BACS3183
Advanced Database Management

Chapter 8

Transaction Processing

Learning Outcomes

- Describe the techniques for data security
- Explain the function, importance and properties of transactions.
- Describe problems and techniques for concurrency control of transactions.
- Describe techniques for database recovery

1. Database Software Security Features

- Views
- Integrity controls
- Authorization
- Encryption
- Authentication schemes
- Backup & recovery

Views

- Views
  - Subset of the database that is presented to one or more users
  - User can be given access privilege to view without allowing access privilege to underlying tables

```
CREATE VIEW low_stock_view AS
   SELECT *
   FROM products
   WHERE quantityInStock<50
   WITH READ ONLY CONSTRAINT low_stock_readOnly;
```

Integrity Controls

- Ensures that data entered into the database is accurate, valid, and consistent.
- Three basic types of integrity controls are:
  - Entity integrity, not allowing multiple rows to have the same identity within a table.
    - ✓ Primary key
  - Domain integrity, restricting data to predefined data types, e.g.: dates. Set allowable values

    Eg CREATE DOMAIN PriceChange AS DECIMAL
    CHECK (VALUE BETWEEN .001 and .15);
  - Referential integrity, requiring the existence of a related row in another table, e.g. a customer for a given customer ID.

Referential Integrity

- **Referential integrity means that, if FK contains a value, that value must match a CK in parent table.**

Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Action taken attempting to update/delete a CK value in parent table with matching rows in child is dependent on referential action specified using ON UPDATE and ON DELETE subclauses:

- CASCADE
- SET DEFAULT
- SET NULL
- NO ACTION

Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

```
CREATE TABLE Staff (
   StaffNo    VARCHAR(4) NOT NULL,
    ……
   BranchNo VARCHAR(4)  DEFAULT 'B005',
   PRIMARY KEY (StaffNo),
   FOREIGN KEY (BranchNo) REFERENCES Branch
      ON UPDATE CASCADE
      ON DELETE  SET DEFAULT
);
```

Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Referential Integrity

CASCADE: Delete row from parent and delete matching rows in child, and so on in cascading manner.

SET NULL: Delete row from parent and set FK column(s) in child to NULL. Only valid if FK columns are NOT NULL.

SET DEFAULT: Delete row from parent and set each component of FK in child to specified default. Only valid if DEFAULT specified for FK columns.

NO ACTION: Reject delete from parent. Default.

Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchN
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Authorization

- **Controls incorporated in the DBMS to restrict**
  - **–access to data**
  - **–actions that people can take on data**

Authorization table for subjects (salespeople)

	Customer records	Order records
Read	Y	Y
Insert	Y	Y
Modify	Y	N
Delete	N	N

Authorization table for objects (orders)

	Salespersons (password BATMAN)	Order entry (password JOKER)	Accounting (password TRACY)
Read	Y	Y	Y
Insert	N	Y	N
Modify	N	Y	Y
Delete	N	N	Y

Oracle privileges

Privilege	Capability
SELECT	Query the object.
INSERT	Insert records into the table/view.
	Can be given for specific columns.
UPDATE	Update records in table/view.
	Can be given for specific columns.
DELETE	Delete records from table/view.
ALTER	Alter the table.
INDEX	Create indexes on the table.
REFERENCES	Create foreign keys that reference the table.
EXECUTE	Execute the procedure, package, or function.

Privileges can be granted to users at the database level or table level. INSERT and UPDATE can be granted at the column level.

Authorization

- **Authorization matrix for:**
  - **Subjects**
  - **Objects**
  - **Actions**
  - **Constraints**

Subject	Object	Action	Constraint
Sales Dept.	Customer record	Insert	Credit limit LE $5000
Order trans.	Customer record	Read	None
Terminal 12	Customer record	Modify	Balance due only
Acctg. Dept.	Order record	Delete	None
Ann Walker	Order record	Insert	Order aml LT $2000
Program AR4	Order record	Modify	None

Authorization

The owner of a relation R is given all privileges on R.

The owner account holder can pass privileges to other users by granting privileges to their accounts.

```
GRANT SELECT, UPDATE (salary)
ON Staff
TO Personnel, Director;
```

```
GRANT   {PrivilegeList | ALL PRIVILEGES}
ON      ObjectName
TO      {AuthorizationIdList | PUBLIC}
[WITH GRANT OPTION]
```

```
SELECT
DELETE
INSERT        [(columnName [, . . . ])]
UPDATE        [(columnName [, . . . ])]
REFERENCES    [(columnName [, . . . ])]
USAGE
```

WITH GRANT ACTION, the users receiving the privilege may in turn grant it to other user

To create a view, the account must have SELECT privilege on all relations involved in the view definition.

Authorization

```
REVOKE [GRANT OPTION FOR] {PrivilegeList | ALL PRIVILEGES}
ON      ObjectName
FROM    {AuthorizationIdList | PUBLIC} [RESTRICT | CASCADE]
```

```
REVOKE ALL PRIVILEGES
ON Staff
FROM Director;
```

Note:
ALL PRIVILEGES refers to all privileges granted to a user by user revoking privileges

An Example

EMPLOYEE

Name	Ssn	Bdate	Address	Sex	Salary	Dno

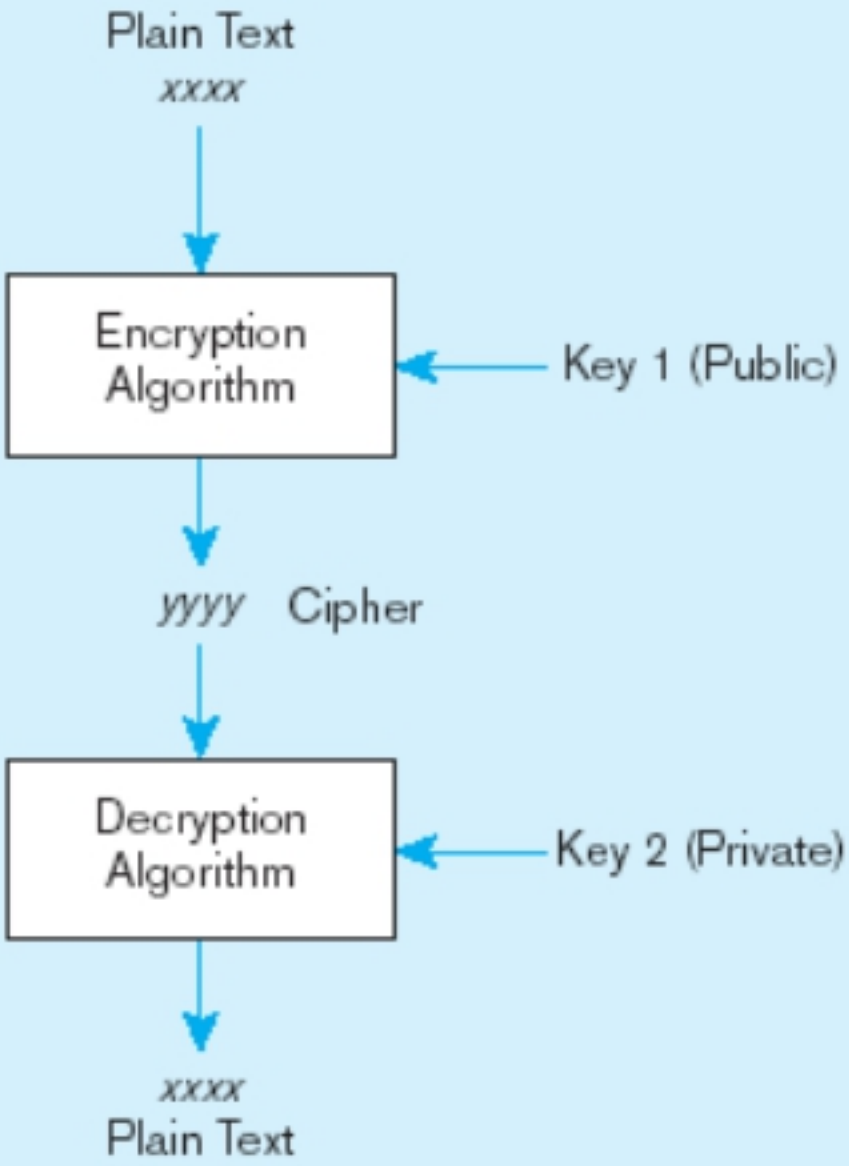
Question: Suppose that A wants to allows B to retrieve only the NAME, BDATE, and ADDRESS attributes and only for the tuples with DNO=5, for B to propagate the privilege.

