No cascading-rollbacks!

Overview over this video

Last video: Recoverable schedules

- I.e. if we follow those, we can avoid breaking durability (because of cascading rollbacks)

Overview over this video

Last video: Recoverable schedules

- I.e. if we follow those, we can avoid breaking durability (because of cascading rollbacks)

This video: Cascadeless schedules

- I.e. if we follow those, we can avoid cascading rollbacks!

Recoverable schedules still cascades

Recoverable schedules still cascades

A recoverable schedule:

$$S_4: w_1(X); w_1(Y); w_2(X); r_2(Y); w_2(Y); c_1; c_2$$

Recoverable schedules still cascades

A recoverable schedule:

$$S_4: w_1(X); w_1(Y); w_2(X); r_2(Y); w_2(Y); c_1; c_2$$

Suppose T_1 needs to be rolled back here

Recoverable schedules still cascades

A recoverable schedule:

$$S_4: w_1(X); w_1(Y); w_2(X); r_2(Y); w_2(Y); c_1; c_2$$

Suppose T_1 needs to be rolled back here

T_1 rolls back → T_2 has to be rolled back

Cascadeless Schedules

Cascadeless Schedules

A schedule is **cascadeless** if each transaction in it reads only values that were written by transactions that have already committed.

Cascadeless Schedules

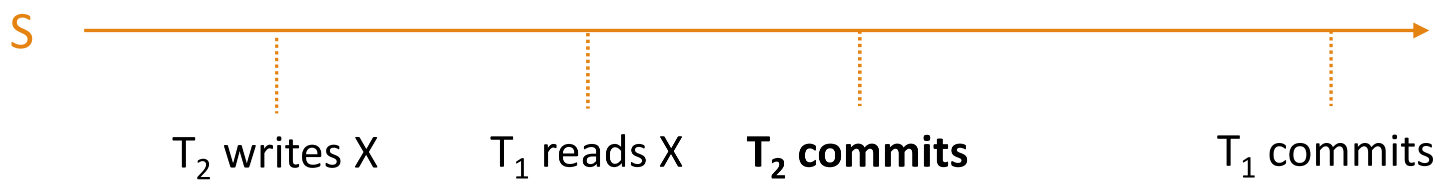
A schedule is **cascadeless** if each transaction in it reads only values that were written by transactions that have already committed.

No reading of “dirty data”.
No cascading rollbacks.

Cascadeless Schedules

A schedule is **cascadeless** if each transaction in it reads only values that were written by transactions that have already committed.

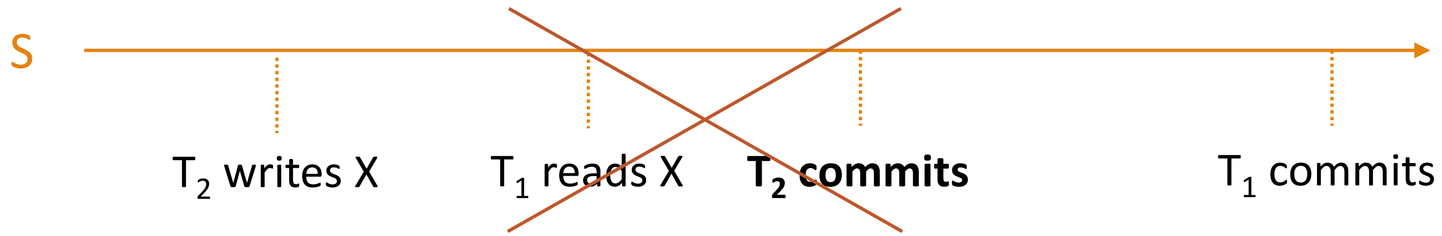
No reading of "dirty data".
No cascading rollbacks.



Cascadeless Schedules

A schedule is **cascadeless** if each transaction in it reads only values that were written by transactions that have already committed.

No reading of “dirty data”.
No cascading rollbacks.



Cascadeless Schedules

A schedule is **cascadeless** if each transaction in it reads only values that were written by transactions that have already committed.

No reading of “dirty data”.
No cascading rollbacks.

Cascadeless Schedules

A schedule is **cascadeless** if each transaction in it reads only values that were written by transactions that have already committed.

No reading of “dirty data”.
No cascading rollbacks.

As for recoverable schedules:
Log records have to reach disk in the right order.

Example

Example

The schedules S_1-S_4 in previous video are **not cascadeless**:

S_1: $w_2(X); w_1(Y); w_1(X); r_2(Y); w_2(Y); c_1; c_2$

S_2: $w_1(X); w_1(Y); w_2(X); r_2(Y); w_2(Y); c_2; c_1$

S_3: $w_2(X); w_1(Y); w_1(X); r_2(Y); w_2(Y); c_2; c_1$

S_4: $w_1(X); w_1(Y); w_2(X); r_2(Y); w_2(Y); c_1; c_2$

Example

The schedules S_1-S_4 in previous video are **not cascadeless**:

reads uncommitted data from T_1

S_1: $w_2(X)$; $w_1(Y)$; $w_1(X)$; $r_2(Y)$; $w_2(Y)$; c_1; c_2

S_2: $w_1(X)$; $w_1(Y)$; $w_2(X)$; $r_2(Y)$; $w_2(Y)$; c_2; c_1

S_3: $w_2(X)$; $w_1(Y)$; $w_1(X)$; $r_2(Y)$; $w_2(Y)$; c_2; c_1

S_4: $w_1(X)$; $w_1(Y)$; $w_2(X)$; $r_2(Y)$; $w_2(Y)$; c_1; c_2

Example

The schedules S_1-S_4 in previous video are **not cascadeless**:

reads uncommitted data from T_1

$$S_1: w_2(X); w_1(Y); w_1(X); r_2(Y); w_2(Y); c_1; c_2$$

$$S_2: w_1(X); w_1(Y); w_2(X); r_2(Y); w_2(Y); c_2; c_1$$

$$S_3: w_2(X); w_1(Y); w_1(X); r_2(Y); w_2(Y); c_2; c_1$$

$$S_4: w_1(X); w_1(Y); w_2(X); r_2(Y); w_2(Y); c_1; c_2$$

This variant of S_1 is **cascadeless**:

