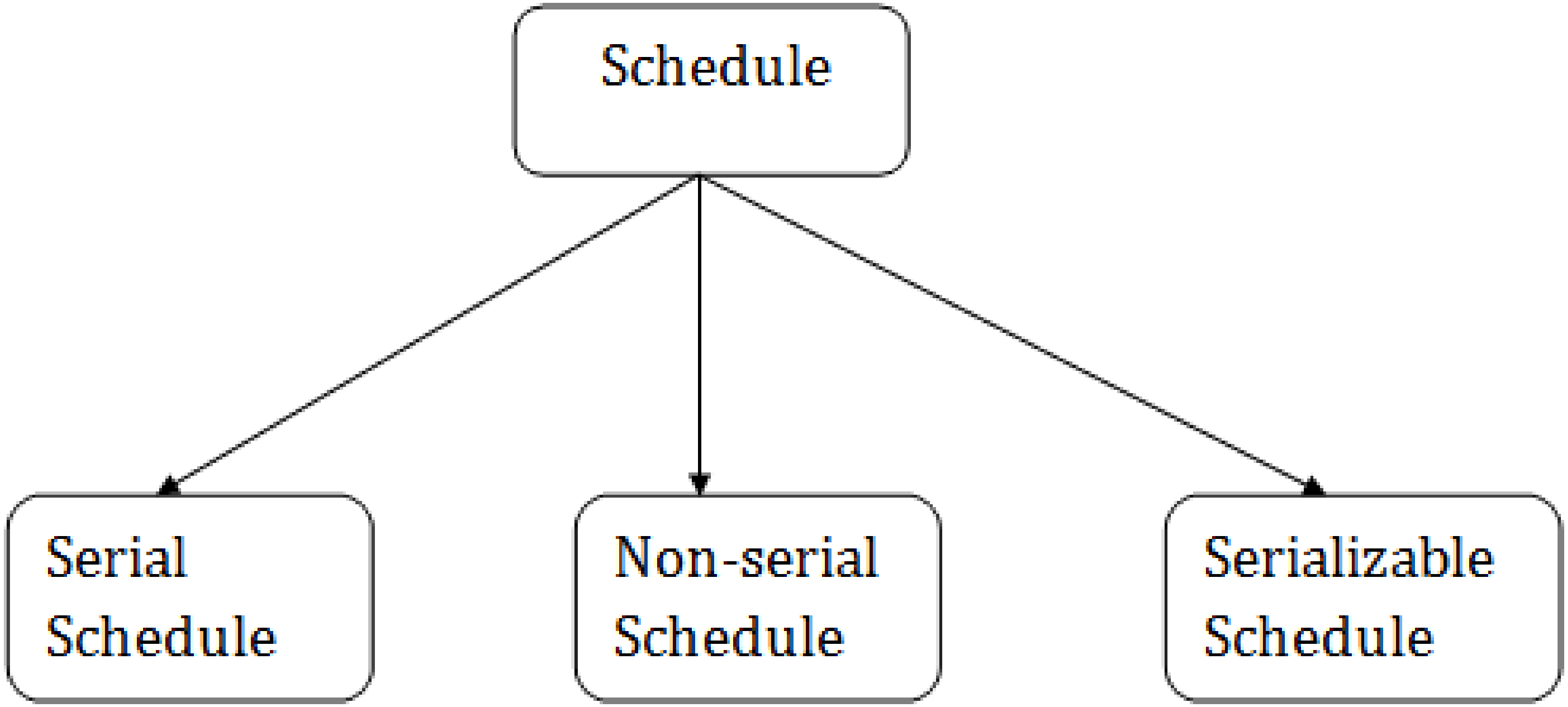
Characterizing Schedules based on Serializability

Characterizing Schedules based on Serializability



Schedules

Serial schedule

- A schedule S is serial if, for every T in S, all the operations of T are executed consecutively in the schedule.

