DATABASE SYSTEMS

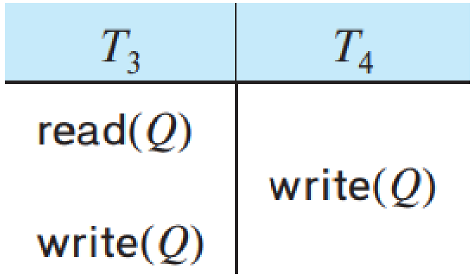
CS – 355/CE – 373

Instructor: Maria N. Samad

October 28th, 2024

CONFLICT SERIALIZABILITY

- A schedule *S* is **conflict serializable** if it is conflict equivalent to a serial schedule.
- Consider the following schedule:



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

- This schedule is not conflict serializable as it is not equivalent to the serial schedule <T_3, T_4> or the serial schedule <T_4, T_3>.

CONFLICT SERIALIZABILITY

- Example: Given three schedules as follows:
  - S1: r1(X), w1(X), r2(X), w2(X), r1(Y), w1(Y), r2(Y), w2(Y)
  - S2: r1(X), w1(X), r1(Y), r2(X), w2(X), w1(Y), r2(Y), w2(Y)
- Check if the above two schedules are conflict serializable, by using swapping techniques
- Solution: On board

CONFLICT SERIALIZABILITY

- Example: Given two schedules as follows:
  - S1: r1(X), r2(Y), w3(Y), w1(X), w2(Y)
  - S2: r1(X), r2(Y), w1(X), w3(Y), w2(Y)
- Check if the schedules are conflict serializable, by using swapping techniques
- Solution: On board

CONFLICT SERIALIZABILITY

- Example: Given two schedules as follows:
  - S1: r1(X), r2(X), w1(X), w2(X), w3(X)
  - S2: r1(X), w1(X), r2(X), w2(X), w3(X)
- Check if the schedules are conflict serializable, by using swapping techniques
- Solution: On board

CONFLICT SERIALIZABILITY

- Class Activity

CONFLICT SERIALIZABILITY

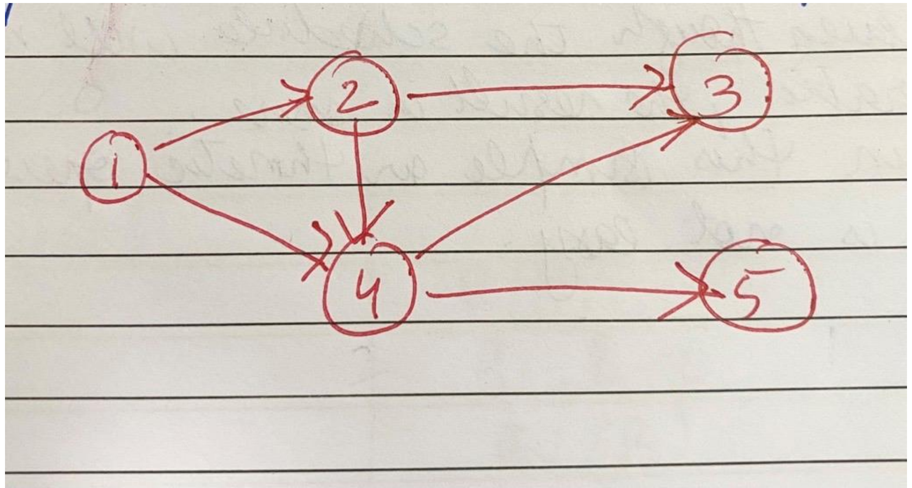
- Class Activity Solution:
  - Conflict Serializable Solution

TOPOLOGICAL SORTING

- Applied to only Directed Acyclic Graphs (DAG)
- Algorithm:
    1. Start with node/vertex having in-degree = 0
    2. Add it to the selected vertices list
    3. Remove it and its associated edges from the graph and update the in-degrees
    4. Repeat steps 2-3 until no nodes/vertices are left

