed by a confirmed transaction are recorded into the DB

- If there is a failure in secondary memory, the changes will be lost.

3.1.1 Recovery from failures of system main memory

Recovery from failures of system main memory

What to do:

- Recover confirmed transactions which have not been recorded.
- Cancel transactions which have failed.

Who:

Recovery module

How:

Most used technique: Use a *journal file (or log, "fichero diario")*.

Transaction implementation

Two ways of implementing transactions:

- Transactions with Immediate Update
  Updates have an immediate effect on secondary memory.
  In case of cancellation, they have to be undone.

- Transactions with Deferred Update
  Updates only have immediate effect on main memory.
  The updates will be transferred to secondary memory
  when confirmed.

- Both need a log file

Log (journal) file

Activities and events are recorded in the log file:

- Record the update operations performed by existing transactions during a period of time (e.g. one day)
- The log (journal) file is stored on disk to avoid loss after a system failure.
- It is dumped periodically into a massive storage unit (magnetic tape, optical disk,…).

Types of entries in a log file

[**start**, T]: A transaction with identifier T has been started.

[**write**, T, X, value_before, value_after]: The T transaction has performed an update instruction on data X.

[**read**, T, X]: The T transaction has read data X.

[**confirm**, T]: The T transaction has been confirmed.

[**cancel**, T]: The T transaction has been cancelled.

Problems:

- Size of log file can increase very quickly.
- Recovery in case of failure is very expensive (many instructions have to be redone).

Solution:

checkpoints *(“puntos de control o verificación”)*

Checkpoints $\rightarrow$ They are recorded in the log file periodically

How it works ?

- Suspend the execution of transactions temporally.
- Record a checkpoint in the log file.
- Record all updates performed by confirmed transactions (copy all main memory buffers to disk).
- Resume the execution of the suspended transactions..

A. INMEDIATE DATABASE UPDATES

Updates have an immediate effect on secondary memory. In case of cancellation, they have to be undone

Failure of a transaction $T \rightarrow$ Undo changes performed by T

Update the data which has been modified by T with its original value (*value_before*): Search for the entries in the logfile [write, T, X, *value_before*, *value_after*]

Unconfirmed transactions

[start, T] in the log file without *[confirm, T]*

→ Undo changes performed by *T* (previous process)

