Database Management Systems

Lecture 9: Concurrency Control

© Spring 2023 – **Dr. Alyaa Hamza**
alyaa,.hamza@sut.edu.eg

Schedules

- **Schedule** – a sequences of instructions that specify the chronological order in which instructions of concurrent transactions are executed
  - A schedule for a set of transactions must consist of all instructions of those transactions
  - Must preserve the order in which the instructions appear in each individual transaction.
- A transaction that successfully completes its execution will have a commit instructions as the last statement
  - By default transaction assumed to execute commit instruction as its last step
- A transaction that fails to successfully complete its execution will have an abort instruction as the last statement

Schedule 1

- Let T_1 transfer $50 from *A* to *B*, and T_2 transfer 10% of the balance from *A* to *B*.
- A serial schedule in which T_1 is followed by T_2 :

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

Schedule 2

- A serial schedule where T_2 is followed by T_1

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	

Schedule 3

- Let T_1 and T_2 be the transactions defined previously. The following schedule is not a serial schedule, but it is *equivalent* to Schedule 1

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

- In Schedules 1, 2 and 3, the sum A + B is preserved.

Schedule 4

- The following concurrent schedule does not preserve the value of $(A + B)$.

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

Serializability

- 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 non-serial schedule.
- Serializable schedule:
  - A schedule S is serializable if it is equivalent to some serial schedule of the same n transactions.
- Result equivalent:
  - Two schedules are called result equivalent if they produce the same final state of the database.

Serializability

- Being serializable is **not** the same as being serial
- Being serializable implies that the schedule is a correct schedule.
  - It will leave the database in a consistent state.
  - The interleaving is appropriate and will result in a state as if the transactions were serially executed, yet will achieve efficiency due to concurrent execution.
- Serializability is hard to check.
  - Interleaving of operations occurs in an operating system through some scheduler
  - Difficult to determine beforehand how the operations in a schedule will be interleaved.

Testing for conflict serializability:

- Looks at only read_Item (X) and write_Item (X) operations
- Constructs a precedence graph (serialization graph) - a graph with directed edges
- An edge is created from Ti to Tj if one of the operations in Ti appears before a conflicting operation in Tj
- The schedule is serializable if and only if the precedence graph has no cycles.

Conflicting Instructions

- 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

- Schedule 3 can be transformed into Schedule 6, a serial schedule where T_2 follows T_1, by series of swaps of non-conflicting instructions. Therefore Schedule 3 is conflict serializable.

