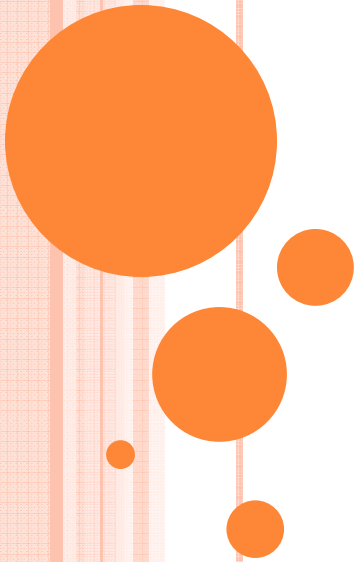


SERIALIZABILITY

Lisna Thomas

BCA

SERIALIZABILITY IN DBMS-

- Some non-serial schedules may lead to inconsistency of the database.
- Serializability is a concept that helps to identify which non-serial schedules are correct and will maintain the consistency of the database.

SERIALIZABLE SCHEDULES

- If a given non-serial schedule of ‘n’ transactions is equivalent to some serial schedule of ‘n’ transactions, then it is called as a **serializable schedule**.

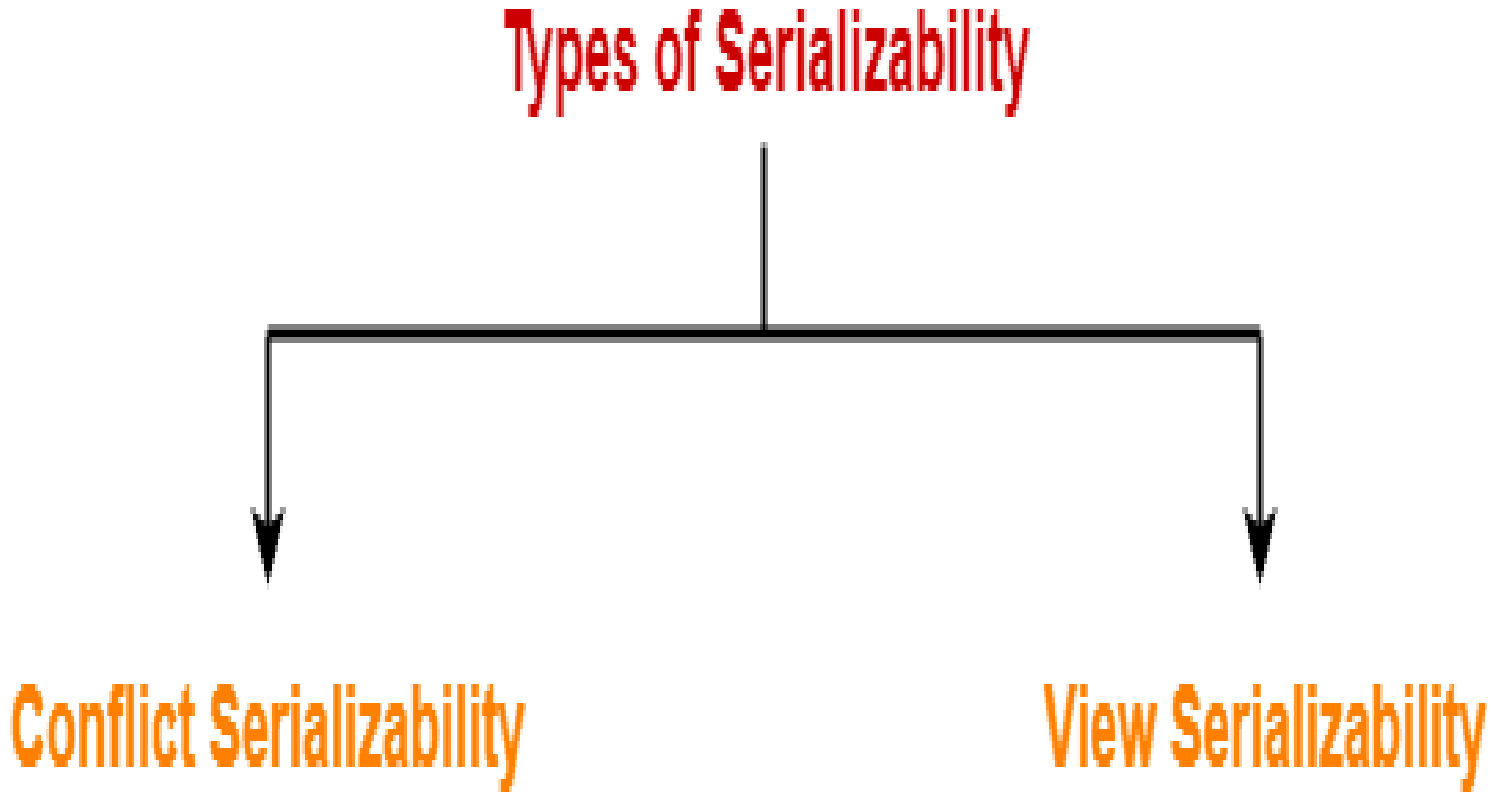
CHARACTERISTICS-

Serializable schedules behave exactly same as serial schedules.

