DBMS UNIT V

Serializability
and
Recoverability

Transactions

- A sequence of operations performed together.
  - Insert, Update, Delete, Select
- Generalized as
  - Read Operation on an Object (table)
  - Write Operation on a Object (table)
- The Reads and Writes are on a database object
  - Represented as:
    - $R_T(O)$: Read an object O(Table) (T is the transaction)
    - $W_T(O)$: Read an object O(Table) (T is the transaction)
    - The transaction is concluded by Commit or Abort(Rollback)

Transaction Schedule

- A Schedule is a list of actions like
  - Read, Write, Commit, Abort, Shared Lock eXclusive Lock for a transaction
- The list of actions appear in the same sequence as in the transaction.

T1	T2
R(A)	
W(A)	
	R(B)
	W(B)
R(C)	
W(C)	

Transaction Schedule

- A Transaction Schedule is a sequences of operations that specify the order in which operations of concurrent transactions are executed.
  - A schedule for a set of transactions must consist of all operations of those transactions.
  - It must preserve the order in which the operations appear in each transaction.
  - A transaction that successfully completes its execution will have a commit operations as the last operation. If not stated, it is assumed to commit.
  - A transaction that fails to successfully complete its execution will have an abort operation as the last operation.

Transaction Schedule

- In this transaction schedule (and the previous example), the operations of the two transactions are interleaved.

T1	T2
R(A)	
W(A)	
	R(C)
	W(C)
R(B)	
W(B)	
	R(D)
	W(D)
	Commit
Commit	

Anomalies due to Interleaved Execution - Review

- Two transactions reading the database(Read-Read) are not affected by interleaving.
- But, The following operations result in anomalies **when the same object** is involved:
  - Write-Read (write followed by a read) or Dirty Read
  - Read-Write (read followed by a write) or Unrepeatable Read
  - Write-Write (write followed by a write) or Lost Update

Definitions

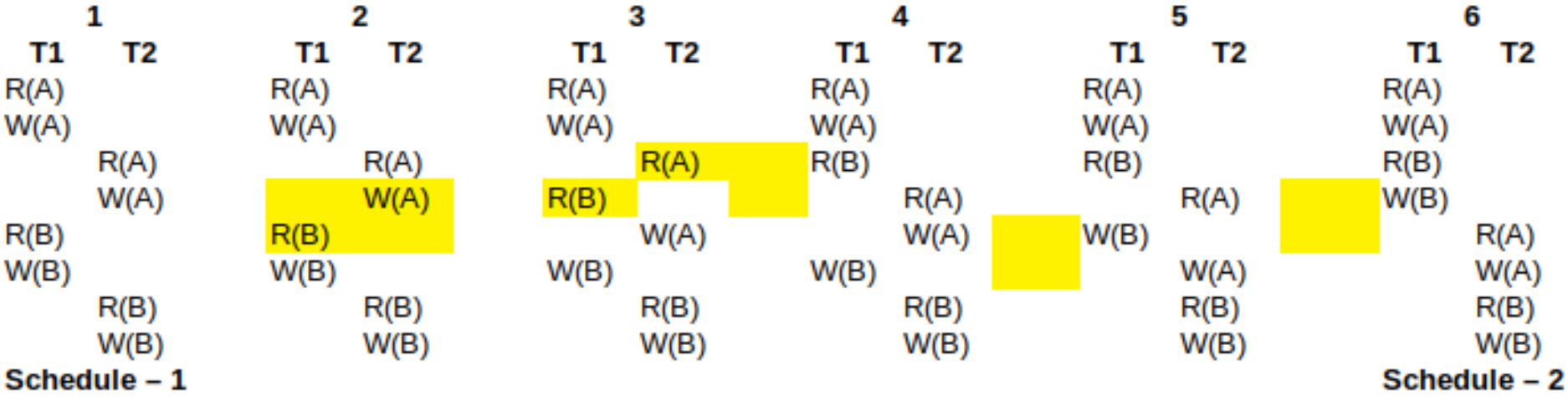
- **Schedule**: A sequence of actions of a set of transaction. A possible execution sequence.
- **Complete Schedule**: A schedule that contains an **Abort or Commit** for every transaction in the schedule.
- **Serial Schedule**: If the **transactions are not interleaved**, that is transactions are executed from start to finish, one by one, one after another.
- **Non-serial Schedule**: Non-serial schedules mean that transactions are interleaved.
- **Serializable schedule**: A serializable schedule over a set of S committed transactions is a schedule whose effect on an consistent database instance guaranteed to be identical to that of some complete serial schedule over S.
- **Conflict Serializable:** A schedule is conflict serializable if it is equivalent to some serial schedule.
- **Recoverable Schedule**: Transactions commit only after all the transactions whose changes they read commit.
- **Unrecoverable Schedule**: Has read, used and committed data from an aborted transaction.
- **Strict Schedule:** A Strict schedule is one where a transaction is not allowed to read or write any uncommitted data.