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

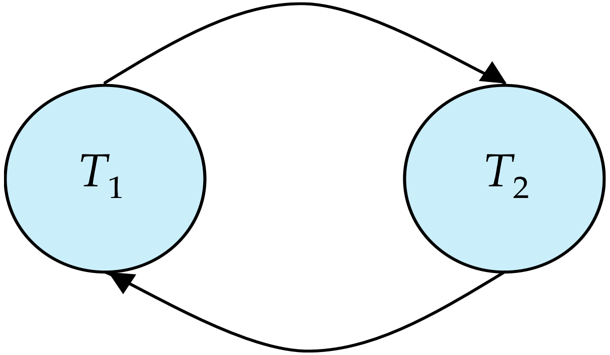
Schedule 3

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

Schedule 6

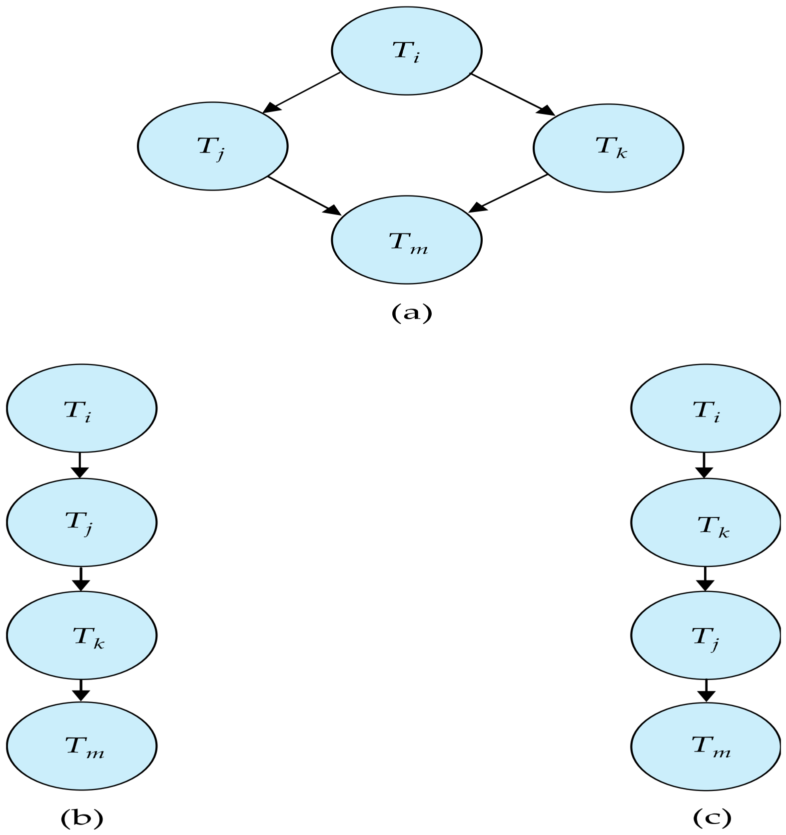
Testing for Serializability

- 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.
- Example of a precedence graph



Test for Conflict Serializability

- A schedule is conflict serializable if and only if its precedence graph is acyclic.
- Cycle-detection algorithms exist which take order n^2 time, where n is the number of vertices in the graph.
  - (Better algorithms take order $n + e$ where e is the number of edges.)
- If precedence graph is acyclic, the serializability order can be obtained by a *topological sorting* of the graph.
  - This is a linear order consistent with the partial order of the graph.
  - For example, a serializability order for Schedule A would be
    $T_5 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_4$
    - Are there others?

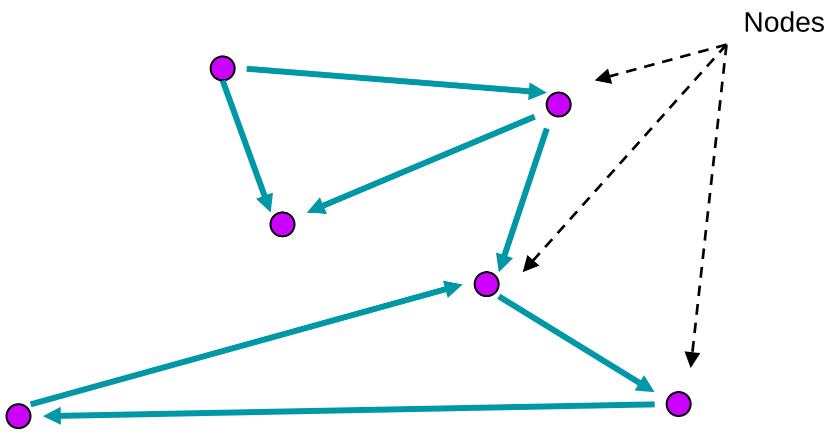


Non-Conflicting Actions

Two actions are non-conflicting if whenever they occur consecutively in a schedule, swapping them does not affect the final state produced by the schedule. Otherwise, they are conflicting.