$$S_5: w_2(X); w_1(Y); w_1(X); c_1; r_2(Y); w_2(Y); c_2$$

Example

The schedules S_1-S_4 in previous video are **not cascadeless**:

reads uncommitted data from T_1

$$S_1: w_2(X); w_1(Y); w_1(X); r_2(Y); w_2(Y); c_1; c_2$$

$$S_2: w_1(X); w_1(Y); w_2(X); r_2(Y); w_2(Y); c_2; c_1$$

$$S_3: w_2(X); w_1(Y); w_1(X); r_2(Y); w_2(Y); c_2; c_1$$

$$S_4: w_1(X); w_1(Y); w_2(X); r_2(Y); w_2(Y); c_1; c_2$$

This variant of S_1 is **cascadeless**:

$$S_5: w_2(X); w_1(Y); w_1(X); c_1; r_2(Y); w_2(Y); c_2$$

reads committed data from T_1

Example

The schedules S_1-S_4 in previous video are **not cascadeless**:

reads uncommitted data from T_1

$$S_1: w_2(X);\ w_1(Y);\ w_1(X);\ r_2(Y);\ w_2(Y);\ c_1;\ c_2$$

$$S_2: w_1(X);\ w_1(Y);\ w_2(X);\ r_2(Y);\ w_2(Y);\ c_2;\ c_1$$

$$S_3: w_2(X);\ w_1(Y);\ w_1(X);\ r_2(Y);\ w_2(Y);\ c_2;\ c_1$$

$$S_4: w_1(X);\ w_1(Y);\ w_2(X);\ r_2(Y);\ w_2(Y);\ c_1;\ c_2$$

This variant of S_1 is **cascadeless**:

$$S_5: w_2(X);\ w_1(Y);\ w_1(X);\ c_1;\ r_2(Y);\ w_2(Y);\ c_2$$

Note: S_5 is **not serialisable**.

reads committed data from T_1

Cascadeless Schedules: Properties

Cascadeless Schedules: Properties

Cascadeless schedules are **recoverable**:

Cascadeless Schedules: Properties

Cascadeless schedules are **recoverable**:

Cascadeless Schedules: Properties

Cascadeless schedules are **recoverable**:

Cascadeless Schedules: Properties

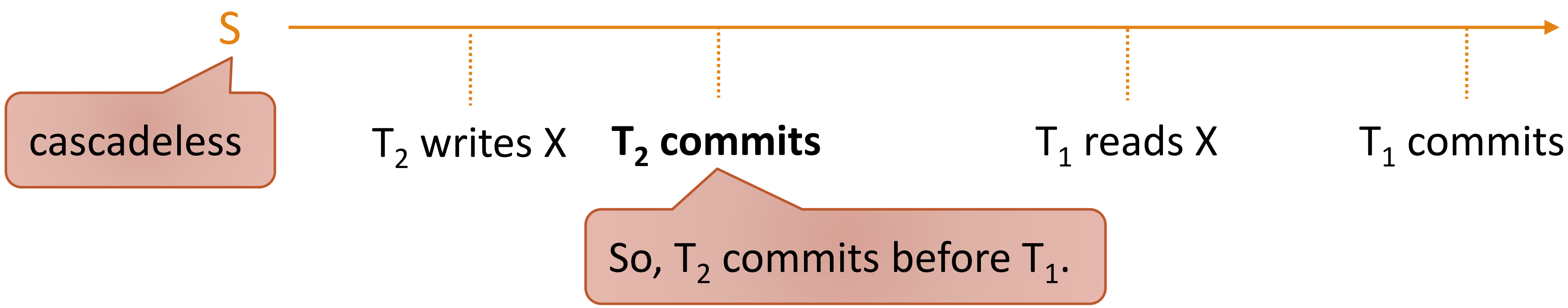
Cascadeless schedules are **recoverable**:

Cascadeless Schedules: Properties

Cascadeless schedules are **recoverable**:

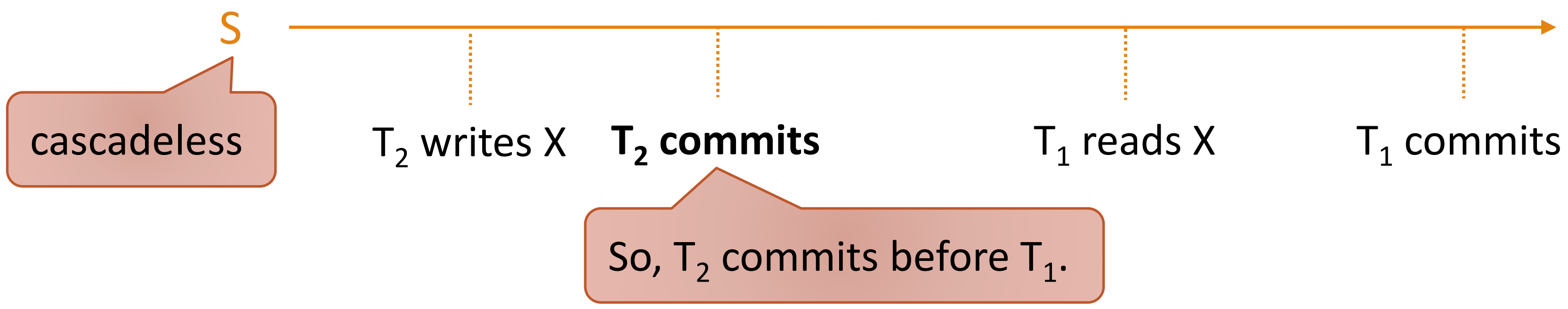
Cascadeless Schedules: Properties

Cascadeless schedules are **recoverable**:



Cascadeless Schedules: Properties

Cascadeless schedules are **recoverable**:



Cascadeless schedules are in general **not serialisable**.
(recall example on previous slide)

Can We Have Both?

Can We Have Both?
No Cascading Aborts & Serialisability?

Strict Schedules

Strict Schedules

A schedule is **strict** if each transaction in it reads and writes only values that were written by transactions that have already committed.

