INT206

Transaction Management

Part 2

Sanit Sirisawatvatana and Sunisa Sathapornwajana

Serializability and Recoverability

- When multiple transactions run concurrently, there is a possible that the database may be left in an inconsistent state.

- The objective of a concurrency control protocol
  - is to schedule transactions in such a way as to avoid any interference between them
  - hence prevent the types of problem described in the previous part.

Serial Schedule vs. Non-serial Scedule

- Schedule
  - A sequence of the operations by a set of concurrent transactions that preserves the order of the operations in each of the individual transctions.

- Serial schedule
  - Is always a serializable schedule.
  - A schedule where opertions of each transaction are executed consecutive without any interleaved operations from other transactions.
  - A transaction only starts when the other transaction finished executed.
- Non-serial schedule
  - A schedule where the operations from a set of current transactions are interleaved.
  - Is said to be serializable schedule, if it is equivalent to the serial schedul of those n tranasctions.

Serializability

- The objective of serializability
  - Is to find nonserial schedules that allow transactions to execute concurrently without interfering with one another, and thereby produce a database state that could be produced by a serial execution.

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

Recoverablility

- Serializability identifies schedules that maintain the consistency of database, assuming that none of the transactions in the schedule fails.
- Recoverablity of transactions within a schedule:
  - If a transaction fails, the atomicity property requires that we undo the effects of transactions.
  - The durability property states that one a transaction commits, its changes cannot be undone (without running another, compensating, transaction).

Unrecoverable Schedule

- G is nonrecoverable schedule, because T2 read the value of A written by T1, and committed.
- T1 later aborted, therefore the value read by T2 is wrong, but since T2 committed.

$$G = \begin{bmatrix} T1 & T2 \\ R(A) & \\ W(A) & \\ & R(A) \\ & W(A) \\ & Com. \\ Abort & \\ & \end{bmatrix}$$

Recoverable Schedule

- A schedule where, for each pair of transactions T_i and T_j, if T_j reads a data item previously written by T_i, then the commit operation of T_i precedes the commit operation of $T_{j.}$

$$F = \begin{bmatrix} T1 & T2 \\ R(A) & \\ W(A) & \\ & R(A) \\ & W(A) \\ Com. & \\ & Com. \\ & \end{bmatrix} \quad F2 = \begin{bmatrix} T1 & T2 \\ R(A) & \\ W(A) & \\ & R(A) \\ & W(A) \\ Abort & \\ & Abort \\ & \end{bmatrix}$$

- F is recoverable because T1 commits before T2, that makes the value read by T2 corrects, Then T2 can commit itself.
- In F2 is recoverable, if T1 aborted, T2 has to abort because the value of A it read is incorrect

Concurrency Control Techniques

- Serializability can be achieved in several ways.
- Two concurrency control techniques
  - Locking
  - Timestamping

- Both are conservative (or pessimistic) approaches
  - They cause transaction to be delayed when they conflict with other transactions

- Optimistic approaches based on:
  - Transaction conflict is rare. So they allow transactions to run unsynchronized and check for conflicts only when the transaction commits at the end

Locking Methods

- Definition:
  - A procedure used to control concurrency access to data. Transaction uses locks to deny access to other transactions to prevent incorrect results.

- Locking methods are the most widely used approach to ensure serializability of concurrent transactions.

- Transaction must request a lock on a data item before it read or write the data item.

- The lock prevents another transaction from modifying or reading the item.