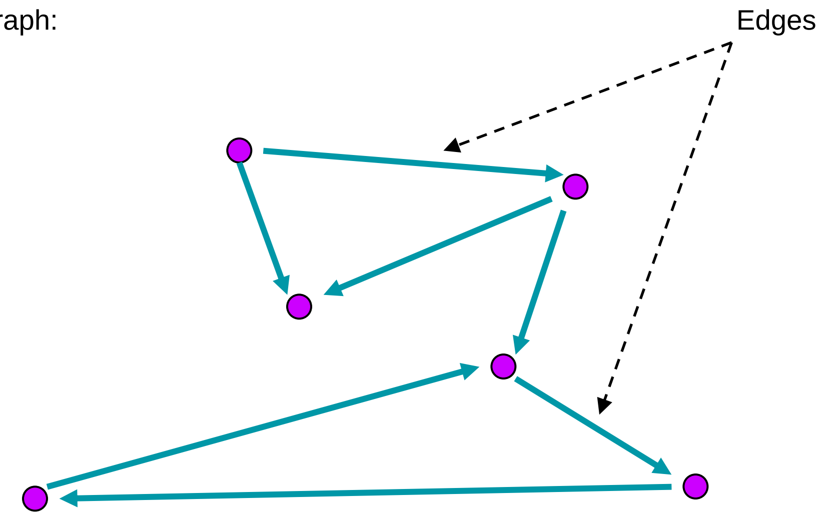
Graph Theory 101

Directed Graph:



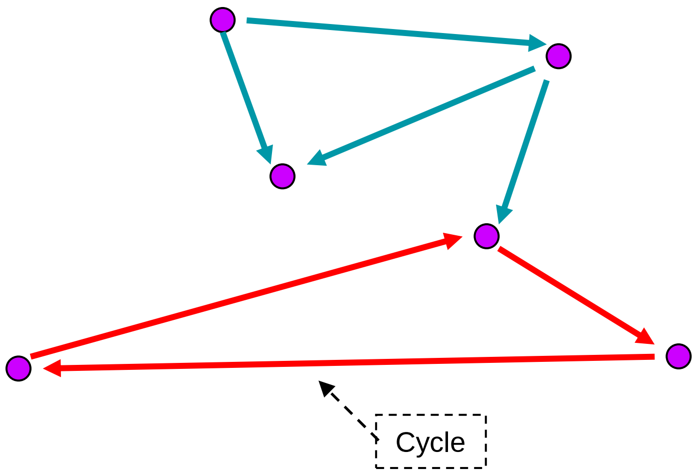
Graph Theory 101

Directed Graph:



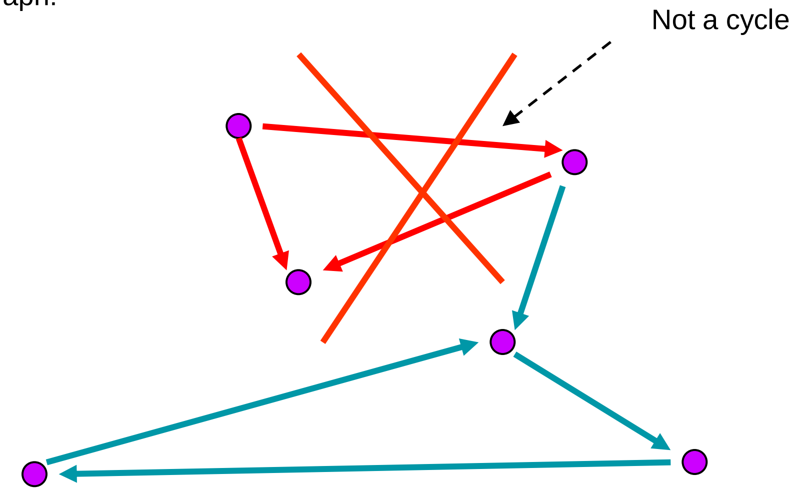
Graph Theory 101

Directed Graph:



Graph Theory 101

Directed Graph:

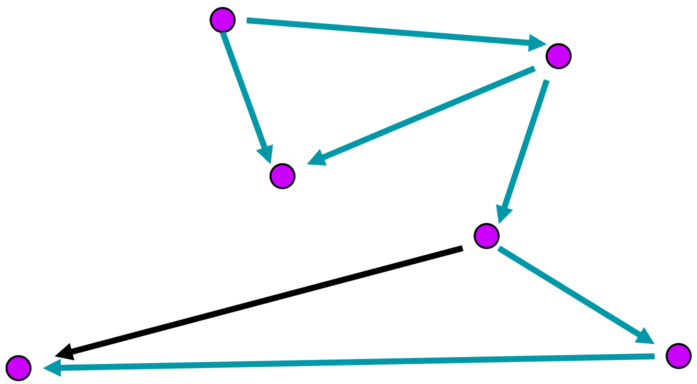


Graph Theory 101

Acyclic Graph: A graph with no cycles

Graph Theory 101

Acyclic Graph:



Testing Conflict Serializability

- Construct precedence graph G for given schedule S
- S is conflict-serializable iff G is acyclic

Precedence Graph

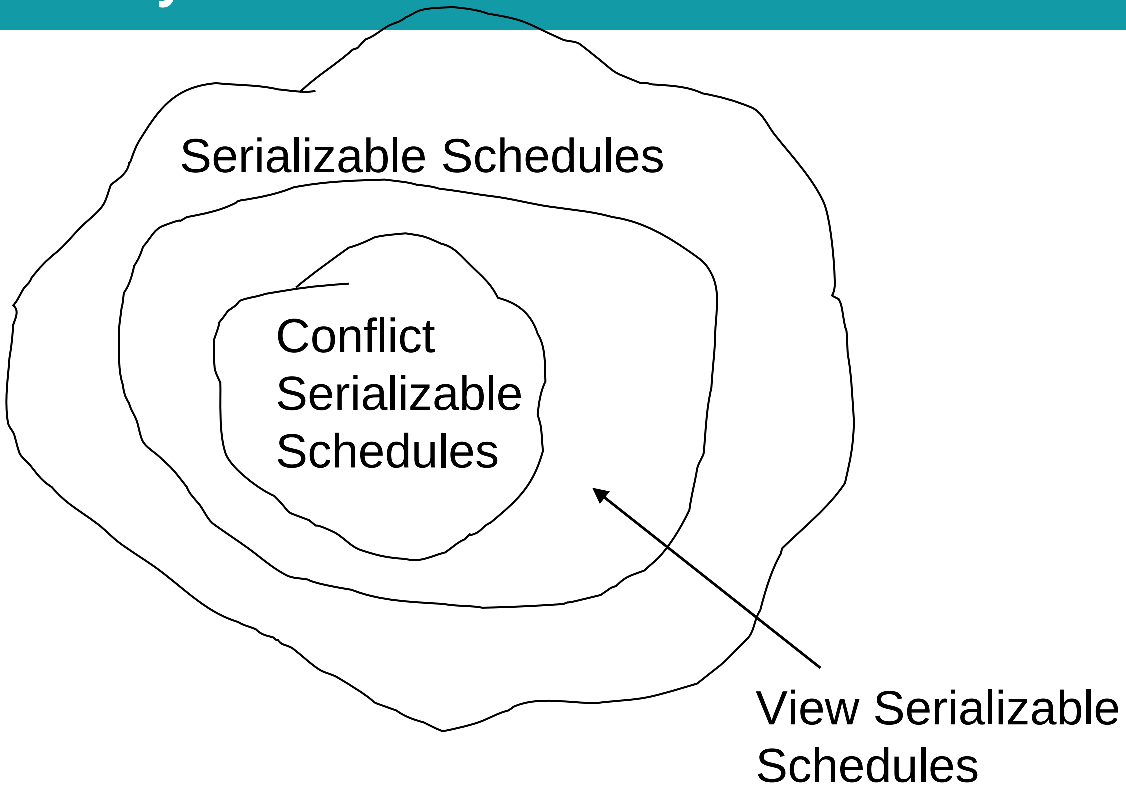
- Precedence graph for schedule S:
  - Nodes: Transactions in S
  - Edges: Ti → Tj whenever
    - S: … ri (X) … wj (X) …
    - S: … wi (X) … rj (X) …
    - S: … wi(X) … wj (X) …

Note: not necessarily consecutive

Testing Conflict Serializability

- Construct precedence graph G for given schedule S
- S is conflict-serializable if G is acyclic