Strict Schedules

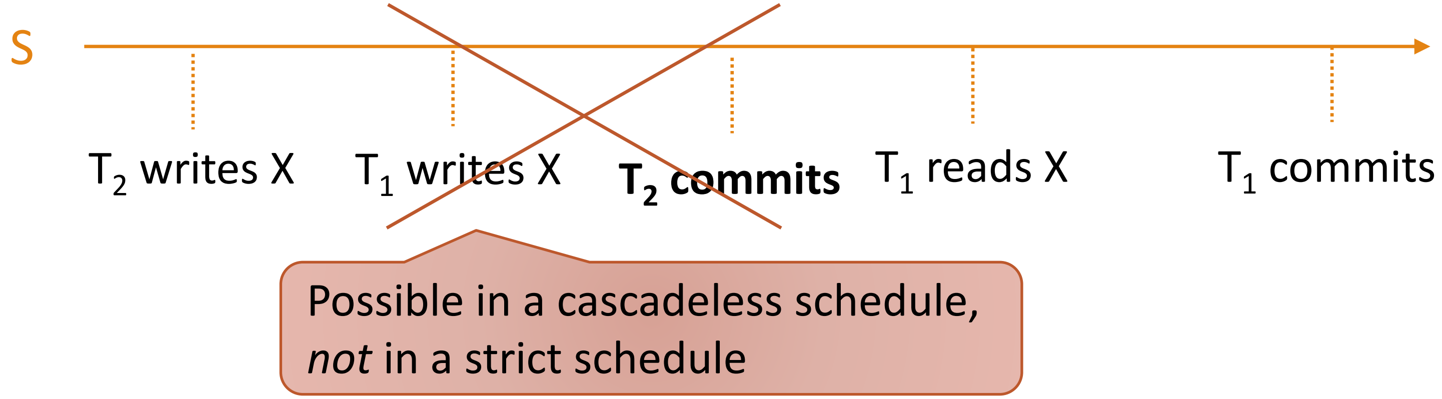
A schedule is **strict** if each transaction in it reads and writes only values that were written by transactions that have already committed.

Strict Schedules

A schedule is **strict** if each transaction in it reads and writes only values that were written by transactions that have already committed.

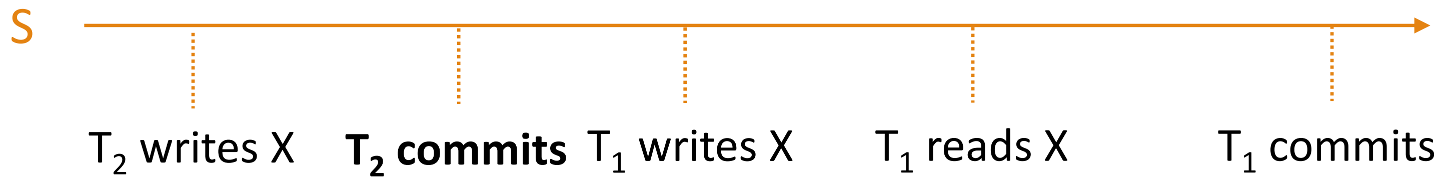
Strict Schedules

A schedule is **strict** if each transaction in it reads and writes only values that were written by transactions that have already committed.



Strict Schedules

A schedule is **strict** if each transaction in it reads and writes only values that were written by transactions that have already committed.



Strict Schedules

A schedule is **strict** if each transaction in it reads and writes only values that were written by transactions that have already committed.

Of course, log records have to reach disk in order.

Strict Two-Phase Locking (Strict 2PL)

Strict Two-Phase Locking (Strict 2PL)

Most popular variant of two-phase locking (2PL)

Strict Two-Phase Locking (Strict 2PL)

Most popular variant of two-phase locking (2PL)

Enforces both:

- Conflict-serialisability
- Strict schedules

Strict Two-Phase Locking (Strict 2PL)

Most popular variant of two-phase locking (2PL)

Enforces both:

- Conflict-serialisability
- Strict schedules

Strict locking condition
(in addition to 2PL condition):

A transaction T **must not release any lock (that allows T to write data)** until:

- T has committed or aborted, and
- the commit/abort log record has been written to disk.

Strict Two-Phase Locking (Strict 2PL)

Most popular variant of two-phase locking (2PL)

Enforces both:

- Conflict-serialisability
- Strict schedules

Strict locking condition
(in addition to 2PL condition):

- with simple locking: **any lock**
- with shared/exclusive locks: **just exclusive locks**

A transaction T **must not release any lock (that allows T to write data)** until:

- T has committed or aborted, and
- the commit/abort log record has been written to disk.

Example 1

Transaction T
lock(X)
read_item(X)
X := X + 100
write_item(X)
lock(Y)
unlock(X)
read_item(Y)
Y := Y + 100
write_item(Y)
unlock(Y)
commit

Example 1

2PL transaction

Transaction T
lock(X)
read_item(X)
X := X + 100
write_item(X)
lock(Y)
unlock(X)
read_item(Y)
Y := Y + 100
write_item(Y)
unlock(Y)
commit

Example 1

2PL transaction

Transaction T
lock(X)
read_item(X)
X := X + 100
write_item(X)
lock(Y)
unlock(X)
read_item(Y)
Y := Y + 100
write_item(Y)
unlock(Y)
commit

For undo logging, we assume that **commit**…

1. Writes all log records to disk
2. Writes all modified database items to disk
3. Writes the commit record to disk

Example 1

2PL transaction

Not strict 2PL

Transaction T
lock(X)
read_item(X)
X := X + 100
write_item(X)
lock(Y)
unlock(X)
read_item(Y)
Y := Y + 100
write_item(Y)
unlock(Y)
commit

For undo logging, we assume that **commit**…

1. Writes all log records to disk
2. Writes all modified database items to disk
3. Writes the commit record to disk

Example 2

2PL transaction

Transaction T
lock(X)
read_item(X)
X := X + 100
write_item(X)
lock(Y)
unlock(X)
read_item(Y)
Y := Y + 100
write_item(Y)
unlock(Y)
commit

New transaction T’
lock(X)
read_item(X)
X := X + 100
write_item(X)
lock(Y)
read_item(Y)
Y := Y + 100
write_item(Y)
commit
unlock(X)
unlock(Y)

Example 2

2PL transaction

Transaction T
lock(X)
read_item(X)
X := X + 100
write_item(X)
lock(Y)
unlock(X)
read_item(Y)
Y := Y + 100
write_item(Y)
unlock(Y)
commit

