TRANSACTION MANAGEMENT

EGCO321 DATABASE SYSTEMS

KANAT POOLSAWASD
DEPARTMENT OF COMPUTER ENGINEERING
MAHIDOL UNIVERSITY

TRANSACTION

- Transaction is a unit of work that should be processed reliably. DBMS provide recovery and concurrency control services to process transactions efficiently and reliably.

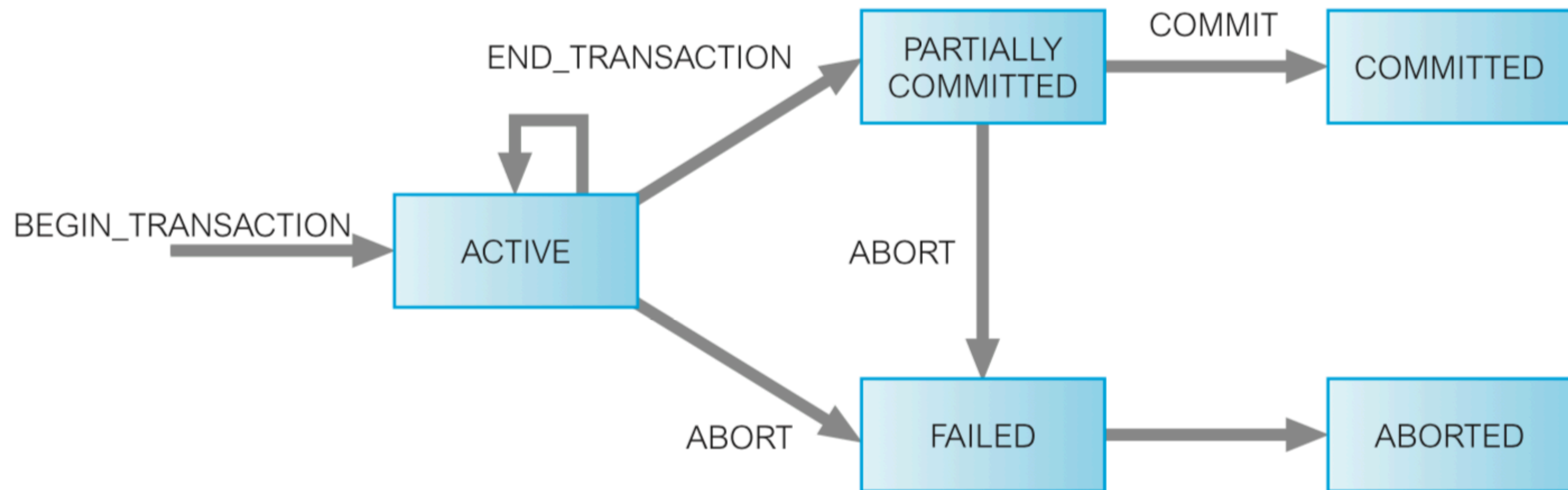
TRANSACTION EXAMPLES

Transaction	Description
Add order	Customer places a new order
Update order	Customer changes details of an existing order
Check status	Customer checks the status of an order
Payment	Payment received from a customer
Shipment	Goods sent to a customer

PROCEDURE FOR AN ATM TRANSACTION

```
START 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
        UPDATE account by posting credit
        INSERT history record
        Display final message and issue cash
    Else
        Write error message
    End If
    On Error: ROLLBACK
COMMIT
```

STATE TRANSITION DIAGRAM



TRANSACTION: PROGRAMMER'S ROLE

TRANSACTION: SYSTEM'S ROLE (1)

- **Atomicity** means that a transaction cannot be subdivided.
- **Consistency** means that if applicable constraints are true before the transaction starts, the constraints will be true after the transaction terminates.
- **Isolation** means that transactions do not interfere with each other except in allowable ways.
- **Durability** means that any changes resulting from a transaction are permanent. No failure will erase any changes after a transaction terminates.

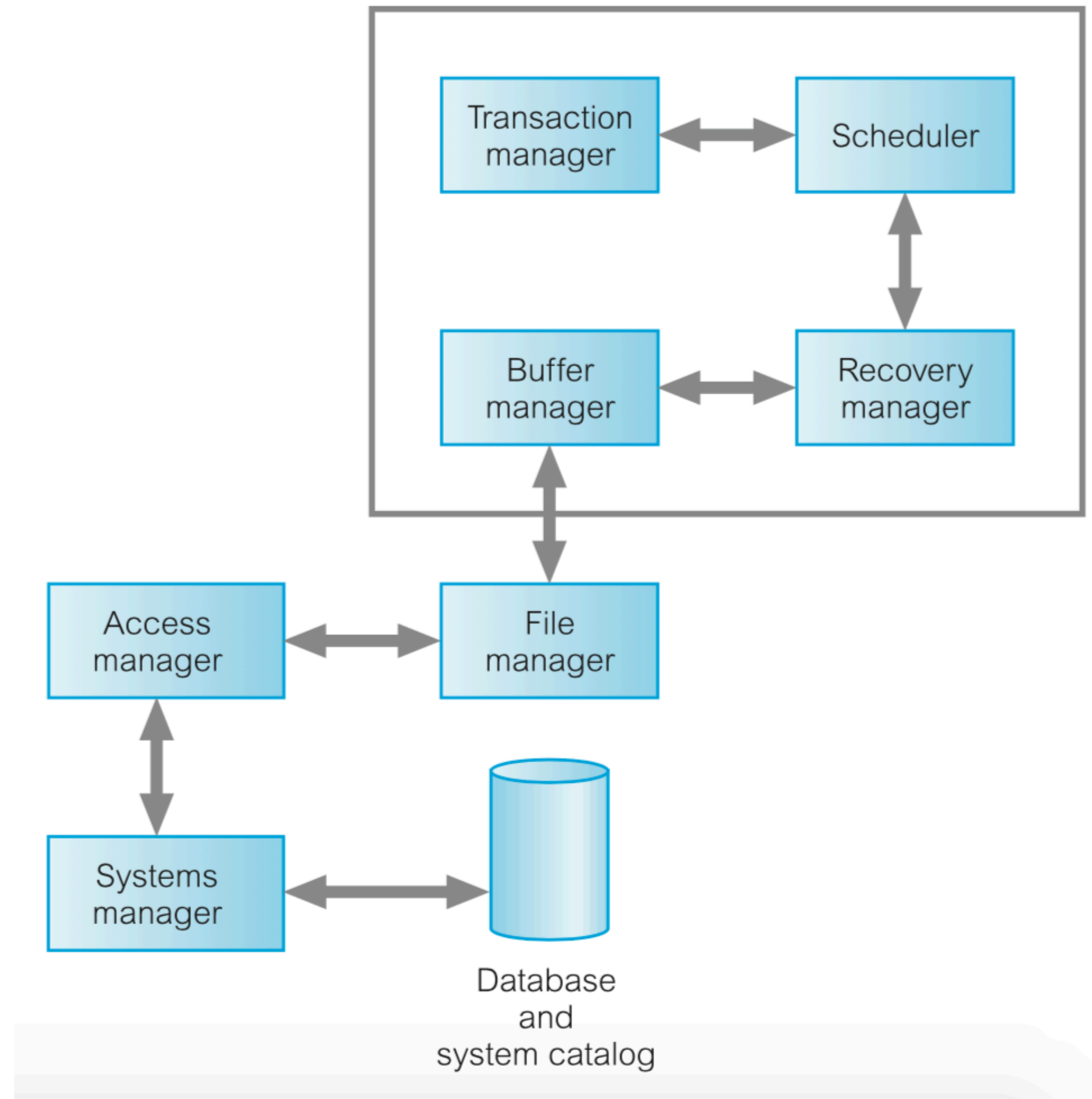
TRANSACTION: SYSTEM'S ROLE (2)