View Serializability



Issues with Concurrency: Example

Bank database: 3 Accounts

Account Balances

A = 500

B = 500

C = 500

Property: A + B + C = 1500

Money does not leave the system

Issues with Concurrency: Example

Transaction T1: Transfer 100 from A to B

A = 500, B = 500, C = 500 $\longrightarrow$

Read (A, t)

t = t - 100

Write (A, t)

Read (B, t)

t = t + 100

Write (B, t)

A = 400, B = 600, C = 500 $\longrightarrow$

Issues with Concurrency: Example

Transaction T2: Transfer 100 from A to C

Read (A, s)

s = s - 100

Write (A, s)

Read (C, s)

s = s + 100

Write (C, s)

Transaction T1	Transaction T2	A	B	C
		500	500	500
Read (A, t)				
t = t - 100				
	Read (A, s)			
	s = s - 100			
	Write (A, s)	400	500	500
Write (A, t)		400	500	500
Read (B, t)				
t = t + 100				
Write (B, t)		400	600	500
	Read (C, s)			
	s = s + 100			
	Write (C, s)	400	600	600

$\boxed{400 + 600 + 600 = 1600}$

Transaction T1	Transaction T2	A	B	C
		500	500	500
Read (A, t)				
t = t - 100				
Write (A, t)		400	500	500
	Read (A, s)			
	s = s - 100			
	Write (A, s)	300	500	500
Read (B, t)				
t = t + 100				
Write (B, t)		300	600	500
	Read (C, s)			
	s = s + 100			
	Write (C, s)	300	600	600

$\boxed{300 + 600 + 600 = 1500}$

Serial Schedule

			A	B	C
	Read (A, t)		500	500	500
	t = t - 100				
T1	Write (A, t)				
	Read (B, t)				
	t = t + 100				
	Write (B, t)		400	600	500
		Read (A, s)			
		s = s - 100			
		Write (A, s)			
T2		Read (C, s)			
		s = s + 100			
		Write (C, s)	300	600	600

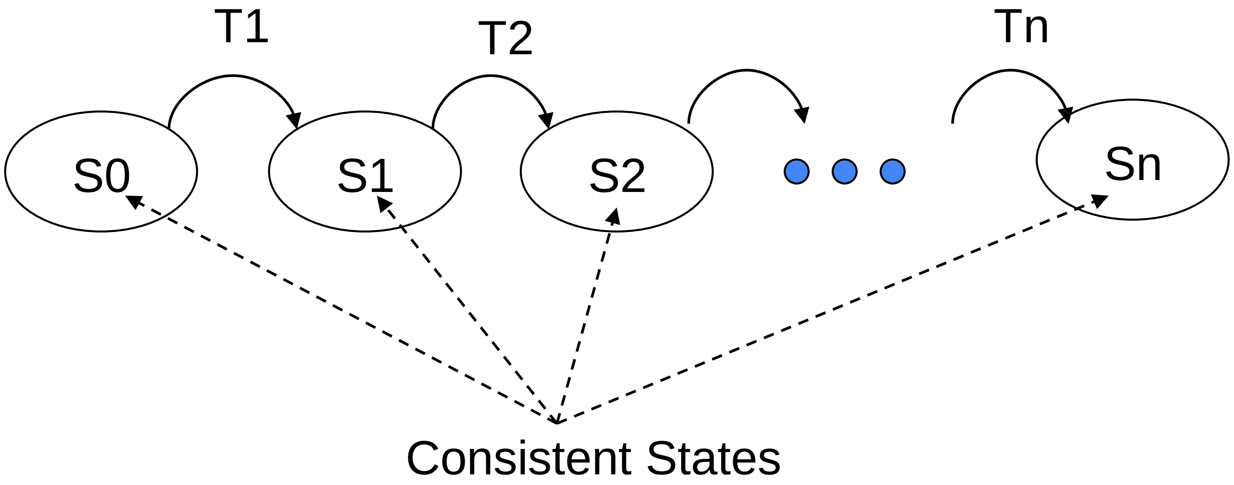
300 + 600 + 600 = 1500

Serial Schedule

Transaction	T1	T2	A	B	C
T2		Read (A, s)	500	500	500
		s = s - 100			
		Write (A, s)			
		Read (C, s)			
		s = s + 100			
		Write (C, s)	400	500	600
T1	Read (A, t)				
	t = t - 100				
	Write (A, t)				
	Read (B, t)				
	t = t + 100				
	Write (B, t)		300	600	600