How can this be achieved?

```
CREATE VIEW BEMPLOYEE AS
        SELECT NAME, BDATE, ADDRESS
        FROM EMPLOYEE
        WHERE DNO = 5;

GRANT SELECT ON BEMPLOYEE TO B
                WITH GRANT OPTION;
```

Encryption



Encryption – the coding or scrambling of data so that humans cannot read them

Secure Sockets Layer (SSL) is a popular encryption scheme for TCP/IP connections

Authentication Schemes

- Goal – obtain a *positive* identification of the user
- Passwords: First line of defense
  - Complexity requirement
  - Should be changed frequently
- Two factor – e.g. smart card plus PIN
- Three factor –e.g. smart card, biometric, PIN
- Biometric devices – use of fingerprints, retinal scans, etc. for positive ID
- Third-party mediated authentication – using secret keys, digital certificates

2. Transaction Support

- **Transaction**
  - ✓ **Action, or series of actions, carried out by user or application, which reads or updates contents of database.**
  - ✓ **A Logical unit of work on the database.**
- **Instructions for specifying boundaries of a transaction:**
  - ***BEGIN TRANSACTION***
  - ***COMMIT*: transaction is completed successfully ie database is transformed from one consistent state to another consistent state.**
  - ***ROLLBACK*: transaction is aborted - database is restored to the consistent state it was in before the transaction started.**

ATM Transaction Example

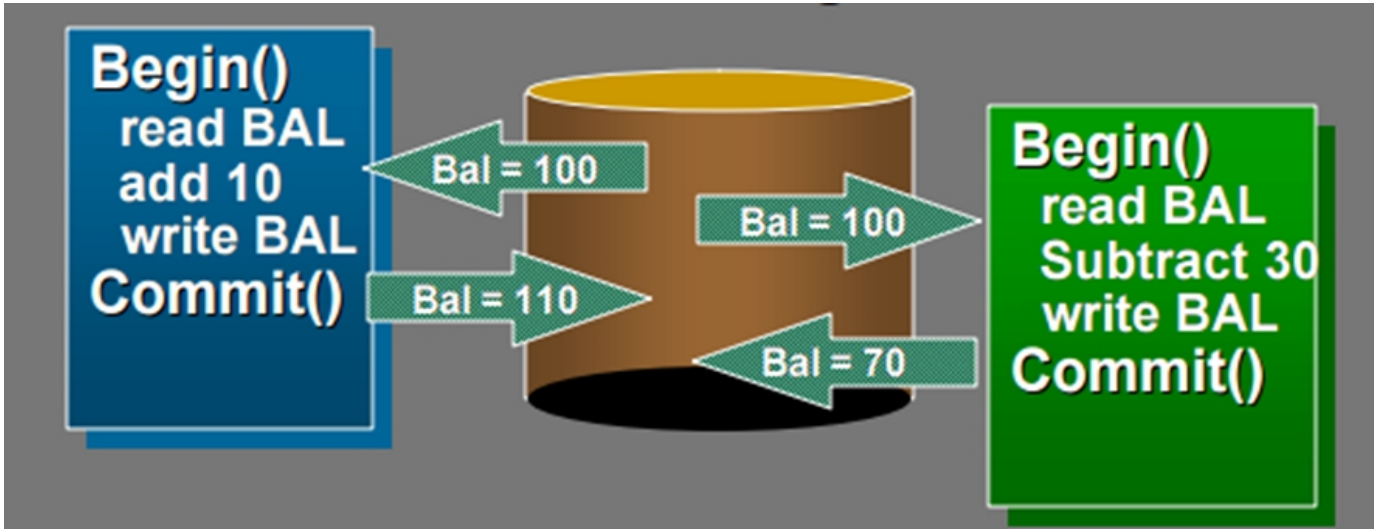
```
BEGIN TRANSACTION
    Display greeting
    Get account number, pin, type, and amount
    SELECT account number, type, and balance
    If balance is sufficient then
        UPDATE account by posting debit
        INSERT history record
        Display message and dispense cash
        Print receipt if requested
    End If
    On Error: ROLLBACK
COMMIT
```

Transaction (ACID) Properties

- **Atomicity: all or nothing**
- **Consistency: database must be consistent before and after a transaction**
- **Isolation: Partial effects of incomplete transactions should not be visible to other transactions.**
- **Durability: Effects of a committed transaction are permanent and must not be lost because of later failure.**

2.1 Concurrency Control

- Process of managing simultaneous operations on the database without having them interfere with one another.
- Prevents interference when two or more users are accessing database simultaneously and at least one is updating data.
- Although two transactions may be correct in themselves, interleaving of operations may produce an incorrect result.

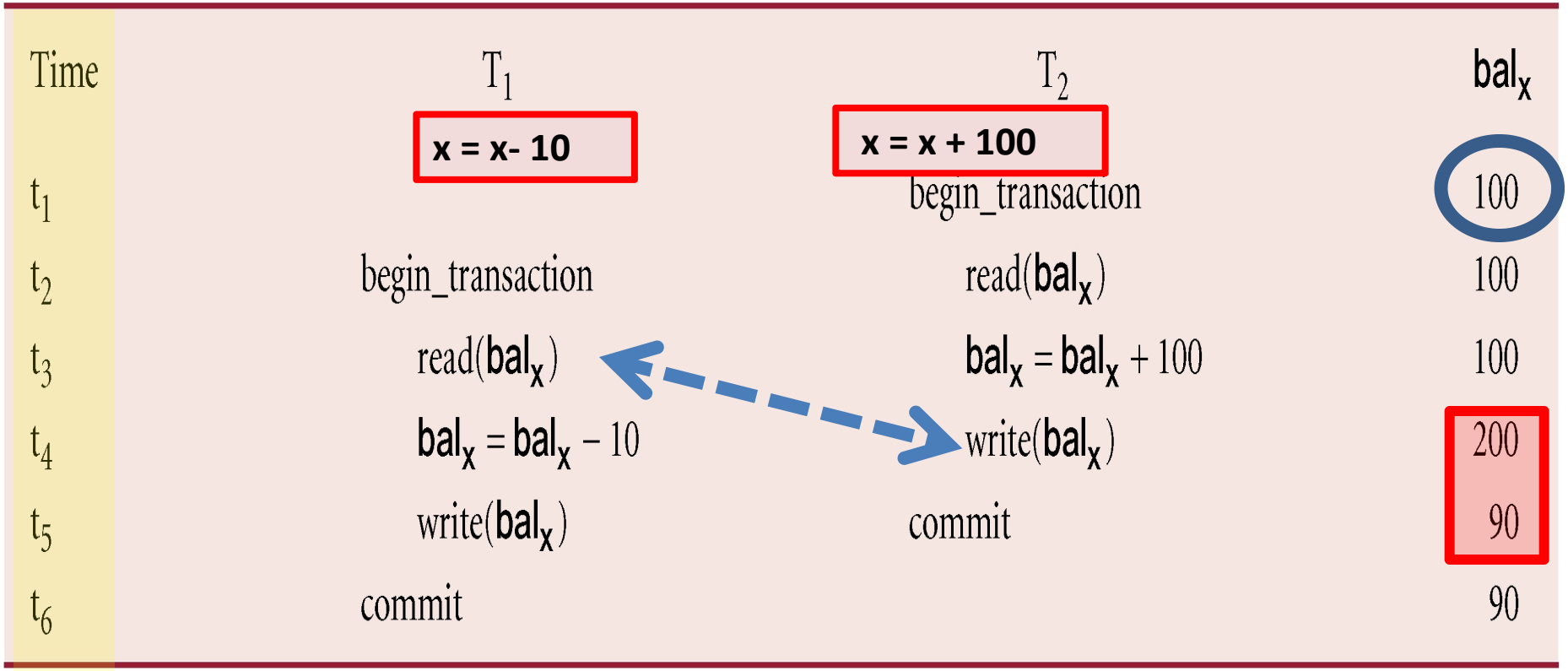


Need for Concurrency Control

- Three examples of potential problems caused by concurrency:
  - Lost update problem.
  - Uncommitted dependency problem (dirty read).
  - Inconsistent analysis problem.

Lost Update Problem

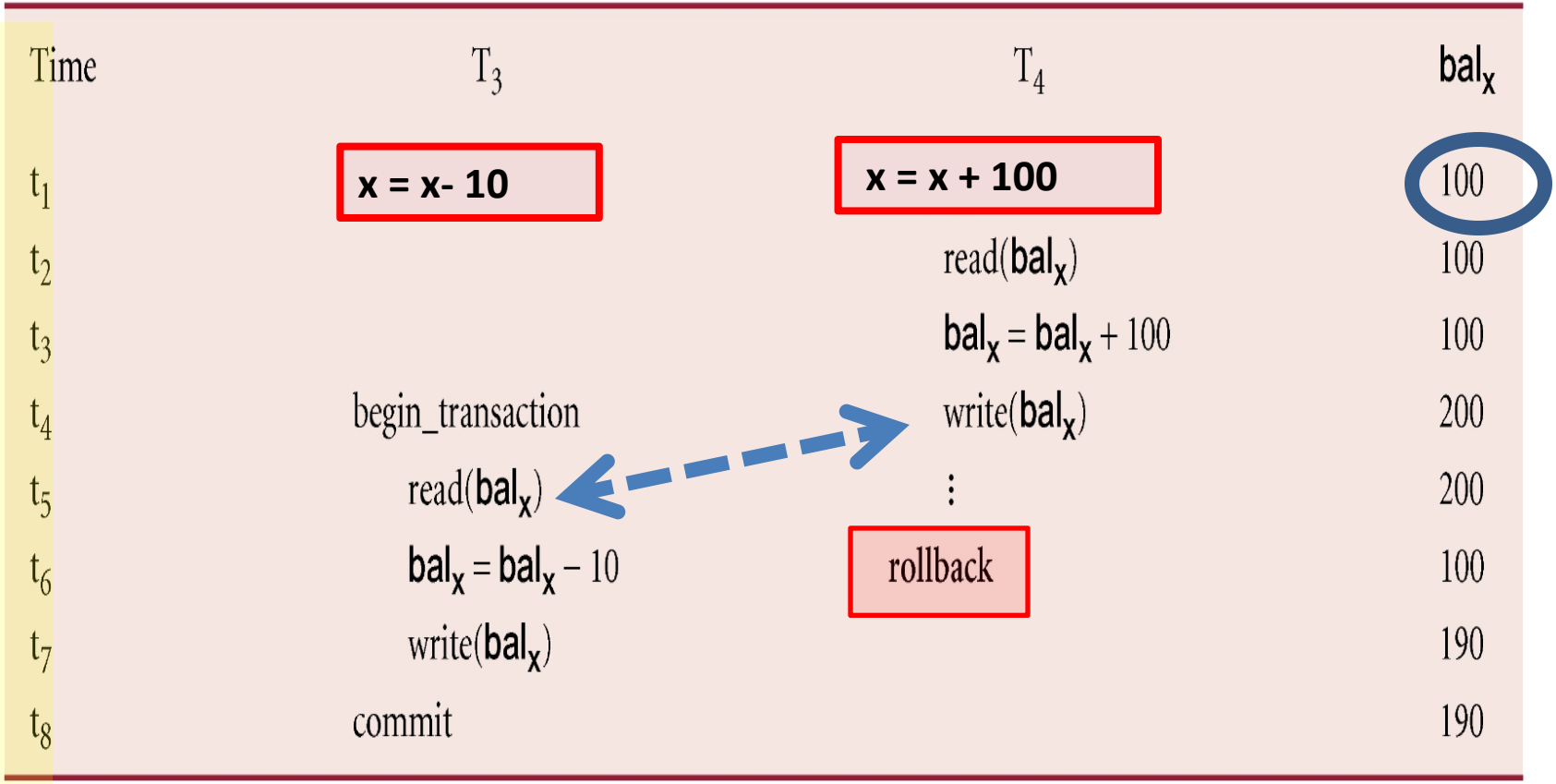
- **Successfully completed update is overridden by another user.**