Complete and Serial Schedule

- Complete (Has a commit or Abort)
  - $R_1(A)$, $W_2(A)$, $R_2(A)$, $W_2(A)$, $Commit_1$, $Abort_2$
- Not Complete (no Commit or Abort)
  - $R_1(A)$, $W_2(A)$, $R_2(A)$, $W_2(A)$
- Serial Schedule (Transactions-one after another)
  - $R_1(A)$, $W_1(A)$, $Commit_1$, $R_2(A)$, $W_2(A)$, $Abort_2$
- Nonserial (Interleaved reads and writes)

Conflict Equivalent

If we can swap non-conflicting operations and get a new schedule, the two schedules are conflict equivalent. Example:



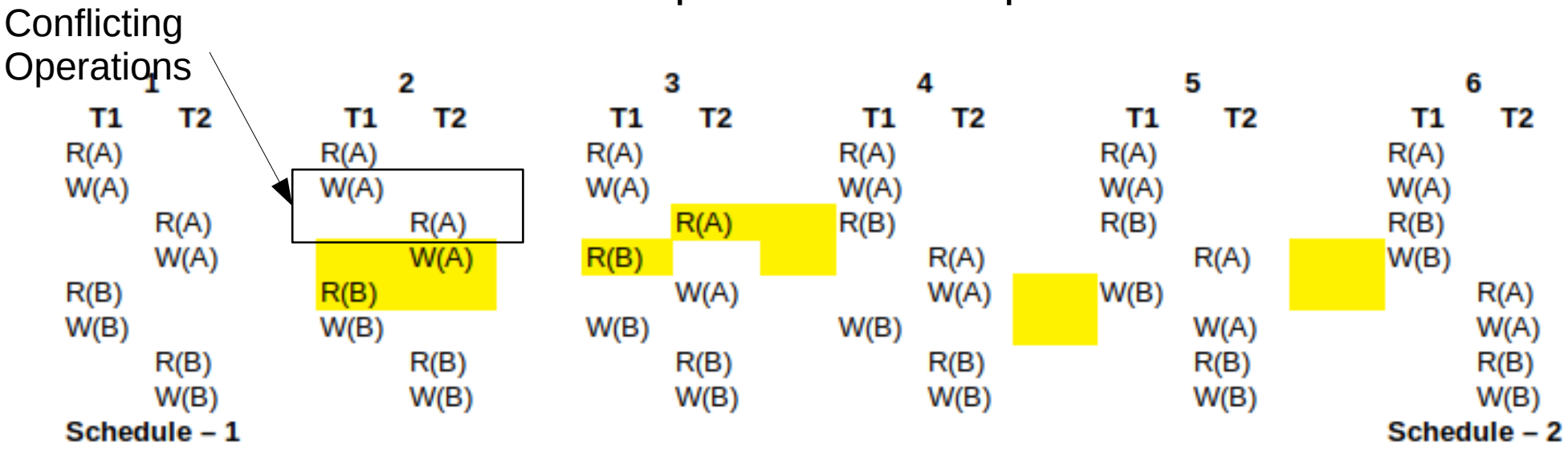
Column 1 is the given Schedule

In column 2, observe that $R_1(B)$ and $W_2(A)$ are not in conflict and hence we can swap to get the schedule in column 3.

