Schedules

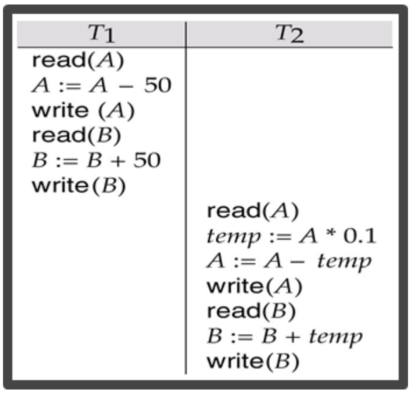
Schedules:
Sequences that indicate the chronological order in which instructions of concurrent transactions are executed
A schedule can have many transactions in it, each consisting of a number of instructions/tasks.

Serial schedule:
A schedule S is serial if, for every transaction T participating in the schedule, all the operations of T are executed consecutively in the schedule. Otherwise, the schedule is called a non serial schedule.

Serializable schedule:
Serializable schedules are always considered to be correct when concurrent transactions are executed.
The main difference between the serial schedule and the serializable schedule is that in serial schedule, no concurrency is allowed whereas in serializable schedule, concurrency is allowed.

Example:
Let *T*1 transfer $50 from *A* to *B*, and *T*2 transfer 10% of the balance from *A* to *B.* The following is a serial schedule, in which *T*1 is followed by *T*2.

Schedule 1



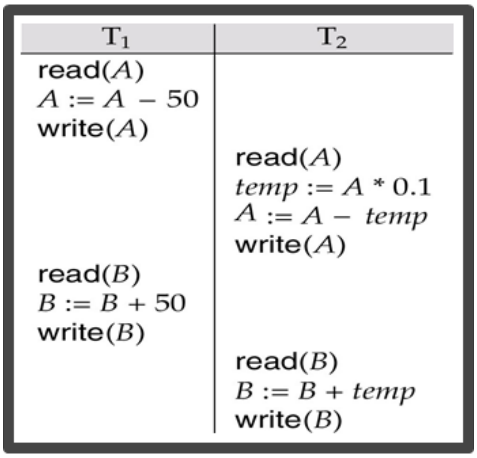
T_1	T_2

read(A)
$A := A - 50$
write (A)
read(B)
$B := B + 50$
write(B)	

	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)

Let *T*1 and *T*2 be the transactions defined previously. The following schedule 2 is not a serial schedule, but it is *equivalent* to Schedule 1.

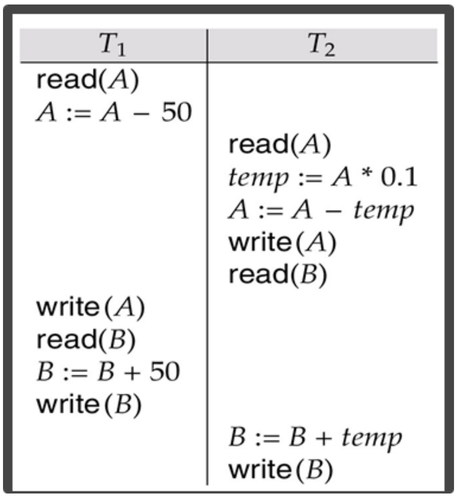
Schedule 2



T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
read(B)	
$B := B + 50$	
write(B)	
	read(B)
	$B := B + temp$
	write(B)

In both Schedule 1 and 2, the sum A + B is preserved.
The following concurrent schedule does not preserve the value of the sum *A* + *B*.

Schedule 3



T_1	T_2
read(A)	
$A := A - 50$	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
write(A)	
read(B)	
$B := B + 50$	
write(B)	
	$B := B + temp$
	write(B)

Serializability
When multiple transactions are being executed by the operating system in a multiprogramming environment, there are possibilities that instructions of one transaction are interleaved with some other transaction.

What is a conflict?

A pair of Operations in a schedule such that if their order is interchanged then the behavior of at least one of the transactions may change.
Operations are conflict, if they satisfy all three of the following conditions :
They belong to different transactions.
They access the same data item .
At least one of the operations is a write operation.

Conflict Equivalence

Schedules are conflict equivalent if they can be transformed one into another by a sequence of non conflicting interchanges adjacent actions.

Conflict Serializability

Instructions I_i and I_j of transactions T_i and T_j respectively, conflict if and only if there exists some item Q accessed by both I_i and I_j, and at least one of these instructions wrote Q.
1.I_i = read(Q), I_j = read(Q). I_i and I_j don't conflict.
2.I_i = read(Q), I_j = write(Q). They conflict.
3.I_i = write(Q), I_j = read(Q). They conflict
4.I_i = write(Q), I_j = write(Q). They conflict
Intuitively, a conflict between I_i and I_j forces a (logical) temporal order between them. If I_i and I_j are consecutive in a schedule and they do not conflict, their results would remain the same even if they had been interchanged in the schedule.

If a schedule S can be transformed into a schedule S´ by a series of swaps of non-conflicting instructions, we say that S and S´ are conflict equivalent.
We say that a schedule S is conflict serializable if it is conflict equivalent to a serial schedule.

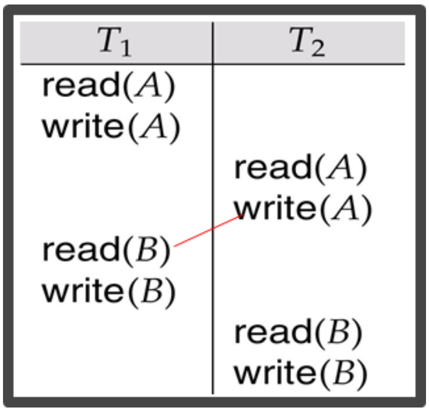
Example of a schedule that is not conflict serializable:

T3	**T4**
READ(Q)	

	WRITE(Q)
WRITE(Q)	

We are unable to swap instructions in the above schedule to obtain either the serial schedule < T_3, T_4 >, or the serial scheSchedule 4 below can be transformed into a serial schedule where T_2 follows T_1, by series of swaps of non-conflicting instructions.
Therefore Schedule 4 is conflict serializable.
dule < T_4, T_3 >.

Schedule 4



T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

View Serializability

Let S and S´ be two schedules with the same set of transactions. S and S´ are view equivalent if the following three conditions are met, for each data item Q,

1.If in schedule S, transaction T_i reads the initial value of Q, then in schedule S' also transaction T_i must read the initial value of Q.
2.If in schedule S transaction T_i executes read(Q), and that value was produced by transaction T_j (if any), then in schedule S' also transaction T_i must read the value of Q that was produced by the same write(Q) operation of transaction T_j .
3.The transaction (if any) that performs the final write(Q) operation in schedule S must also perform the final write(Q) operation in schedule S'.

As can be seen, view equivalence is also based purely on reads and writes alone

A schedule S is viewed as serializable if it is view equivalent to a serial schedule.
Every conflict serializable schedule is also view serializable.
Below is a schedule which is view-serializable but not conflict serializable.

T3	T4	T6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)