New transaction T’
lock(X)
read_item(X)
X := X + 100
write_item(X)
lock(Y)
read_item(Y)
Y := Y + 100
write_item(Y)
commit
unlock(X)
unlock(Y)

Example 2

2PL transaction

Transaction T
lock(X)
read_item(X)
X := X + 100
write_item(X)
lock(Y)
unlock(X)
read_item(Y)
Y := Y + 100
write_item(Y)
unlock(Y)
commit

New transaction T’
lock(X)
read_item(X)
X := X + 100
write_item(X)
lock(Y)
read_item(Y)
Y := Y + 100
write_item(Y)
commit
unlock(X)
unlock(Y)

Example 2

2PL transaction

Transaction T
lock(X)
read_item(X)
X := X + 100
write_item(X)
lock(Y)
unlock(X)
read_item(Y)
Y := Y + 100
write_item(Y)
unlock(Y)
commit

New transaction T’
lock(X)
read_item(X)
X := X + 100
write_item(X)
lock(Y)
read_item(Y)
Y := Y + 100
write_item(Y)
commit
unlock(X)
unlock(Y)

Example 2

2PL transaction

Transaction T
lock(X)
read_item(X)
X := X + 100
write_item(X)
lock(Y)
unlock(X)
read_item(Y)
Y := Y + 100
write_item(Y)
unlock(Y)
commit

Strict 2PL transaction

New transaction T’
lock(X)
read_item(X)
X := X + 100
write_item(X)
lock(Y)
read_item(Y)
Y := Y + 100
write_item(Y)
commit
unlock(X)
unlock(Y)

Example 2

2PL transaction

Transaction T
lock(X)
read_item(X)
X := X + 100
write_item(X)
lock(Y)
unlock(X)
read_item(Y)
Y := Y + 100
write_item(Y)
unlock(Y)
commit

Strict 2PL transaction

New transaction T'
lock(X)
read_item(X)
X := X + 100
write_item(X)
lock(Y)
read_item(Y)
Y := Y + 100
write_item(Y)
commit
unlock(X)
unlock(Y)

Locks released only after fully committed, and all log records written to disk

Example With Shared/Exclusive Locks

Transaction T
s-lock(X)
read_item(X)
x-lock(Y)
unlock(X)
read_item(Y)
Y := X + Y
write_item(Y)
commit
unlock(Y)

Example With Shared/Exclusive Locks

Transaction T
s-lock(X)
read_item(X)
x-lock(Y)
unlock(X)
read_item(Y)
Y := X + Y
write_item(Y)
commit
unlock(Y)

Strict 2PL

Example With Shared/Exclusive Locks

Transaction T
s-lock(X)
read_item(X)
x-lock(Y)
unlock(X)
read_item(Y)
Y := X + Y
write_item(Y)
commit
unlock(Y)

Strict 2PL

T can release the shared lock on X here (shared locks do not allow T to write data)

Example With Shared/Exclusive Locks

Transaction T
s-lock(X)
read_item(X)
x-lock(Y)
unlock(X)
read_item(Y)
Y := X + Y
write_item(Y)
commit
unlock(Y)

Strict 2PL

T can release the shared lock on X here (shared locks do not allow T to write data)

T is allowed to release the exclusive lock on Y only here.

Strict Two-Phase Locking *Enforces* Conflict-Serialisable & Strict Schedules

Strict Two-Phase Locking *Enforces* Conflict-Serialisable & Strict Schedules

If S is a schedule consisting of strict 2PL transactions:

- S is conflict-serialisable.
- S is strict.

Strict Two-Phase Locking *Enforces* Conflict-Serialisable & Strict Schedules

If S is a schedule consisting of strict 2PL transactions:

- S is conflict-serialisable.
- S is strict.

Strictness:

Strict Two-Phase Locking *Enforces* Conflict-Serialisable & Strict Schedules

If S is a schedule consisting of strict 2PL transactions:

- S is conflict-serialisable.
- S is strict.

Strictness:

S

Strict Two-Phase Locking *Enforces* Conflict-Serialisable & Strict Schedules

If S is a schedule consisting of strict 2PL transactions:

- S is conflict-serialisable.
- S is strict.

Strictness:

Strict Two-Phase Locking *Enforces* Conflict-Serialisable & Strict Schedules

If S is a schedule consisting of strict 2PL transactions:

- S is conflict-serialisable.
- S is strict.

Strictness:

Strict Two-Phase Locking *Enforces* Conflict-Serialisable & Strict Schedules

If S is a schedule consisting of strict 2PL transactions:

- S is conflict-serialisable.
- S is strict.

Strictness:

Strict Two-Phase Locking *Enforces* Conflict-Serialisable & Strict Schedules

If S is a schedule consisting of strict 2PL transactions:

- S is conflict-serialisable.
- S is strict.

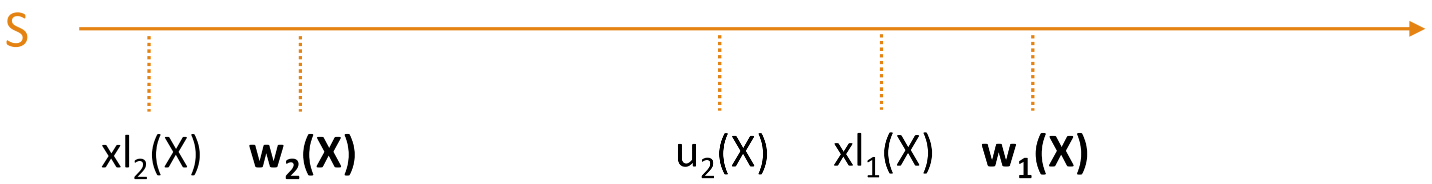
Strictness:

Strict Two-Phase Locking *Enforces* Conflict-Serialisable & Strict Schedules

If S is a schedule consisting of strict 2PL transactions:

- S is conflict-serialisable.
- S is strict.

