Recognizing a conflict-serializable schedule

Overview of this video

This video will show you how to recognize if a schedule is conflict-serializable or not

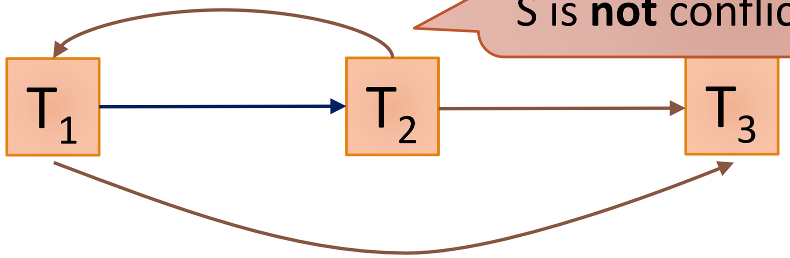
Example

A schedule that is **not** conflict-serializable, but serializable:

S: $w_2(X); w_1(X); w_1(Y); w_2(Y); w_3(X);$

has a cycle →
S is **not** conflict-serialisable

How to show: Not conflict-serializable?



Identify the transactions in S

Identify the conflicts in S

Conflicts impose constraints on the order of the transactions in any conflict-equivalent serial schedule

Precedence Graph

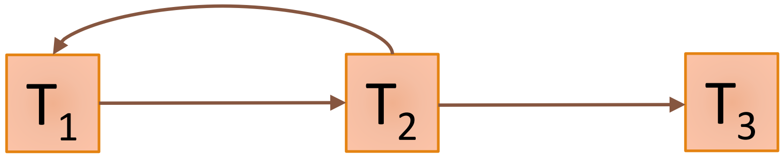
The precedence graph for a schedule S is defined as follows:

- It is a **directed graph.**
- Its **nodes** are the transactions that occur in S.
- It has an **edge** from transaction T_i to transaction T_j if there is a conflicting pair of operations op_1 and op_2 in S such that
  - op_1 appears before op_2 in S
  - op_1 belongs to transaction T_i
  - op_2 belongs to transaction T_j.

Example:

S: $r_2(X); r_1(Y); w_2(X); r_2(Y); r_3(X); w_1(Y); w_3(X); w_2(Y)$

Precedence graph for S:



Testing Conflict-Serializability

To test if a schedule S is **conflict-serializable**:

- Construct the precedence graph for S.
- If the precedence graph is **no cycle**, then S is conflict-serializable. Otherwise not.

Example 1:

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

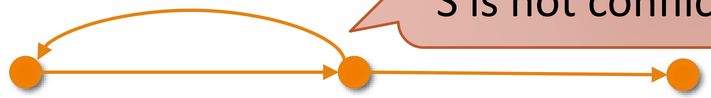
Precedence graph for S:

has no cycle → S is conflict-serializable

Example 2:

S: $r_2(X); r_1(Y); w_2(X); r_2(Y); r_3(X); w_1(Y); w_3(X); w_2(Y)$

Precedence graph for S:



contains a cycle → S is not conflict-serializable

Why does this work?

$$T_1 \longrightarrow T_2$$

This says:

There is a conflict between an operation in T_1 (that appears first) and an operation in T_2

All conflict-equivalent schedulers: operation in T_1 is before operation in T_2

Why does this work?

$T_1 \rightarrow T_2$

All conflict-equivalent schedulers: operation x in T_1 is before operation y in T_2

Proof by contradiction: Assume there is a conflict-equivalent schedule S' where this is not so

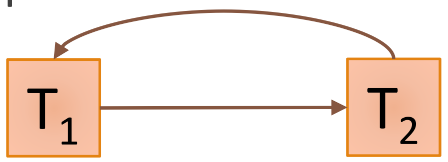
Consider first *consecutive* swap between S and S' where x goes from being before y to being after y

In that swap, either:

- We swap x and y (not legal since they conflict)
- Or we swap at most 1 of them, but then either x is swapped with something before y or y is swapped with something after x and in either cases that swap can't have put y before x. This is a contradiction! This means that our assumption is wrong and there is no such schedule

Implication of a cycle

A cycle in the precedence graph



Consider some serial schedule

It must put some transaction T in the cycle first by definition

Since there is a cycle, there must be a transaction S in the cycle that points to T

By last slide, one of the operations in S must be before one of the operations in T

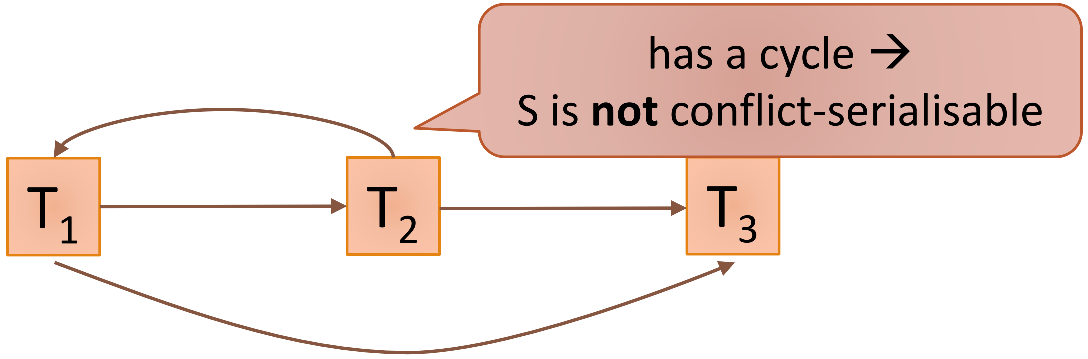
But then T is not the first transaction in the cycle