$300 + 600 + 600 = 1500$

Serial Schedule



Is this Serializable?

Transaction T1	Transaction T2
Read (A, t)	
t = t - 100	
Write (A, t)	
	Read (A, s)
	s = s - 100
	Write (A, s)
Read (B, t)	
t = t + 100	
Write (B, t)	
	Read (C, s)
	s = s + 100
	Write (C, s)

Equivalent Serial Schedule

Transaction T1	Transaction T2
Read (A, t)	
t = t - 100	
Write (A, t)	
Read (B, t)	
t = t + 100	
Write (B, t)	
	Read (A, s)
	s = s - 100
	Write (A, s)
	Read (C, s)
	s = s + 100
	Write (C, s)

Is this Serializable?

Transaction T1	Transaction T2
Read (A, t)	
t = t - 100	
	Read (A, s)
	s = s - 100
	Write (A, s)
Write (A, t)	
Read (B, t)	
t = t + 100	
Write (B, t)	
	Read (C, s)
	s = s + 100
	Write (C, s)

No. In fact, it leads to inconsistent state

Is this Serializable?

Transaction T1	Transaction T2
Read (A, t)	
t = t - 100	
	Read (A, s)
	s = s - ~~100~~ 0
	Write (A, s)
Write (A, t)	
Read (B, t)	
t = t + 100	
Write (B, t)	
	Read (C, s)
	s = s + ~~100~~ 0
	Write (C, s)

Is this Serializable?

Transaction T1	Transaction T2
Read (A, t)	
t = t - 100	
	Read (A, s)
	s = s - 0
	Write (A, s)
Write (A, t)	
Read (B, t)	
t = t + 100	
Write (B, t)	
	Read (C, s)
	s = s + 0
	Write (C, s)