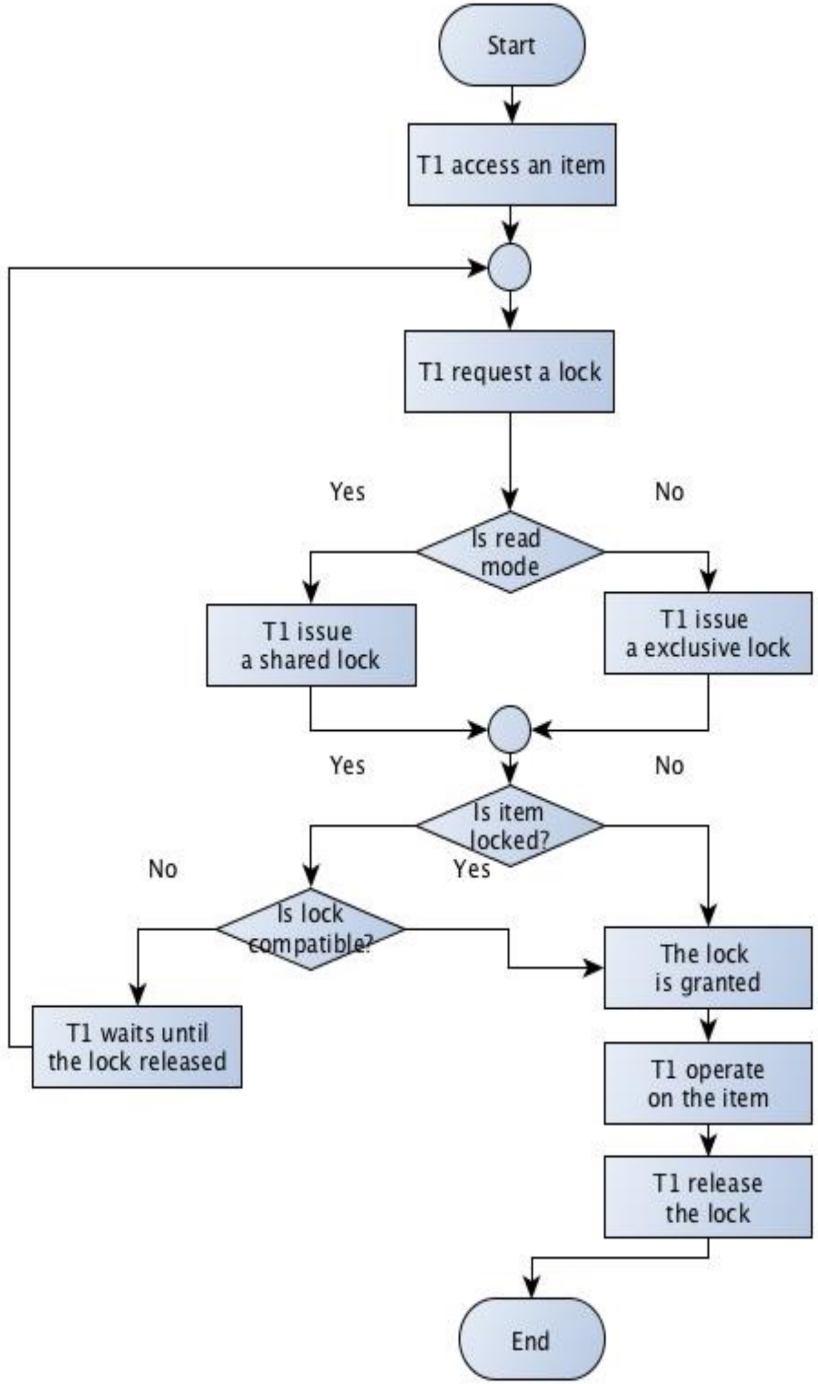
- The other transaction must wait until the lock is released.