Strictness:

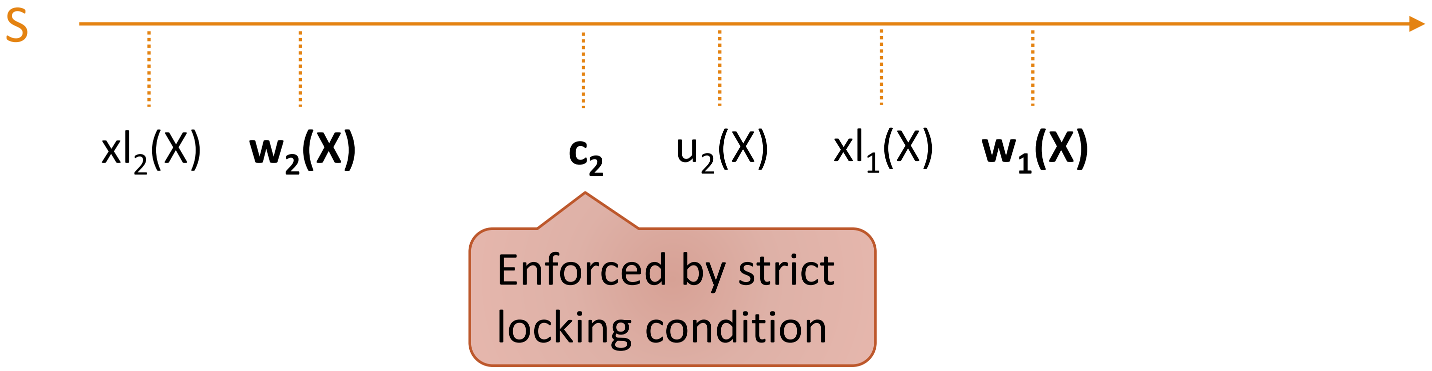


Strict Two-Phase Locking *Enforces* Conflict-Serialisable & Strict Schedules

If S is a schedule consisting of strict 2PL transactions:

- S is conflict-serialisable.
- S is strict.

Strictness:

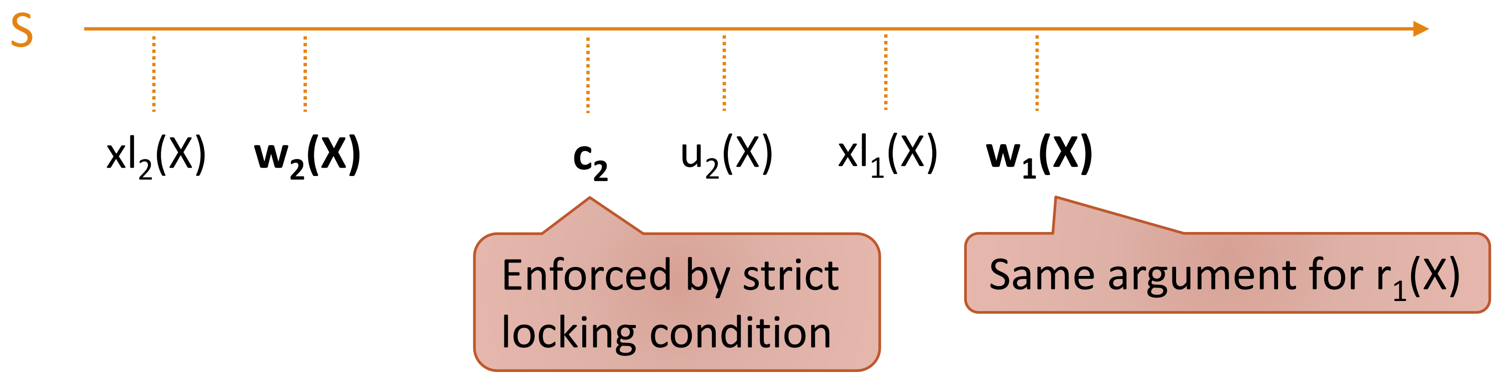


Strict Two-Phase Locking *Enforces* Conflict-Serialisable & Strict Schedules

If S is a schedule consisting of strict 2PL transactions:

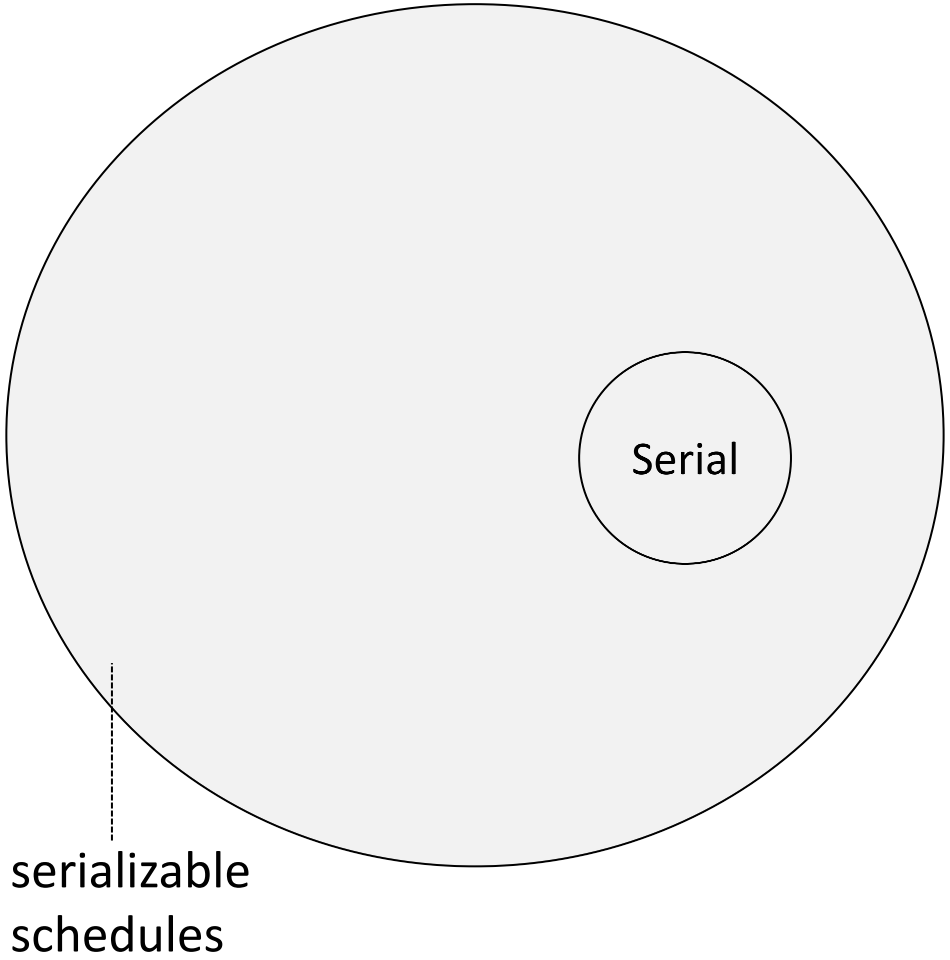
- S is conflict-serialisable.
- S is strict.