Thus, serializable schedules are always-

- Consistent
- Recoverable
- Casacadeless
- Strict

Types of Serializability



CONFLICT SERIALIZABILITY

- If a given non-serial schedule can be converted into a serial schedule by swapping its non-conflicting operations, then it is called as a **conflict serializable schedule**.

View Serializability

- If a given schedule is found to be view equivalent to some serial schedule, then it is called as a view serializable schedule.

CONCURRENCY-CONTROL PROTOCOLS

- allow concurrent schedules, but ensure that the schedules are conflict/view serializable, and are recoverable and maybe even cascadeless.
  These protocols do not examine the precedence graph as it is being created, instead a protocol imposes a discipline that avoids non-seralizable schedules.

Lock Based Protocols

- **Shared Lock(S)**

- **Exclusive Lock (X)**

SHARED LOCK(S)

- also known as Read-only lock. As the name suggests it can be shared between transactions because while holding this lock the transaction does not have the permission to update data on the data item. S-lock is requested using lock-S instruction.

EXCLUSIVE LOCK (X)

- Data item can be both read as well as written.This is Exclusive and cannot be held simultaneously on the same data item. X-lock is requested using lock-X instruction.

Lock Compatibility Matrix

	S	X
S	✓	✗
X	✗	✗

- A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions.
- Any number of transactions can hold shared locks on an item, but if any transaction holds an exclusive(X) on the item no other transaction may hold any lock on the item.
- If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. Then the lock is granted.

UPGRADE / DOWNGRADE LOCKS

A transaction that holds a lock on an item **A** is allowed under certain condition to change the lock state from one state to another.
Upgrade: A S(A) can be upgraded to X(A) if T_i is the only transaction holding the S-lock on element A.
Downgrade: We may downgrade X(A) to S(A) when we feel that we no longer want to write on data-item A. As we were holding X-lock on A, we need not check any conditions.

So, by now we are introduced with the types of locks and how to apply them. But wait, just by applying locks if our problems could've been avoided then life would've been so simple! If you have done Process Synchronization under OS you must be familiar with one consistent problem, starvation and Deadlock! We'll be discussing them shortly, but just so you know we have to apply Locks but they must follow a set of protocols to avoid such undesirable problems. Shortly we'll use 2-Phase Locking (2-PL) which will use the concept of Locks to avoid deadlock. So, applying simple locking, we may not always produce Serializable results, it may lead to Deadlock Inconsistency.

PROBLEM

T_1	T_2
1. lock-X(B)	
2. read(B)	
3. B:=B-50	
4. write(B)	
5.	lock-S
6.	read(A)
7.	lock-S(B)
8. lock-X(A)	
9. …….	…….

DEADLOCK

- consider the above execution phase.
  Now, $\mathbf{T_1}$ holds an Exclusive lock over B,
  and $\mathbf{T_2}$ holds a Shared lock over A. Consider Statement 7, $\mathbf{T_2}$ requests for lock on B, while in Statement 8 $\mathbf{T_1}$ requests lock on A. This as you may notice imposes a **Deadlock** as none can proceed with their execution.

STARVATION

- is also possible if concurrency control manager is badly designed. For example: A transaction may be waiting for an X-lock on an item, while a sequence of other transactions request and are granted an S-lock on the same item. This may be avoided if the concurrency control manager is properly designed.

Concurrency Control Protocol

Two Phase Locking –

A transaction is said to follow Two Phase Locking protocol if Locking and Unlocking can be done in two phases.

- **Growing Phase:** New locks on data items may be acquired but none can be released.
- **Shrinking Phase:** Existing locks may be released but no new locks can be acquired.

PROBLEM

T_1	T_2
1. LOCK-S(A) 2. 3. LOCK-X(B) 4. ..... 5. UNLOCK(A) 6. 7. UNLOCK(B) 8. 9. 10.....	LOCK-S(A) ..... LOCK-X(C) UNLOCK(A) UNLOCK(C) .....

This is just a skeleton transaction which shows how unlocking and locking works with 2-PL. Note for:

- **Transaction T_1:**

  Growing Phase is from steps 1-3.

  Shrinking Phase is from steps 5-7.

  Lock Point at 3

- **Transaction T_2:**

  Growing Phase is from steps 2-6.

  Shrinking Phase is from steps 8-9.

  Lock Point at 6


Thank
You