Types of a lock

- Two types:
  - Shared lock (read lock)
    - Is used for read-only mode (reading purpose)
    - Transaction requests shared lock on a data item in order to read its content.
    - Updating the data is not allowed

  - Exclusive lock (write lock)
    - Is used for write mode
    - Only the transaction that requests a lock can read and update the data item
    - The other transactions can not read , write or lock the data and must wait until the lock is released

Flowchart of locking

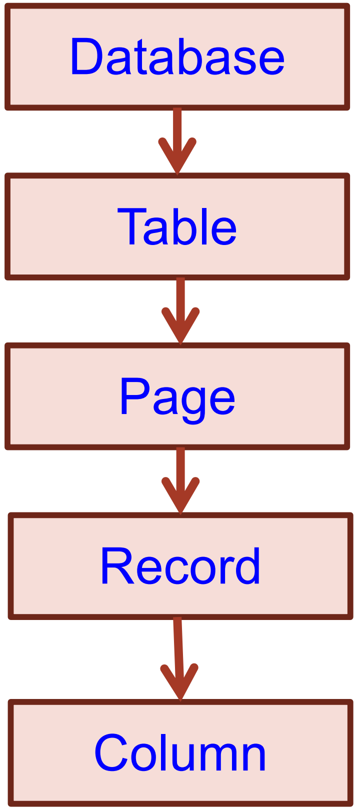


Lock Matrix

	T1 issue a lock at t = 0	
T2 issue a lock at t = 10	Shared	Exclusive
Shared	Both are granted shared locks Both only Read	Only T1 is granted an exclusive lock T2 wait for a lock T1 Read/Write
Exclusive	Only T1 is granted a shared lock T2 wait for a lock T1 only Read	Only T1 is granted an exclusive lock T2 wait for a lock T1 Read/Write

Some systems allow transaction to upgrade shared (read) lock to an exclusive lock, or downgrade exclusive lock to a shared lock.

Locking Granularity



Example : Incorrect locking schedule

Transactions release locks too soon				
Time	T9	T10	balx	baly
t1	write_lock(balx)		100	400
t2	read(balx)		100	400
t3	balx = balx +100		200	400
t4	write(balx)		200	400
t5	unlock(balx)		200	400
t6		write_lock(balx)	200	400
t7		read(balx)	200	400
t8		balx = 1.1*balx	220	400
t9		write(balx)	220	400
t10		unlock(balx)	220	400
t11		write_lock(baly)	220	400
t12		read(baly)	220	400
t13		baly = 1.1*baly	220	440
t14		write(baly)	220	440
t15		unlock(baly)	220	440
t16		commit	220	440
t17	write_lock(baly)		220	440
t18	read(baly)		220	440
t19	baly = baly - 100		220	340
t20	write(baly)		220	340
t21	unlock(baly)		220	340
t22	commit		220	340

T9 executes before T10

T9 executes before T10				
Time	T9	T10	balx	baly
t1	write_lock(balx)		100	400
t2	read(balx)		100	400
t3	balx = balx +100		200	400
t4	write(balx)		200	400
t5		write_lock(balx)	200	400
t6		wait	200	400
t7		wait	200	400
t8		wait	200	400
t9		wait	200	400
t10		wait	200	400
t11		wait	200	400
t12		wait	200	400
t13		wait	200	400
t14		wait	200	400
t15		wait	200	400
t16	write_lock(baly)	wait	200	400
t17	read(baly)	wait	200	400
t18	baly = baly - 100	wait	200	300
t19	write(baly)	wait	200	300
t20	commit	wait	200	300
t21		read(balx)	200	300
t22		balx = 1.1*balx	220	300
t23		write(balx)	220	300
t24		write_lock(baly)	220	300
t25		read(baly)	220	300
t26		baly = 1.1*baly	220	330
t27		write(baly)	220	330
t28		commit	220	330

T10 executes before T9

T10 executes before T9

Time	T9	T10	balx	baly
t1		write_lock(balx)	100	400
t2	write_lock(balx)	read(balx)	100	400
t3	wait	balx = 1.1*balx	110	400
t4	wait	write(balx)	110	400
t5	wait	write_lock(baly)	110	400
t6	wait	read(baly)	110	400
t7	wait	baly = 1.1*baly	110	440
t8	wait	write(baly)	110	440
t9	wait	commit	110	440
t10	read(balx)		110	440
t11	balx = balx +100		210	440
t12	write(balx)		210	440
t13	write_lock(baly)		210	440
t14	read(baly)		210	440
t15	baly = baly - 100		210	340
t16	write(baly)		210	340
t17	commit		210	340

Locking

- Locking in transactions does not guarantee serializability of schedules

- The problem of incorrect locking schedule is that
  - The transactions release locks too soon after executing read/write operations
  - It allows transactions to interfere with one another
  - This results in loss of total isolation and atomicity.

