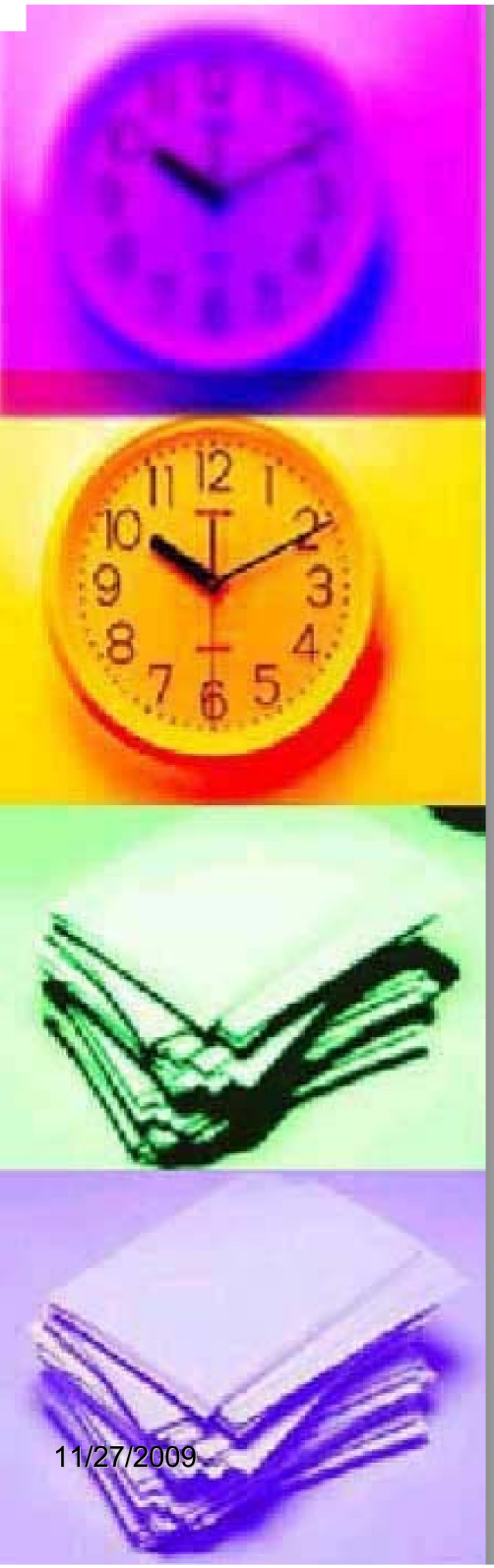
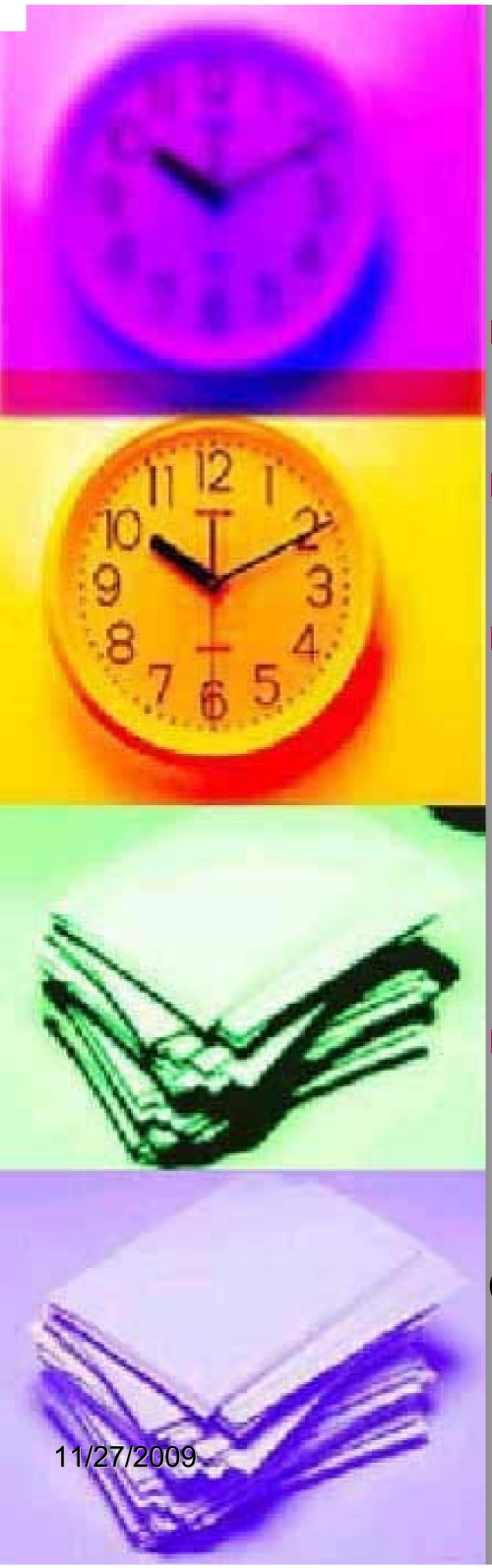


Contents

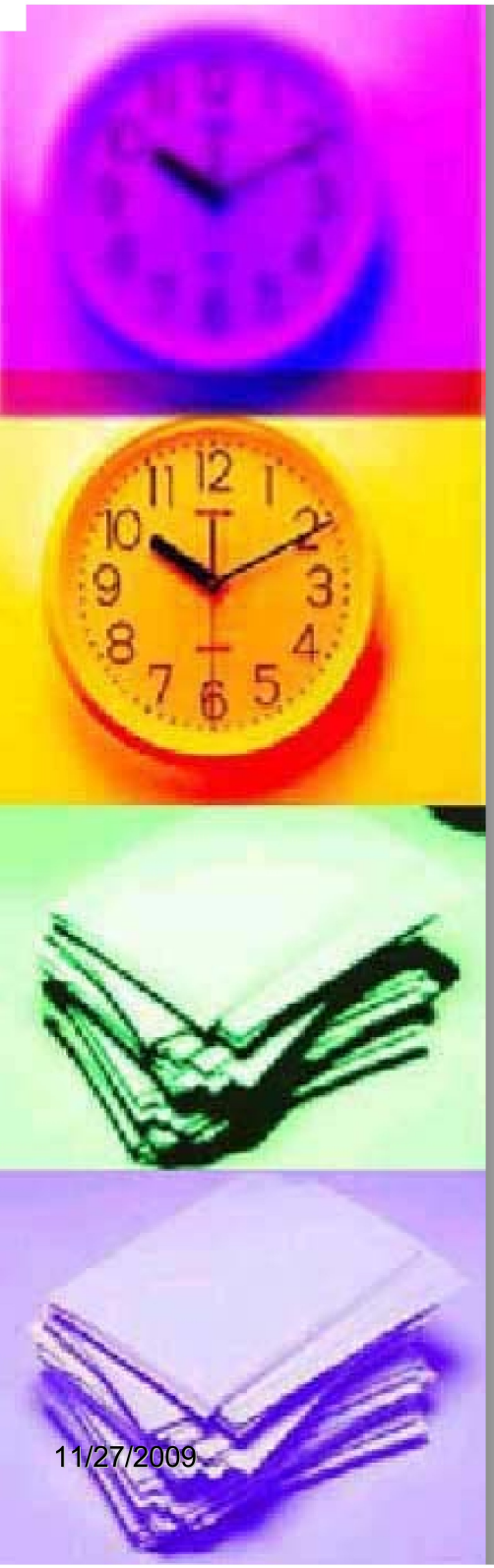
- Transaction Concept
- Transaction State
- Implementation of Atomicity and Durability
- Concurrent Executions
- Serializability
- Recoverability



Serializability

- Basic Assumption – Each transaction preserves database consistency.
- Thus serial execution of a set of transactions preserves database consistency.
- A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule. Different forms of schedule equivalence give rise to the notions of:
  1. conflict serializability
  2. view serializability
- We ignore operations other than **read** and **write** instructions, and we assume that transactions may perform arbitrary computations on data in local buffers in between reads and writes.