- **Loss of T_2's update can be avoided by preventing T_1 from reading bal_x until after update.**

Uncommitted Dependency (Dirty Read) Problem

- Occurs when one transaction can see intermediate results of another transaction before it has committed.



- Problem can be avoided by preventing T_3 from reading bal_x until after T_4 commits or aborts.

Inconsistent Analysis Problem

- Occurs when a transaction **reads** several values but second transaction **updates** some of them during execution of first.

Time	T_5	T_6	bal_x	bal_y	bal_z	sum
	x = x-10 z = z+10	Sum = x+y+z				
t_1		begin_transaction	100	50	25	
t_2	begin_transaction	sum = 0	100	50	25	0
t_3	read(bal_x)	read(bal_x)	100	50	25	0
t_4	$bal_x = bal_x - 10$	sum = sum + bal_x	100	50	25	100
t_5	write(bal_x)	read(bal_y)	90	50	25	100
t_6	read(bal_z)	sum = sum + bal_y	90	50	25	150
t_7	$bal_z = bal_z + 10$		90	50	25	150
t_8	write(bal_z)		90	50	35	150
t_9	commit	read(bal_z)	90	50	35	150
t_{10}		sum = sum + bal_z	90	50	35	185
t_{11}		commit	90	50	35	185

- Problem can be avoided by preventing T_6 from reading bal_x and bal_z until after T_5 completed updates.

2.2 Serializability

Schedule: A sequence of reads/writes by set of concurrent transactions.

Serial Schedule: operations of each transaction are executed consecutively without any interleaved operations from other transactions.

Non-serial Schedule: operations from set of concurrent transactions are interleaved.

Time	T_7	T_8
t_1	begin_transaction	
t_2	read(bal_x)	
t_3	write(bal_x)	
t_4		begin_transaction
t_5		read(bal_x)
t_6		write(bal_x)
t_7	read(bal_y)	
t_8	write(bal_y)	
t_9	commit	
t_{10}		read(bal_y)
t_{11}		write(bal_y)
t_{12}		commit

Example

Non-Serial Schedule

Time	T_7	T_8
t_1	begin_transaction	
t_2	read(bal_x)	
t_3	write(bal_x)	
t_4		begin_transaction
t_5		read(bal_x)
t_6		write(bal_x)
t_7	read(bal_y)	
t_8	write(bal_y)	
t_9	commit	
t_{10}		read(bal_y)
t_{11}		write(bal_y)
t_{12}		commit

(a)

T_7	T_8
begin_transaction	
read(bal_x)	
write(bal_x)	
	begin_transaction
	read(bal_x)
read(bal_y)	
	write(bal_x)
write(bal_y)	
commit	
	read(bal_y)
	write(bal_y)
	commit

(b)

Serial Schedule

T_7	T_8
begin_transaction	
read(bal_x)	
write(bal_x)	
read(bal_y)	
write(bal_y)	
commit	
	begin_transaction
	read(bal_x)
	write(bal_x)
	read(bal_y)
	write(bal_y)
	commit

(c)

Serializability

- **Objective of serializability is to find non-serial schedules that allow transactions to execute concurrently without interfering with one another.**
- **In other words, want to find non-serial schedules that are equivalent to *some* serial schedule. Such a schedule is called *serializable*.**

Time	T_7	T_8
t_1	begin_transaction	
t_2	read(bal_x)	
t_3	write(bal_x)	
t_4		begin_transaction
t_5		read(bal_x)
t_6		write(bal_x)
t_7	read(bal_y)	
t_8	write(bal_y)	
t_9	commit	
t_{10}		read(bal_y)
t_{11}		write(bal_y)
t_{12}		commit

(a)

T_7	T_8
begin_transaction	
read(bal_x)	
write(bal_x)	
	begin_transaction
	read(bal_x)
read(bal_y)	
	write(bal_x)
write(bal_y)	
commit	
	read(bal_y)
	write(bal_y)
	commit

(b)

T_7	T_8
begin_transaction	
read(bal_x)	
write(bal_x)	
read(bal_y)	
write(bal_y)	
commit	
	begin_transaction
	read(bal_x)
	write(bal_x)
	read(bal_y)
	write(bal_y)
	commit

(c)

Serializability

In serializability, ordering of read/writes is important:

(a) If two transactions only read a data item, they do not conflict and order is not important.

(b) If two transactions either read or write separate data items, they do not conflict and order is not important.

(c) If one transaction writes a data item and another reads or writes same data item, order of execution is important.

2.3 Concurrency Control Techniques

- **Conservative approaches**: **delay transactions** in case they conflict with other transactions.
  - Locking,
  - Timestamping.

- **Optimistic** methods assume conflict is rare and only check for conflicts at commit
  - Versioning (multi-version)

2.3.1 Locking

- Most widely used approach to ensure serializability.

- Two types of lock
  - shared (read) lock: a transaction with a shared lock on a data item can read the item but not update it. Used when just reading to prevent another user from placing an exclusive lock on the record.

  - exclusive (write) lock: a transaction with exclusive lock on a data item can both read and update the item.

Lock Compatibility Matrix

	User 2 Requests	
User 1 Holds	***Shared/Read Lock***	***Exclusive/Write Lock***
Shared/Read Lock	**Lock granted**	**User 2 waits**
Exclusive/Write Lock	**User 2 waits**	**User 2 waits**

- ✓ **Any number of users can hold a shared lock on the same item.**
- ✓ **Only one user can hold an exclusive lock.**

Locking Mechanisms

Locking level:

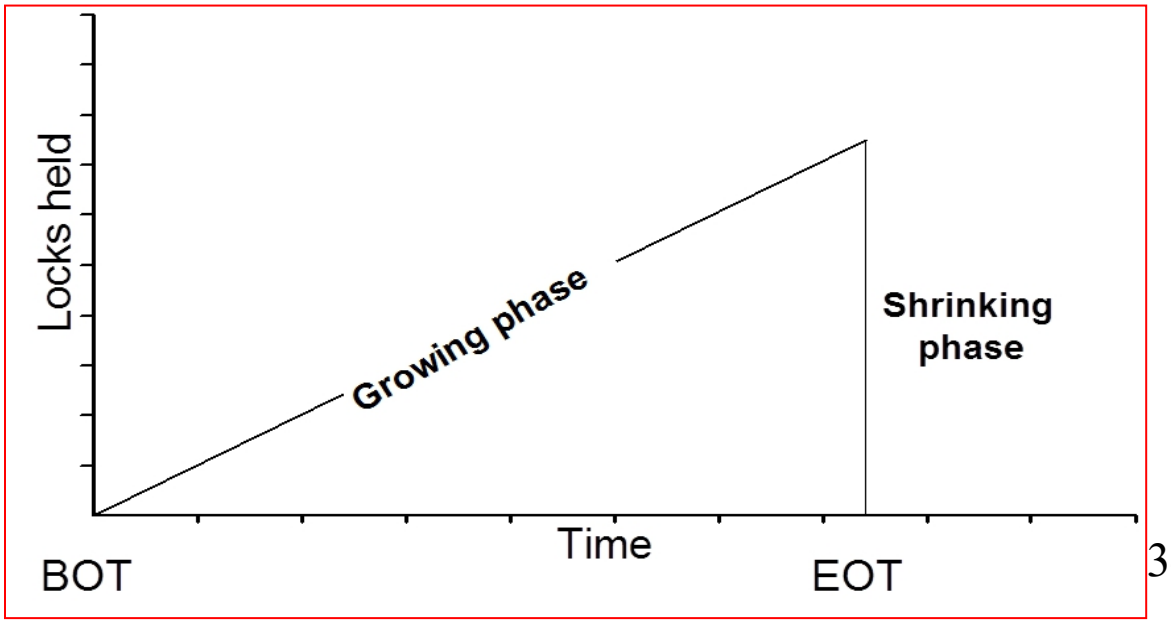
- **Database – it prevents the use of any tables in the database by transaction T2 while transaction T1 is being executed**
  - suitable for batch processes but not on-line multi-user DBMSs.
- **Table –it prevents the access to any row by transaction T2 while transaction T1 is using the table; not suitable for multi-user DBMS**
- **Page/block - A table can span several pages, and a page can contain several rows (tuples) of one or more tables.**
  Most suitable for multi-user DBMSs.
- **Record– The DBMS allows concurrent transactions to access different rows of the same table, even if the rows are located on the same page**
- **Attribute –allows concurrent transactions to access the same row, as long as they require the use of different attributes within the rowrequires significant overhead; impractical**