- DBMSs provide two services, **recovery transparency** and **concurrency transparency**
- **Recovery Transparency** means that the DBMS automatically restores a database to a consistent state after a failure.
- **Concurrency Transparency** means that users perceive the database as a single-user system even though there may be many simultaneous users.

OBJECTIVE OF CONCURRENCY CONTROL

- **Transaction Throughput** is the number of transactions processed per time interval. It is an important measure of transaction processing performance.
- **Hot Spot** is common data that multiple users try to change. Without adequate concurrency control, users may interfere with each other on hot spots.

DBMS TRANSACTION SUBSYSTEM



INTERFERENCE PROBLEMS (1)

- Lost update is a concurrency control problem in which one user's update overwrites another user's update.

Time	T_1	T_2	$\mathbf{bal_x}$
t_1		begin_transaction	100
t_2	begin_transaction	read($\mathbf{bal_x}$)	100
t_3	read($\mathbf{bal_x}$)	$\mathbf{bal_x} = \mathbf{bal_x} + 100$	100
t_4	$\mathbf{bal_x} = \mathbf{bal_x} - 10$	write($\mathbf{bal_x}$)	200
t_5	write($\mathbf{bal_x}$)	commit	90
t_6	commit		90

INTERFERENCE PROBLEMS (2)

- Uncommitted dependency is a concurrency control problem in which one transactions read data written by another transaction before the other transaction commits. If the second transaction aborts, the first transaction may rely on data that will no longer exist.

Time	T_3	T_4	$\mathbf{bal_x}$
t_1		begin_transaction	100
t_2		read($\mathbf{bal_x}$)	100
t_3		$\mathbf{bal_x} = \mathbf{bal_x} + 100$	100
t_4	begin_transaction	write($\mathbf{bal_x}$)	200
t_5	read($\mathbf{bal_x}$)	⋮	200
t_6	$\mathbf{bal_x} = \mathbf{bal_x} - 10$	**rollback**	100
t_7	write($\mathbf{bal_x}$)		190
t_8	commit		190

INTERFERENCE PROBLEMS (3)

- Incorrect summary is a concurrency control problem in which a transaction reads several values, but another transaction updates some of the values while the first transaction is still executing.

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

SCHEDULE

- A schedule is a **sequence of the operations** by a set of concurrent transactions that preserves the order of operations in each of the individual transactions
- A serial schedule is a schedule where operations of each transaction are executed consecutively without any interleaved operations from other transactions (each transaction commits before the next one is allowed to begin)

SERIAL SCHEDULES

- Serial schedules are guaranteed to avoid interference and keep the database consistent
- However databases need concurrent access which means interleaving operations from different transactions

SERIAL AND SERIALIZABLE

Interleaved Schedule	Serial Schedule
T1 Read(X)	T2 Read(X)
T2 Read(X)	T2 Read(Y)
T2 Read(Y)	T2 Read(Z)
T1 Read(Z)	
T1 Read(Y)	T1 Read(X)
T2 Read(Z)	T1 Read(Z)
	T1 Read(Y)

This schedule is serialisable:

SERIALIZABILITY (1)

- **Schedule** is a sequence of the operations by a set of concurrent transactions that preserves the order of the operations in each of the individual transactions.
  - **Serial schedule** is a schedule where the operations of each transaction are executed consecutively without any interleaved operations from other transactions.
  - **Non-serial schedule** is a schedule where the operations from a set of concurrent transactions are interleaved.

SERIALIZABILITY (2)

- In serializability, the ordering of read and write operations is important:
  - If two transactions only read a data item, they do not conflict and order is not important.
  - If two transactions either read or write completely separate data items, they do not conflict and order is not important.
  - If one transaction writes a data item and another either reads or writes the same data item, the order of execution is important.

SERIALIZABILITY (3)

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)

CONFLICT SERIALIZABILITY (1)

- Two transactions have a conflict:
  - NO If they refer to different resources
  - NO If they are reads
  - YES If at least one is a write and they use the same resource
- A schedule is conflict serializable if transactions in the schedule have a conflict but the schedule is still serializable

CONFLICT SERIALIZABILITY (2)

- Conflict serializable schedules are the main focus of concurrency control
- They allow for interleaving and at the same time they are guaranteed to behave as a serial schedule
- Important questions: how to determine whether a schedule is conflict serializable
- How to construct conflict serializable schedules