Time

T_1	T_2
read_item(X);	
$X := X - N$;	
write_item(X);	
read_item(Y);	
$Y := Y + N$;	
write_item(Y);	
	read_item(X);
	$X := X + M$;
	write_item(X);

Schedule A

(b)

Time

T_1	T_2
	read_item(X);
	$X := X + M$;
	write_item(X);
read_item(X);	
$X := X - N$;	
write_item(X);	
read_item(Y);	
$Y := Y + N$;	
write_item(Y);	

Schedule B

Schedules

Non-Serial Schedule

- A schedule S is non-serial if the operations from different transactions are **<u>interleaved in *S*.</u>**

	T_1	T_2
Time ↓	read_item(X);	
	$X := X - N$;	
		read_item(X);
		$X := X + M$;
	write_item(X);	
	read_item(Y);	
		write_item(X);
	$Y := Y + N$;	
	write_item(Y);	

The operations of each T_i in S must appear in the same order in which they occur in T_i.

S_a: $r_1(X)$; $r_2(X)$; $w_1(X)$; $r_1(Y)$; $w_2(X)$; $w_1(Y)$;

Schedules

We would like to determine which of the non-serial schedules *always* give a correct result and which do not .

Time

T_1	T_2
read_item(X); $X := X - N$;	
	read_item(X); $X := X + M$;
write_item(X); read_item(Y);	
	write_item(X);
$Y := Y + N$; write_item(Y);	

Schedule C

Time

T_1	T_2

read_item(X);
$X := X - N$;
write_item(X);	

	read_item(X);
	$X := X + M$;
	write_item(X);

read_item(Y);
$Y := Y + N$;
write_item(Y);	

Schedule D

Schedules

Serializable schedule

- A schedule S is serializable if it is equivalent to some serial schedule of the same n transactions.

Time

T_1	T_2
read_item(X); $X := X - N$;	
	read_item(X); $X := X + M$;
write_item(X); read_item(Y);	
	write_item(X);
$Y := Y + N$; write_item(Y);	

Schedule C

Time

T_1	T_2

read_item(X);
$X := X - N$;
write_item(X);	

	read_item(X);
	$X := X + M$;
	write_item(X);

read_item(Y);
$Y := Y + N$;
write_item(Y);	

Schedule D

A nonserial schedule *S* is serializable means it is correct

Schedules- **conflict**

- Two operations in S are in **conflict** if
  - they belong to ***different transactions***
  - they **access the same *item X*** and
  - *at least one* of the operations is a write_item(X).

S: $r_1(x)$ $w_1(X)$ $r_2(X)$ $w_2(x)$ $r_1(y)$ a_1

- **Conflicting operations**
  - $r_1(X)$ and $w_2(X)$
  - $r_2(X)$ and $w_1(X)$
  - $w_1(X)$ and $w_2(X)$

- **Non-conflicting operations**
  - $r_1(X)$ and $r_2(X)$
  - $w_2(X)$ and $w_1(Y)$
  - $r_1(X)$ and $w_1(X)$

Intuitively, two operations are conflicting if changing their order can result in a different outcome.

Types of Conflict: **read-write conflict and write-write conflict**

Characterizing Schedules based on Serializability

- **Conflict equivalent:** Two schedules are said to be conflict equivalent if the order of any two conflicting operations is same in both schedules.

Two operations on the same data items conflict if at least one of them is a write

- $r(X)$ and $w(X)$
- w(X) and r(X)
- $w(X)$ and $w(X)$

- If two conflicting operations are applied in *different orders* in two schedules, the effect can be different

Serializability

- A schedule S is said to be conflict serializable if it is conflict equivalent to some serial schedule S'.

T_1	T_2
read_item(X);	
$X := X - N$;	
write_item(X);	
read_item(Y);	
$Y := Y + N$;	
write_item(Y);	
	read_item(X);
	$X := X + M$;
	write_item(X);

Schedule A

(b)

T_1	T_2
	read_item(X);
	$X := X + M$;
	write_item(X);
read_item(X);	
$X := X - N$;	
write_item(X);	
read_item(Y);	
$Y := Y + N$;	
write_item(Y);	

Schedule B

T_1	T_2
read_item(X);	
$X := X - N$;	
	read_item(X);
	$X := X + M$;
write_item(X);	
read_item(Y);	
	write_item(X);
$Y := Y + N$;	
write_item(Y);	

Schedule C

T_1	T_2
read_item(X);	
$X := X - N$;	
write_item(X);	
	read_item(X);
	$X := X + M$;
	write_item(X);
read_item(Y);	
$Y := Y + N$;	
write_item(Y);	

Schedule D

Serializability

- A schedule S is said to be conflict serializable if it is conflict equivalent to some serial schedule S’.