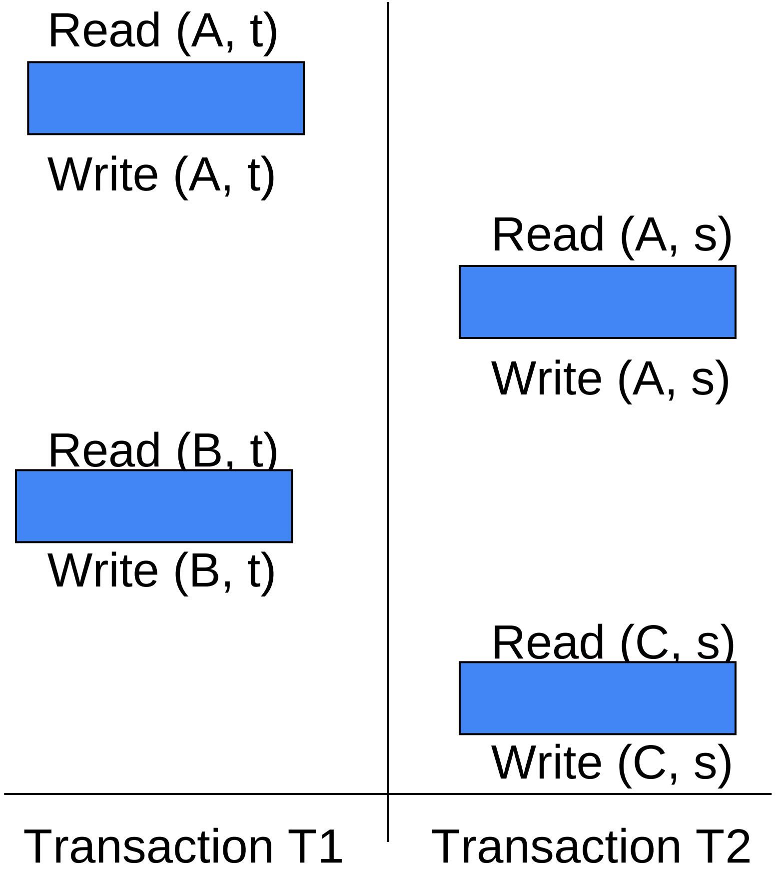
Yes, T2 is no-op

Serializable Schedule

Transaction T1	Transaction T2
Read (A, t)	
t = t - 100	
	Read (A, s)
	s = s - 0
	Write (A, s)
Write (A, t)	
Read (B, t)	
t = t + 100	
Write (B, t)	
	Read (C, s)
	s = s + 0
	Write (C, s)

Serializability depends on code details

Serializable Schedule



Still Serializable!

Example

Consider the following four schedules due to three transactions (indicated by the subscript) using read and write on a data item X, denoted by r(X) and w(X) respectively. Which one of them is conflict serializable ?

$S_1 : r_1(X); r_2(X); w_1(X); r_3(X); w_2(X)$

$S_2 : r_2(X); r_1(X); w_2(X); r_3(X); w_1(X)$

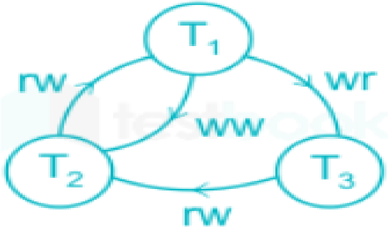
$S_3 : r_3(X); r_2(X); r_1(X); w_2(X); w_1(X)$

$S_4 : r_2(X); w_2(X); r_3(X); r_1(X); w_1(X)$

Option_1 – Not conflict serializable

T_1	T_2	T_3
r(X)		
	r(X)	
w(X)		
		r(X)
	w(X)	

$$S_1 : r_1(X); r_2(X); w_1(X); r_3(X); w_2(X)$$

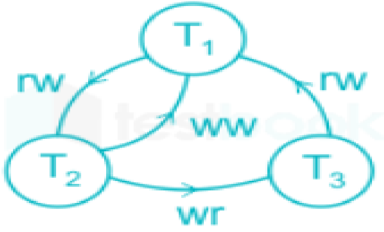


- Cycle between T1 and T2.
- Cycle between T1->T3->T2

Option_2 – Not conflict serializable

T_1	T_2	T_3
	r(X)	
r(X)		
	w(X)	
		r(X)
w(X)		

$S_2 : r_2(X); r_1(X); w_2(X); r_3(X); w_1(X)$

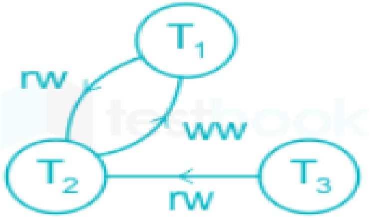


- Cycle between T1 and T2.
- Cycle between T1->T2->T3

Option-3 - Not conflict serializable

T_1	T_2	T_3
		r(X)
	r(X)	
r(X)		
	w(X)	
w(X)		

$$S_3 : r_3(X); r_2(X); r_1(X); w_2(X); w_1(X)$$

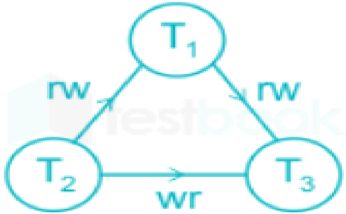


- Cycle between T1 and T2.

Option-4 - Conflict serializable

T_1	T_2	T_3
	r(X)	
	w(X)	
		r(X)
r(X)		
w(X)		

$$S_4 : r_2(X); w_2(X); r_3(X); r_1(X); w_1(X)$$



Correct Answer is S4

- No cycle


Question
&
Answer