TOPOLOGICAL SORTING

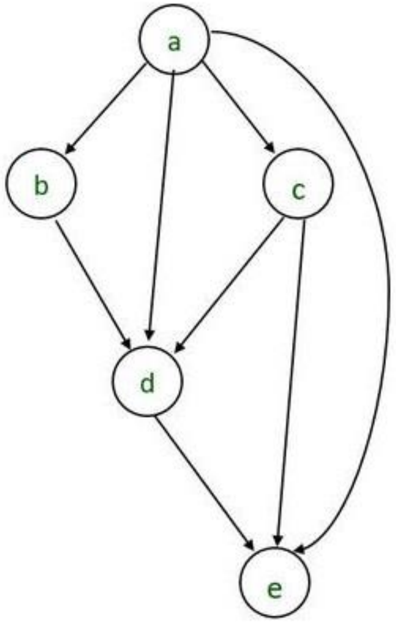
- Example: Given a DAG, find its topological sorting order of the vertices



- Solution: On board

TOPOLOGICAL SORTING

- Example: Given a DAG, find its topological sorting order of the vertices



- Solution: On board

TOPOLOGICAL SORTING

- Class Activity

TOPOLOGICAL SORTING

- Class Activity Solution:
  - Topological Sorting Solution

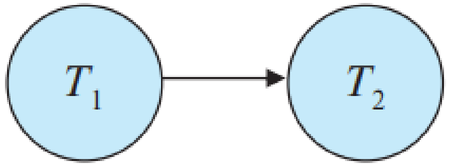
CONFLICT SERIALIZABILITY – PRECEDENCE GRAPHS

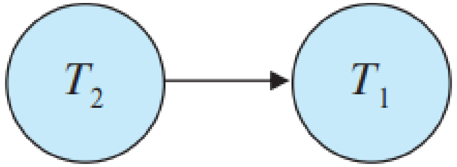
- To find whether a schedule is **conflict serializable**, we use precedence graphs.
- This graph $G = (V, E)$, has V as its set of vertices and E as its set of edges. The set of vertices consists of all the transactions participating in the schedule.
- The set of edges consists of all edges $T_i \rightarrow T_j$ for which one of three conditions holds:
  - T_i executes *write*(Q) before T_j executes *read*(Q)
  - T_i executes *read*(Q) before T_j executes *write*(Q)
  - T_i executes *write*(Q) before T_j executes *write*(Q)
- If an edge $T_i \rightarrow T_j$ exists in the precedence graph, then, in any serial schedule S' equivalent to S, T_i must appear before T_j

CONFLICT SERIALIZABILITY – PRECEDENCE GRAPHS

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

Precedence Graph of the (left) schedule

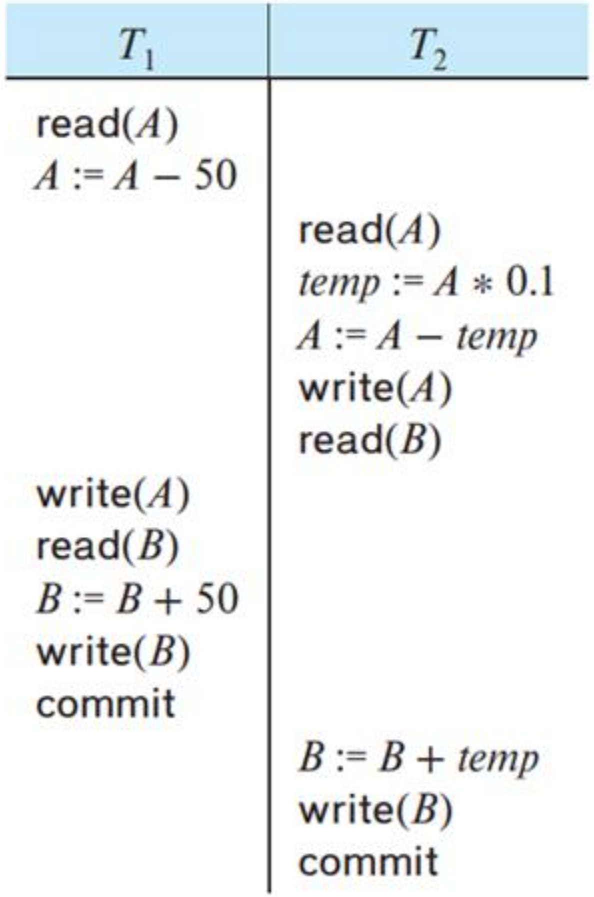




Precedence Graph of the (right) schedule

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

CONFLICT SERIALIZABILITY – PRECEDENCE GRAPHS



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)	
commit	
	$B := B + temp$
	write(B)
	commit

Figure 17.5 Schedule 4—a concurrent schedule resulting in an inconsistent state.