Our simplified schedules consist of only **read** and **write** instructions

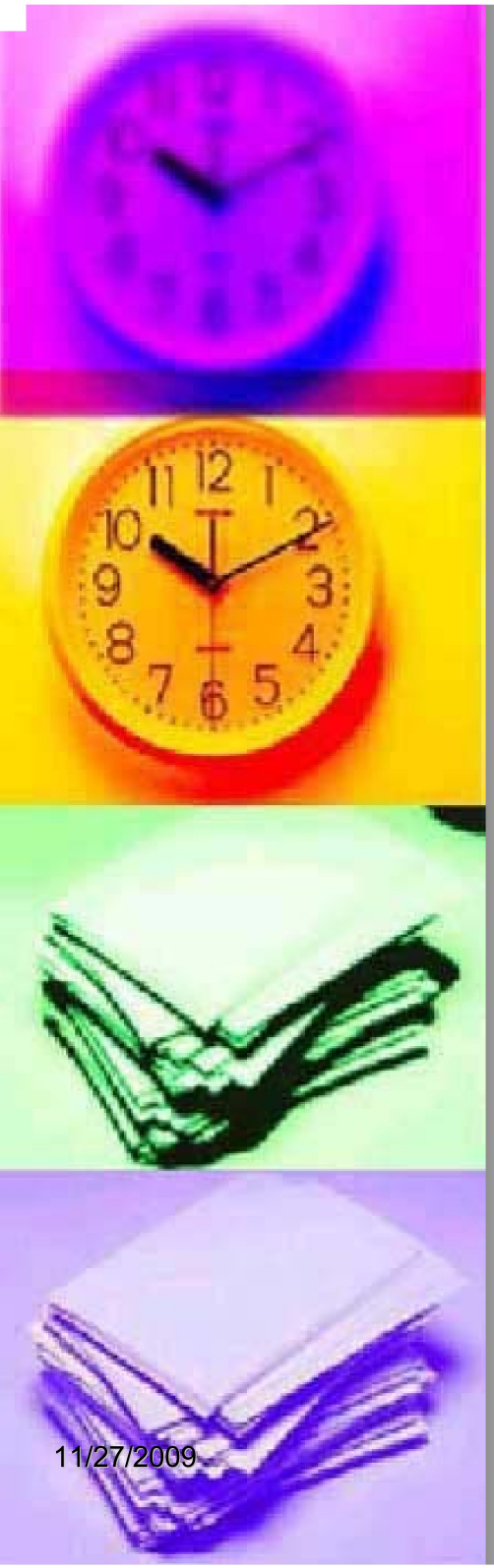


Conflict Serializability

- Instructions l_i and l_j of transactions T_i and T_j respectively, **conflict** if and only if there exists some item Q accessed by both l_i and l_j, and at least one of these instructions wrote Q.

  1. l_i = **read**(Q), l_j = **read**(Q). l_i and l_j don't conflict.
  2. l_i = **read**(Q), l_j = **write**(Q). They conflict.
  3. l_i = **write**(Q), l_j = **read**(Q). They conflict
  4. l_i = **write**(Q), l_j = **write**(Q). They conflict

- Intuitively, a conflict between l_i and l_j forces a (logical) temporal order between them. If l_i and l_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.

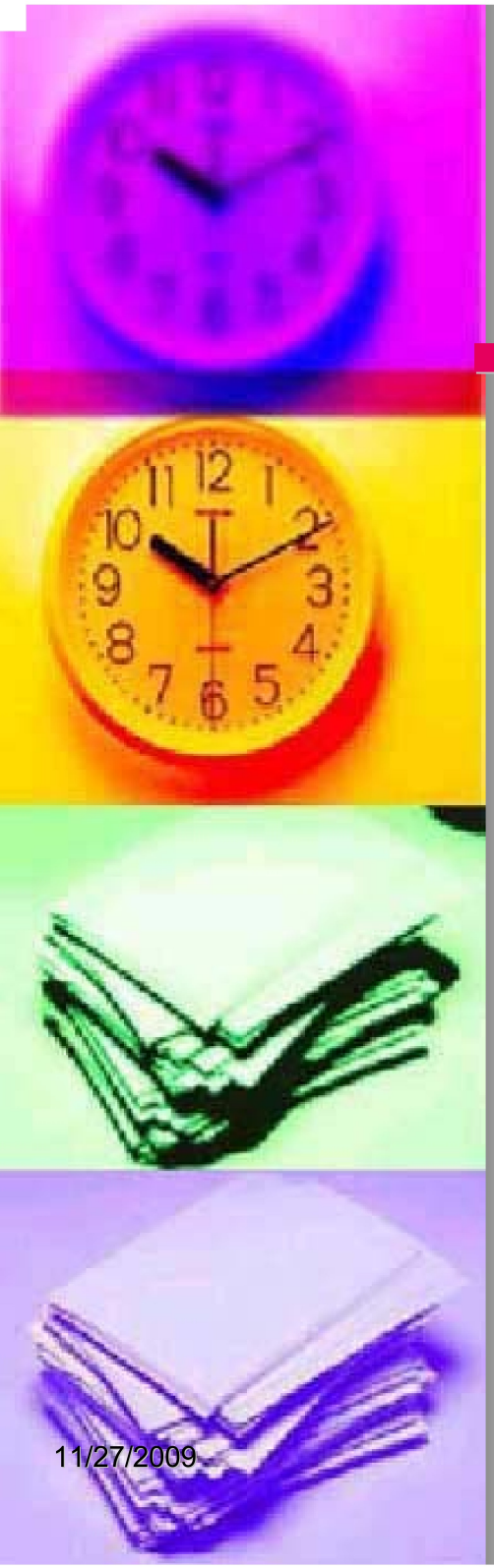


Conflict Serializability (Cont.)

- 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' ar **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:

T_3	T_4
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 schedule $< T_4, T_3 >$.