me

T_1	T_2
read_item(X);	
$X := X - N$;	
write_item(X);	
read_item(Y);	
$Y := Y + N$;	
write_item(Y);	
	read_item(X);
	$X := X + M$;
	write_item(X);

Schedule A

(b)

Time

T_1	T_2
	read_item(X);
	$X := X + M$;
	write_item(X);
read_item(X);	
$X := X - N$;	
write_item(X);	
read_item(Y);	
$Y := Y + N$;	
write_item(Y);	

Schedule B

me

T_1	T_2
read_item(X);	
$X := X - N$;	
	read_item(X);
	$X := X + M$;
write_item(X);	
read_item(Y);	
	write_item(X);
$Y := Y + N$;	
write_item(Y);	

Schedule C

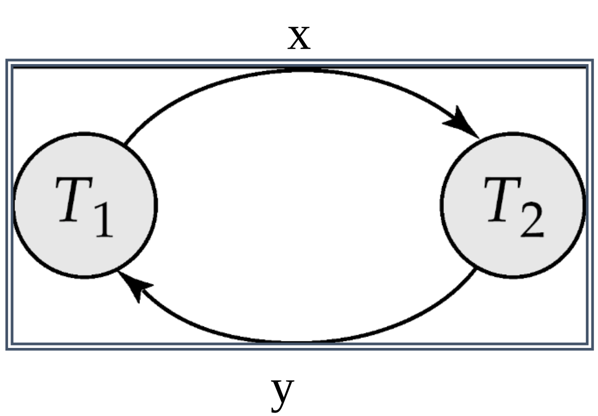
Time

T_1	T_2
read_item(X);	
$X := X - N$;	
write_item(X);	
	read_item(X);
	$X := X + M$;
	write_item(X);
read_item(Y);	
$Y := Y + N$;	
write_item(Y);	

Schedule D

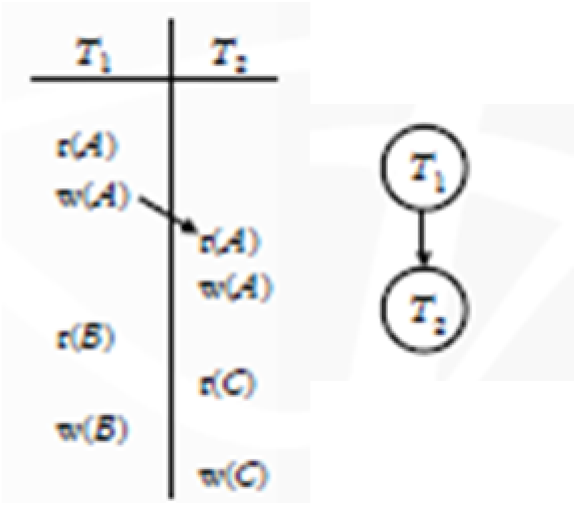
Testing for Serializability

- Consider some schedule of a set of transactions T_1, T_2, ..., T_n
- **Precedence graph**
  - a ***directed graph*** where the vertices ***are*** the *transactions.*
- Draw an arc from T_i to T_j
  - if the two transactions 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**.



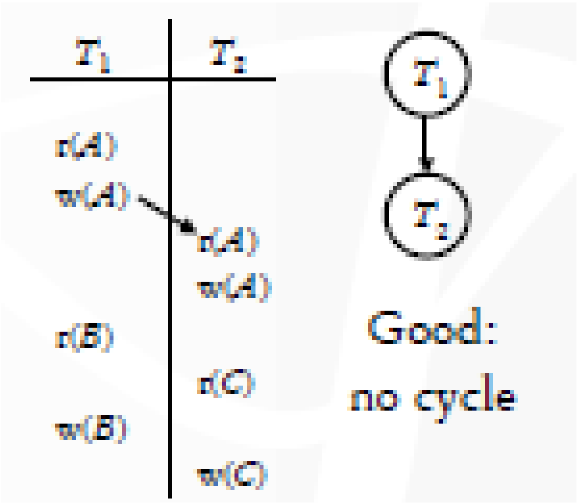
Precedence Graph

- A node for each transaction
- A directed edge from T_i to T_j if an operation of T_i precedes and conflicts with an operation of T_j in the schedule