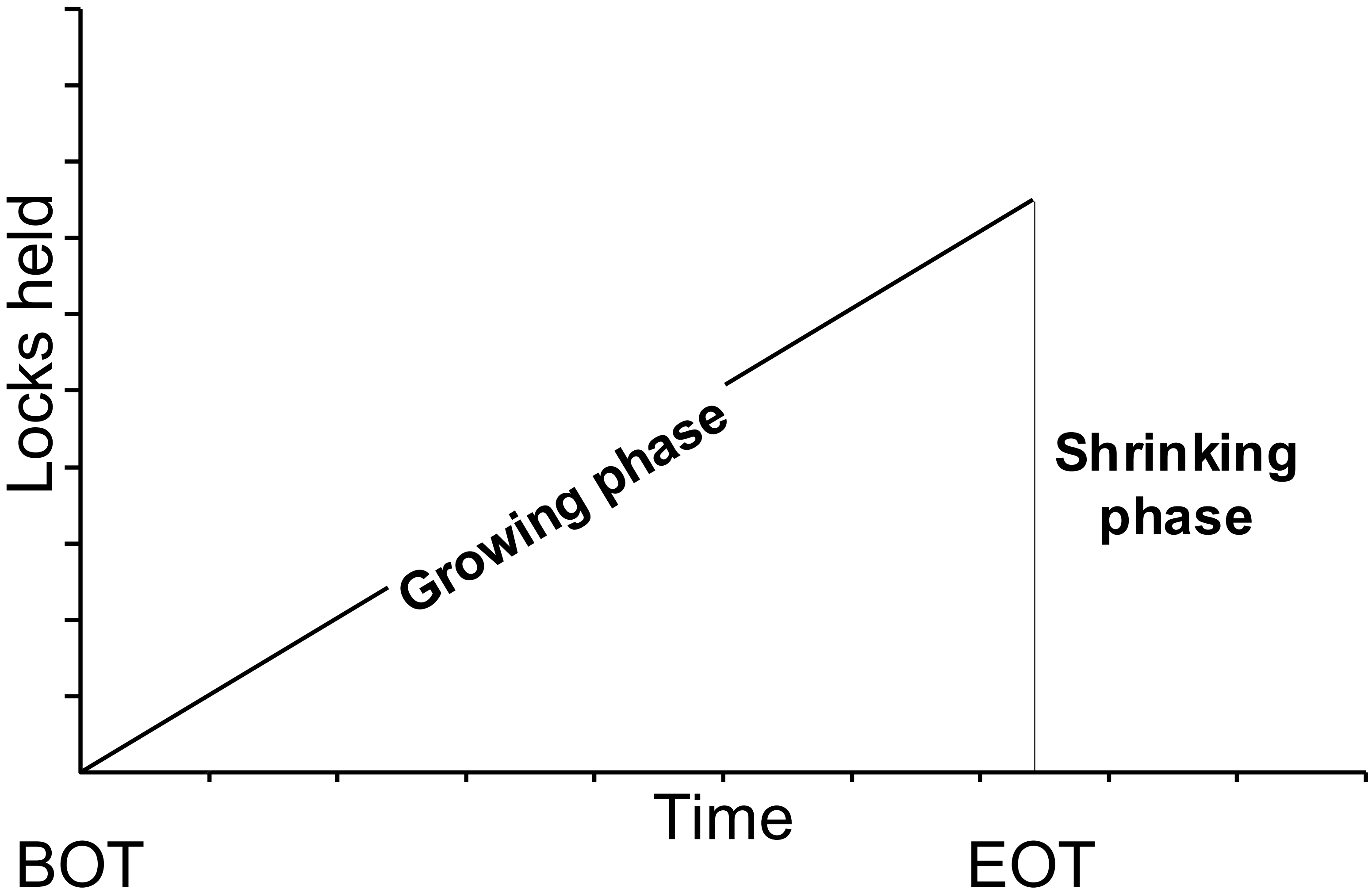
- To guarantee serializability
  - Needs an additional protocol concerning the positioning of lock and unlock operations in every transaction.
  - The best known protocol is two-phase locking (2PL)