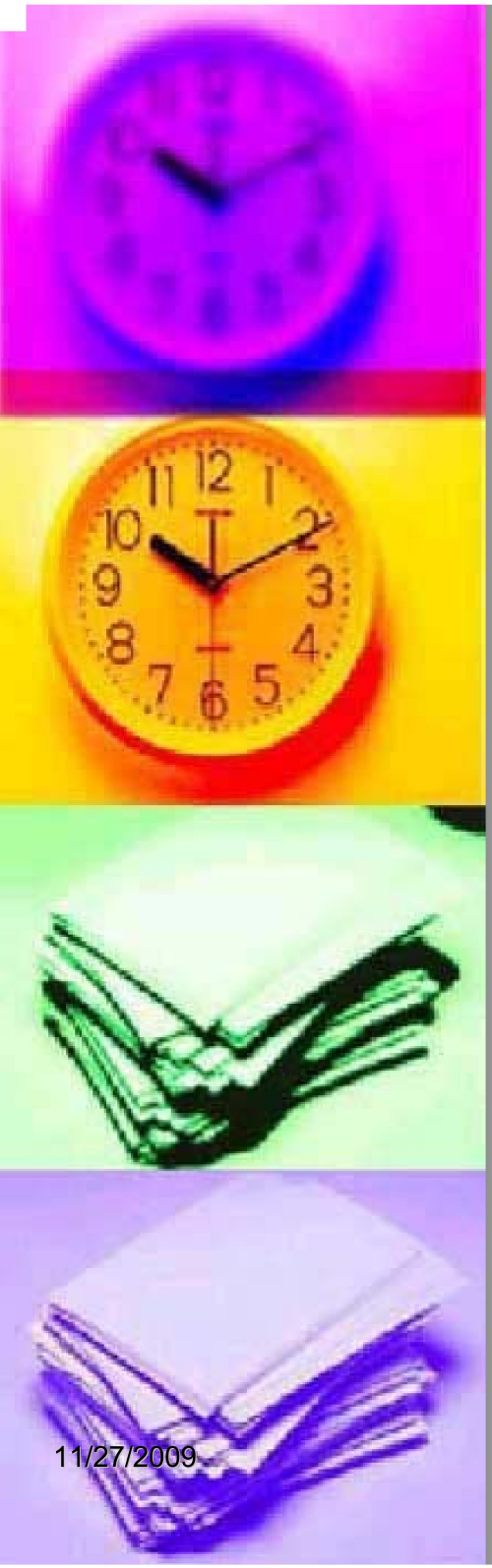


Conflict Serializability (Cont.)

- Schedule1 below can be transformed into Schedule5 , a serial schedule where T_2 follows T_1, by series of swaps of non-conflicting instructions.

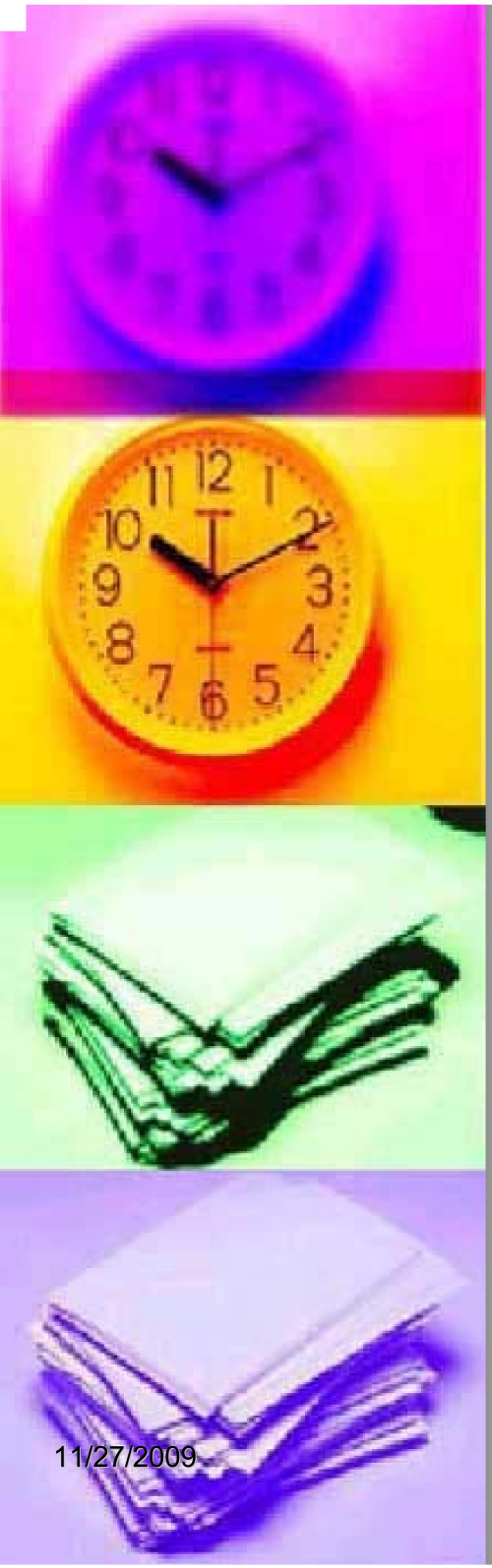
Therefore Schedule1 is conflict serializable.

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



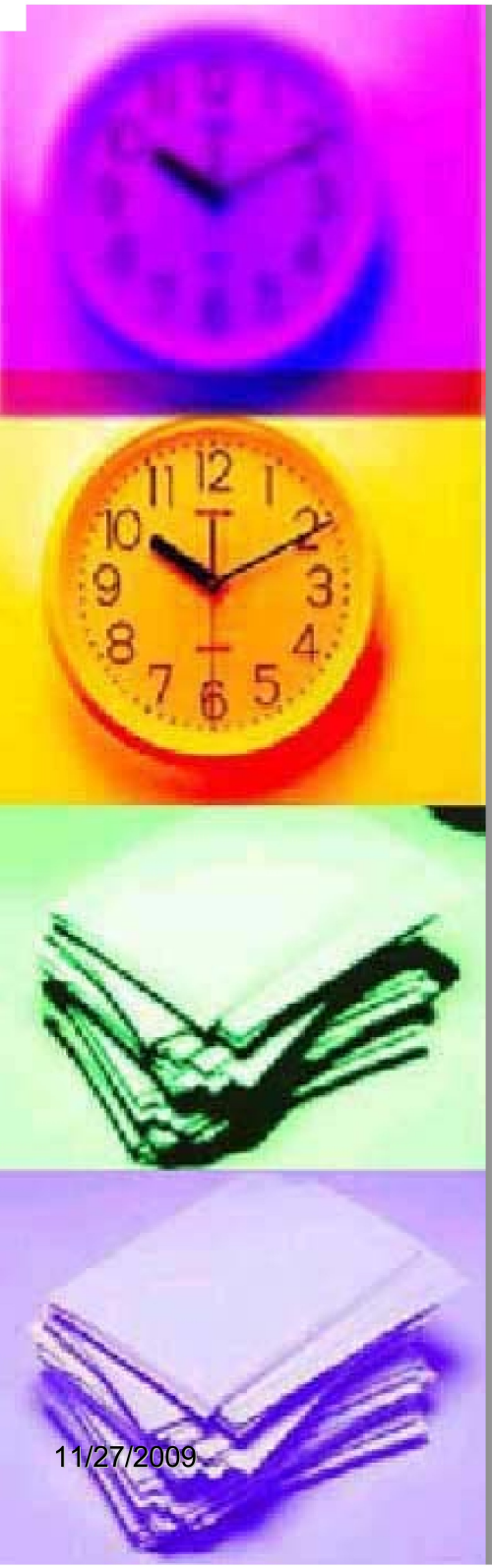
Schedule1

T(1)	T(2)
READ(A)	
WRITE(A)	
	READ(A)
	WRITE(A)
READ(B)	
WRITE(B)	
	READ(B)
	WRITE(B)