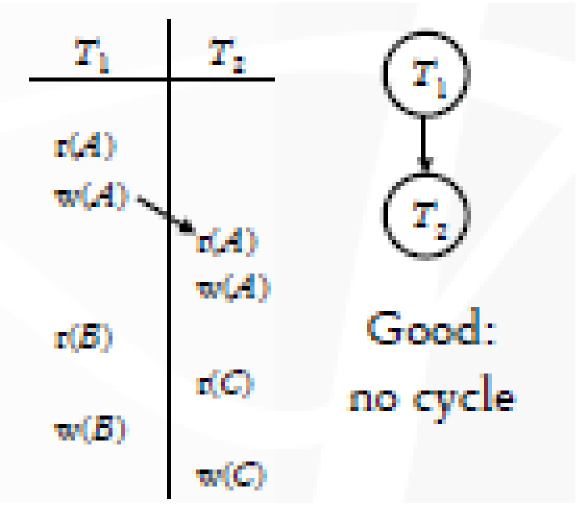
Precedence Graph

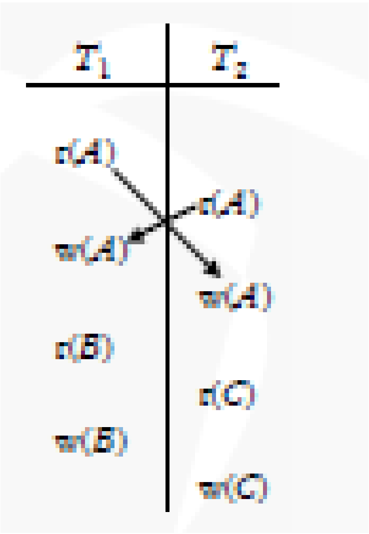
- A node for each transaction
- A directed edge from T_i to T_j if an operation of T_i precedes and conflicts with an operation of T_j in the schedule



Precedence Graph

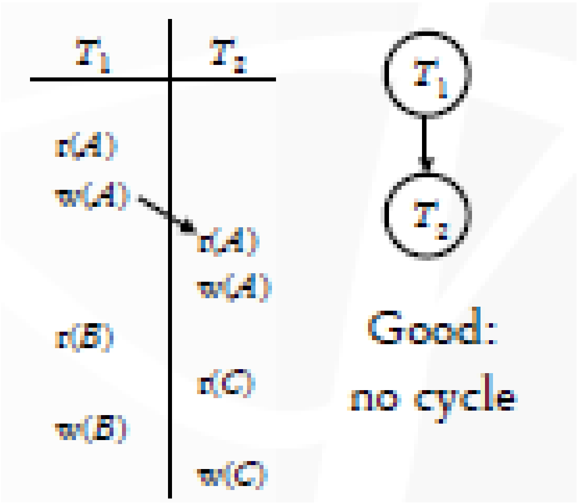
- A node for each transaction
- A directed edge from T_i to T_j if an operation of T_i precedes and conflicts with an operation of T_j in the schedule

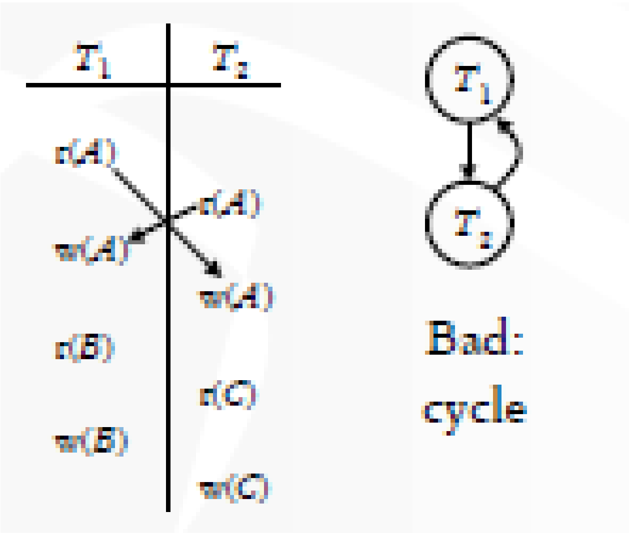




Precedence Graph

- A node for each transaction
- A directed edge from T_i to T_j if an operation of T_i precedes and conflicts with an operation of T_j in the schedule





Constructing the Precedence Graphs

me

T_1	T_2

read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);	

	read_item(X);
	$X := X + M$;
	write_item(X);