TESTING FOR CONFLICT SERIALIZABILITY

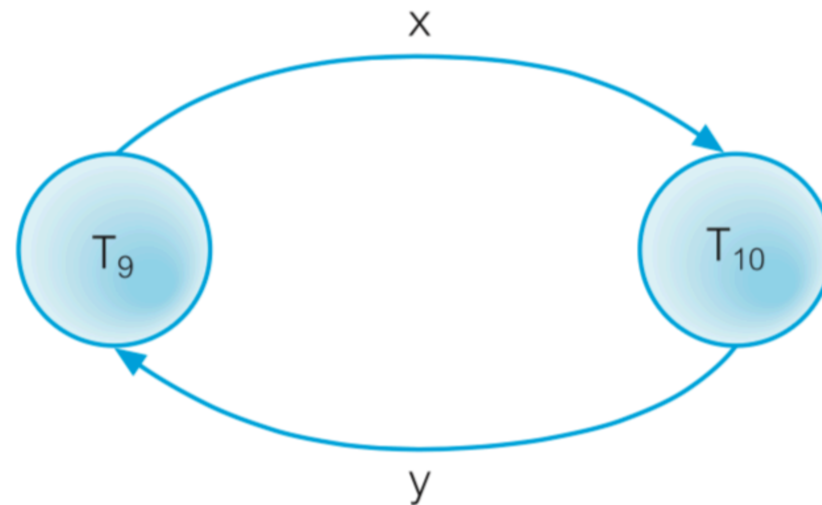
- Create a node for each transaction.
- Create a directed edge $T_i \rightarrow T_j$, if T_j reads the value of an item written by T_i.
- Create a directed edge $T_i \rightarrow T_j$, if T_j writes a value into an item after it has been read by T_i.
- Create a directed edge $T_i \rightarrow T_j$, if T_j writes a value into an item after it has been written by T_i.

NON-CONFLICT SERIALIZABLE SCHEDULE (1)

Time	T_9	T_{10}
t_1	begin_transaction	
t_2	read($\mathbf{bal_x}$)	
t_3	$\mathbf{bal_x} = \mathbf{bal_x} + 100$	
t_4	write($\mathbf{bal_x}$)	begin_transaction
t_5		read($\mathbf{bal_x}$)
t_6		$\mathbf{bal_x} = \mathbf{bal_x} * 1.1$
t_7		write($\mathbf{bal_x}$)
t_8		read($\mathbf{bal_y}$)
t_9		$\mathbf{bal_y} = \mathbf{bal_y} * 1.1$
t_{10}		write($\mathbf{bal_y}$)
t_{11}	read($\mathbf{bal_y}$)	commit
t_{12}	$\mathbf{bal_y} = \mathbf{bal_y} - 100$	
t_{13}	write($\mathbf{bal_y}$)	
t_{14}	commit	

NON-CONFLICT SERIALIZABLE SCHEDULE (2)

- Precedence graph for previous slide showing a cycle, so schedule is not conflict serializable.

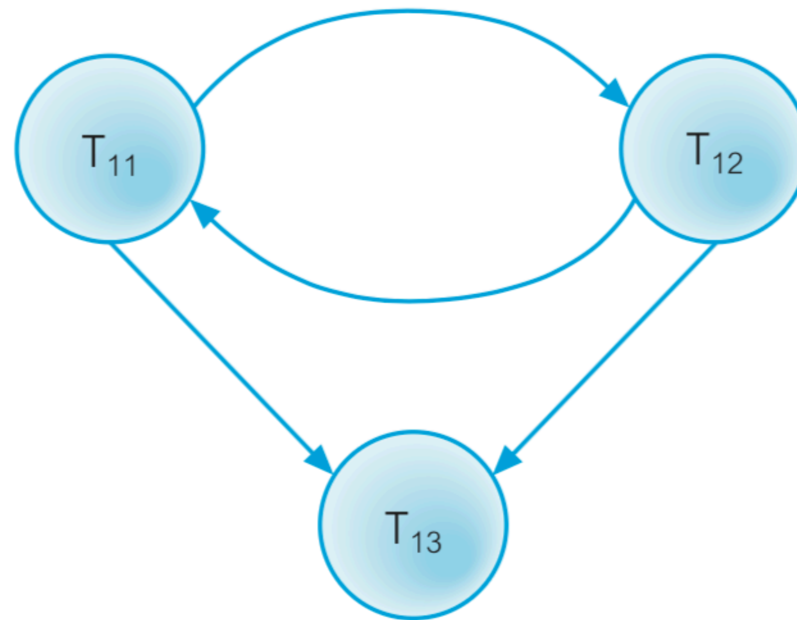


EXAMPLE (1)

Time	T_{11}	T_{12}	T_{13}
t_1	begin_transaction		
t_2	read(**bal_x**)		
t_3		begin_transaction	
t_4		write(**bal_x**)	
t_5		commit	
t_6	write(**bal_x**)		
t_7	commit		
t_8			begin_transaction
t_9			write(**bal_x**)
t_{10}			commit

EXAMPLE (2)

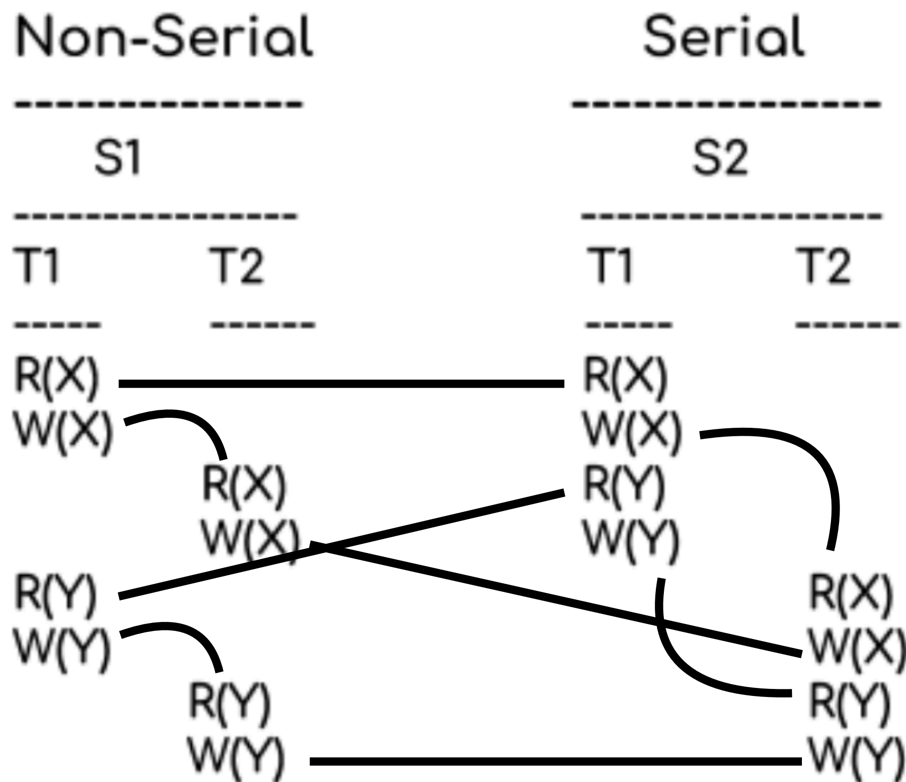
- Precedence graph for previous slide showing a cycle, so schedule is not conflict serializable.