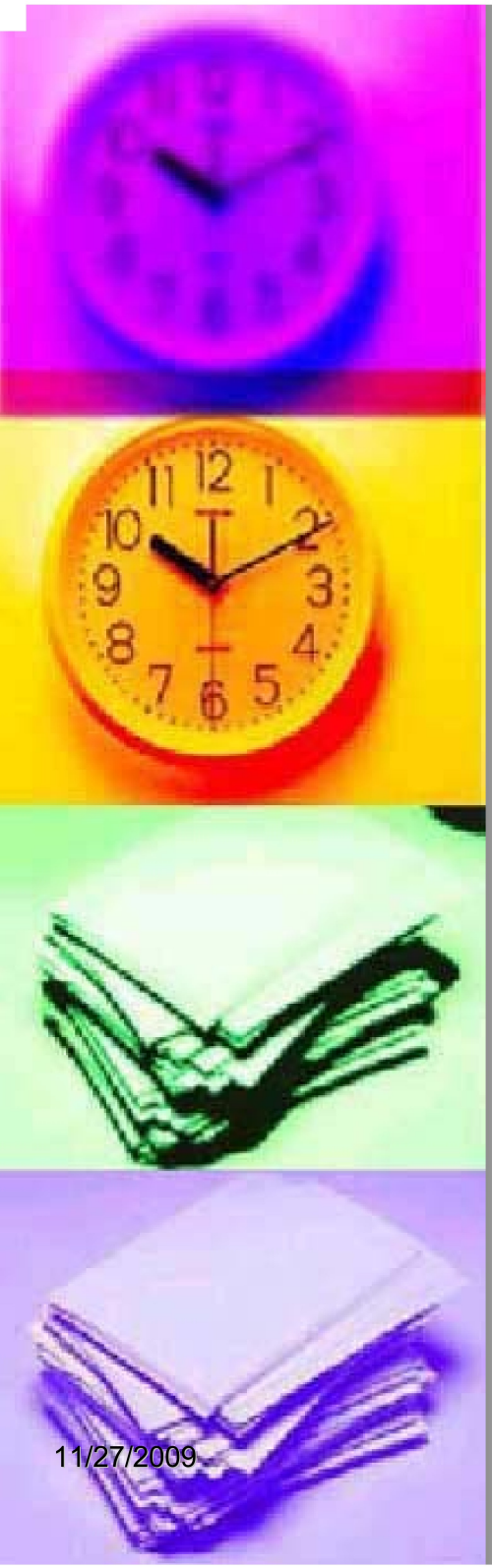
Schedule1

T(1)	T(2)
READ(A)	
WRITE(A)	
	READ(A)
	WRITE(A)
READ(B)	
WRITE(B)	
	READ(B)
	WRITE(B)



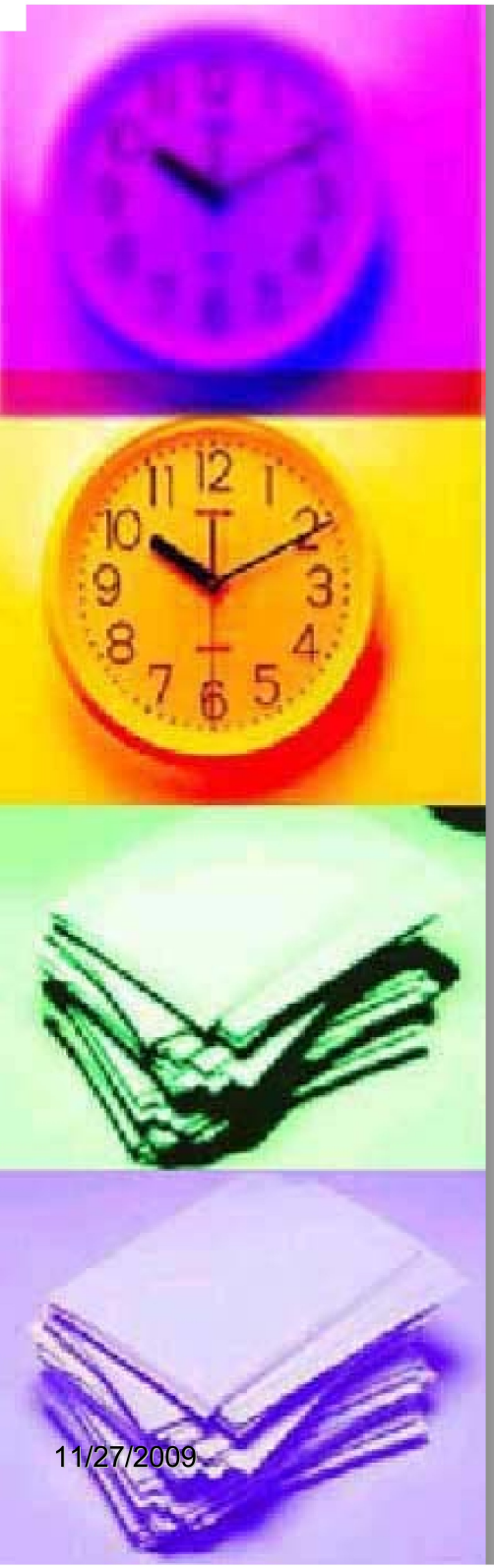
Schedule1

T(1)	T(2)
READ(A)	
WRITE(A)	
	READ(A)
	WRITE(A)
READ(B)	
WRITE(B)	
	READ(B)
	WRITE(B)



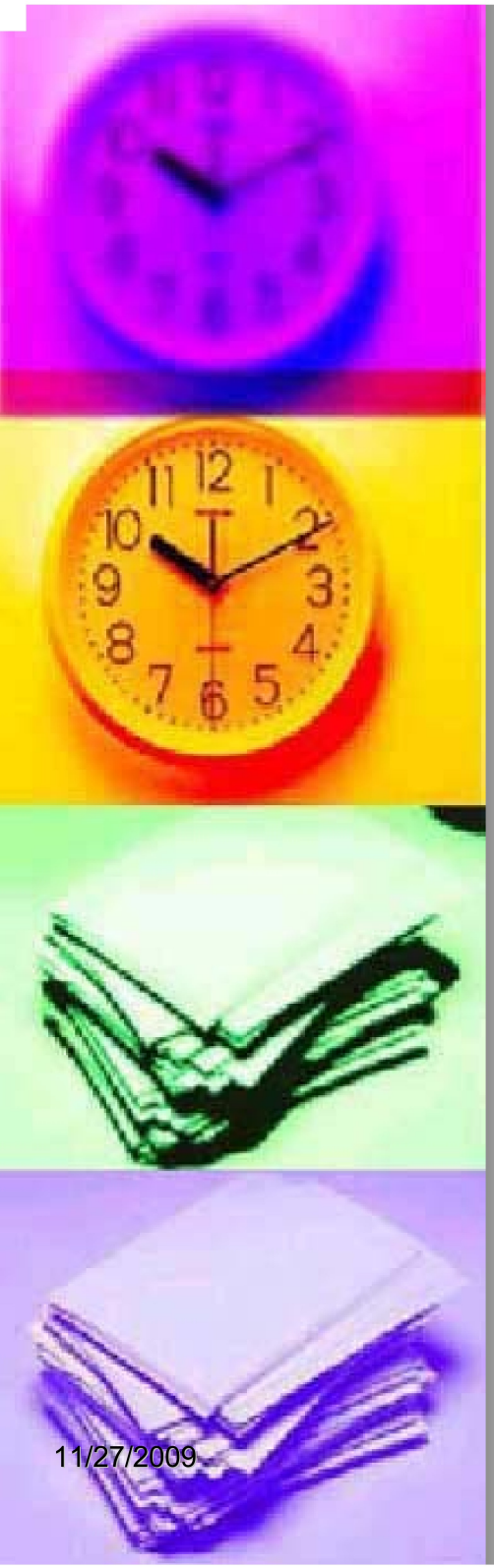
Schedule1

T(1)	T(2)
READ(A)	
WRITE(A)	
	READ(A)
	WRITE(A)
READ(B)	
WRITE(B)	
	READ(B)
	WRITE(B)