Schedule A

(b)

Time

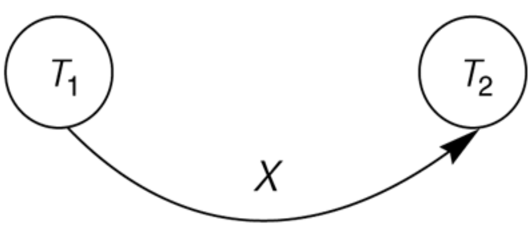
T_1	T_2

	read_item(X);
	$X := X + M$;
	write_item(X);

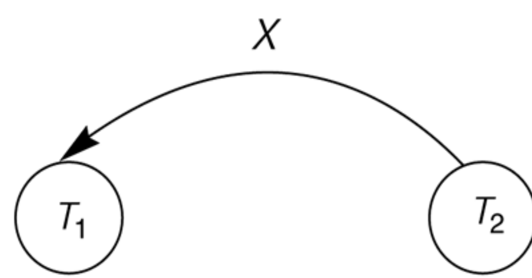
read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);	

Schedule B

(a)



(b)



Constructing the Precedence Graphs

me

T_1	T_2
read_item(X); $X := X - N$;	
	read_item(X); $X := X + M$;
write_item(X); read_item(Y);	
	write_item(X);
$Y := Y + N$; write_item(Y);	

Schedule C

Time

T_1	T_2

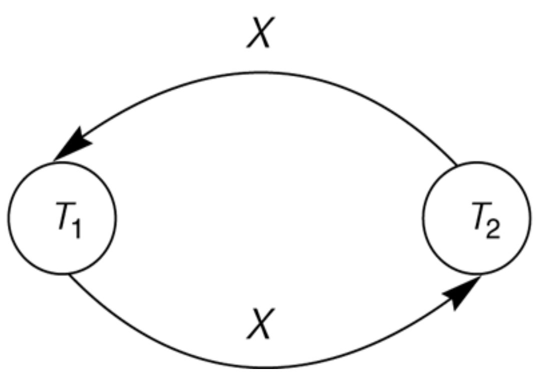
read_item(X);
$X := X - N$;
write_item(X);	

	read_item(X);
	$X := X + M$;
	write_item(X);

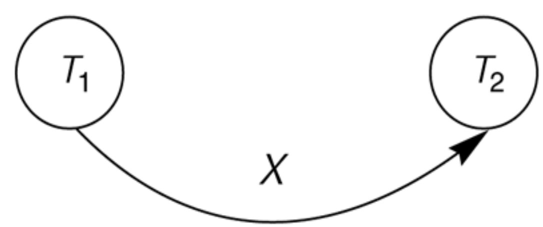
read_item(Y);
$Y := Y + N$;
write_item(Y);	

Schedule D

(c)



(d)



Example of serializability Testing

Transaction T_1
read_item(X);
write_item(X);
read_item(Y);
write_item(Y);

Transaction T_2
read_item(Z);
read_item(Y);
write_item(Y);
read_item(X);
write_item(X);

Transaction T_3
read_item(Y);
read_item(Z);
write_item(Y);
write_item(Z);

Example of Serializability Testing

Time

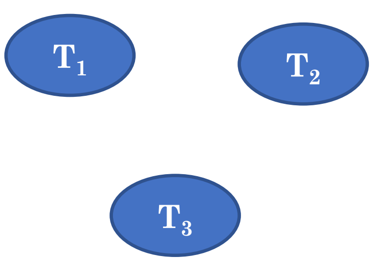
Transaction T_1	Transaction T_2	Transaction T_3
	read_item(Z);	
	read_item(Y);	
	write_item(Y);	
		read_item(Y);
		read_item(Z);
read_item(X);		
write_item(X);		
		write_item(Y);
		write_item(Z);
	read_item(X);	
read_item(Y);		
write_item(Y);		
	write_item(X);	

Schedule E

Example of Serializability Testing

Time	Transaction T_1	Transaction T_2	Transaction T_3
↓		read_item(Z);	
		read_item(Y);	
		write_item(Y);	
			read_item(Y);
			read_item(Z);
	read_item(X);		
	write_item(X);		
			write_item(Y);
			write_item(Z);
		read_item(X);	
	read_item(Y);		
	write_item(Y);		
		write_item(X);	