VIEW SERIALIZABILITY (1)

- Two schedules T1 and T2 are said to be view equivalent, if they satisfy all the following conditions:
  - **Initial Read**: Initial read of each data item in transactions must match in both schedules.
  - **Final Write**: Final write operations on each data item must match in both the schedules.
  - **Update Read:** If in schedule S1, the transaction T1 is reading a data item updated by T2 then in schedule S2, T1 should read the value after the write operation of T2 on same data item.

VIEW SERIALIZABILITY (2)



S2 is the serial schedule of S1. If we can prove that they are view equivalent then we we can say that given schedule S1 is view Serializable

CONFLICT VS. VIEW SERIALIZABILITY

Conflict Serializability	View Serializability
Two schedules are said to be conflict equivalent if all the conflicting operations in both the schedule get executed in the same order. If a schedule is a conflict equivalent to its serial schedule then it is called Conflict Serializable Schedule.	Two schedules are said to be view equivalent if the order of initial read, final write and update operations is the same in both the schedules. If a schedule is view equivalent to its serial schedule then it is called View Serializable Schedule.
If a schedule is view serializable then it may or may not be conflict serializable.	If a schedule is conflict serializable then it is also view serializable schedule.
Conflict equivalence can be easily achieved by reordering the operations of two transactions therefore, Conflict Serializability is easy to achieve.	View equivalence is rather difficult to achieve as both transactions should perform similar actions in a similar manner. Thus, View Serializability is difficult to achieve.

CONCURRENCY CONTROL

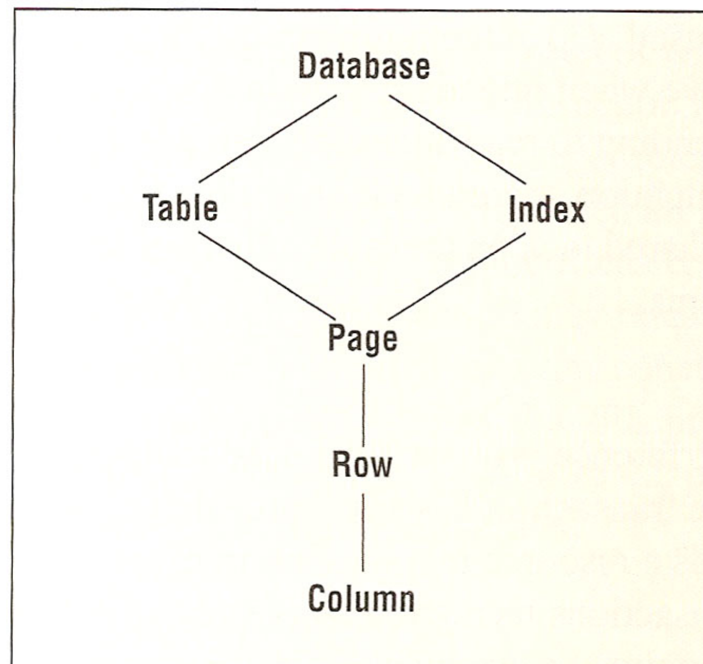
- This section describes two tools, **locks and two-phase locking protocol (2PL)**, used by most DBMSs to prevent the three interference problems discussed in previous section.
- In addition to the two tools, the deadlock problem is presented because it can result through the use of locks.

LOCKING METHODS (1)

- Locking is a procedure used to control concurrent access to data. When one transaction is accessing the database, a lock may deny access to other transactions to prevent incorrect results.
  - **Shared (S) Lock**: If a transaction has a shared lock on a data item, it can read the item but not update it.
  - **Exclusive (X) Lock**: If a transaction has an exclusive lock on a data item, it can both read and update the item.

LOCKING METHODS (2)

- Locking granularity is the size of the database item locked. Locking granularity is a trade-off between waiting time (amount of concurrency permitted) and overhead (number of locks held)



EXAMPLE 1 - LOST UPDATE

Time	T_1	T_2	bal_x
t_1		begin_transaction	100
t_2	begin_transaction	write_lock(bal_x)	100
t_3	write_lock(bal_x)	read(bal_x)	100
t_4	WAIT	$bal_x = bal_x + 100$	100
t_5	WAIT	write(bal_x)	200
t_6	WAIT	commit/unlock(bal_x)	200
t_7	read(bal_x)		200
t_8	$bal_x = bal_x - 10$		200
t_9	write(bal_x)		190
t_{10}	commit/unlock(bal_x)		190

EXAMPLE 2 - UNCOMMITTED DEPENDENCY

Time	T_3	T_4	bal_x
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

EXAMPLE 3 - INTERFERENCE

Time	T_5	T_6	bal_x	bal_y	bal_z	sum
t_1		begin_transaction	100	50	25	
t_2	begin_transaction	sum = 0	100	50	25	0
t_3	write_lock(bal_x)		100	50	25	0
t_4	read(bal_x)	read_lock(bal_x)	100	50	25	0
t_5	$bal_x = bal_x - 10$	WAIT	100	50	25	0
t_6	write(bal_x)	WAIT	90	50	25	0
t_7	write_lock(bal_z)	WAIT	90	50	25	0
t_8	read(bal_z)	WAIT	90	50	25	0
t_9	$bal_z = bal_z + 10$	WAIT	90	50	25	0
t_{10}	write(bal_z)	WAIT	90	50	35	0
t_{11}	commit/unlock(bal_x, bal_z)	WAIT	90	50	35	0
t_{12}		read(bal_x)	90	50	35	0
t_{13}		sum = sum + bal_x	90	50	35	90
t_{14}		read_lock(bal_y)	90	50	35	90
t_{15}		read(bal_y)	90	50	35	90
t_{16}		sum = sum + bal_y	90	50	35	140
t_{17}		read_lock(bal_z)	90	50	35	140
t_{18}		read(bal_z)	90	50	35	140
t_{19}		sum = sum + bal_z	90	50	35	175
t_{20}		commit/unlock(bal_x, bal_y, bal_z)	90	50	35	175

TWO-PHASE LOCKING (2PL)

- Two-phase locking (2PL) is a concurrency control method used in database systems to ensure that transactions are executed in a way that maintains database consistency, even when multiple transactions are occurring simultaneously.
- It consists of two main phases: the growing phase and the shrinking phase.