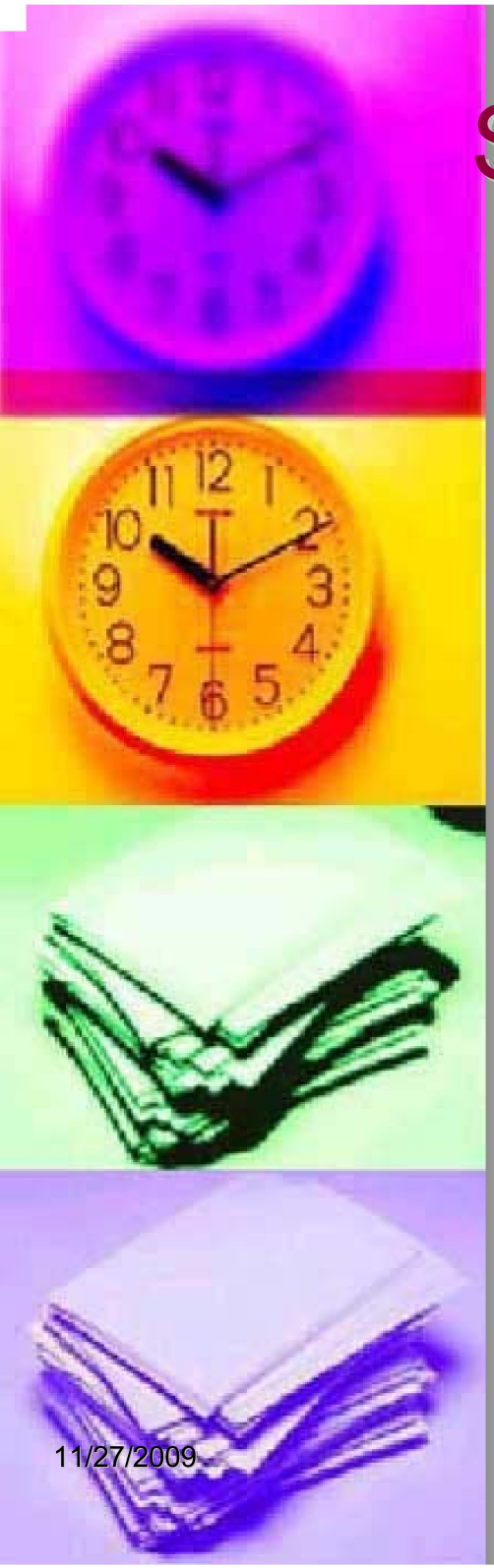
Schedule1:
Blue – NonConflicting,
Red - Conflicting

T(1)	T(2)
READ(A)	
WRITE(A)	
	READ(A)
	WRITE(A)
READ(B)	
WRITE(B)	
	READ(B)
	WRITE(B)



Schedule1:
Black- NonConflicting(Not Swapped)
Red - Conflicting

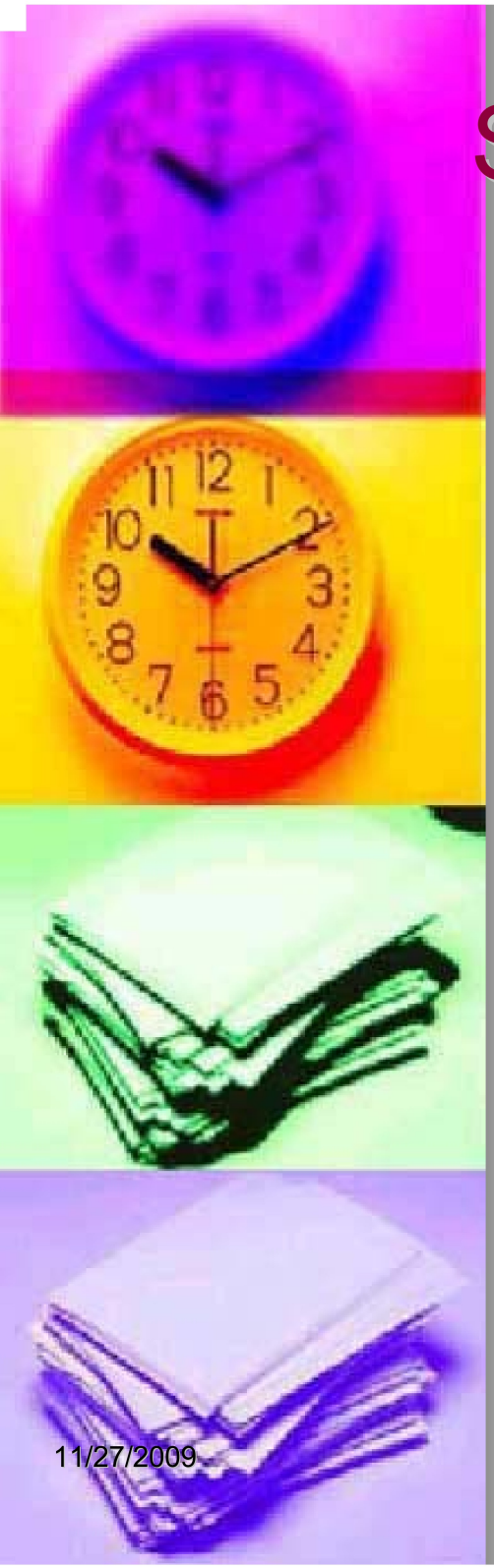
T(1)	T(2)
READ(A)	
WRITE(A)	
	READ(A)
	WRITE(A)
READ(B)	
WRITE(B)	
	READ(B)
	WRITE(B)



Schedule 2:

Swapping of T1(READ(B)) and T2(WRITE(A))

T(1)	T(2)
READ(A)	
WRITE(A)	
	READ(A)
READ(B)	
	WRITE(A)
WRITE(B)	
	READ(B)
	WRITE(B)



Schedule 2:

Swapping of T1(READ(B)) and T2(WRITE(A))

T(1)	T(2)
READ(A)	
WRITE(A)	
	READ(A)
READ(B)	
	WRITE(A)
WRITE(B)	
	READ(B)
	WRITE(B)

Schedule 3

Swapping of T1(READ(B)) and T2(READ(A))