Confirmed transactions

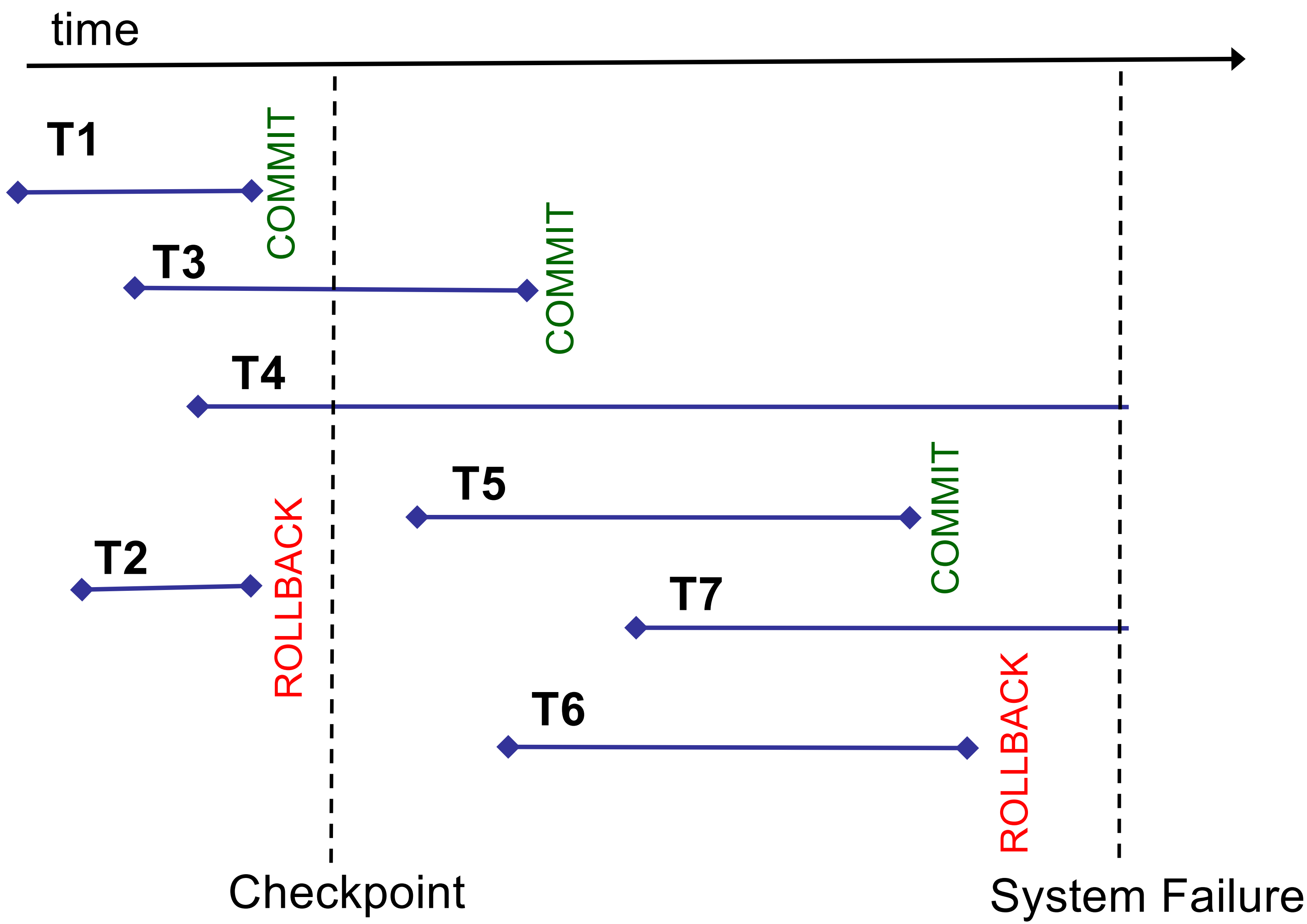
[confirm, T]

→ Execute (redo) them again:

[write, T, X, *value_before*, *value_after*]

DB recovery with **immediate** Updates

Recovery the DB from the last checkpoint



DB recovery with **immediate** Update

Basic considerations:

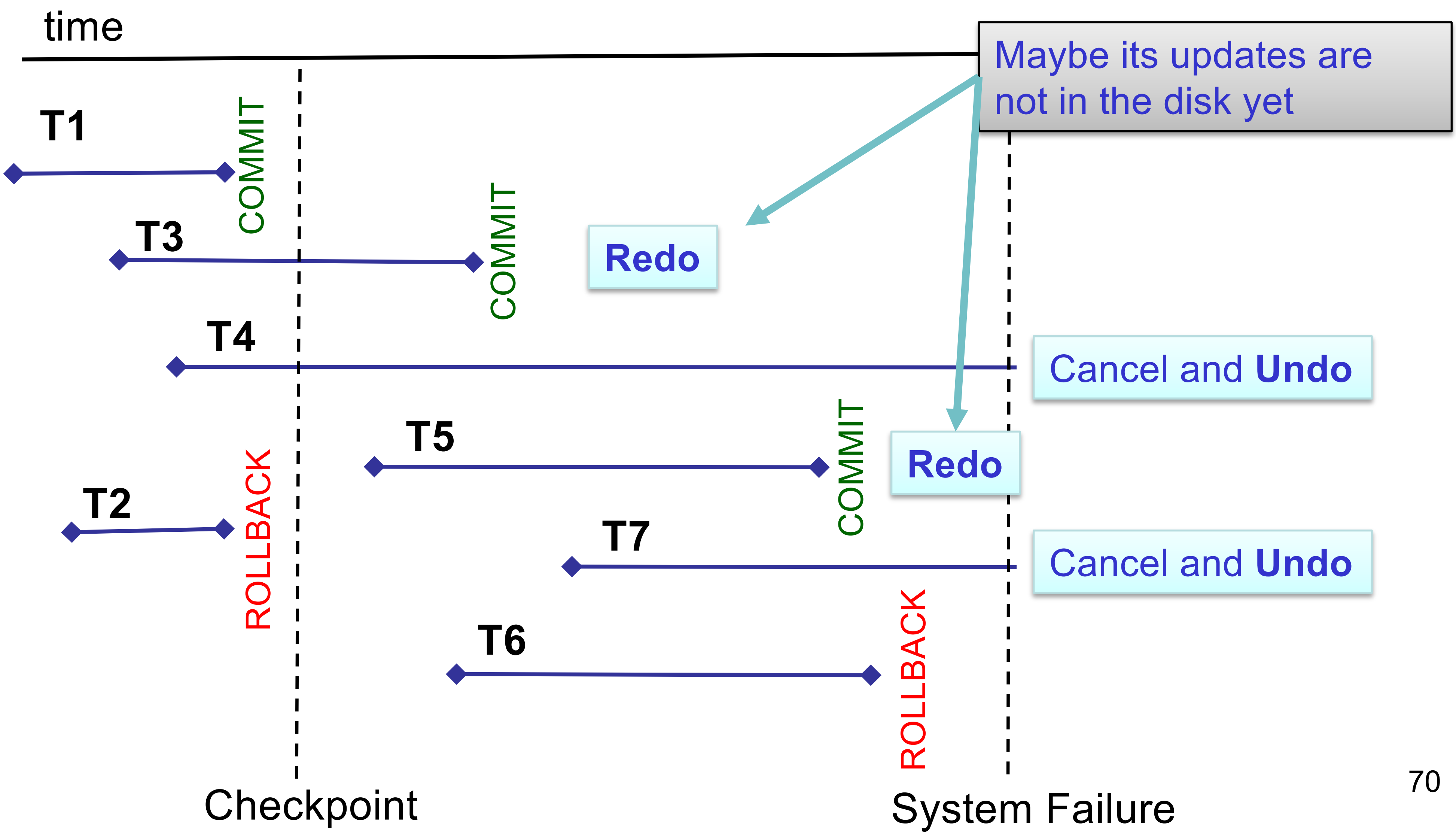
- Updates performed by confirmed transaction (*a transaction commit appears in the log file)* could have not been transferred to disk because the buffer block where they are, has not been recorded yet: **Redo**

- Updates performed by non-confirmed transactions (there is no transaction commit in the log file) could be in disk because their main memory blocks were transferred to disk: **Undo** (only used in immediate updates)

- When a checkpoint is recorded, the DBMS records all the updates performed by the confirmed transaction.

DB recovery with **immediate** Update

Recovery the DB from the last checkpoint



B. DEFERRED DATABASE UPDATES

Updates only have immediate effect on main memory. The updates are only written to the secondary memory until confirmed (after the commit).

- Unconfirmed transactions

  [start, T] in the log file without *[confirm, T]*

  → do nothing they are not in secondary memory

- Confirmed transactions

  [confirm, T]

  → Execute (redo) them again:

  [write, T, X, value_before, value_after]

DB recovery with **deferred** Update

Basic considerations:

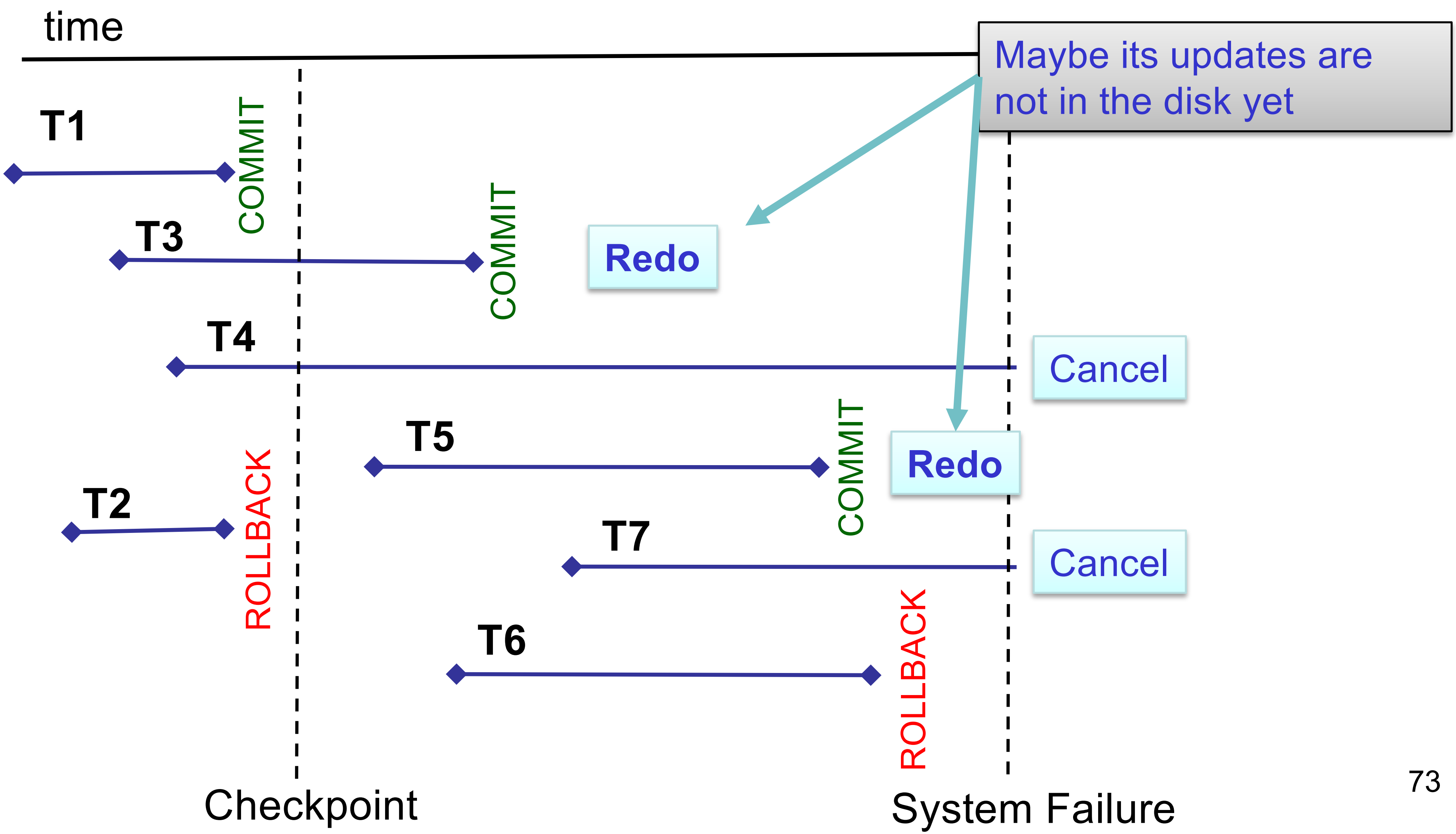
- Updates performed by confirmed transaction (*a transaction commit appears in the log file)* could have not been transferred to disk because the buffer block where they are, has not been recorded yet: **Redo**

- Updates performed by non-confirmed transactions (there is no transaction commit in the log file) are not in disk : **Do nothing**

- When a checkpoint is recorded, the DBMS records all the updates performed by the confirmed transaction.