Two-Phase Locking (2PL)

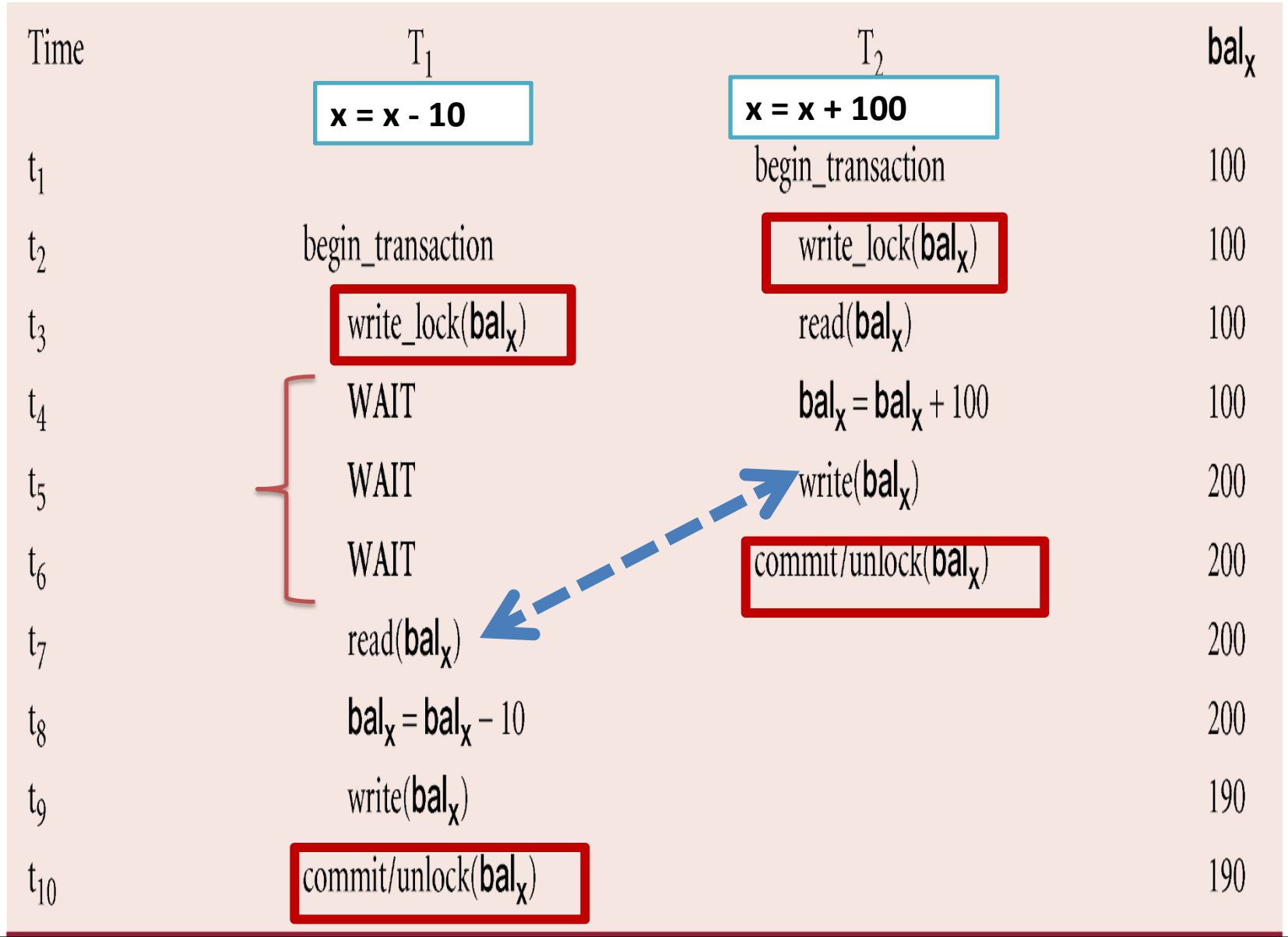
A transaction follows 2PL protocol if all locking operations precede first unlock operation in the transaction.

Two phases for every transaction:

- **Growing phase - acquires locks without releasing any locks.**
- **Shrinking phase - releases locks without acquiring new locks.**



Preventing Lost Update Problem using 2PL



Preventing Uncommitted Dependency Problem using 2PL

Time	T_3	T_4	bal_x
	x = x - 10	x = x + 100	
t_1		begin_transaction	100
t_2		write_lock(bal_x)	100
t_3		read(bal_x)	100
t_4	begin_transaction	$bal_x = bal_x + 100$	100
t_5	write_lock(bal_x)	write(bal_x)	200
t_6	WAIT	rollback/unlock(bal_x)	100
t_7	read(bal_x)		100
t_8	$bal_x = bal_x - 10$		100
t_9	write(bal_x)		90
t_{10}	commit/unlock(bal_x)		90

Inconsistent Analysis Problem (recap)

- Occurs when a transaction **reads** several values but second transaction **updates** some of them during execution of first.

Time	T_5	T_6	bal_x	bal_y	bal_z	sum
	x = x-10 z = z+10	Sum = x+y+z				
t_1		begin_transaction	100	50	25	
t_2	begin_transaction	sum = 0	100	50	25	0
t_3	read(bal_x)	read(bal_x)	100	50	25	0
t_4	$bal_x = bal_x - 10$	sum = sum + bal_x	100	50	25	100
t_5	write(bal_x)	read(bal_y)	90	50	25	100
t_6	read(bal_z)	sum = sum + bal_y	90	50	25	150
t_7	$bal_z = bal_z + 10$		90	50	25	150
t_8	write(bal_z)		90	50	35	150
t_9	commit	read(bal_z)	90	50	35	150
t_{10}		sum = sum + bal_z	90	50	35	185
t_{11}		commit	90	50	35	185

- Problem can be avoided by preventing T_6 from reading bal_x and bal_z until after T_5 completed updates.

Deadlock

An impasse that may result when two (or more) transactions are each waiting for locks held by the other to be released.

Time	x = x - 10 y = y + 20	y = y + 100 x = x + 70
t_1	begin_transaction	
t_2	write_lock(bal_x)	begin_transaction
t_3	read(bal_x)	write_lock(bal_y)
t_4	$bal_x = bal_x - 10$	read(bal_y)
t_5	write(bal_x)	$bal_y = bal_y + 100$
t_6	write_lock(bal_y)	write(bal_y)
t_7	WAIT	write_lock(bal_x)
t_8	WAIT	WAIT
t_9	WAIT	WAIT
t_{10}	⋮	WAIT
t_{11}	⋮	⋮

Deadlock

- **Three general techniques for handling deadlock:**
  - **Timeouts.**
  - **Deadlock prevention.**
  - **Deadlock detection and recovery.**

Timeouts

- Transaction that requests lock will only wait for a system-defined period of time.
- If lock has not been granted within this period, lock request times out.
- DBMS aborts and automatically restarts the transaction.

t_1	begin_transaction	
t_2	write_lock(bal_x)	begin_transaction
t_3	read(bal_x)	write_lock(bal_y)
t_4	$bal_x = bal_x - 10$	read(bal_y)
t_5	write(bal_x)	$bal_y = bal_y + 100$
t_6	write_lock(bal_y)	write(bal_y)
t_7	WAIT	write_lock(bal_x)
t_8	WAIT	WAIT
t_9	WAIT	WAIT
t_{10}	⋮ aborted	WAIT
t_{11}	⋮	⋮

Deadlock Prevention

- DBMS looks ahead to see if transaction would cause deadlock and never allows deadlock to occur.
- Could order transactions using transaction timestamps:
  - Wait-Die
  - Wound-Wait

Wait-Die

Condition 1: **If timestamp of T1 is smaller than the timestamp of T2, then allow T1 to wait for T2 to release lock**

Condition 2: **If timestamp of T1 is larger than the timestamp of T2, then T1 is aborted (rollback and restart)**

Transaction T1 TS1 – "2014-03-15 23:50:56" *Older Transaction*	Transaction T2 TS2 – "2014-03-15 23:51:12" *Younger Transaction*
Read (X)	
Read (Y)	
	Read (X)
	Read (Y)
Write (X)	
	Write (Y)
Request for write_lock (Y) and wait	**Request for write_lock (X) T2 is aborted**

Wound-Wait

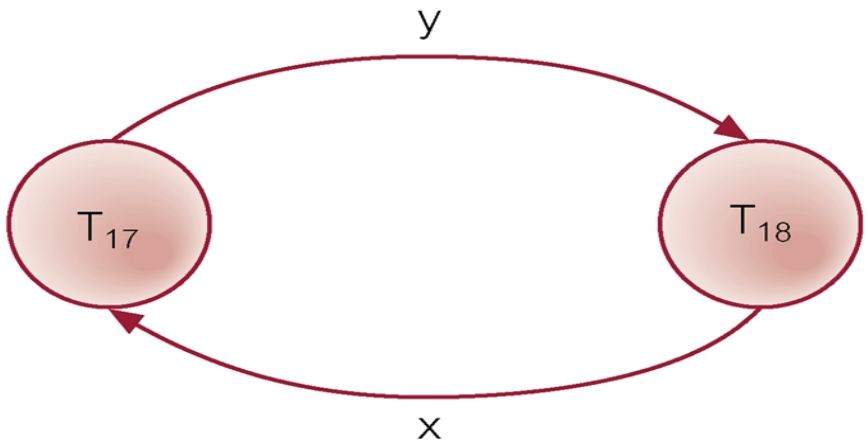
***Condition 1*: If timestamp of T1 is larger than the timestamp of T2, allow T1 to wait for T2 to release lock**

