

Conflict-serializable schedules

Overview of this video

Serializable schedules are too complex, so what is DBMS doing instead?

How much concurrency can we allow
while satisfying Isolation and Consistency
... while being able to check it fast?

Conflict-Serializability

Stronger form of serializability that is used/enforced by most commercial DBMS

Based on the notion of a **conflict**.

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.

Example:

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

conflict

Do not write commit/abort
(will deal with these later)

Observation: Operations can only be in conflict if one of them is a write & they access the same item.

Conflict – Characterisation

A **conflict** in a schedule is a pair of operations

1. from different transactions
2. that access the same item
3. and at least one of them is a write operation

Example (from previous slide):

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



Can you find any other conflicts?

conflict in S

HINT: there are 2 more...

Conflict-Serializability

Two schedules S and S' are **conflict-equivalent** if S' can be obtained from S by swapping any number of (1) *consecutive* (2) non-conflicting operations from (3) different transactions.

Example:

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

Conflict-serialisable
schedule

$r_1(X); w_1(X); w_2(X); r_1(Y); r_2(Y); w_1(Z)$

$r_1(X); w_1(X); r_1(Y); w_2(X); r_2(Y); w_1(Z)$

$r_1(X); w_1(X); r_1(Y); w_2(X); w_1(Z); r_2(Y)$

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

Serial schedule

A schedule is **conflict-serialisable** if it is conflict-equivalent to a serial schedule.

Example

Is the schedule S below conflict-serializable?

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

Example

Is the schedule S below conflict-serializable?

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

Yes: Conflict-serialisable

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

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

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

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

Serial

Example 2

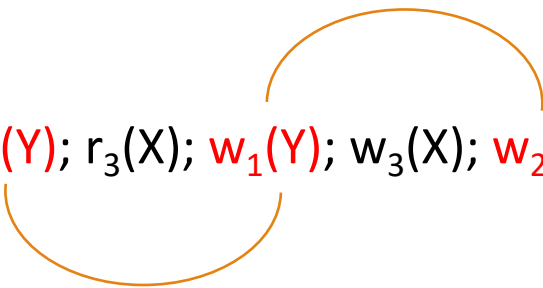
What about this one?

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

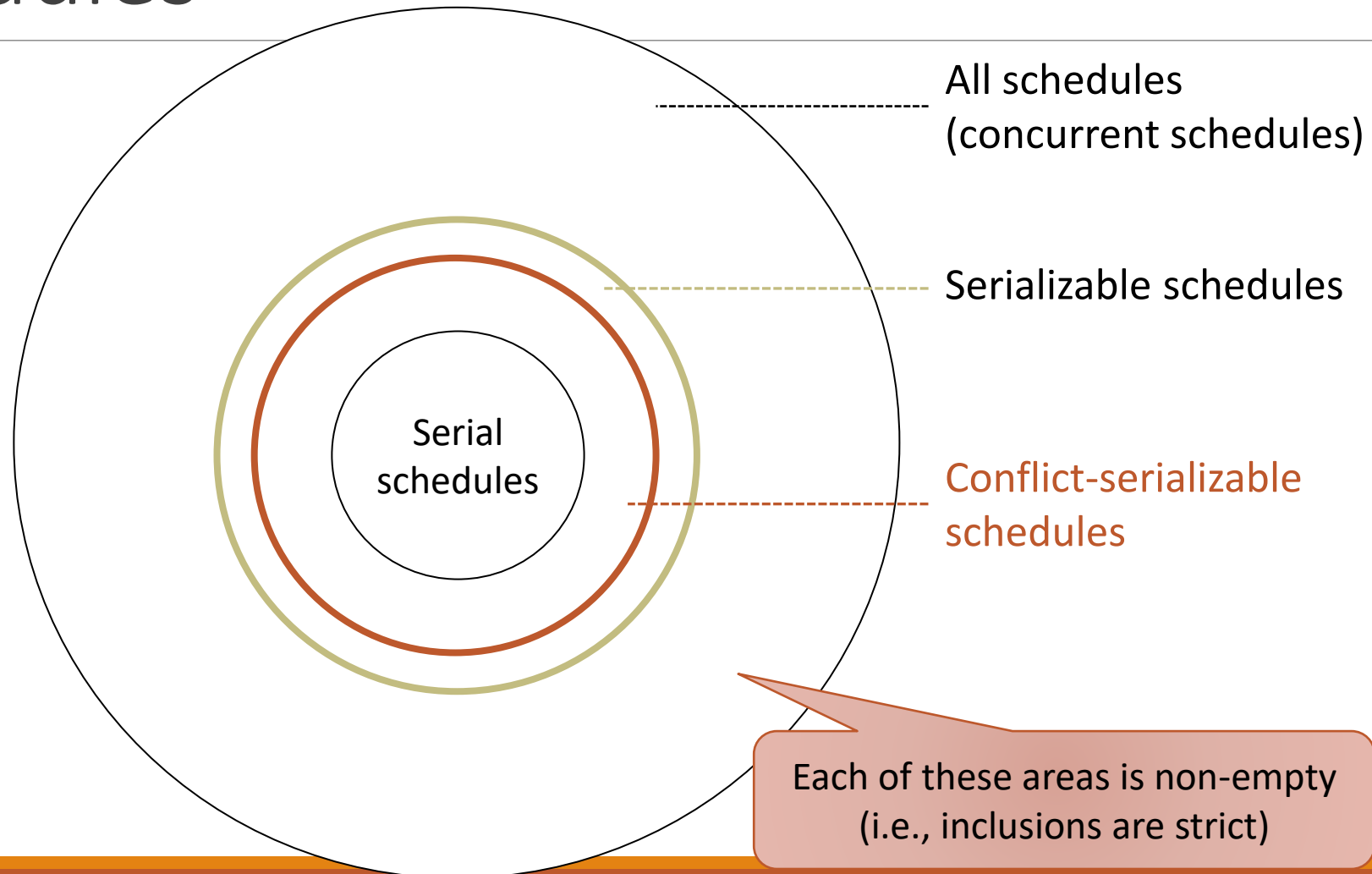
Example 2

What about this one?

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



Schedules



Example

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

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

Why is it serializable?

- The change in the database is given by $w_2(Y); w_3(X);$ and neither transaction reads anything
- Hence, it is equivalent to $w_1(X); w_1(Y); w_2(X); w_2(Y); w_3(X);$

Why is it **not** conflict-serializable?

- We will see in the next video how to check if a schedule is conflict-serializable!
 - Of course, that will also mean that we can see if something is not conflict-serializable!

Summary

We want a schedule that allows as much concurrency as possible while following Isolation and Consistency!

Such a schedule is called a serializable schedule! (as discussed in the last video)

... but we can't find it (or even recognize it) in general (just a claim though)

Therefore, DBMS implementations use conflict-serializable schedules instead!

- Next video will show how to recognize those

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.

A schedule is **conflict-serializable** if you can in some way repeatably swap consecutive pairs of non-conflicting operations from different transactions and get a serial schedule