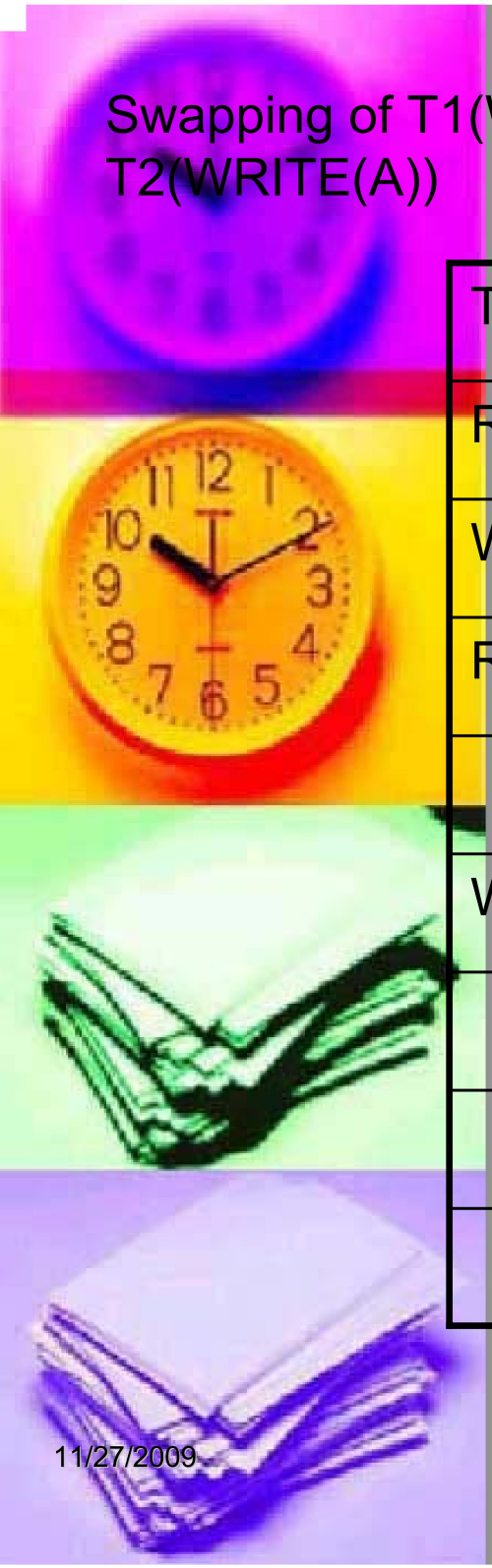
T(1)	T(2)
READ(A)	
WRITE(A)	
READ(B)	
	READ(A)
	WRITE(A)
WRITE(B)	
	READ(B)
	WRITE(B)

OR

Swapping of T1(WRITE(B)) and T2(WRITE(A))

T(1)	T(2)
READ(A)	
WRITE(A)	
	READ(A)
READ(B)	
WRITE(B)	
	WRITE(A)
	READ(B)
	WRITE(B)

Any path will lead to same situation in next diagram



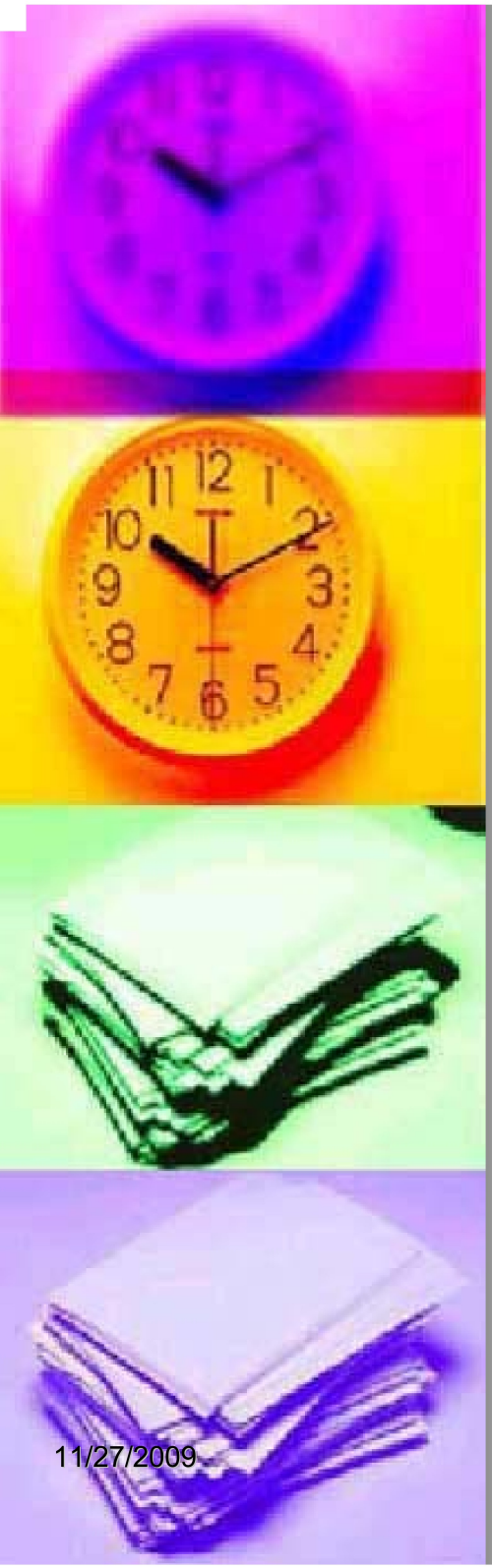
Swapping of T1(WRITE(B)) and T2(WRITE(A))

OR

Swapping of T1(READ(B)) and T2(READ(A))

Schedule 4

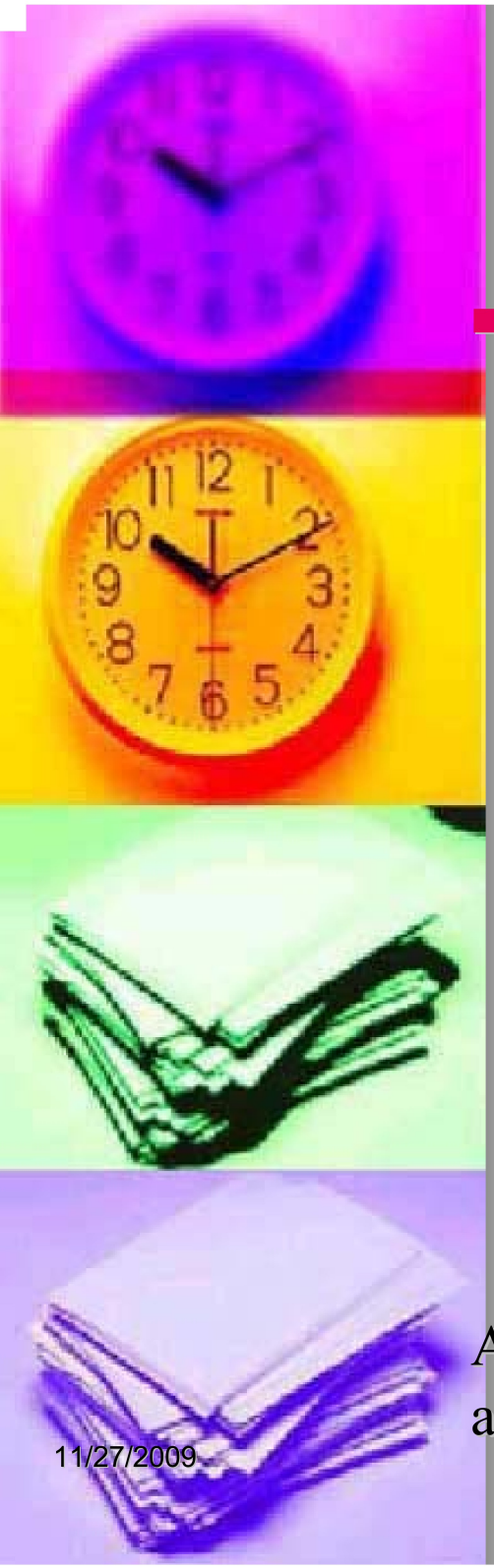
T(1)	T(2)
READ(A)	
WRITE(A)	
READ(B)	
	READ(A)
WRITE(B)	
	WRITE(A)
	READ(B)
	WRITE(B)