***Condition 2*: If timestamp of T1 is smaller than the timestamp of T2, T2 is aborted (wounded). T2 is rolled-back.**

Transaction T1 TS1 – "2014-03-15 23:50:56" *Older Transaction*	Transaction T2 TS2 – "2014-03-15 23:51:12" *Younger Transaction*
Read (X)	
Read (Y)	
	Read (X)
	Read (Y)
Write (X)	
	Write (Y)
Request Write_lock(Y) and preempts T2. ie T2 is aborted.	**Request Write_lock(X) and wait.**

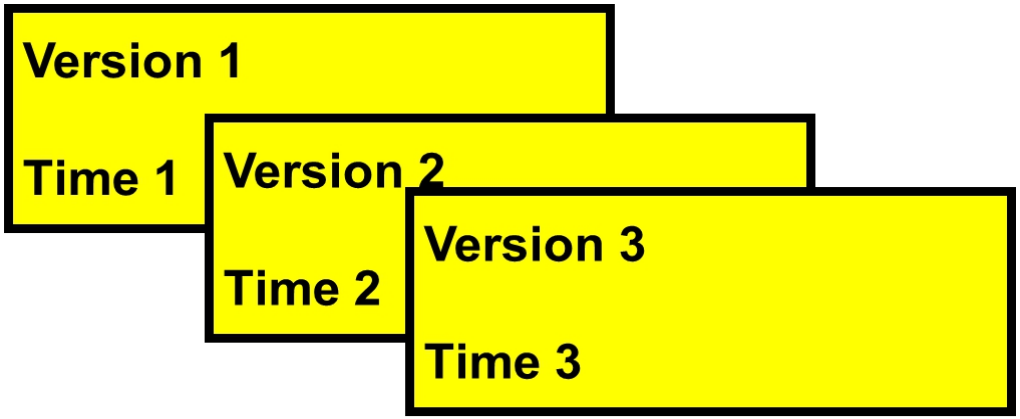
Deadlock Detection and Recovery

- DBMS allows deadlock to occur but recognizes it and breaks it.
- Usually handled by construction of wait-for graph (WFG) showing transaction dependencies:
  - Create a node for each transaction.
  - Create edge T_i -> T_j, if T_i waiting to lock item locked by T_j.
- Deadlock exists if and only if WFG contains cycle.
- WFG is created at regular intervals.

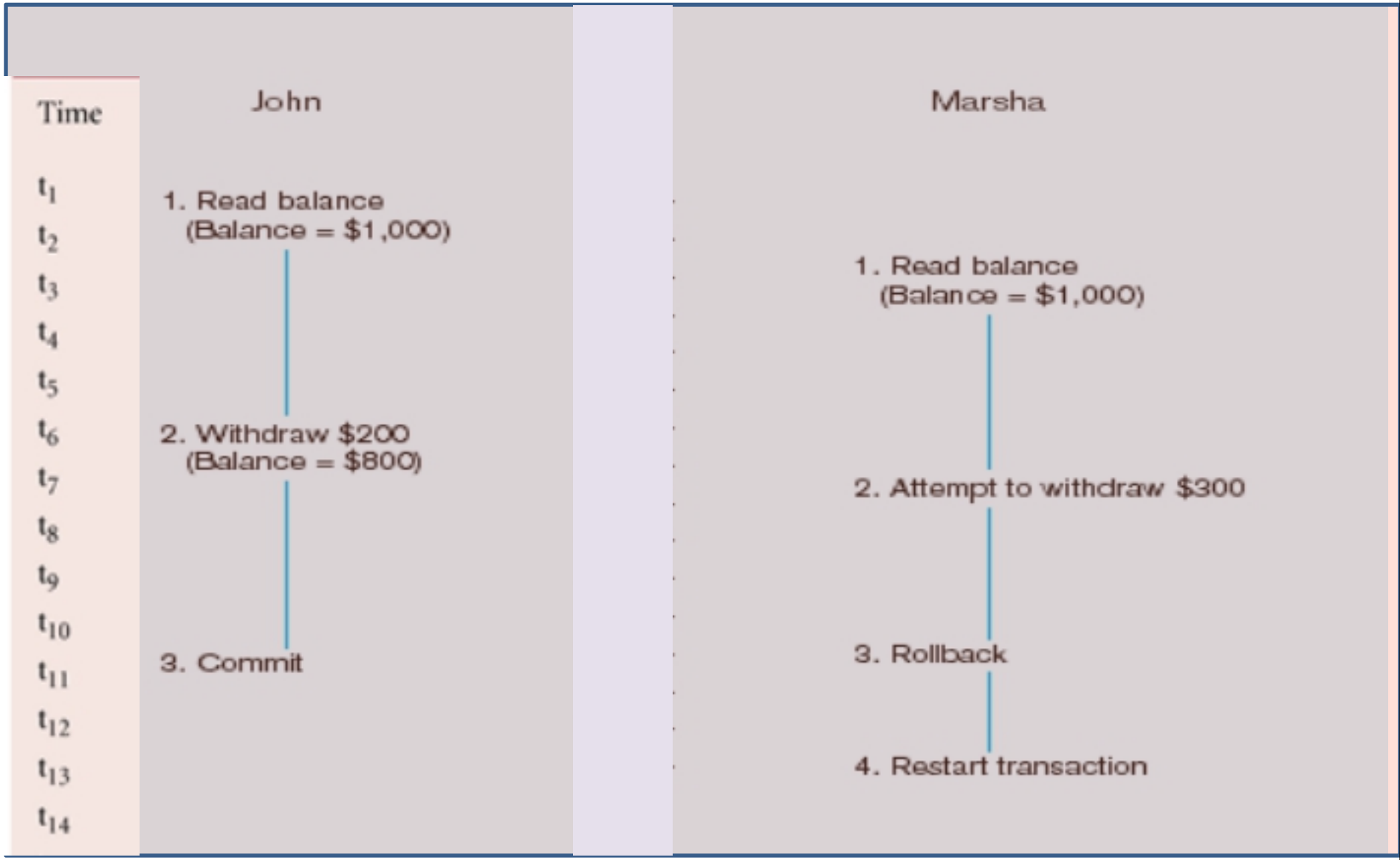


2.3.2 Versioning

- Optimistic approach to concurrency control
- Increase concurrency - multiple transactions can read and write different versions of same data item
- Assumption is that simultaneous updates will be infrequent
- Each transaction can attempt an update as it wishes.
- At commit, check is made to determine whether conflict has occurred. If yes, transaction is rolled back and restarted.



Versioning

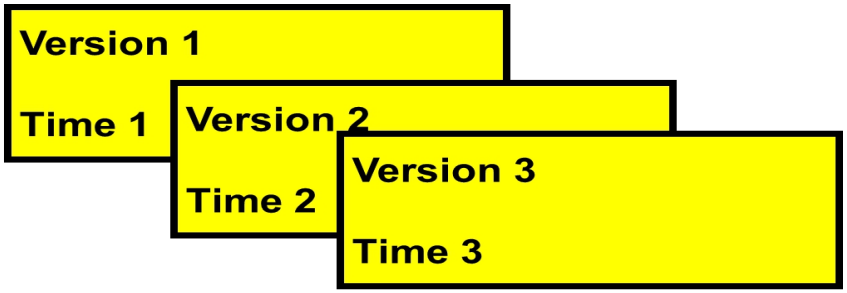

Time
John
Marsha
t1
t2
t3
t4
t5
t6
t7
t8
t9
t10
t11
t12
t13
t14
1. Read balance
(Balance = $1,000)
2. Withdraw $200
(Balance = $800)
3. Commit
1. Read balance
(Balance = $1,000)
2. Attempt to withdraw $300
3. Rollback
4. Restart transaction

Versioning

Three phases:

– Read

- reads values from database and stores them in local variables. Updates are applied to a local copy of the data.



– Validation

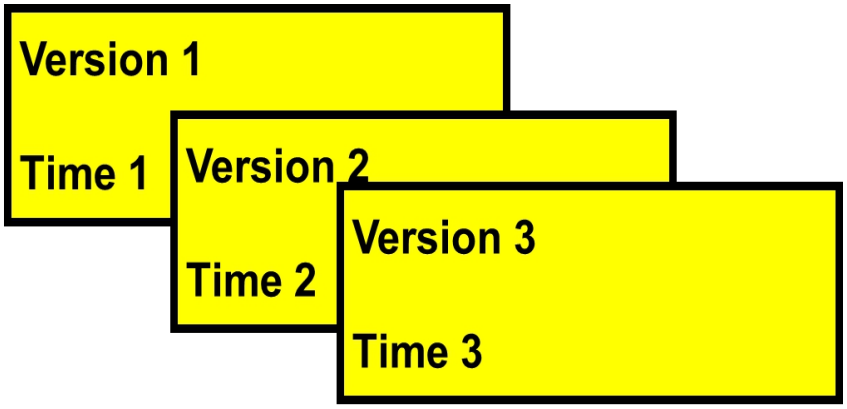
- For read-only transaction, checks that data read are still current values. For update transaction, checks transaction leaves database in a consistent state, with serializability maintained. If not, transaction is aborted and restarted.