Strictness:

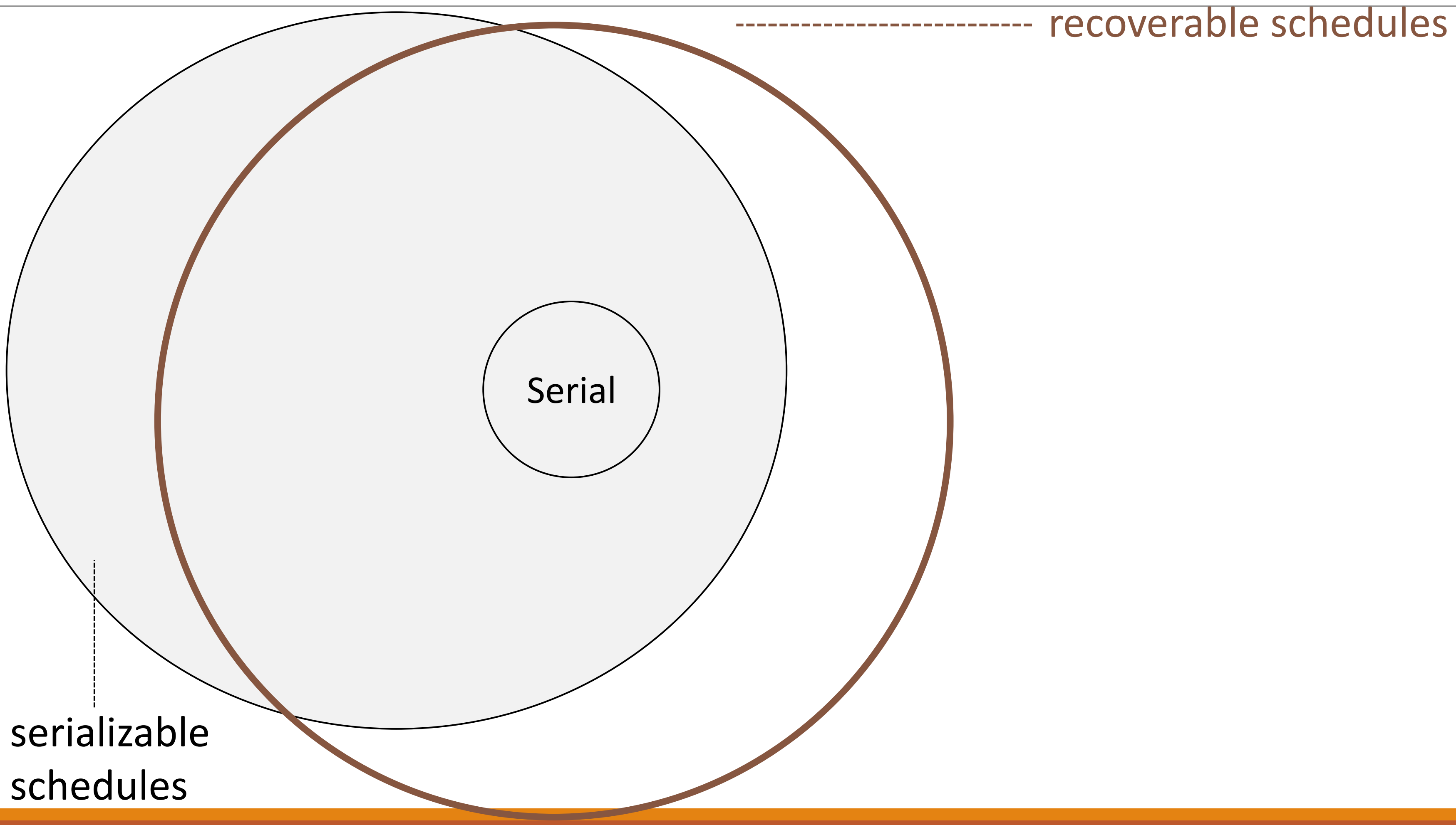


How the Types of Schedules are Related

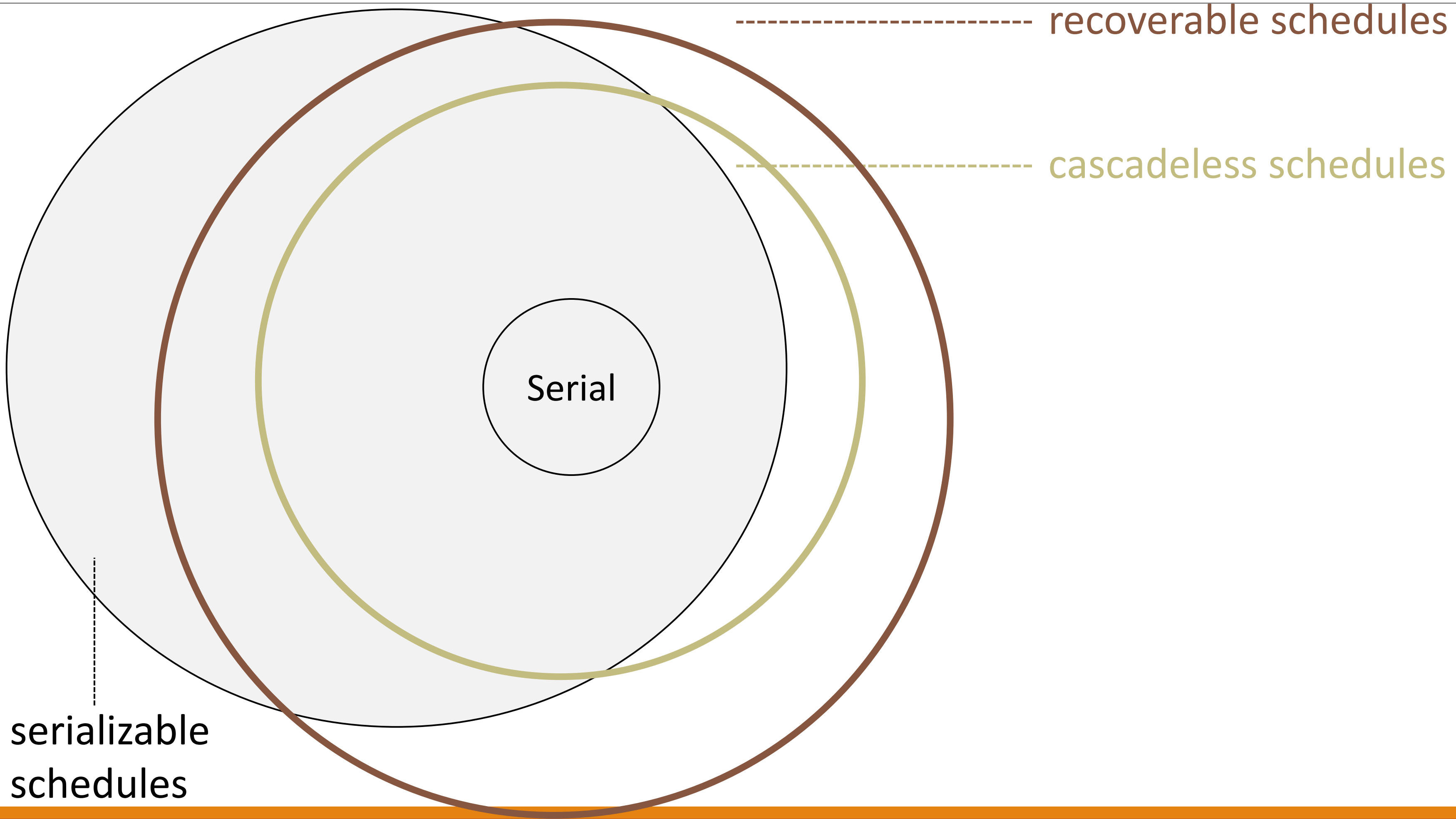
How the Types of Schedules are Related



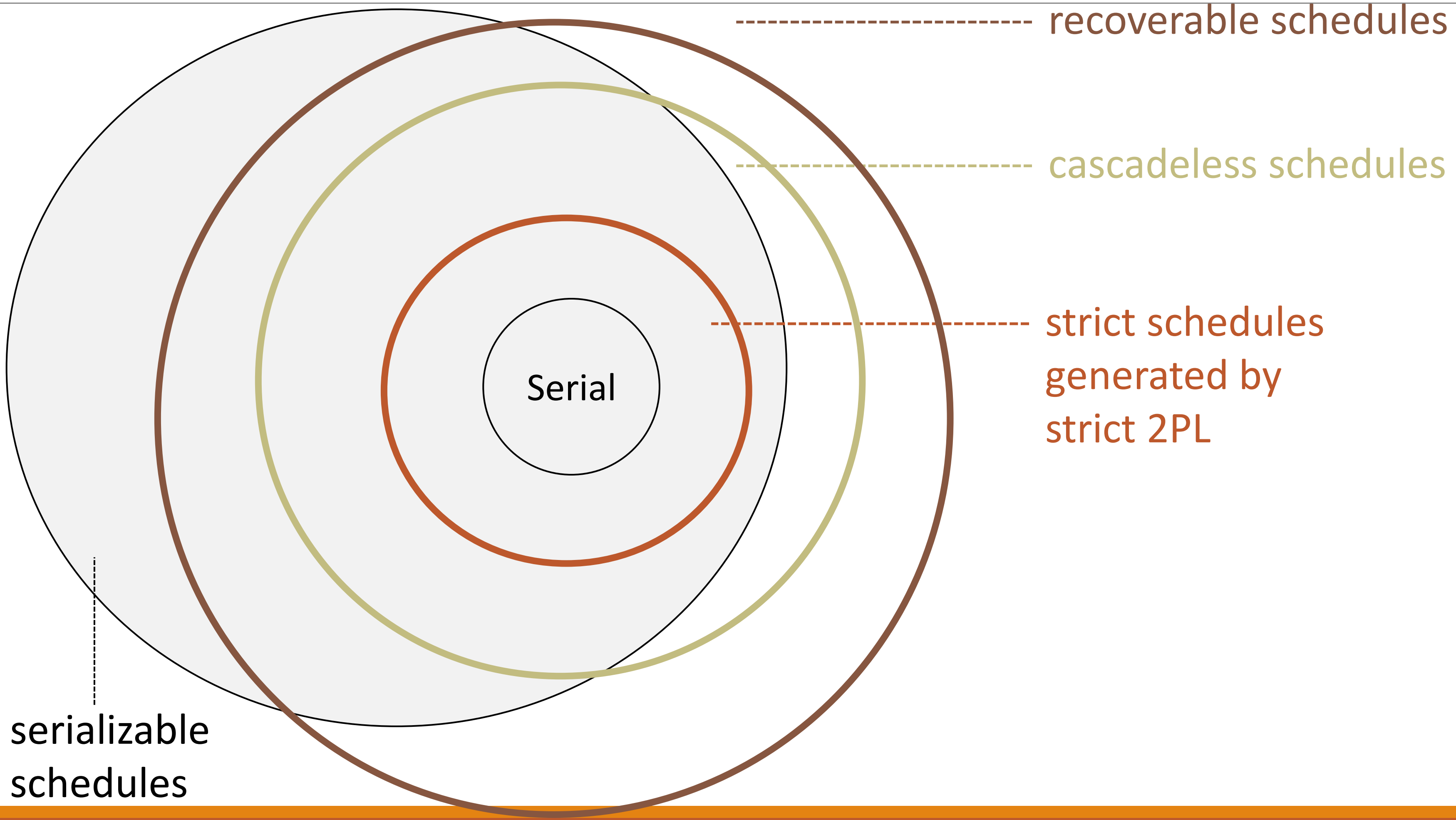
How the Types of Schedules are Related



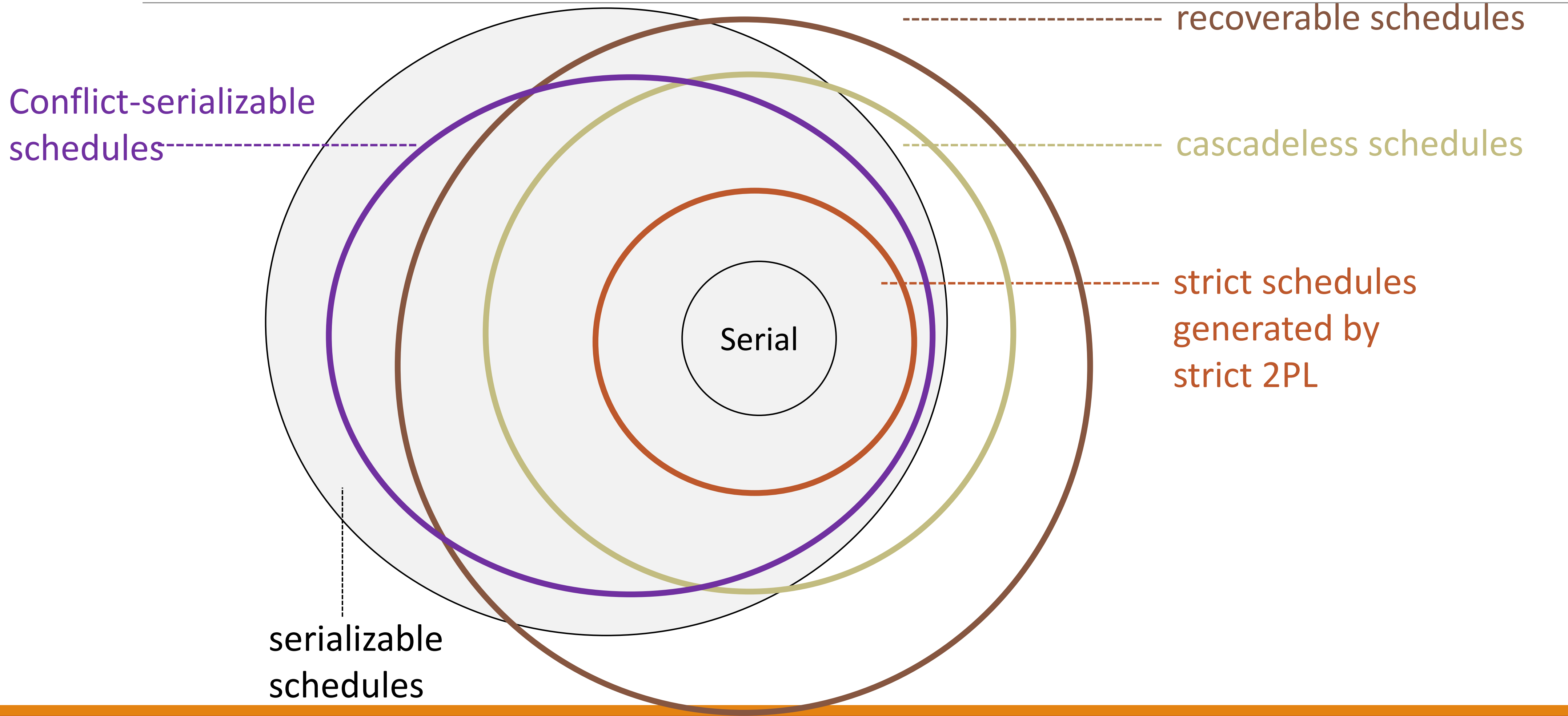
How the Types of Schedules are Related



How the Types of Schedules are Related



How the Types of Schedules are Related



Examples for serializable schedules

Examples for serializable schedules

1. example: $w_2(X); w_1(X); w_1(Y); w_2(Y); r_3(Y); w_3(X); c_3; c_2; c_1$

Examples for serializable schedules

1. example: $w_2(X); w_1(X); w_1(Y); w_2(Y); r_3(Y); w_3(X); c_3; c_2; c_1$
   - This is serializable, but not conflict-serializable or recoverable

Examples for serializable schedules