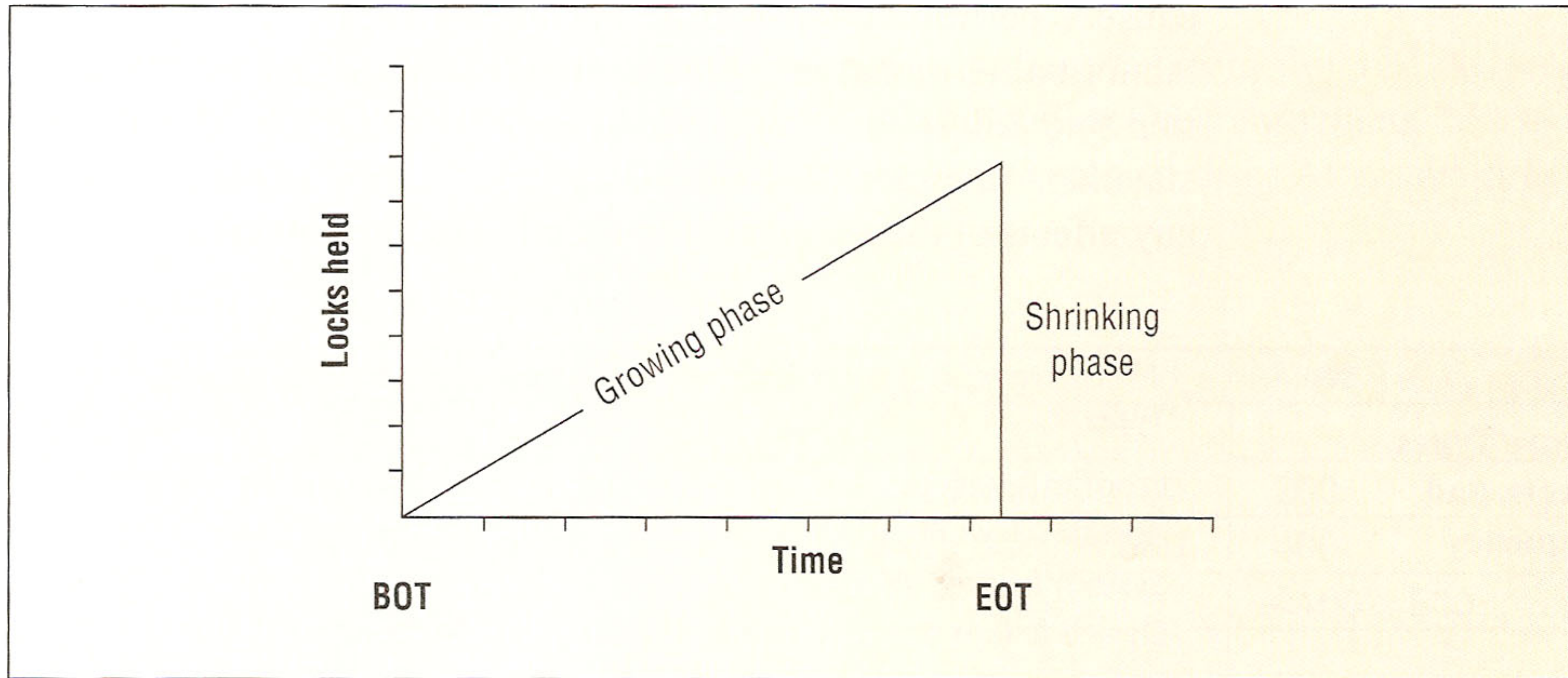
TWO-PHASE LOCKING (2PL)

- **Growing Phase**
  - During this phase, a transaction can acquire locks on data items but cannot release any locks.
  - The transaction continues to acquire locks as needed to ensure it can read or modify the required data.
- **Shrinking Phase**
  - Once a transaction releases its first lock, it enters the shrinking phase.
  - In this phase, the transaction can release locks but cannot acquire any new locks.

TWO-PHASE LOCKING (2PL)

```
Locking Activity
  Acquire Lock A
  Acquire Lock B
  Read/Write Data
  ——————————————————————> (End of Growing Phase)
  Release Lock A
  Release Lock B
  (Shrinking Phase)
```

GROWING AND SHRINKING PHASES OF 2PL



BOT = Begin of Transaction

EOT = End of Transaction

KEY CHARACTERISTICS

- Deadlock Avoidance: While 2PL helps in maintaining consistency, it doesn't inherently prevent deadlocks. Additional mechanisms (like timeout or wait-die schemes) are often needed.
- Serializability: Transactions executed under strict two-phase locking are guaranteed to produce a serializable schedule, meaning the end result is the same as if the transactions were executed one after the other, rather than concurrently.
- Locking: 2PL can be implemented using various locking mechanisms (e.g., exclusive locks for writes, shared locks for reads).