– Write

- For successful validation of update transactions, the updates made to the local copy are applied to the database.

Versioning

- **Advantage:**
  - Deadlock free
  - Maximum parallelism
- **Disadvantage:**
  - Rerun transaction if aborts
  - Probability of conflict rises substantially at high loads

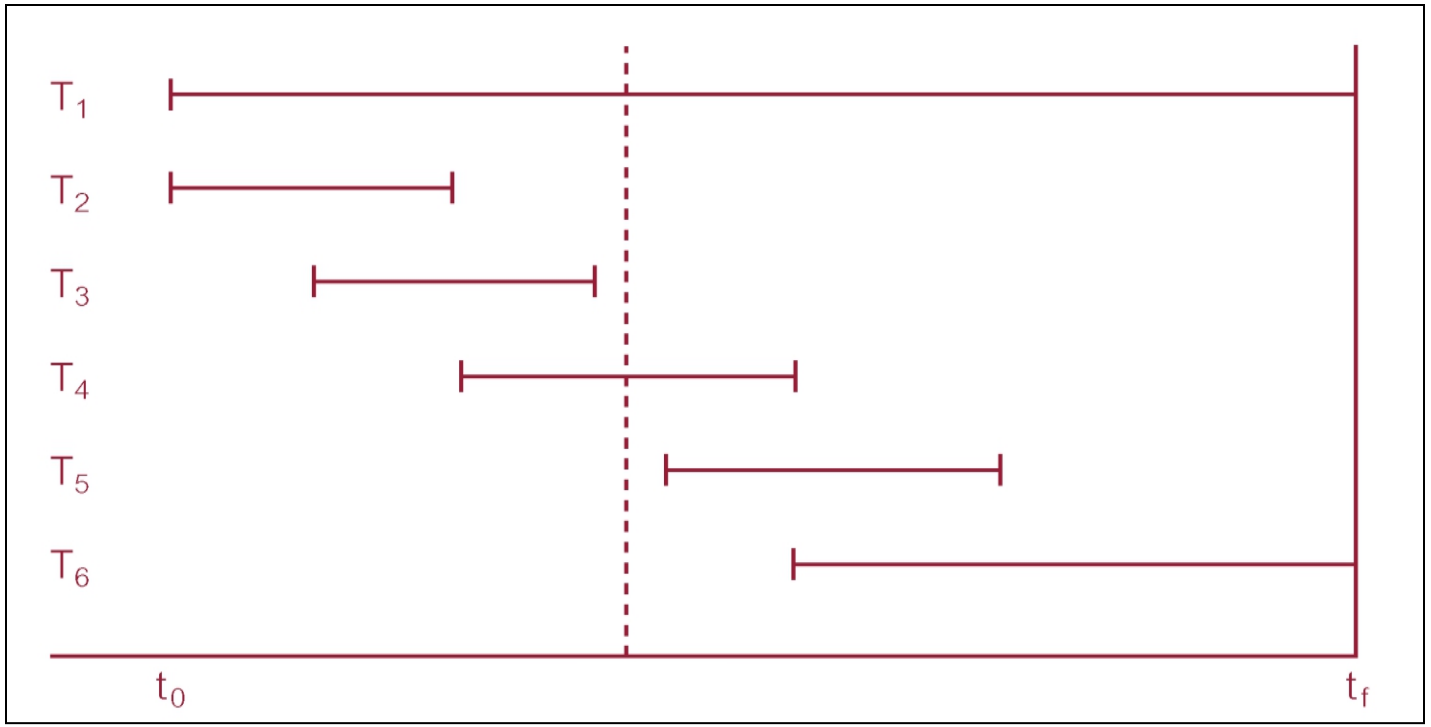


3. Database Recovery

Types of Failures

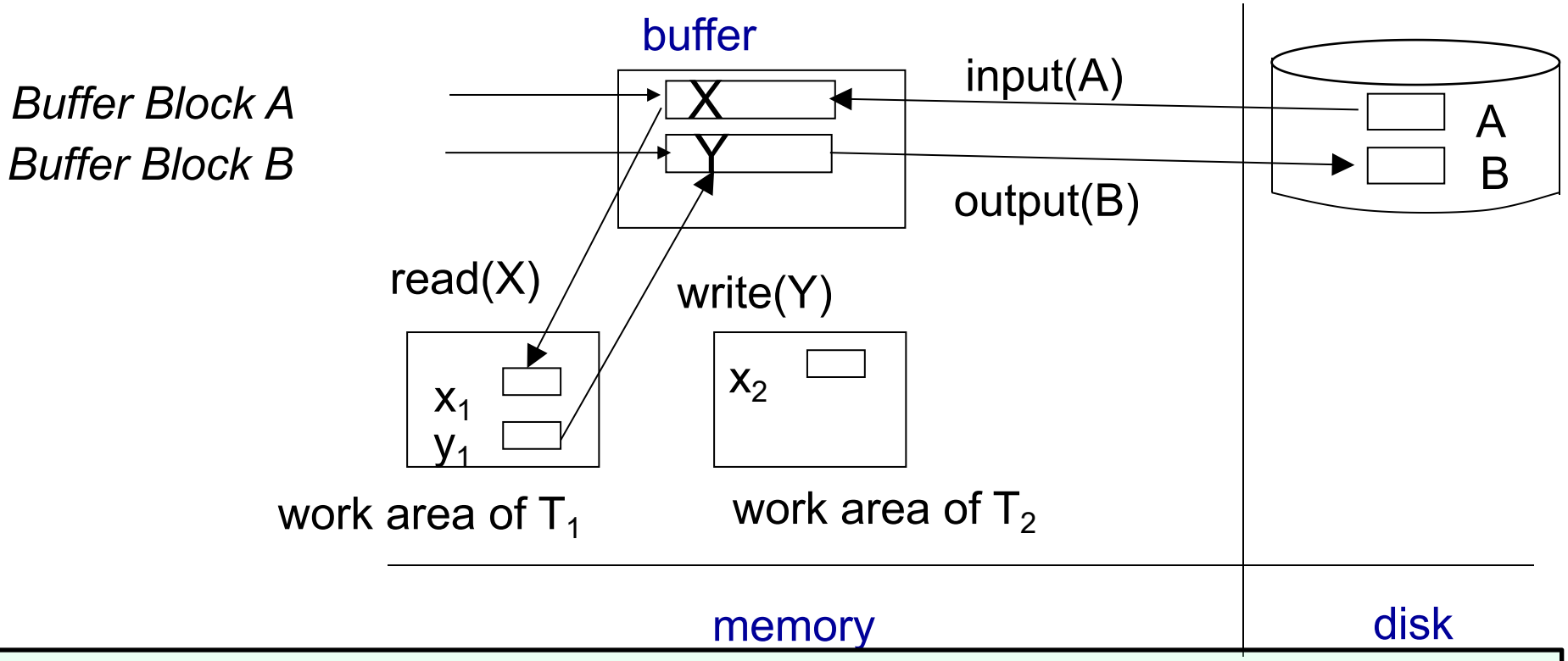
- System crashes, resulting in loss of main memory.
- Media failures, resulting in loss of parts of secondary storage.
- Application software errors.
- Natural physical disasters.
- Carelessness or unintentional destruction of data or facilities.
- Sabotage.

Database Recovery



- DBMS starts at time t_0, but fails at time t_f.
- T_1 and T_6 have to be undone (*rollback) ie to undo* any effects of that transaction for atomicity.
- In absence of any other information, T_2, T_3, T_4, and T_5 have to be redone (rollforward), to endure durability

Example of Data Access with Concurrent transactions



Read operation: e.g. read(X)

1. **find the address of the block or page that contains the record with the data item X.**
2. **transfer the block into a buffer in main memory.**
3. **copy the value of X from the buffer into the program variable X.**

Write operation: e.g. write(Y)

1. **and 2. same as above**
3. **copy the value from the program variable Y into the buffer.**
4. **write the buffer back to the block on disk.**

Recovery Facilities

- DBMS should provide the following facilities to assist with recovery:
  - **Backup mechanism**, which makes periodic backup copies of database.
  - **Logging facilities**, which keep track of current state of transactions and database changes.
  - **Checkpoint facility**, which enables updates to database in progress to be made permanent.
  - **Recovery manager**, which allows DBMS to restore database to consistent state following a failure.

3.1 Log File

- Contains information about all updates to database:
  - Transaction identifier.
  - Type of log record (transaction start, insert, update, delete, abort, commit).
  - Identifier of data item affected by database action (insert, delete, and update operations).
  - Before-image of data item.
  - After-image of data item.