Schedule 5 :

Swapping of T1(WRITE(B)) and T2(READ(A))

T(1)	T(2)
READ(A)	
WRITE(A)	
READ(B)	
WRITE(B)	
	READ(A)
	WRITE(A)
	READ(B)
	WRITE(B)

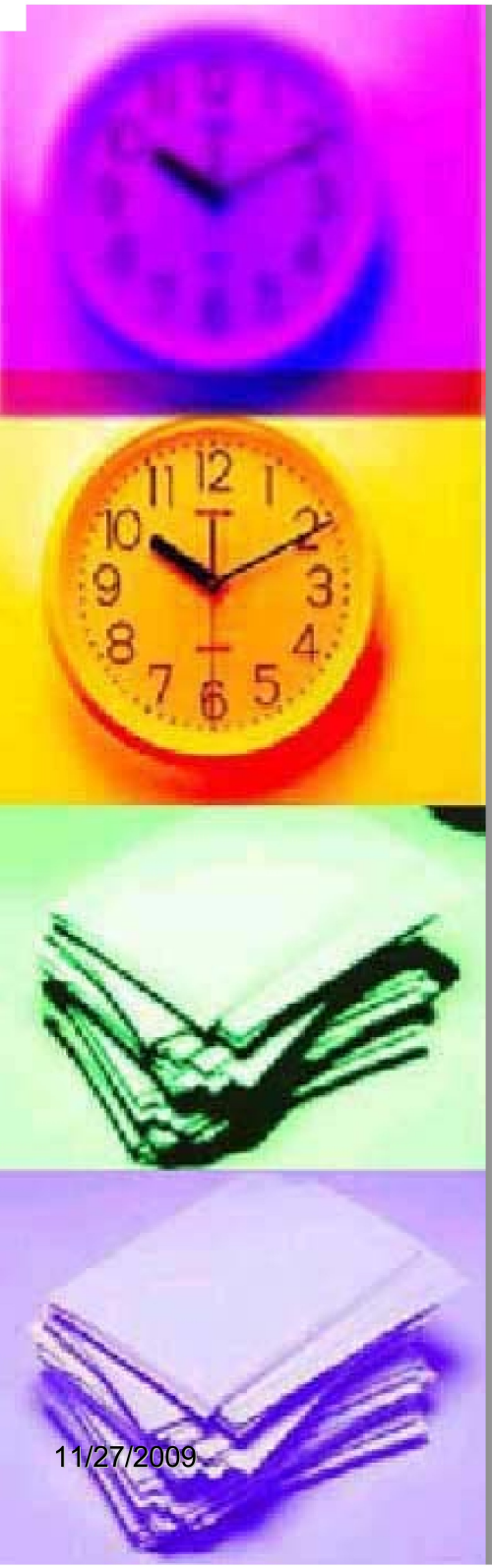


View Serializability

- 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:

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

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

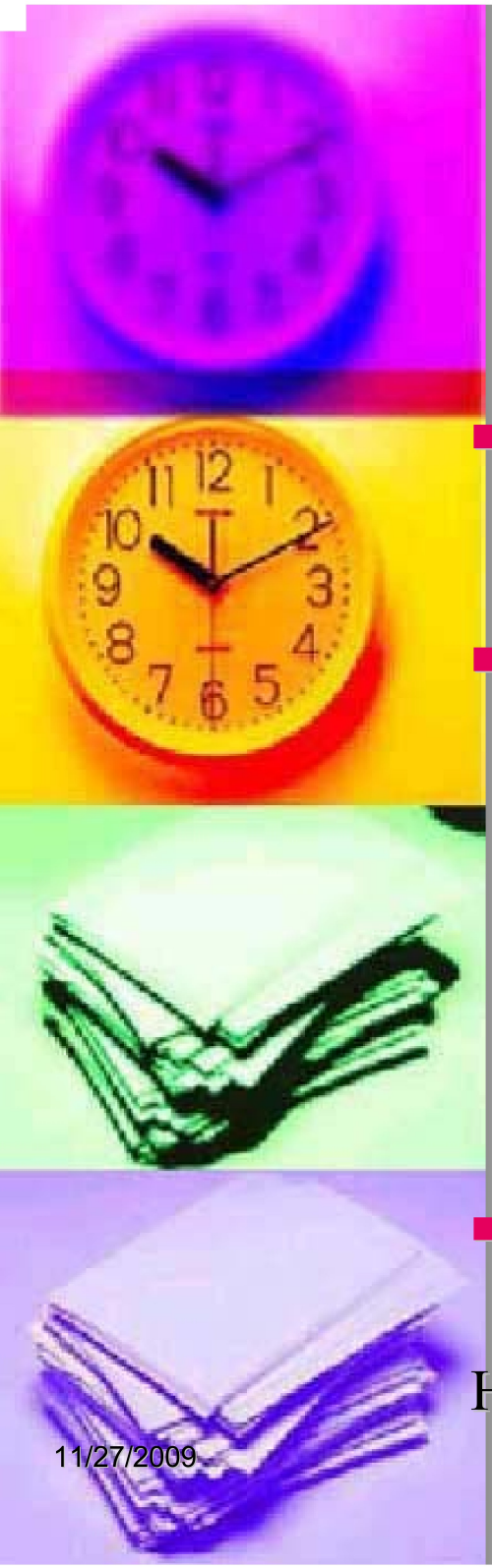


View Serializability (Cont.)

- A schedule S is **view serializable** it is view equivalent to a seria schedule.
- **Every conflict serializable schedule is also view serializable.**
- The following schedule is view-serializable
- but *not* conflict serializable.

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

- Every view serializable schedule that is not conflict serializable has **blind writes.**



Recoverability

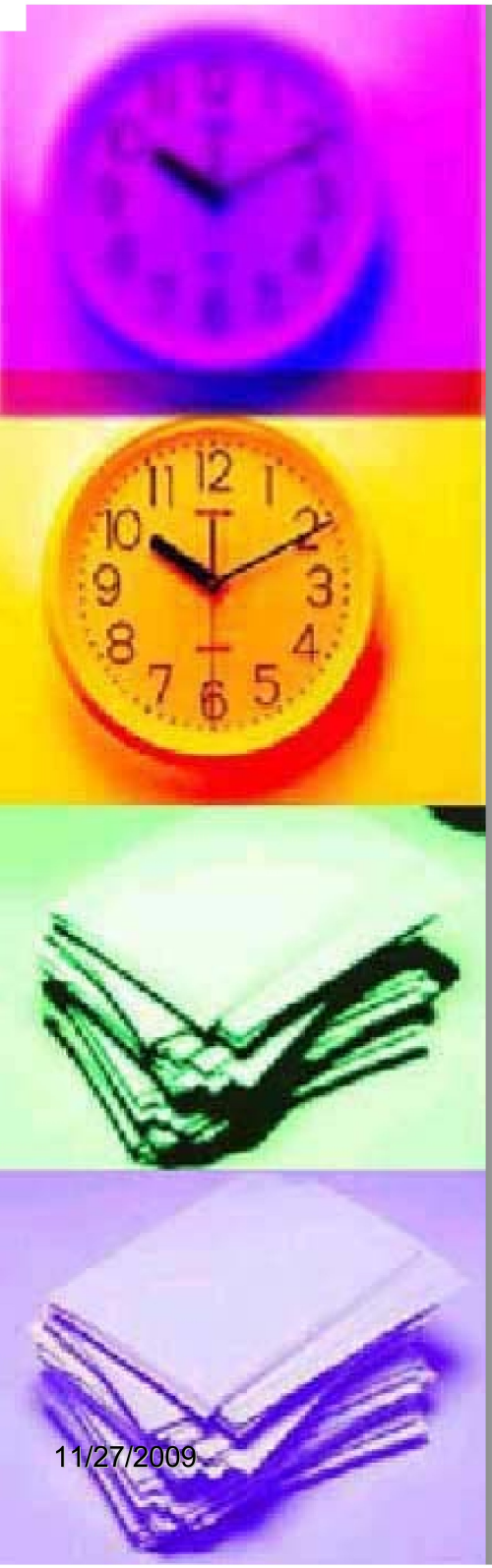
Need to address the effect of transaction failures on concurrently running transactions.