CASCADING ROLLBACK

Time	T_{14}	T_{15}	T_{16}
t_1	begin_transaction		
t_2	write_lock($\mathbf{bal_x}$)		
t_3	read($\mathbf{bal_x}$)		
t_4	read_lock($\mathbf{bal_y}$)		
t_5	read($\mathbf{bal_y}$)		
t_6	$\mathbf{bal_x} = \mathbf{bal_y} + \mathbf{bal_x}$		
t_7	write($\mathbf{bal_x}$)		
t_8	unlock($\mathbf{bal_x}$)	begin_transaction	
t_9	⋮	write_lock($\mathbf{bal_x}$)	
t_{10}	⋮	read($\mathbf{bal_x}$)	
t_{11}	⋮	$\mathbf{bal_x} = \mathbf{bal_x} + 100$	
t_{12}	⋮	write($\mathbf{bal_x}$)	
t_{13}	⋮	unlock($\mathbf{bal_x}$)	
t_{14}	⋮	⋮	
t_{15}	**rollback**	⋮	
t_{16}		⋮	begin_transaction
t_{17}		⋮	read_lock($\mathbf{bal_x}$)
t_{18}		**rollback**	⋮
t_{19}			**rollback**

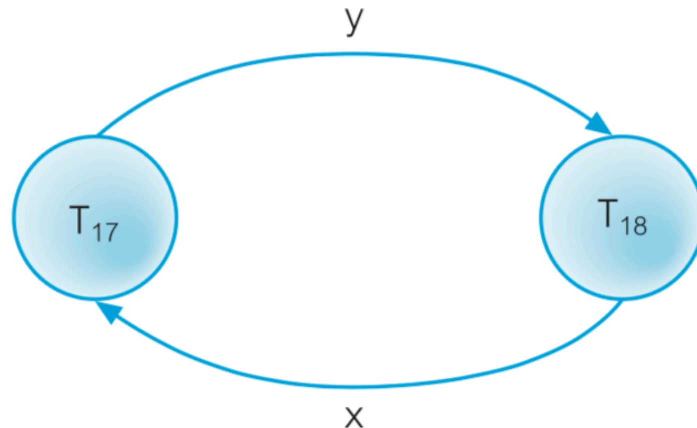
DEAD LOCK

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

Time	T_{17}	T_{18}
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 PREVENTION

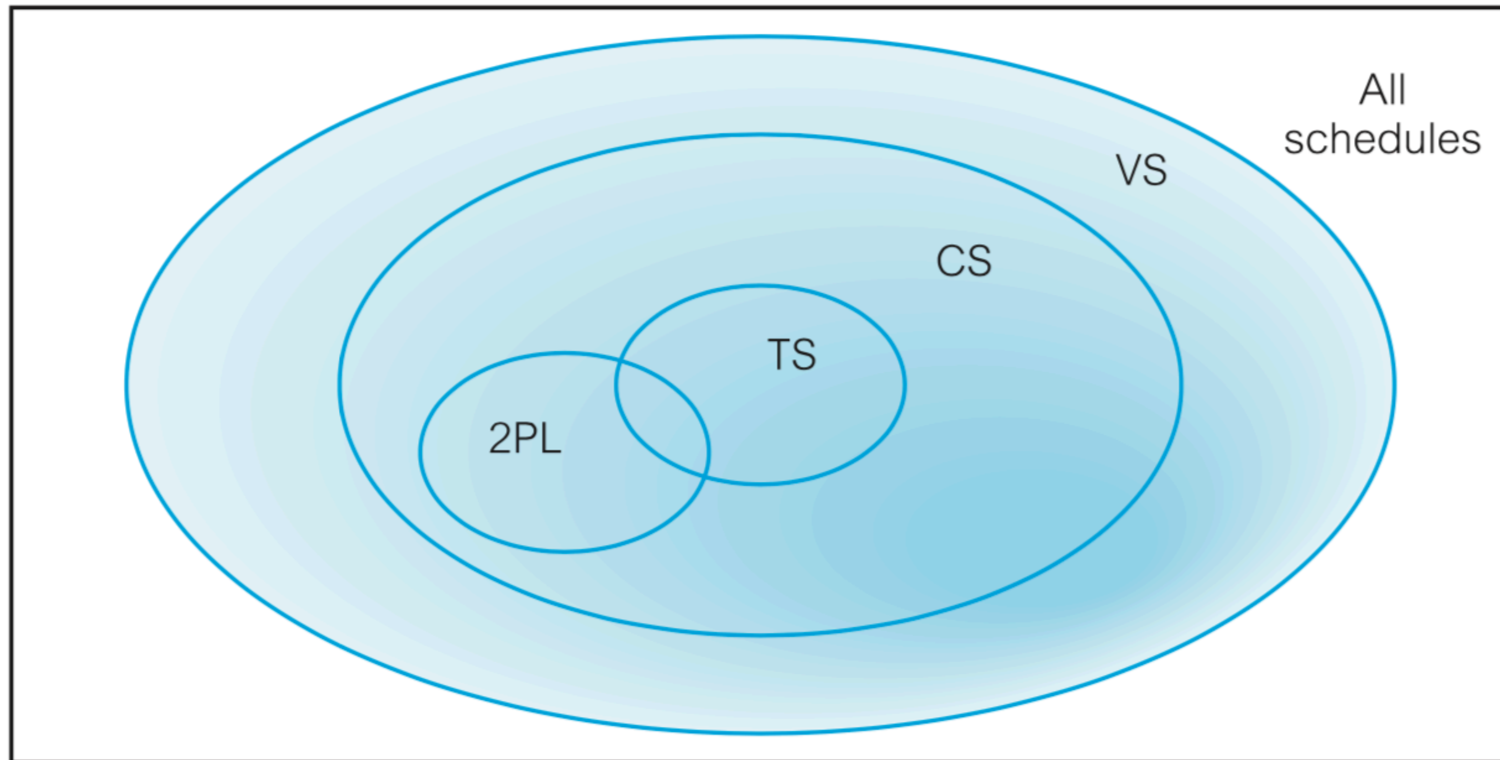
- A simple approach to deadlock prevention is based on lock timeouts.
- Another possible approach to deadlock prevention is to order transactions using transaction timestamps, which we discuss in “Timestamps”
- Deadlock detection is usually handled by the construction of a wait-for graph (WFG)



TIME STAMPING METHODS

- Timestamp is a unique identifier created by the DBMS that indicates the relative starting time of a transaction.
- Timestamping is a concurrency control protocol that orders transactions in such a way that older transactions, transactions with smaller timestamps, get priority in the event of conflict.

COMPARISON OF METHODS



OPTIMISTIC APPROACHES (1)

- Optimistic concurrency control approaches assume that conflicts are rare. If conflicts are rare, it is more efficient to check for conflicts rather than use locks to force waiting.
- In optimistic approaches, transactions are permitted to access the database without acquiring locks. Instead, the concurrency control manager checks whether a conflict has occurred. The check can be performed either just before a transaction commits or after each read and write.