Tid	Time	Operation	Object	Before image	After image	pPtr	nPtr
T1	10:12	START				0	2
T1	10:13	UPDATE	STAFF SL21	(old value)	(new value)	1	8
T2	10:14	START				0	4
T2	10:16	INSERT	STAFF SG37		(new value)	3	5
T2	10:17	DELETE	STAFF SA9	(old value)		4	6
T2	10:17	UPDATE	PROPERTY PG16	(old value)	(new value)	5	9
T3	10:18	START				0	11
T1	10:18	COMMIT				2	0
	10:19	CHECKPOINT	T2, T3				
T2	10:19	COMMIT				6	0
T3	10:20	INSERT	PROPERTY PG4		(new value)	7	12
T3	10:21	COMMIT				11	0

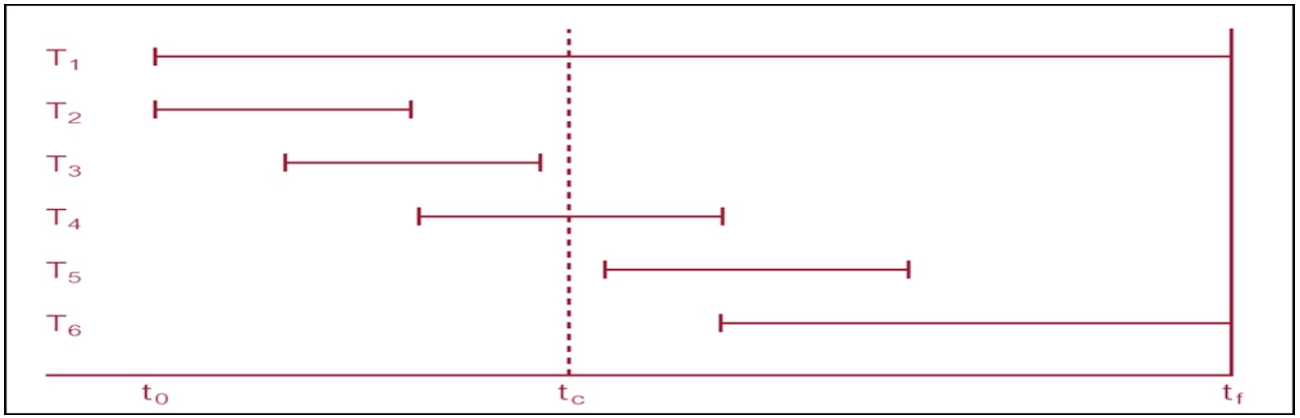
3.2 Checkpointing

- **Checkpoint**

  Point of synchronization between database and log file. All buffers are force-written to secondary storage.

- **Steps:**
  1. Suspend execution of transactions temporarily.
  2. Force write modified buffer data to disk.
  3. Write a [checkpoint] record to the log, save the log to disk.
  4. Resume normal transaction execution.

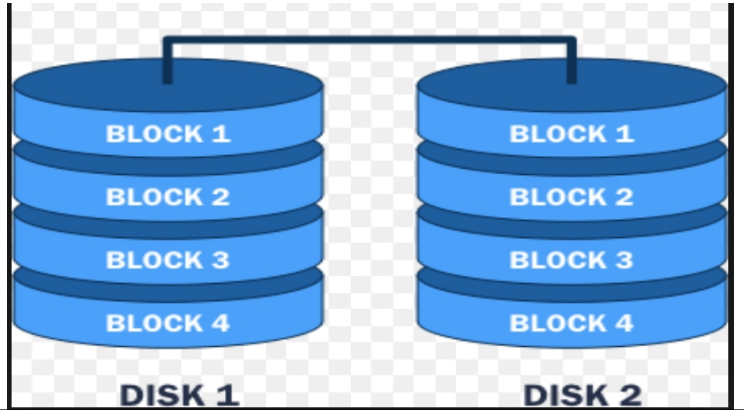
- **Checkpoint record** is created containing identifiers of all active transactions.



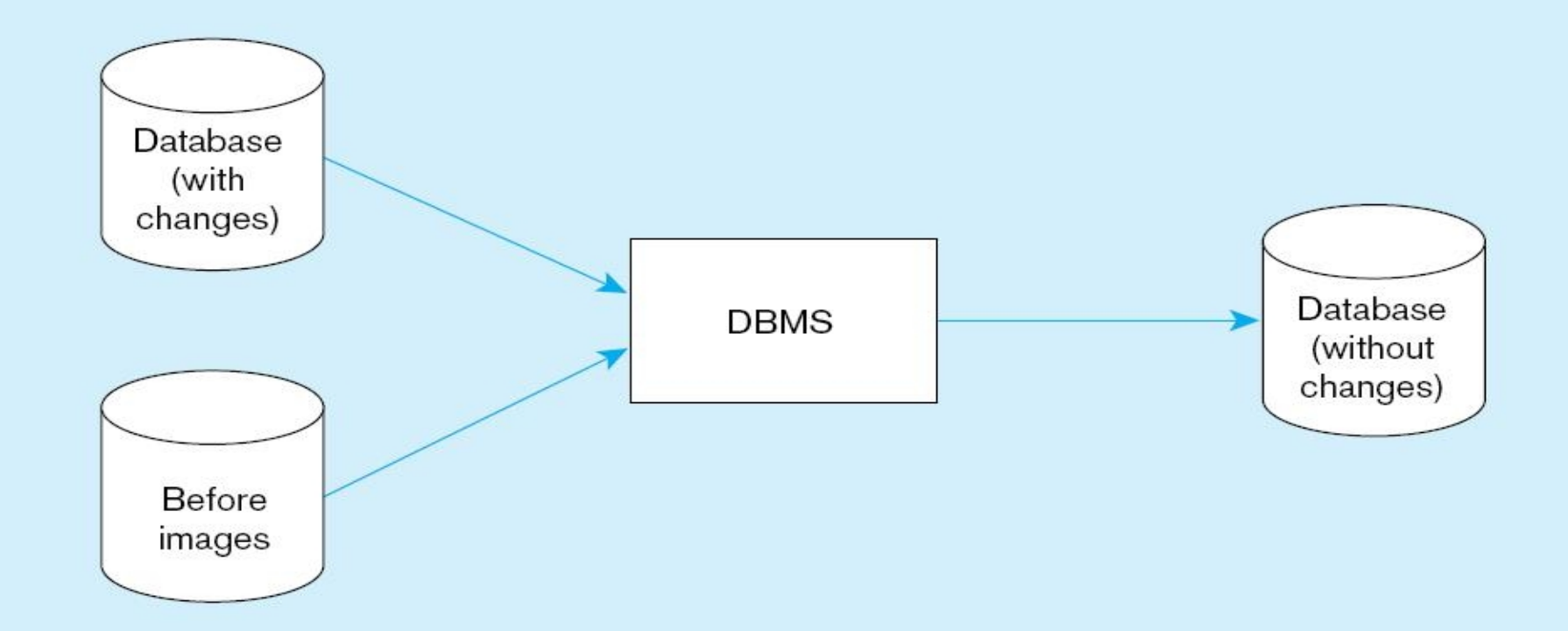
4. Database Recovery

- Recovery Manager – DBMS module that restores the database to a correct condition when a failure occurs and then resumes processing user requests
- Recovery and Restart Procedures
  - Disk Mirroring – switch between identical copies of databases
  - Restore/Rerun–reprocess transactions against the backup (only done as a last resort)
  - Backward Recovery (Rollback)–apply before images
  - Forward Recovery (RollForward)–apply after images (preferable to restore/rerun)