Two-phase locking (2PL)

- Definition:
  - All locking operations precede the first unlock operations in the transaction.

- Rules
  - Every transaction can be divided into two phases
  - Growing phase
    - Transaction must acquire a lock before operating (read/write) on the item.
    - It acquires all the locks but can not release any locks
    - There is no requirement that all locks be obtained at the same time
    - Transaction acquires some locks, does some processing, and goes on to acquire additional locks as needed.
  - Shrinking phase
    - When no new locks are needed, it releases its locks but can not acquire any new locks.

Two-phase locking (2PL)



Preventing the lost update problem using 2PL

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

Preventing Uncommitted Dependency Problem using 2PL

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

Preventing Inconsistent Analysis Problem using 2PL

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

Deadlock

- Definition:
  - Two (or more) transactions wait for locks on items to be released that are held by the other transactions
- This is a problem of using two-phase locking, which applies to all locking-based schemes
- When deadlock occurs, the applications involved cannot resolve the problem
- The DBMS has to recognize the deadlock and break the deadlock in some way.
- The only way to break deadlock is aborting one or more of transactions (requiring to undoing all the changes made by aborted transactions)

Example : Deadlock

Time	T_{17}	T_{18}
t_1	begin_transaction	
t_2	write_lock($\mathbf{bal_x}$)	begin_transaction
t_3	read($\mathbf{bal_x}$)	write_lock($\mathbf{bal_y}$)
t_4	$\mathbf{bal_x} = \mathbf{bal_x} - 10$	read($\mathbf{bal_y}$)
t_5	write($\mathbf{bal_x}$)	$\mathbf{bal_y} = \mathbf{bal_y} + 100$
t_6	write_lock($\mathbf{bal_y}$)	write($\mathbf{bal_y}$)
t_7	WAIT	write_lock($\mathbf{bal_x}$)
t_8	WAIT	WAIT
t_9	WAIT	WAIT
t_{10}	⋮	WAIT
t_{11}	⋮	⋮

Techniques for handling deadlock

- Three techniques:
  - Timeouts
  - Deadlock prevention
  - Deadlock detection

Timeouts

- Simple approach is based on lock timeouts
- A transaction that requests a lock will wait for a system-defined period of time
- If the lock has been granted within this period, the lock request times out
- Can abort transactions that are not deadlocked
- Timeouts is used by several commercial DBMSs

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 algorithm
  - Wound-Wait algorithm

Deadlock detection

- DBMS allows deadlock to occur but recognizes it and breaks it.

- To detect the deadlock
  - Is to check allocation against resource availability for all possible allocation sequences to determine if the system is in deadlock state.

- Once a deadlock is detected, the DBMS needs to abort it and needs to be a way to recovery the aborted transactions.

- Used by enterprise DBMSs such as Oracle

Cascading Rollback

- Every transaction in a schedule follows 2PL, schedule is serializable.
- However, problems can occur with interpretation of when locks can be released.
- Cascading rollback occurs when a transaction (T1) causes a failure and a rollback must be performed.
- Other transactions dependent on T1's actions must also be rollbacked due to T1's failure.

Cascading Rollbak

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

Timestamping Methods

- Definition:
  - A unique identifier created by DBMS that indicates relative starting time of a transaction.