Schedule E



Example of Serializability Testing

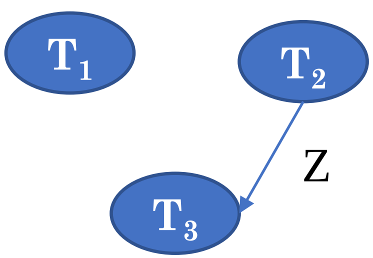
Transaction T_1	Transaction T_2	Transaction T_3

	read_item(Z);
	read_item(Y);
	write_item(Y);	

		read_item(Y); read_item(Z);
read_item(X); write_item(X);		
		write_item(Y); write_item(Z);
	read_item(X);	
read_item(Y); write_item(Y);		
	write_item(X);	

Time

Schedule E



Example of Serializability Testing

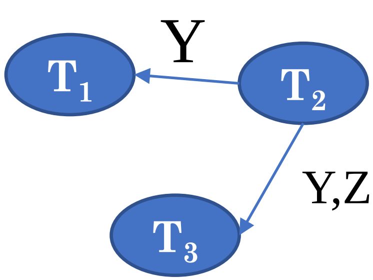
Time

Transaction T_1	Transaction T_2	Transaction T_3

	read_item(Z);
	read_item(Y);
	write_item(Y);	

		read_item(Y); read_item(Z);
read_item(X); write_item(X);		
		write_item(Y); write_item(Z);
	read_item(X);	
read_item(Y); write_item(Y);		
	write_item(X);	

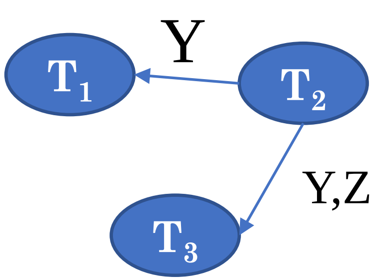
Schedule E



Example of Serializability Testing

Time	Transaction T_1	Transaction T_2	Transaction T_3
↓		read_item(Z);	
		read_item(Y);	
		write_item(Y);	
			read_item(Y);
			read_item(Z);
	read_item(X);		
	write_item(X);		
			write_item(Y);
			write_item(Z);
		read_item(X);	
	read_item(Y);		
	write_item(Y);		
		write_item(X);	

Schedule E

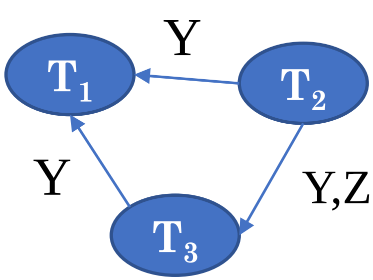


Example of Serializability Testing

Time

Transaction T_1	Transaction T_2	Transaction T_3
	read_item(Z);	
	read_item(Y);	
	write_item(Y);	
		read_item(Y);
		read_item(Z);
read_item(X);		
write_item(X);		
		write_item(Y);
		write_item(Z);
	read_item(X);	
read_item(Y);		
write_item(Y);		
	write_item(X);	

Schedule E



Example of Serializability Testing

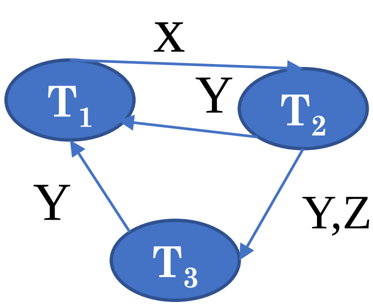
Transaction T_1	Transaction T_2	Transaction T_3

	read_item(Z);
	read_item(Y);
	write_item(Y);	

		read_item(Y); read_item(Z);
read_item(X); write_item(X);		
		write_item(Y); write_item(Z);
	read_item(X);	
read_item(Y); write_item(Y);		
	write_item(X);	

Time

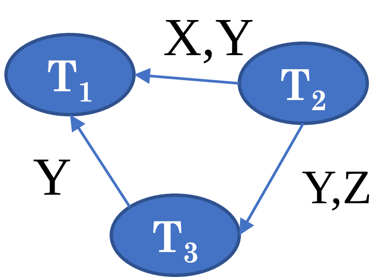
Schedule E



Example of Serializability Testing

Time	Transaction T_1	Transaction T_2	Transaction T_3
↓		read_item(Z);	
		read_item(Y);	
		write_item(Y);	
			read_item(Y);
			read_item(Z);
	read_item(X);		
	write_item(X);		
			write_item(Y);
			write_item(Z);
		read_item(X);	
	read_item(Y);		
	write_item(Y);		
		write_item(X);	