Continue this step: in column 3, swap $R_1(B)$ and $R_2(A)$ which are not in conflict and hence get schedule in column 4. Continue to get a schedule that is a serial schedule- T1 after T2. (col 6)

Conflict Equivalent

If we can swap non-conflicting operations and get a new schedule, the two schedules are conflict equivalent. Example:



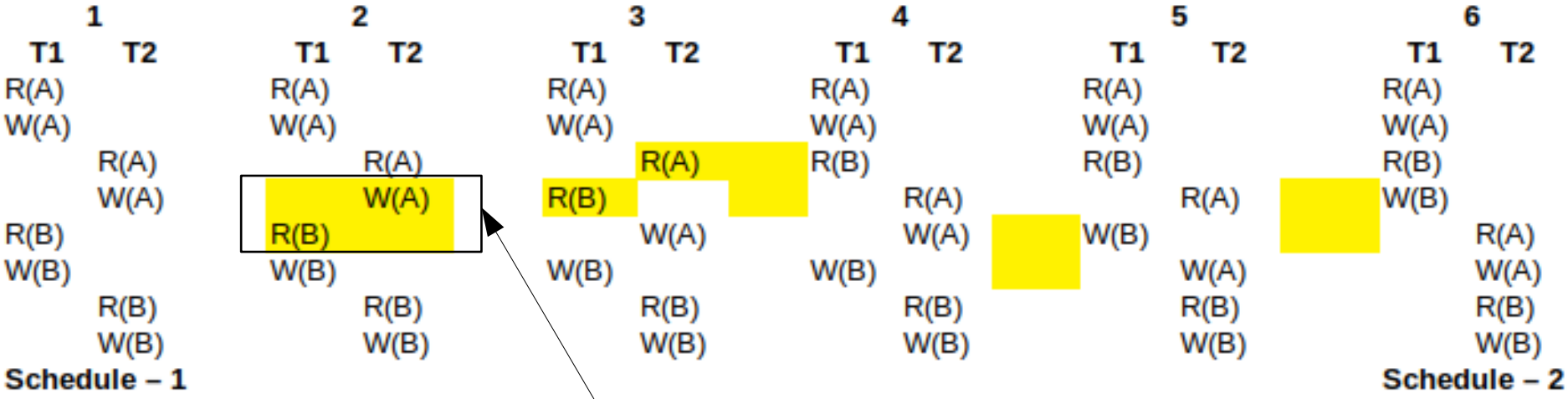
Column 1 is the given Schedule

In column 2, observe that $R_1(B)$ and $W_2(A)$ are not in conflict and hence we can swap to get the schedule in column 3.

Continue this step: in column 3, swap $R_1(B)$ and $R_2(A)$ which are not in conflict and hence get schedule in column 4. Continue to get a schedule that is a serial schedule- T1 after T2. (col 6)

Conflict Equivalent

If we can swap non-conflicting operations and get a new schedule, the two schedules are conflict equivalent. Example:



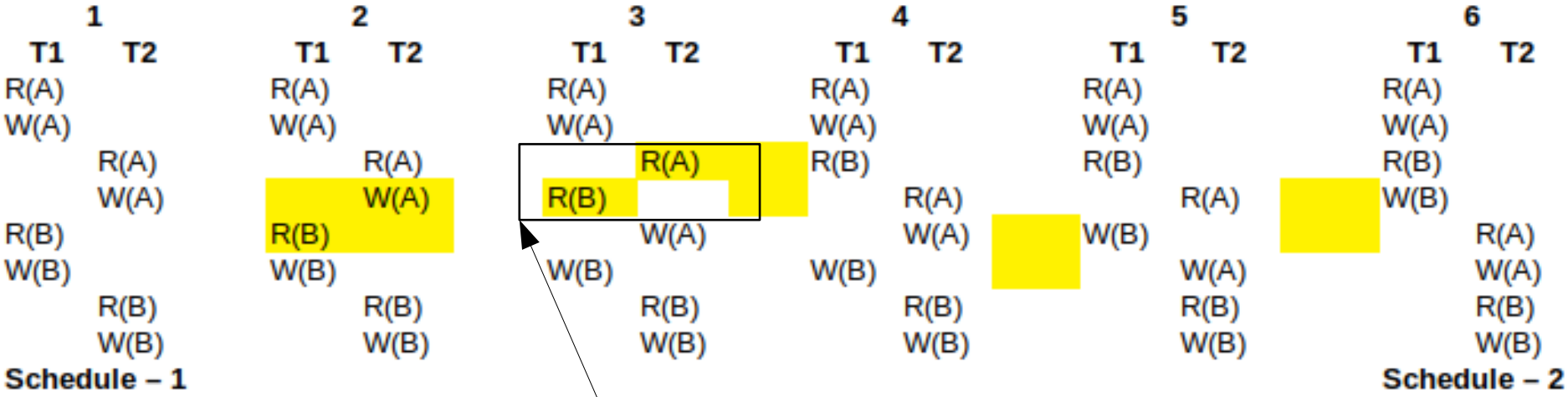
Column 1 is the given Schedule

In column 2, observe that $R_1(B)$ and $W_2(A)$ are not in conflict and hence we can swap to get the schedule in column 3.

Continue this step: in column 3, swap $R_1(B)$ and $R_2(A)$ which are not in conflict and hence get schedule in column 4. Continue to get a schedule that is a serial schedule- T1 after T2. (col 6)

Conflict Equivalent

If we can swap non-conflicting operations and get a new schedule, the two schedules are conflict equivalent. Example:



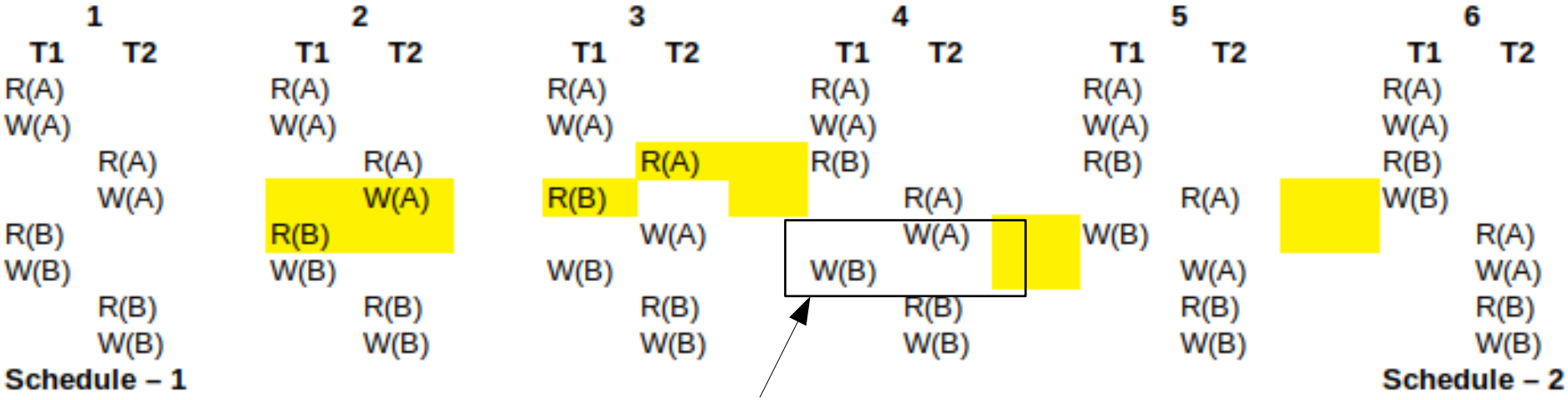
Column 1 is the given Schedule

In column 2, observe that $R_1(B)$ and $W_2(A)$ are not in conflict and hence we can swap to get the schedule in column 3.