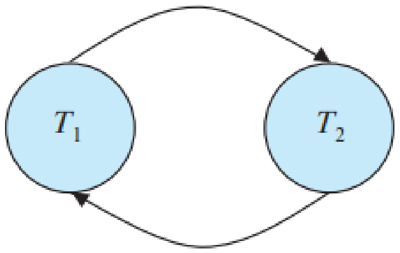


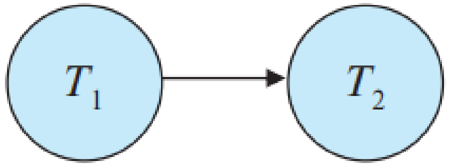
Figure 17.11 Precedence graph for schedule 4.

- The precedence graph for Schedule 4 (left) is shown (above).

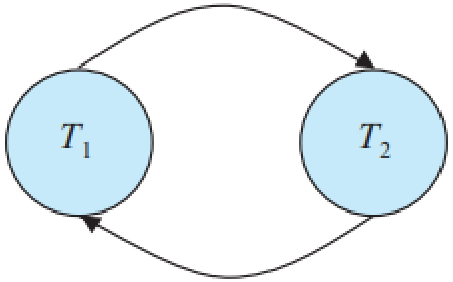
- Observe that it contains a cycle ($T_1 \rightarrow T_2 \rightarrow T_1$).

CONFLICT SERIALIZABILITY – PRECEDENCE GRAPHS

- If the precedence graph for *S* has a cycle, then schedule *S* is not conflict serializable.
- If the graph contains no cycles, then the schedule *S* is conflict serializable.
- Therefore, a schedule with the following (→) precedence graph is conflict serializable



- However, a schedule with the following precedence graph (→) is not conflict serializable, as it contains cycles.



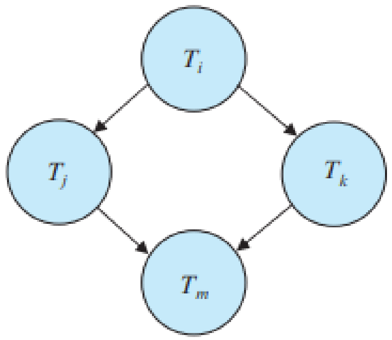
SERIALIZABILITY ORDER

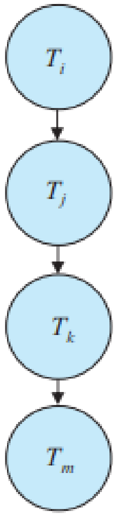
- A serializability order of the transactions can be obtained by finding a linear order consistent with the partial order of the precedence graph.

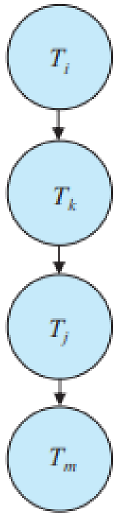
- This process is called **topological sorting**.

- There are, in general, several possible linear orders that can be obtained through a topological sort.

- For example, the graph shown here (top-right) has the two acceptable linear orderings as shown below the graph (right).







PRECEDENCE GRAPH

- Example: Given a schedules as follows:
  - S1: r1(X), w1(X), r2(X), w2(X), r1(Y), w1(Y), r2(Y), w2(Y)
- Check if the schedule is conflict serializable by using precedence graph. If the schedule is serializable, state its equivalent serial schedule
- Solution: On board

PRECEDENCE GRAPH

- Example: Given a schedules as follows:
  - S1: r1(A), r3(B), r3(A), r2(B), r2(C), w3(B), w2(C), r1(C), w1(A), w1(C)
- Check if the schedule is conflict serializable by using precedence graph. If the schedule is serializable, state its equivalent serial schedule
- Solution: On board

PRECEDENCE GRAPH

- Class Activity

PRECEDENCE GRAPH

- Class Activity Solution:
  - Precedence Graph Solution