- Is another approach to guarantee serializability by ordering transaction execution for an equivalent serial schedule

- No locking are involved, No deadlock
- Transactions are not waiting for locking
- Transactions involved in conflict are simply rolled back and restarted

Optimistic Techniques

- Based on assumption that conflict is rare and more efficient to let transactions proceed without delays to ensure serializability.

- At commit, check is made to determine whether conflict has occurred.

- If there is a conflict, transaction must be rolled back and restarted.

- Potentially allows greater concurrency than traditional protocols.

Database Recovery

- Definition:
  - Process of restoring database to a correct state in the event of a failure.

- Transactions represent basic unit of recovery.

- Recovery manager responsible for atomicity and durability.

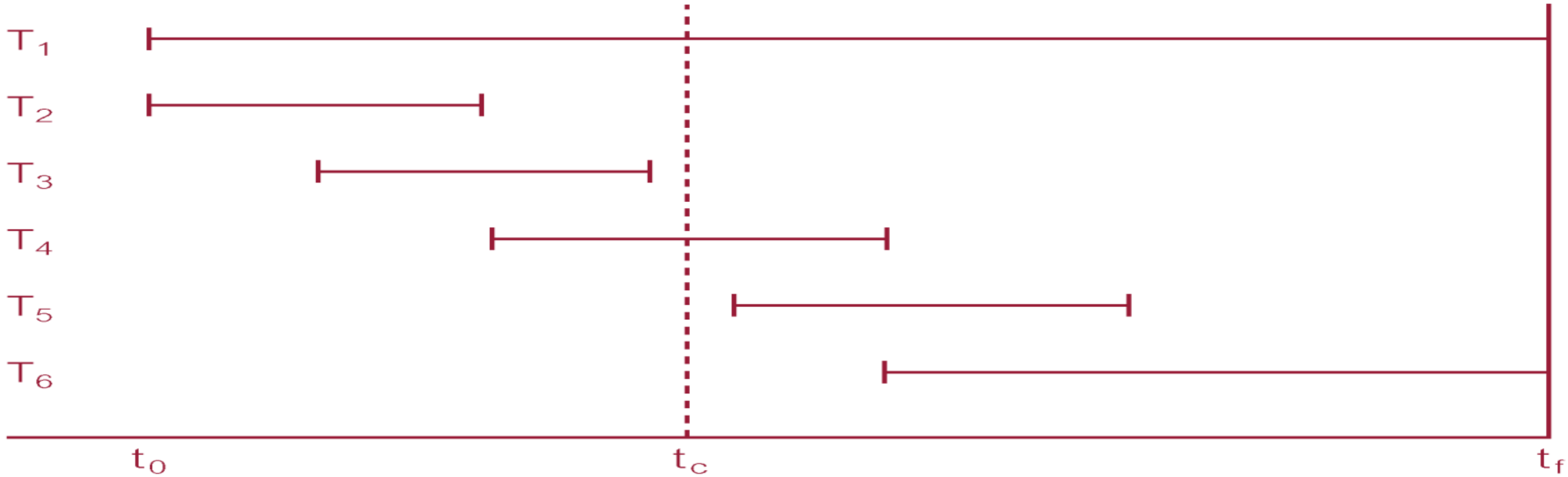
Types of Failures

- 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.

Transactions and Recovery

- If failure occurs between commit and database buffers being flushed to secondary storage then, to ensure durability, recovery manager has to redo (rollforward) transaction's updates.
- If transaction had not committed at failure time, recovery manager has to undo (rollback) any effects of that transaction for atomicity.
- Partial undo - only one transaction has to be undone.
- Global undo - all transactions have to be undone.

Example



- DBMS starts at time t_0, but fails at time t_f. Assume data for transactions T_2 and T_3 have been written to secondary storage.
- T_1 and T_6 had not committed t_f. Therefore at restart the recovery manager must undo transactions T1 and T6.
- In absence of any other information, recovery manager has to redo T_2, T_3, T_4, and T_5.