Continue this step: in column 3, swap $R_1(B)$ and $R_2(A)$ which are not in conflict and hence get schedule in column 4. Continue to get a schedule that is a serial schedule- T1 after T2. (col 6)

Conflict Equivalent

If we can swap non-conflicting operations and get a new schedule, the two schedules are conflict equivalent. Example:



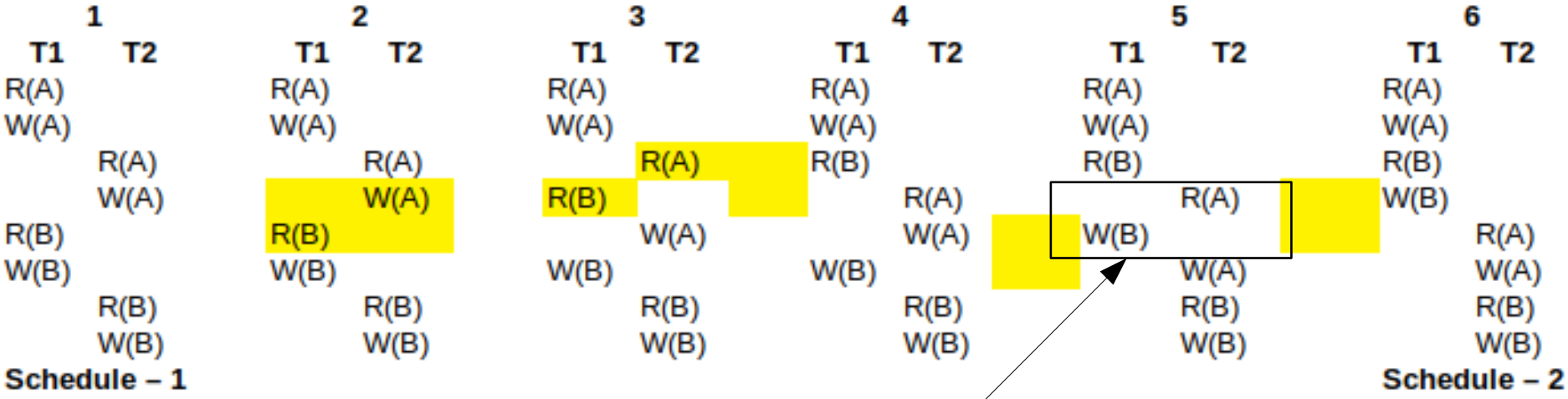
Column 1 is the given Schedule

In column 2, observe that $R_1(B)$ and $W_2(A)$ are not in conflict and hence we can swap to get the schedule in column 3.

Continue this step: in column 3, swap $R_1(B)$ and $R_2(A)$ which are not in conflict and hence get schedule in column 4. Continue to get a schedule that is a serial schedule- T1 after T2. (col 6)

Conflict Equivalent

If we can swap non-conflicting operations and get a new schedule, the two schedules are conflict equivalent. Example:



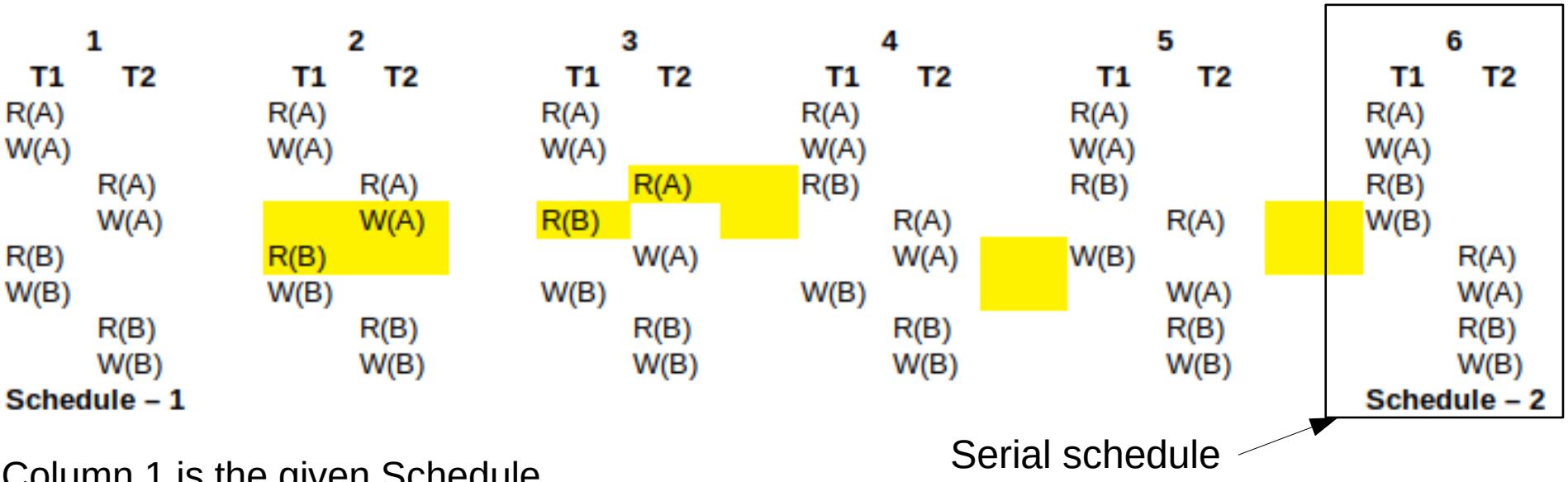
Column 1 is the given Schedule

In column 2, observe that $R_1(B)$ and $W_2(A)$ are not in conflict and hence we can swap to get the schedule in column 3.

Continue this step: in column 3, swap $R_1(B)$ and $R_2(A)$ which are not in conflict and hence get schedule in column 4. Continue to get a schedule that is a serial schedule- T1 after T2. (col 6)

Conflict Equivalent

If we can swap non-conflicting operations and get a new schedule, the two schedules are conflict equivalent. Example:



Column 1 is the given Schedule

In column 2, observe that $R_1(B)$ and $W_2(A)$ are not in conflict and hence we can swap to get the schedule in column 3.

Continue this step: in column 3, swap $R_1(B)$ and $R_2(A)$ which are not in conflict and hence get schedule in column 4. Continue to get a schedule that is a serial schedule- T1 after T2. (col 6)

Conflict Serializable

- Conflict: Two actions conflict if they **operate on the same object and one of them is a Write.**
- Conflict equivalent
  - Two schedules are said to be **conflict equivalent** if the order of two conflicting operations is the same in both the schedules.
  - Alternate definition: Two Schedules are conflict equivalent if they involve the actions of the same transactions and they order every pair of conflicting actions of two committed transactions in the same way.
- Conflict serializable
  - A schedule is conflict seriaizable, if it is conflict equivalent to some serial schedule.
  - The schedule in the previous slide is conflict serializable.
  - Not all schedules are conflict serializable

Recoverable Schedule

- Consider two transactionsT_1 and T_2, If T_2 reads an object written by T_1 and T_2 commits first, and T_1 is aborted.
  - $R_1(X)$, $W_1(X)$, $R_2(X)$, $R_1(Y)$, $W_2(X)$, $Commit_2$, $Abort_1$
- If there is a system crash, since T_2 has read uncommited data from T_1, and T_1 was aborted (or incomplete due to a crash) we have to Abort T_2.
- This is called cascading aborts.
- Such a schedule is not recoverable.
- ***In a recoverable schedule, no committed transaction ever needs to be rolled back.***
- A strict schedule avoids cascading aborts.

