

Serializability

- [Serializability](#)
- [Conflict Serializability](#)
- [Conflict Serializability Example](#)
- [View Serializability](#)
- [View Serializability Example](#)

❖ Serializability

Serializable schedule:

- concurrent schedule for $T_1..T_n$ with final state S
- S is also a final state of a possible serial schedule for $T_1..T_n$

i.e. we know it is valid (because it can be achieved via a serial instead of concurrent schedule)

Abstracting this needs a notion of schedule equivalence.

Two common formulations of serializability:

- conflict serializability (read/write operations occur in the "right" order)
- view serializability (read operations see the correct version of data)

❖ Conflict Serializability

Consider two transactions T_1 and T_2 acting on data item X .

Possible orders for read/write operations by T_1 and T_2 :

T_1 first	T_2 first	Equiv?
$R_1(X) R_2(X)$	$R_2(X) R_1(X)$	yes <small>reads don't change anything</small>
$R_1(X) W_2(X)$	$W_2(X) R_1(X)$	no <small>seeing before update vs after update</small>
$W_1(X) R_2(X)$	$R_2(X) W_1(X)$	no <small>seeing after vs before</small>
$W_1(X) W_2(X)$	$W_2(X) W_1(X)$	no <small>last write wins strategy</small>

If T_1 and T_2 act on different data items, result is always equivalent.

❖ Conflict Serializability (cont)

Two transactions have a potential conflict if

- they perform operations on the same data item
- at least one of the operations is a write operation

In such cases, the order of operations affects the result.

If no conflict, can swap order without affecting the result.

If we can transform a schedule

- by swapping the order of non-conflicting operations
- such that the result is a serial schedule

then we say that the schedule is conflict serializable.

❖ Conflict Serializability (cont)

Example: transform a concurrent schedule to serial schedule

```

T1: R(A) W(A)      R(B)      W(B)
T2:      R(A)      W(A)      R(B) W(B)
swap
T1: R(A) W(A) R(B)      W(B)
T2:      R(A) W(A)      R(B) W(B)
swap
T1: R(A) W(A) R(B)      W(B)
T2:      R(A)      W(A) R(B) W(B)
swap
T1: R(A) W(A) R(B) W(B)
T2:      R(A) W(A) R(B) W(B)

```

❖ Conflict Serializability (cont)

Checking for conflict-serializability:

- show that ordering in concurrent schedule
- cannot be achieved in any serial schedule

Method for doing this:

- build a precedence-graph
- nodes represent transactions
- arcs represent order of action on shared data
- arc from $T_1 \rightarrow T_2$ means T_1 acts on X before T_2
- a cycle indicates *not* conflict-serializable.

❖ Conflict Serializability Example

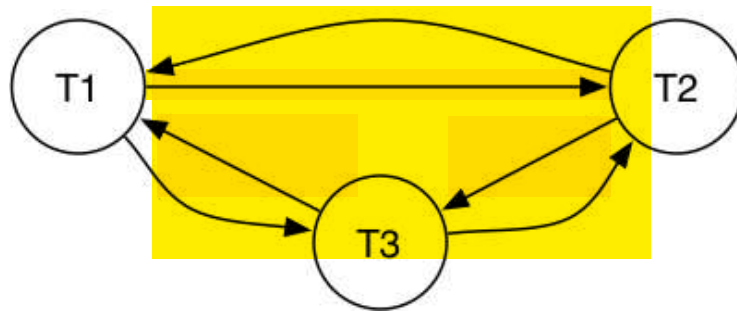
Example schedule which is not conflict serializable:

T1:	R(X)		R(Y)	W(X)		W(Y)
T2:		R(X)		W(X)		
T3:		R(X)				W(X)

attempted swaps

T1:		R(X)	W(X)		R(Y)	W(Y)
T2:		R(X)		W(X)		
T3:	R(X)				W(X)	

Precedence graph for the above schedule:



❖ View Serializability

View Serializability is

- an alternative formulation of serializability
- that is less conservative than conflict serializability (CS)
(some safe schedules that are view serializable are not conflict serializable)

As with CS, it is based on a notion of schedule equivalence

- a schedule is "safe" if *view equivalent* to a serial schedule

The idea: if, across the two schedules ...

- they read the same version of a shared object
- they write the same final version of an object

then they are view equivalent

COMP3311 20T3 ♦ Serializability ♦ [7/10]

so this is more of a practical approach. We only care that it ends up being correct and that the shared object is never read to be in the wrong state.

❖ View Serializability (cont)

Two schedules S and S' on $T_1..T_n$ are view equivalent iff

- for each shared data item X
 - if, in S , T_j reads the initial value of X , then, in S' , T_j also reads the initial value of X
 - if, in S , T_j reads X written by T_k , then, in S' , T_j also reads the value of X written by T_k in S'
 - if, in S , T_j performs the final write of X , then, in S' , T_j also performs the final write of X

To check serializability of S ...

- find a serial schedule that is *view equivalent* to S
- from among the $n!$ possible serial schedules

❖ View Serializability Example

Example: consider the following concurrent schedule

```

T1: R(A) W(A)      R(B)      W(B)
T2:      R(A)      W(A)      R(B) W(B)
  
```

If view serializable, the read/write behaviour must be like one of

1. T1: R(A) W(A) R(B) W(B)
T2: R(A) W(A) R(B) W(B)
2. T1: R(A) W(A) R(B) W(B)
T2: R(A) W(A) R(B) W(B)

❖ View Serializability Example (cont)

Reminder of concurrent schedule

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

In the concurrent schedule

- A: T1 reads initial, T2 reads T1's write, T2 writes final
- B: T1 reads initial, T2 reads T1's write, T2 writes final

In T1;T2

- A: T1 reads initial, T2 reads T1's write, T2 writes final
- B: T1 reads initial, T2 reads T1's write, T2 writes final

So, concurrent schedule is view equivalent to T1;T2

Produced: 15 Nov 2020