That is a contradiction, and we can therefore not have any serial schedule

Example 1

Consider this schedule from before:

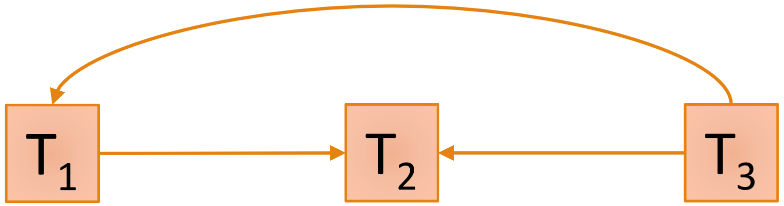
S: $w_2(X)$; $w_1(X)$; $w_1(Y)$; $w_2(Y)$; $w_3(X)$;



Example 2

Consider the following schedule:

S: $r_1(Y), r_3(Y), r_1(X), r_2(X), w_2(X), r_3(Z), w_3(Z), r_1(Z), w_1(Y), r_2(Z)$



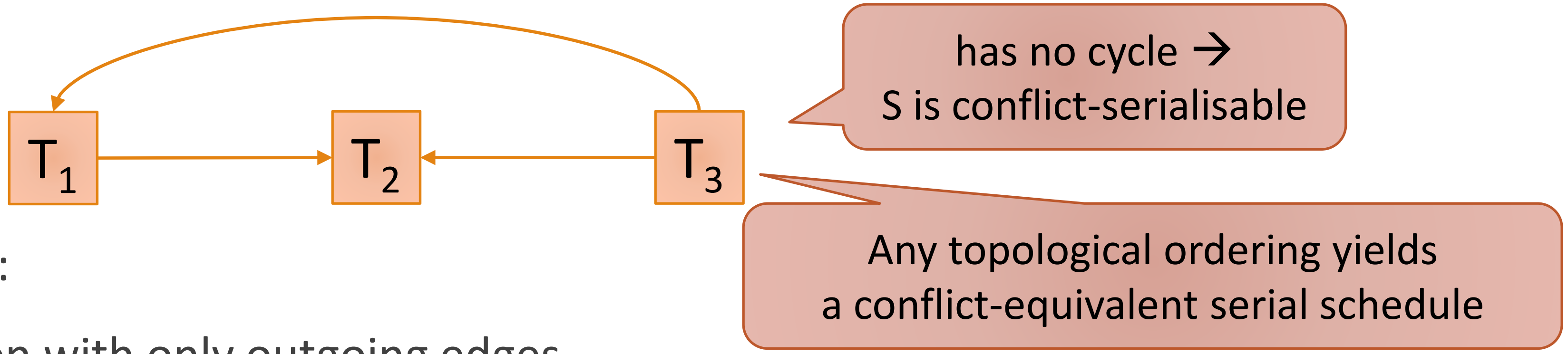
has no cycle →
S is conflict-serialisable

Any topological ordering yields
a conflict-equivalent serial schedule

Example 2

Consider the following schedule:

$$S:\ r_1(Y), r_3(Y), r_1(X), r_2(X), w_2(X), r_3(Z), w_3(Z), r_1(Z), w_1(Y), r_2(Z)$$



Find serial schedule:

1. Find a transaction with only outgoing edges
2. You put it next in your schedule, remove it and all outgoing edges from the graph and repeat

Serial schedule: $r_3(Y), r_3(Z), w_3(Z)$

Example 2

Consider the following schedule:

S: $r_1(Y), r_3(Y), r_1(X), r_2(X), w_2(X), r_3(Z), w_3(Z), r_1(Z), w_1(Y), r_2(Z)$

$$T_1 \longrightarrow T_2$$

Find serial schedule:

1. Find a transaction with only outgoing edges
2. You put it next in your schedule, remove it and all outgoing edges from the graph and repeat

Serial schedule: $r_3(Y), r_3(Z), w_3(Z), r_1(Y), r_1(X), r_1(Z), w_1(Y)$

Example 2

Consider the following schedule:

$$S: \ r_1(Y), r_3(Y), r_1(X), r_2(X), w_2(X), r_3(Z), w_3(Z), r_1(Z), w_1(Y), r_2(Z)$$

T_2

Find serial schedule:

1. Find a transaction with only outgoing edges
2. You put it next in your schedule, remove it and all outgoing edges from the graph and repeat

Serial schedule: $r_3(Y), r_3(Z), w_3(Z), r_1(Y), r_1(X), r_1(Z), w_1(Y), r_2(X), w_2(X), r_2(Z)$

Summary

A schedule is conflict-serializable if there is no cycle in the precedence graph

> RECALL: A **conflict** in a schedule is a pair of operations from different transactions *that cannot be swapped* without changing the behaviour of at least one of the transactions

The precedence graph is defined as follows:

Have a state for each transaction

There is an edge from transaction 1 to transaction 2 iff there is a conflict involving them with the operation from transaction 1 being the first occurring one in the schedule