OPTIMISTIC APPROACHES (2)

- By reviewing the relative time of reads and writes, the concurrency control manager can determine whether a conflict has occurred.
- If a conflict occurs, the concurrency control manager issues a rollback and restarts the offending transaction.

RECOVERY MANAGEMENT

- Recovery management is a service to restore a database to a consistent state after a failure.
- This section describes the kinds of failure to prevent, the tools of recovery management, and the recovery processes that use the tools.

THE NEED FOR RECOVERY

- System crashes due to hardware or software errors, resulting in loss of main memory.
- Media failures, such as head crashes or unreadable media, resulting in the loss of parts of secondary storage.
- Application software errors, such as logical errors in the program that is accessing the database, which cause one or more transactions to fail.
- Natural physical disasters, such as fires, floods, earthquakes, or power failures.
- Carelessness or unintentional destruction of data or facilities by operators or users.
- Sabotage, or intentional corruption or destruction of data, hardware, or software facilities.

TRANSACTION LOG

- A transaction log provides a history of database changes. Every change to database is also recorded in the log.
- The log is a hidden table not available to normal users.

LSN	TransNo	Action	Time	Table	Row	Column	Old	New
1	101001	START	10:29					
2	101001	UPDATE	10:30	Acct	10001	AcctBal	100	200
3	101001	UPDATE	10:30	Acct	15147	AcctBal	500	400
4	101001	INSERT	10:32	Hist	25045	*		<1002, 500, . . . >
5	101001	COMMIT	10:33					

CHECKPOINT

- Checkpoint is the act of writing a checkpoint record to the log and writing log and database buffers to disk.
- All transaction activity cases while a checkpoint occurs
- The checkpoint interval should be chosen to balance restart time with checkpoint overhead.

DATABASE BACKUP

- A backup is a copy of all or part if a disk. The backup is used when the disk containing the database or log is damaged.
- A backup is usually made on magnetic tape because it is less expensive and more reliable than disk.
- Periodically, a backup should be made for both the database and the log.

RECOVERY PROCESSES

- The recovery process depends on the kind of failure. Recovery from a device failure is simple but can be time-consuming, as listed below:
  - The database is restored from the most recent backup
  - Then, the recovery manager applies the redo operator to all committed transactions after the backup. Because the backup may be several hours to days old, the log must be consulted to restore transactions committed after the backup.
  - The recovery process finishes by restarting incomplete transactions.

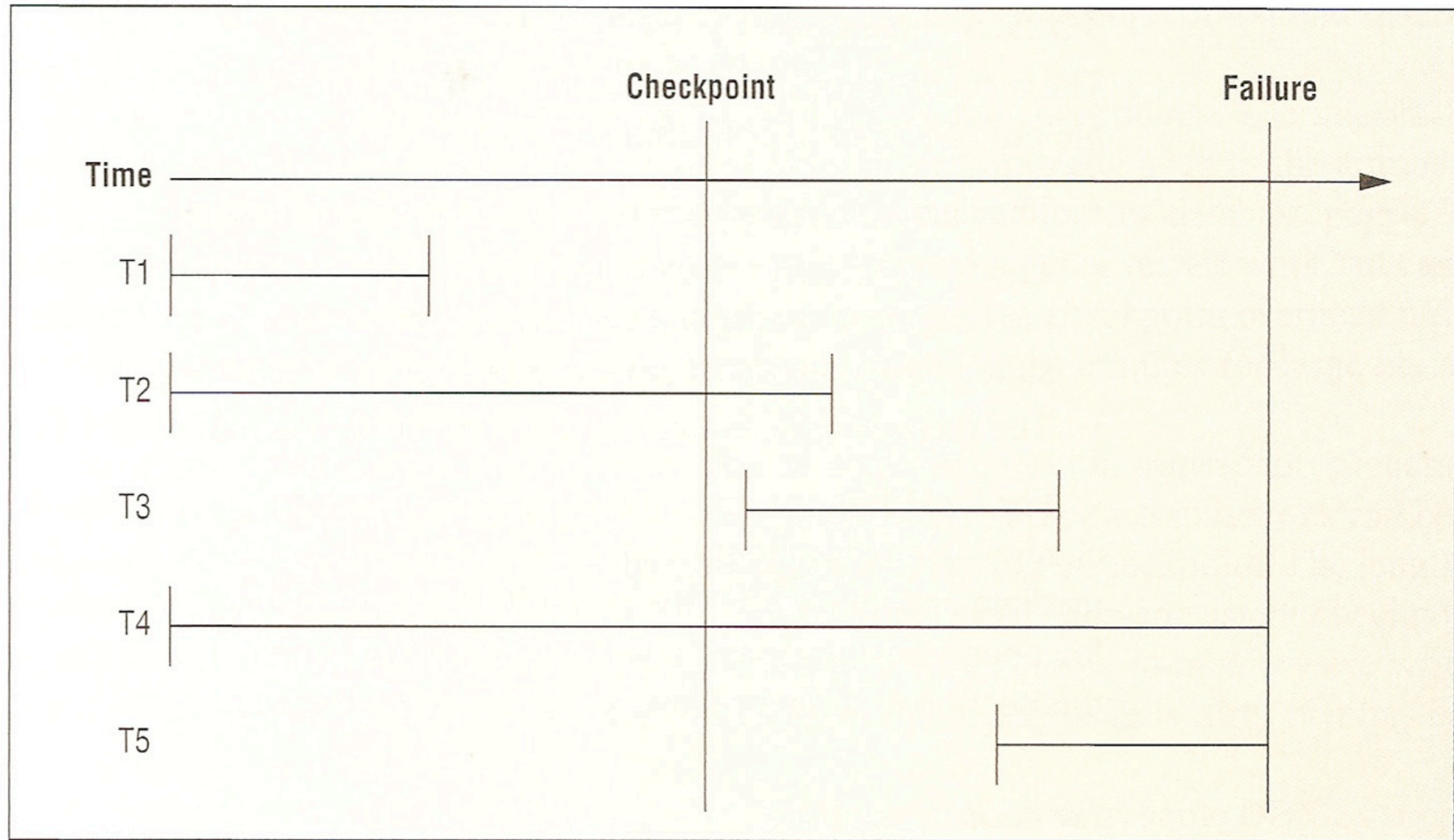
IMMEDIATE UPDATE

- In the immediate update approach, database updates are written to disk when they occur but after the corresponding log updates.
- To restore a database, both undo and redo operations may be needed.