Schedule E

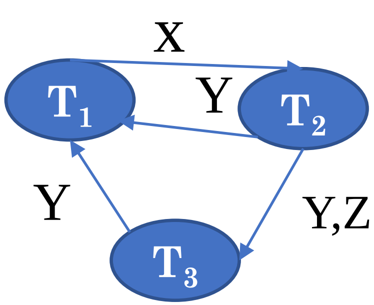


Example of Serializability Testing

Time

Transaction T_1	Transaction T_2	Transaction T_3
	read_item(Z);	
	read_item(Y);	
	write_item(Y);	
		read_item(Y);
		read_item(Z);
read_item(X);		
write_item(X);		
		write_item(Y);
		write_item(Z);
	read_item(X);	
read_item(Y);		
write_item(Y);		
	write_item(X);	

Schedule E



Example of Serializability Testing

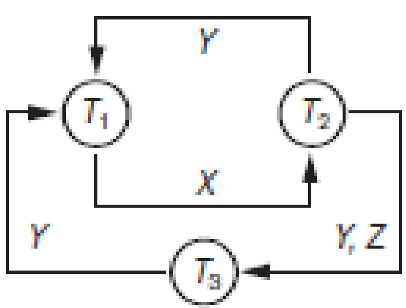
Time ↓

Transaction T_1	Transaction T_2	Transaction T_3

	read_item(Z);
	read_item(Y);
	write_item(Y);	

		read_item(Y); read_item(Z);
read_item(X); write_item(X);		
		write_item(Y); write_item(Z);
	read_item(X);	
read_item(Y); write_item(Y);		
	write_item(X);	

Schedule E



Equivalent serial schedules

None

Reason

Cycle $X(T_1 \rightarrow T_2), Y(T_2 \rightarrow T_1)$

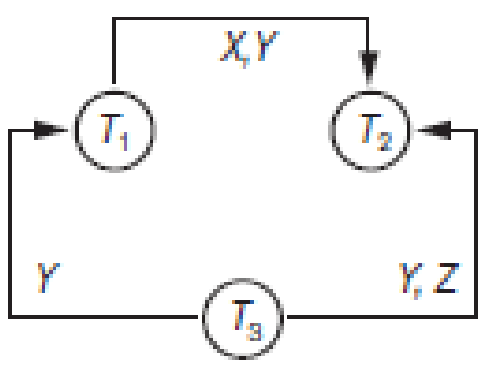
Cycle $X(T_1 \rightarrow T_2), YZ\ (T_2 \rightarrow T_3), Y(T_3 \rightarrow T_1)$

Example of Serializability Testing

Time ↓	Transaction T_1	Transaction T_2	Transaction T_3
			read_item(Y); read_item(Z);
	read_item(X); write_item(X);		
			write_item(Y); write_item(Z);
		read_item(Z);	
	read_item(Y); write_item(Y);		

		read_item(Y);
		write_item(Y);
		read_item(X);
		write_item(X);	

Schedule F



Equivalent serial schedules

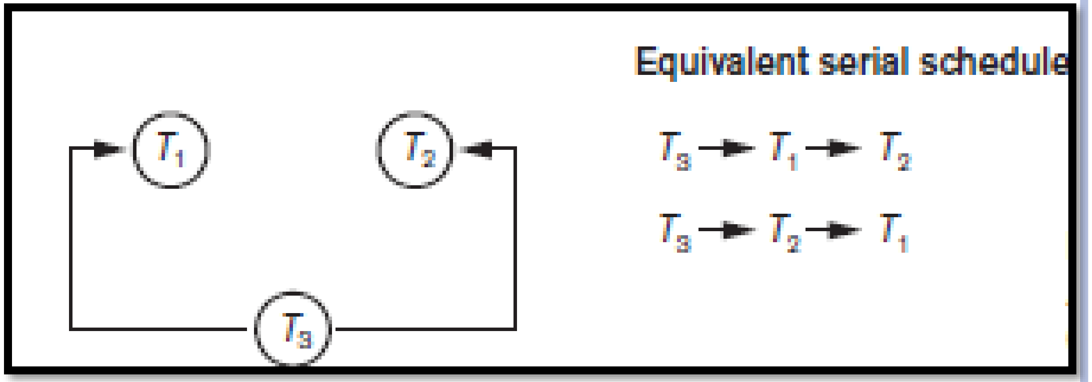
$T_3 \rightarrow T_1 \rightarrow T_2$

Example of Serializability Testing

Transaction T_1
read_item(X);
write_item(X);
read_item(Y);
write_item(Y);