Recovery Facilities

- DBMS should provide 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.

Log File

- Contains information about all updates to database:
  - Transaction records.
  - Checkpoint records.

- Is often used for purposes:
  - Recovery
  - Performance monitoring and auditing

- To recover quickly from minor failures
  - This requires that the log file be stored online on a fast direct-access storage device

Checkpointing

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

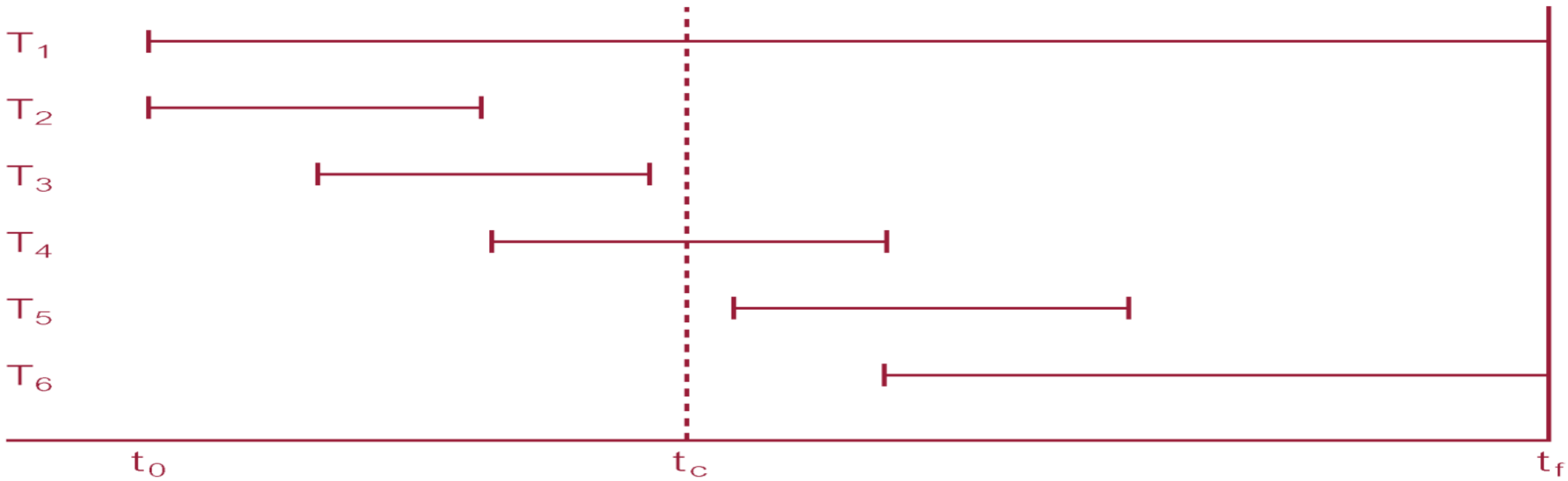
- Checkpoints are scheduled at predefined intervals and involve the following operations:
  - writing all log records in main memory to secondary storage
  - writing the modified blocks in the databases buffers to secondary storage
  - writing a checkpoint record to the log file. This record contains the identifiers of all transactions that are active at the time of the checkpoint

- When failure occurs, redo all transactions that committed since the checkpoint and undo all transactions active at time of crash.

Checkpointing

- In previous example, with checkpoint at time tc, changes made by T2 and T3 have been written to secondary storage.

- .



  - The recovery manager omits the redo for T2 and T3.
  - only redo T4 and T5 since the checkpoint.
  - undo transactions T1 and T6.

Recovery Techniques

- If database has been damaged:
  - Need to restore last backup copy of database and reapply updates of committed transactions using log file.

- If database is only inconsistent:
  - For example, the system crashed while transactions were executing.
  - Need to undo changes that caused inconsistency. May also need to redo some transactions to ensure updates reach secondary storage.
  - Do not need backup,but can restore database using the log file.