Testing for Serializability

- How can we say if a schedule is conflict serializable or not?
- Using precedence graph.
- A topological sort on the precedence graph gives the serial schedule.
- A cycle in the graph indicates that the schedule does not have a conflict serializable schedule

Precedence Graph

- Serializability Graph or Precedence Graph
  - A node for each committed transaction in S
  - An arc from T_i to T_j if an action of T_i precedes and conflicts (RW, WR, WW) with one of T_j's actions
- A schedule S is conflict serializable if and only if its precedence graph is acyclic
- Strict 2PL ensures that the precedence graph for any schedule that it allows is acyclic.

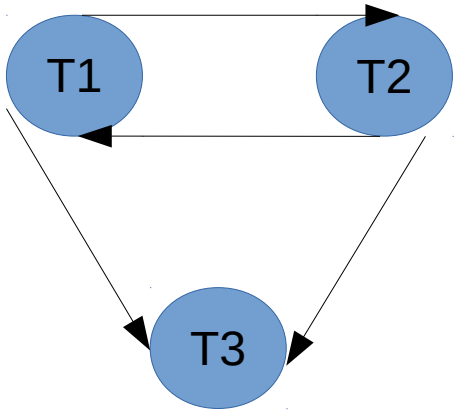
Precedence Graph Examples

T1	T2
R(A)	
W(A)	
R(B)	
W(B)	
Commit	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit

T3	T4
R(A)	
	R(A)
	R(B)
	W(B)
	Commit
R(C)	
W(C)	
Commit	

Precedence Graph Examples

T1	T2	T3
R(A)		
	W(A)	
W(A)		
		W(A)
Commit		
	Commit	
		Commit



Arc	Conflict
T1 to T2	Read-Write
T2 to T1	Write-Write
T2 to T3	Write-Write
T1 to T3	Write-Write

Using Precedence Graph

- The equivalent serial schedule is:
  - T1, T2 for example 1
  - T1, T2 or T2, T1 for example 2
  - T1, T2, T3 for example 3
- The examples 1 and 2 are conflict serializable
- Example 3 is not conflict serializable
  - Because the ***precedence graph has a loop***
  - The serial schedule T1, T2, T3 is not conflict-equivalent to the given schedule
  - ***All schedules are serializable but are not conflict serializable.***

Review Question

- Given a schedule: $R_1(Y)$, $R_2(X)$, $R_2(Y)$, $W_2(X)$, $R_1(X)$, $Commit_1$, $Commit_2$
  - Is the Schedule complete?
  - What is the equivalent serial schedule?
  - Is there a conflict, if yes what kind of conflict is it?
  - Is the schedule conflict serializable?
    - If yes, give the conflict serializable schedule.
  - Is it a strict schedule?

Solution

T1	T2
$R_1(Y)$	
	$R_2(X)$
	$R_2(Y)$
	$W_2(X)$
$R_1(X)$	
$Commit_1$	
	$Commit_2$

- Yes. The schedule is complete. Both transactions have commit.
- T2, T1 (try using the swap method or the precedence graph method)
- Write-Read or Dirty Read
- From the precedence graph, there is no cycle, hence yes it is conflict serializable. The schedule is T2, T1
- No. Not strict because $R_1(X)$ is following $W_2(X)$ but T_2 is not committed yet.

More Examples

- S_1 : $R_1(X)$, $R_2(X)$, $W_2(X)$, $W_1(X)$, $Commit_2$, $Commit_1$
  - Conflict Serializable,
  - Not Strict,
  - Read-Write (unrepeatable read) as well as Write-Write (lost update)

Questions