- **Recoverable schedule** — if a transaction T_j reads a data items previously written by a transaction T_i, the commit operation of T_i appears before the commit operation of T_j.
- The following schedule is not recoverable if T_9 commits immediately after the read

T_8	T_9
read(A)	
write(A)	
	read(A)
read(B)	

- If T_8 should abort, T_9 would have read (and possibly shown to the user) an inconsistent database state.

Hence database must ensure that schedules are recoverable.



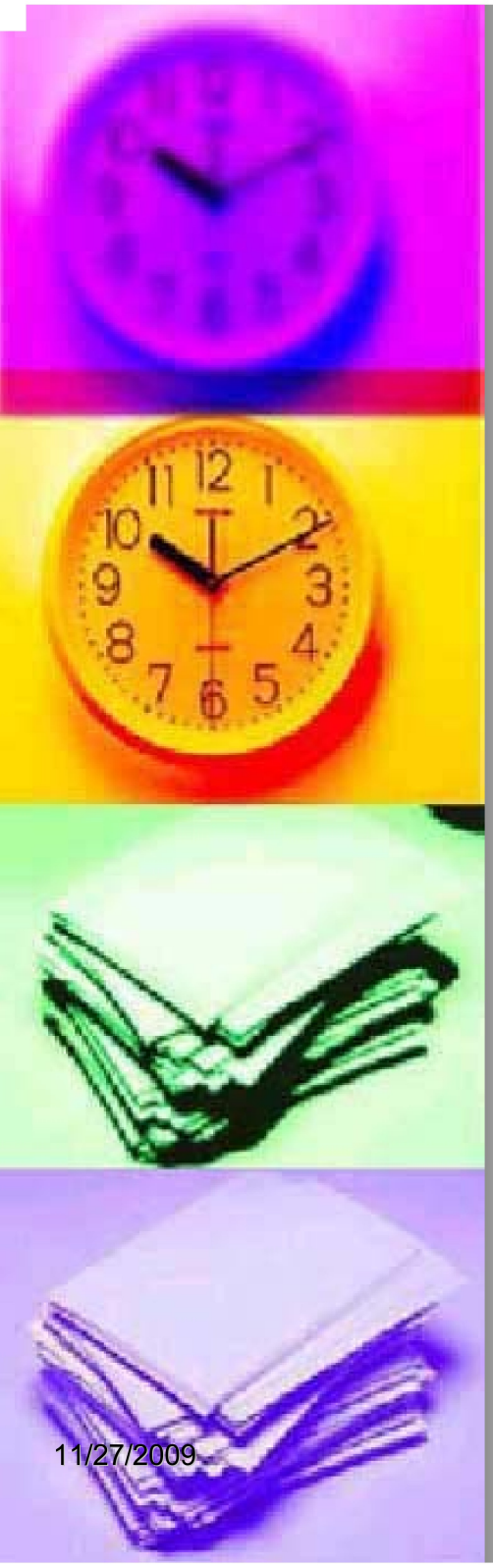
Recoverability (Cont.)

- **Cascading rollback** – a single transaction failure leads to a series of transaction rollbacks. Consider the following schedule where none of the transactions has yet committed (so the schedule is recoverable)

T_{10}	T_{11}	T_{12}
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)

If T_{10} fails, T_{11} and T_{12} must also be rolled back.

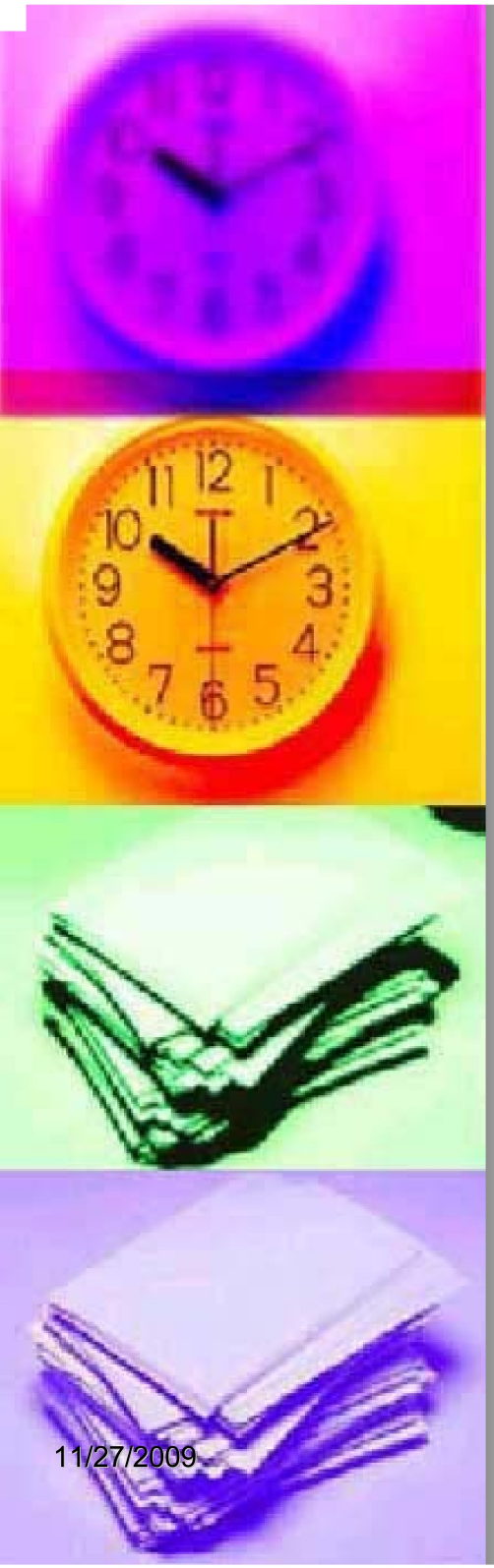
- Can lead to the undoing of a significant amount of work



Recoverability (Cont.)

- **Cascadeless schedules** — cascading rollbacks cannot occur; for each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the read operation of T_j.

- Every cascadeless schedule is also recoverable



Testing for Serializability

- Consider some schedule of a set of transactions $T_1, T_2, ..., T_n$
- **Precedence graph** — a direct graph where the vertices are the transactions (names).
- We draw an arc from T_i to T_j if the two transaction conflict, and T_i accessed the data item on which the conflict arose earlier.
- We may label the arc by the item that was accessed
- A schedule is conflict serializable if and only if its precedence graph is acyclic.