Transaction T_2
read_item(Z);
read_item(Y);
write_item(Y);
read_item(X);
write_item(X);

Transaction T_3
read_item(Y);
read_item(Z);
write_item(Y);
write_item(Z);



To find an equivalent serial schedule:

- start with a node that does not have any incoming edges, and then
- make sure that the node order for every edge is not violated.

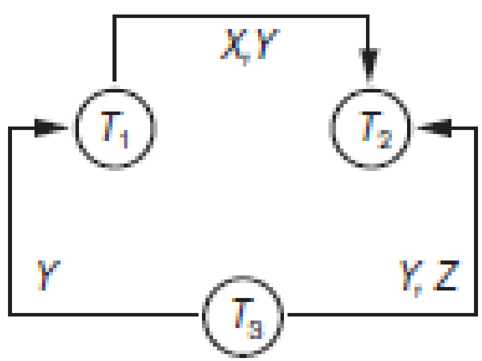
Example of Serializability Testing

Transaction T_1	Transaction T_2	Transaction T_3
		read_item(Y); read_item(Z);
read_item(X); write_item(X);		
		write_item(Y); write_item(Z);
	read_item(Z);	
read_item(Y); write_item(Y);		

	read_item(Y);
	write_item(Y);
	read_item(X);
	write_item(X);	

Time

Schedule F

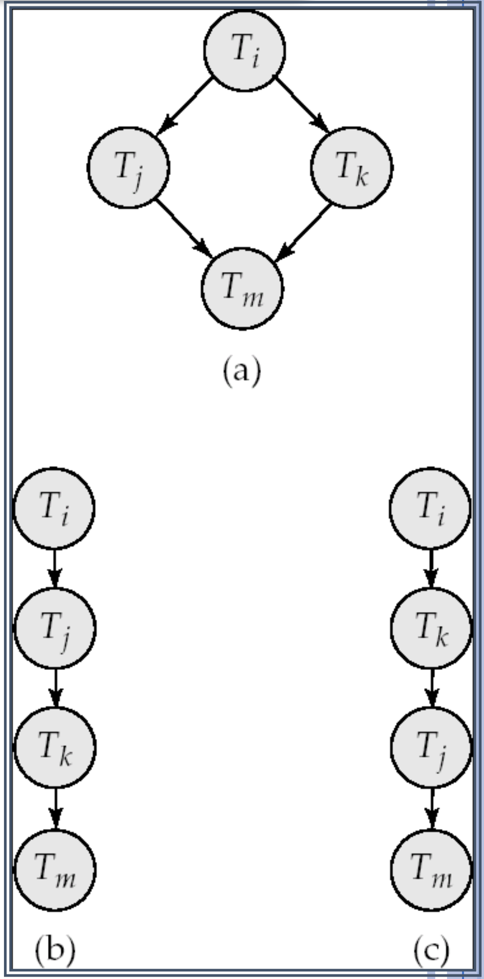


Equivalent serial schedules

$T_3 \rightarrow T_1 \rightarrow T_2$

Test for Conflict Serializability

- A schedule is ***conflict serializable***
  - **if** and only **if** its *precedence graph is* ***acyclic***.
- **Cycle-detection algorithm:**
  - Use ***Depth-first search*** to detect cycle
    - DFS for a connected graph produces a tree.
    - There is a cycle in a graph only if there is a **back-edge** present in the graph
- **If** *precedence graph* is **acyclic**, the *serializability order* can be obtained by a ***topological sorting*** of the graph.



A **topological ordering** of a directed graph is a linear ordering of its <u>*vertices*</u> such that for <u>*every directed edge*</u> *uv* from vertex *u* to vertex *v*, *u* comes before *v* in the ordering.

Characterizing Schedules based on Serializability

- 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.

- Current approach used in most DBMSs:
  - Use of locks with **Two Phase locking Protocol**