Serializibility Example:

Serial Schedule Vs Concurrent Schedule?

Serial schedules have less resource utilization and low throughput. To improve it, two are more transactions are run concurrently. But concurrency of transactions may lead to inconsistency in database. To avoid this, we need to check whether these concurrent schedules are serializable or not.

Conflict Serializable: A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations.
Conflicting operations: Two operations are said to be conflicting if all conditions satisfy:

- They belong to different transactions
- They operate on the same data item
- At Least one of them is a write operation

Example: –

- **Conflicting** operations pair ($R_1(A)$, $W_2(A)$) because they belong to two different transactions on same data item A and one of them is write operation.
- Similarly, ($W_1(A)$, $W_2(A)$) and ($W_1(A)$, $R_2(A)$) pairs are also **conflicting**.
- On the other hand, ($R_1(A)$, $W_2(B)$) pair is **non-conflicting** because they operate on different data item.
- Similarly, (($W_1(A)$, $W_2(B)$) pair is **non-conflicting**

Example 1:

Consider the following schedule:

S1: $R_1(A)$, $W_1(A)$, $R_2(A)$, $W_2(A)$, $R_1(B)$, $W_1(B)$, $R_2(B)$, $W_2(B)$

If O_i and O_j are two operations in a transaction and $O_i < O_j$ (O_i is executed before O_j), same order will follow in schedule as well. Using this property, we can get two transactions of schedule S1 as:

T1: $R_1(A)$, $W_1(A)$, $R_1(B)$, $W_1(B)$
T2: $R_2(A)$, $W_2(A)$, $R_2(B)$, $W_2(B)$

Possible Serial Schedules are: T1->T2 or T2->T1

-> **Swapping non-conflicting operation**s $R_2(A)$ and $R_1(B)$ in S1, the schedule becomes,

S11: $R_1(A)$, $W_1(A)$, $R_1(B)$, **$W_2(A)$,** $R_2(A)$, **$W_1(B)$,** $R_2(B)$, $W_2(B)$

-> Similarly, s**wapping non-conflicting operations** $W_2(A)$ and $W_1(B)$ in S11, the schedule becomes,

S12: $R_1(A)$, $W_1(A)$, $R_1(B)$, $W_1(B)$, $R_2(A)$, $W_2(A)$, $R_2(B)$, $W_2(B)$

S12 is a serial schedule in which all operations of T1 are performed before starting any operation of T2. Since S has been transformed into a serial schedule S12 by swapping non-conflicting operations of S1, S1 is conflict serializable.

Example 2:

Let us take another Schedule:

S2: $R_2(A)$, $W_2(A)$, $R_1(A)$, $W_1(A)$, $R_1(B)$, $W_1(B)$, $R_2(B)$, $W_2(B)$

Two transactions will be:

T1: $R_1(A)$, $W_1(A)$, $R_1(B)$, $W_1(B)$
T2: $R_2(A)$, $W_2(A)$, $R_2(B)$, $W_2(B)$

Possible Serial Schedules are: T1->T2 or T2->T1

Original Schedule is:

S2: $R_2(A)$, $W_2(A)$, **$R_1(A)$,** $W_1(A)$, $R_1(B)$, $W_1(B)$, **$R_2(B)$,** $W_2(B)$

Swapping non-conflicting operations $R_1(A)$ and $R_2(B)$ in S2, the schedule becomes,

S21: $R_2(A)$, $W_2(A)$, $R_2(B)$, **$W_1(A)$,** $R_1(B)$, $W_1(B)$, $R_1(A)$, **$W_2(B)$**

Similarly, swapping non-conflicting operations $W_1(A)$ and $W_2(B)$ in S21, the schedule becomes,

S22: $R_2(A)$, $W_2(A)$, $R_2(B)$, $W_2(B)$, $R_1(B)$, $W_1(B)$, $R_1(A)$, $W_1(A)$

In schedule S22, all operations of T2 are performed first, but operations of T1 are not in order (order should be $R_1(A)$, $W_1(A)$, $R_1(B)$, $W_1(B)$). So S2 is not conflict serializable.