Disk Mirroring

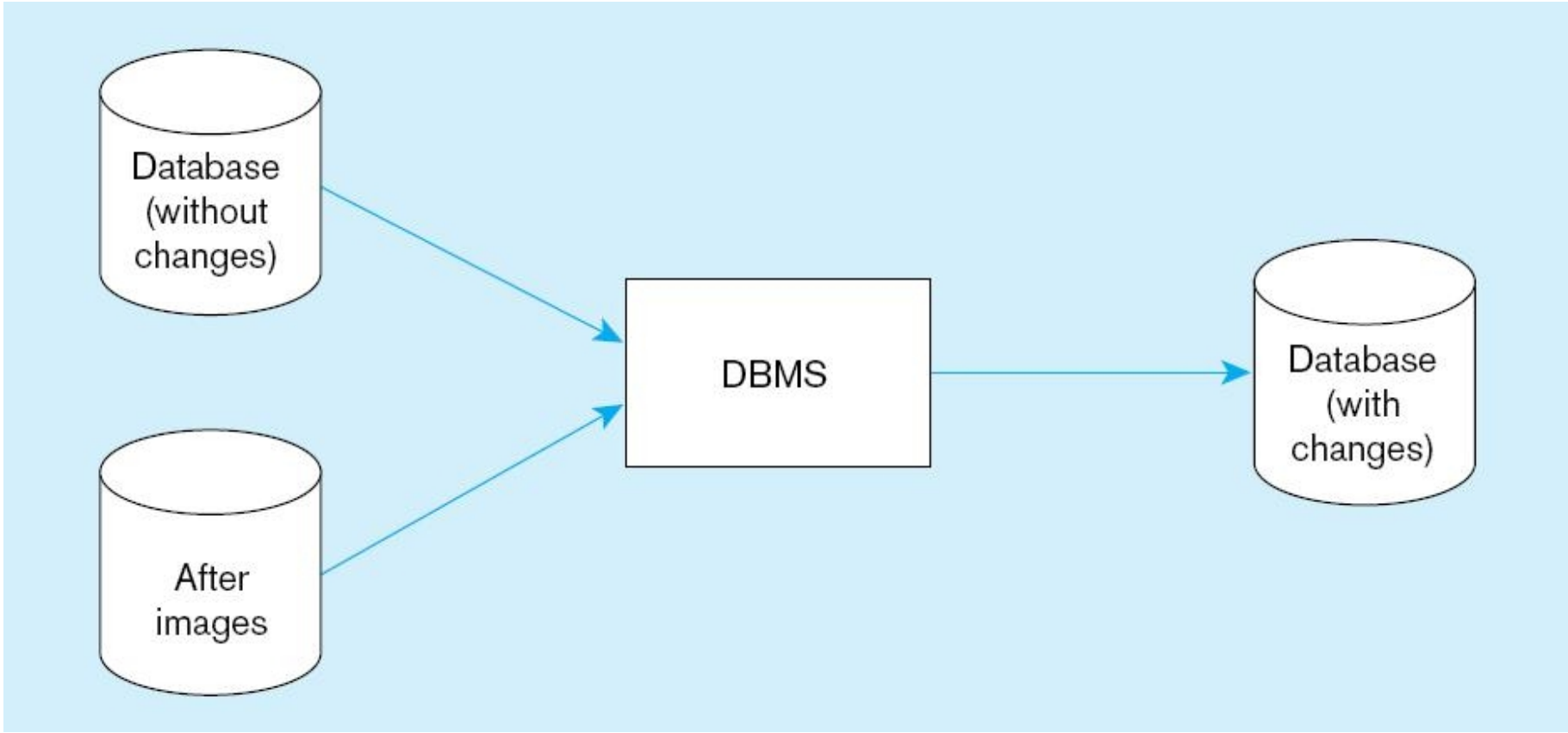


Rollback/Backward Recovery/Undo



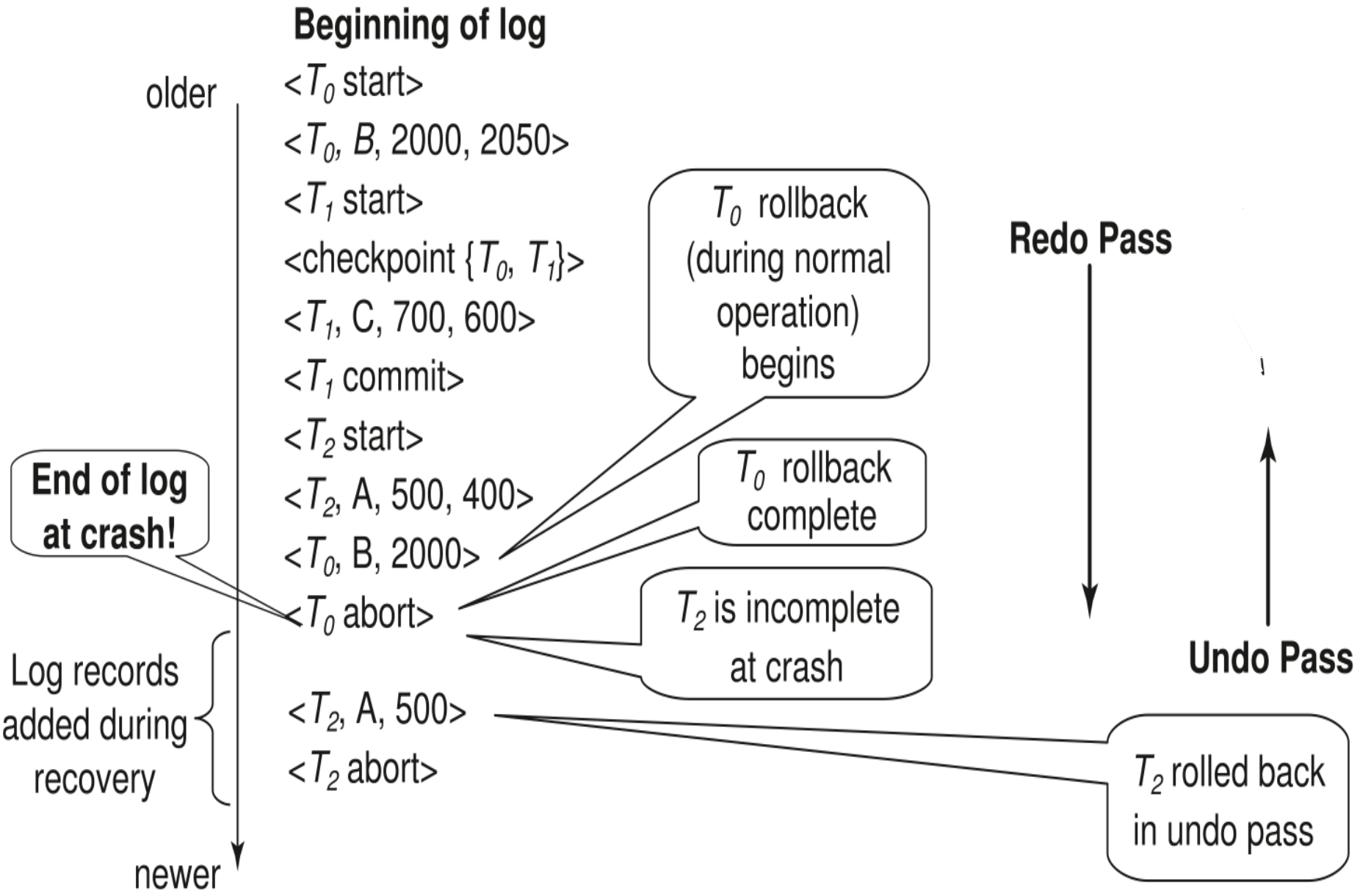
- **Undo of unwanted changes to a database. Before images of the records that have been changed are applied to the database, and the database is returned to an earlier state ie Reverse the changes made by transactions that have been aborted, or terminated abnormally.**
- **The undo operations are performed *in the reverse order to which they were written to the log***

Rollforward /Forward Recovery/Redo



A technique that starts with an earlier copy of a database. After images (the results of good transactions) are applied to the database, and the database is quickly moved forward to a later state.

Example of Recovery

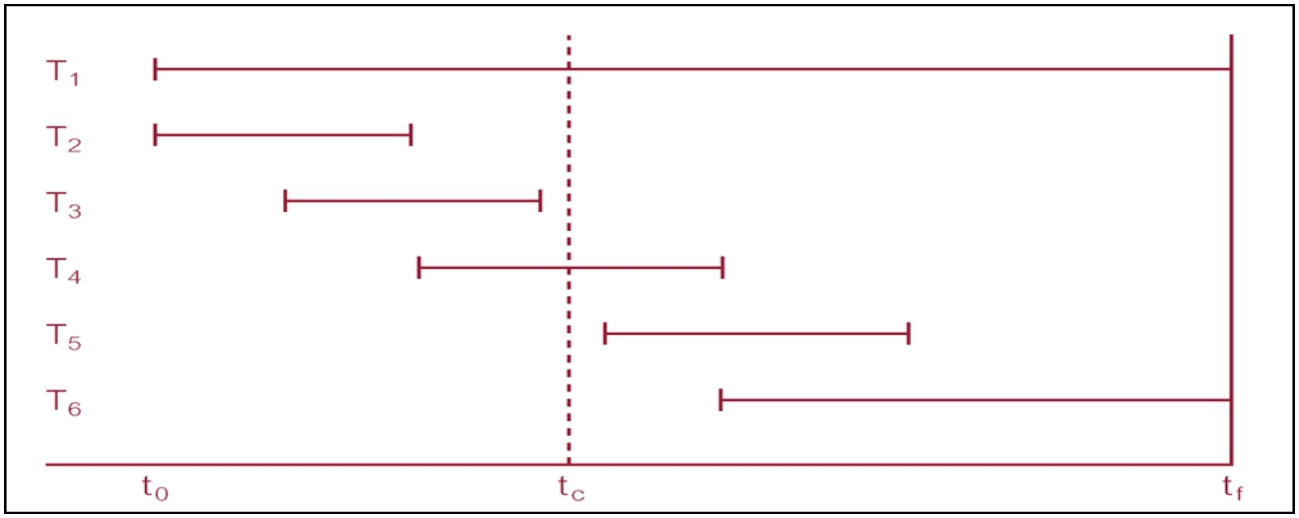


Recovery

- **Redo/rollforward phase:**
  1. Find last <checkpoint *L*> record
  2. Scan forward from above <checkpoint *L*> record
  3. Whenever a record $<T_i, X_j, V_1, V_2>$ is found, redo it by writing V_2 to X_j

- **Undo/rollback phase:**
  1. Scan log backwards from end
  2. Whenever a log record $<T_i, X_j, V_1, V_2>$ is found, perform
     a. undo by writing V_1 to X_j.
     b. write a log record $<T_i, X_j, V_1>$

Undo/Redo Recovery



- When failure occurs,
  - redo all transactions that committed since the checkpoint
  - undo all transactions active at time of crash.
- With checkpoint at time t_c, changes made by T_2 and T_3 have been written to secondary storage.
- Thus:
  - only redo T_4 and T_5,
  - undo transactions T_1 and T_6.

Database Failure Responses

- *Aborted transactions*
  - Causes - human error, input of invalid data, hardware failure, and deadlock
  - Preferred recovery: rollback
  - Alternative: Rollforward to state just prior to abort
- *Incorrect (but valid) data*
  - Causes – carelessness or unintentional human mistake etc
  - Preferred recovery: rollback if the error is discovered soon enough
  - Alternative 1: compensating transactions through human intervention to correct the errors if only a few errors have occurred
  - Alternative 2: reprocess transactions from the most recent checkpoint before the error occurred.

Database Failure Responses

- *System failure (database intact)*
  - ➢ *Causes :* power loss, operator error, loss of communications transmission, and system software failure
  - – Preferred recovery: switch to duplicate database
  - – Alternative 1: rollback
  - – Alternative 2: restart from checkpoint
- *Database destruction*
  - ➢ Causes : disk drive failure (or head crash).
  - – Preferred recovery: switch to duplicate database
  - – Alternative 1: rollforward
  - – Alternative 2: reprocess transactions

References

- ***Database Systems: A Practical Approach to Design, Implementation and Management*.** **6^{th}edn. Connolly, T. M. and Begg, C. E. Pearson.**
- ***Modern Database Management.*** **Hoffer, J.A., Prescott, M., and McFadden,F. *12^{th}edn*.**

  Prentice Hall
- ***Fundamentals of Database Systems*.** **Elmasri, R. and Navathe, S.B.. Addison-Wesley.**