DB recovery with **deferred** Update

Recovery the DB from the last checkpoint



3.1.2 Recovery from failures of secondary memory

Recovery from failures of secondary memory

When a failure of the storage system occurs, the database might be damaged totally or partially.

Technique:

Reconstruction of the database:

- Using the most recent backup
- From the backup instant, the system uses the log file to redo all the instructions performed by the confirmed transactions.

3.2. DB Security

3.2. Security

Objective:

Information can only be accessed by the people and processes that are authorized and in the authorized way

Techniques

- User identification
- Establishment of allowed accesses:

Modes:

- Authorization list associated to each user containing the allowed objects and operations. *(GRANT)*
- Level of authorization (less flexible). There several users groups with different authorization.

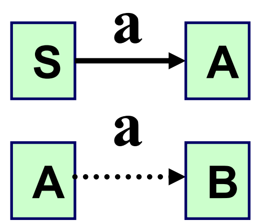
- Management of transferrable authorizations:

  Handover of authorizations from one user to another. *(WITH GRANT OPTION)*

Management of transferrable authorizations

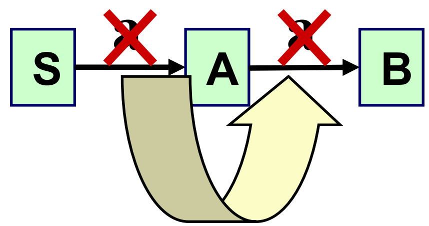
It is necessary to know the access authorizations of each user (some authorizations will be transferable to other users).

- One authorization can be transferred to other user in mode transferable or not

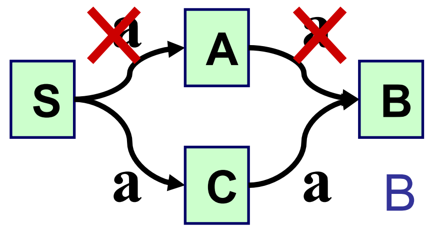


When one authorization is revoked:

- If the authorization was transferable, it is necessary to revoke all the transferred authorizations



if one user receives more than one authorization, each of the authorizations can be independently revoked.



B maintains the authorization a

3.2.1. Privacy and Security

Ethical and legal implications

Extreme care on:

- Protection against the access or spreading of personal data to non-authorized users.
- Control the flow to third parties of information that can contain personal data or information that is apparently aggregated (parameterized queries) but that might reveal particular information for some parameters.
- Custody of security backups, retired or malfunctioning disks, etc.
- Small devices (USB disks or sticks, smart phones, tablets, etc.): lost or stolen very easily.

Personal Data:

- Personal data is "any information relating to an identified or identifiable natural person."
- In the European Union: The EU General Data Protection Regulation (GDPR) became enforceable across Europe on 25 May 2018
- Spain *"**Agencia de Protección de Datos**"*.

Exercises

1.- The physical schema of a database is modified due to a change of disks. If an application uses a view that in turn uses tables that are stored in any of these disks, what happens to the application?

Justify the answer according to the ANSI/SPARC architecture and the concept of independence.

2.- Consider two transactions T1 and T2, which are running concurrently, both working on a piece of data X. Indicate whether one or more properties of transactions are not satisfied here. Briefly justify the answer.

T1	X in T1	T2	X in T2
read(X)	5		
X=X+1	6		
write(X)	6		
	6	read(X)	6
confirm	6	…	6
…			

Time

3. How can be recovered a DB from a main memory failure using a logfile and checkpoints ? (Assume immediate updates)

4. Consider the following time diagram, and assuming a DBMS with immediate update. If a system failure occurs, as illustrated in the following figure

a) What should the DBMS do if the system failure is a main memory loss?

b) What should the DBMS od if the failure is a secondary memory loss ?