DEFERRED UPDATE

- In the deferred update approach, database updates are written only after a transaction commits.
- To restore a database, only redo operations are used.

TRANSACTION TIMELINE



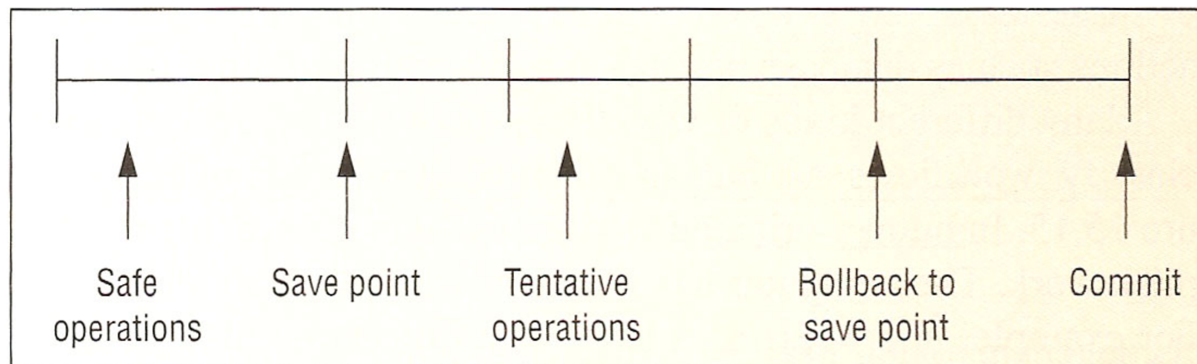
SUMMARY OF RESTART WORK FOR THE IMMEDIATE AND DEFERRED UPDATE APPROACH

Class	Description	Restart Work
T1	Finished before CP	None
T2	Started before CP; finished before failure	Redo forward from the most recent checkpoint
T3	Started after CP; finished before failure	Redo forward from the most recent checkpoint
T4	Started before CP; not yet finished	Undo backwards from most recent log record
T5	Started after CP; not yet finished	Undo backwards from most recent log record

Class	Description	Restart Work
T1	Finished before CP	None
T2	Started before CP; finished before failure	Redo forward from the first log record
T3	Started after CP; finished before failure	Redo forward from the first log record
T4	Started before CP; not yet finished	None
T5	Started after CP; not yet finished	None

SAVE POINTS

- Some transactions have tentative actions that can be canceled by user actions or other events.
- For example, a user may cancel an item on an order after discovering that the item is out of stock.
- MS-SQL provides the SAVEPOINT statement to allow partial rollback of a transaction.



TRANSACTION DESIGN

- With recovery and concurrency services, it may be surprising that the transaction designer still has important design decisions.
- The transaction designer can be a database administrator, a programmer, or a programmer in consultation with a database administrator.
- The design decisions can have a significant impact on transaction processing performance.

TRANSACTION BOUNDARY AND HOT SPOTS

- Transaction boundary is an important decision of transaction design in which an application consisting of a collection of SQL statements is divided into one or more transactions.
- Hot spots can be classified as either system independent or system dependent.
  - System-independent hot spots are parts of a table that many users simultaneously may want to change.
  - System-dependent hot spots depend on the DBMS. Usually, system-dependent hot spots involve parts of the database hidden to normal users.

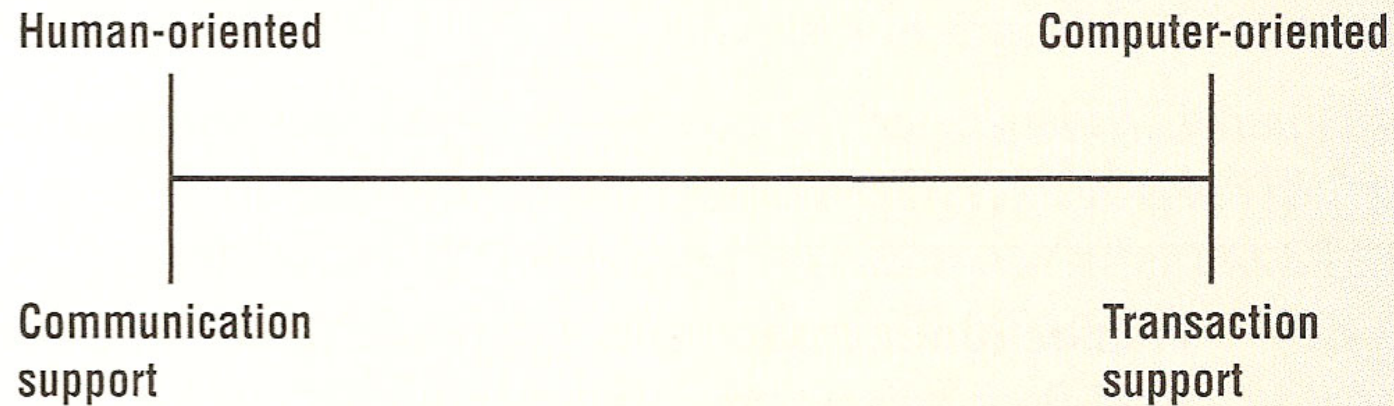
PSEUDOCODE FOR REDESIGNED AIRLINE RESERVATION TRANSACTION

```
Display greeting
Get reservation preferences
SELECT departure and return flight records
if reservation is acceptable then
    START TRANSACTION
    UPDATE seats remaining of departure flight record
    UPDATE seats remaining of return flight record
    INSERT reservation record
End If
On Error: ROLLBACK
COMMIT
Send receipt to customer
```

WORKFLOW MANAGEMENT

- Workflow is a collection of related tasks structured to accomplish a business process.
- In human-oriented workflows, humans provide most of the judgement to accomplish work.
- In computer-oriented task, the computer determines the processing of work.

CLASSIFICATION OF WORKFLOW BY TASK TASK PERFORMANCE



CLASSIFICATION OF WORKFLOW BY TASK STRUCTURE AND COMPLEXITY