1. example: $\boxed{w_2(X);}$ $\boxed{w_1(X);}$ $\boxed{w_1(Y);}$ $\boxed{w_2(Y);}$ $r_3(Y)$; $w_3(X)$; c_3; c_2; c_1
   - This is serializable, but not conflict-serializable or recoverable

Examples for serializable schedules

1. example: $\boxed{w_2(X);}$ $\boxed{w_1(X);}$ $\boxed{w_1(Y);}$ $\boxed{w_2(Y);}$ $r_3(Y)$; $w_3(X)$; c_3; c_2; c_1
   - This is serializable, but not conflict-serializable or recoverable

Examples for serializable schedules

1. example: $\boxed{w_2(X);}$ $\boxed{w_1(X);}$ $\boxed{w_1(Y);}$ $\boxed{w_2(Y);}$ $r_3(Y); w_3(X); c_3; c_2; c_1$
   - This is serializable, but not conflict-serializable or recoverable
2. example: $w_1(X); w_1(Y); w_2(X); w_2(Y); r_3(Y); w_3(X); c_3; c_2; c_1$

Examples for serializable schedules

1. example: $\boxed{w_2(X);}\ \boxed{w_1(X);}\ \boxed{w_1(Y);}\ \boxed{w_2(Y);}\ r_3(Y); w_3(X); c_3; c_2; c_1$
   - This is serializable, but not conflict-serializable or recoverable
2. example: $w_1(X); w_1(Y); w_2(X); w_2(Y); r_3(Y); w_3(X); c_3; c_2; c_1$
   - This is conflict-serializable, but not recoverable

Examples for serializable schedules

1. example: $\boxed{w_2(X);}\ \boxed{w_1(X);}\ \boxed{w_1(Y);}\ \boxed{w_2(Y);}\ r_3(Y); w_3(X); c_3; c_2; c_1$
   - This is serializable, but not conflict-serializable or recoverable
2. example: $w_1(X); w_1(Y); w_2(X); w_2(Y); r_3(Y); w_3(X); c_3; c_2; c_1$
   - This is conflict-serializable, but not recoverable

Examples for serializable schedules

1. example: $\boxed{w_2(X);}$ $\boxed{w_1(X);}$ $\boxed{w_1(Y);}$ $\boxed{w_2(Y);}$ $r_3(Y)$; $w_3(X)$; c_3; c_2; c_1
   - This is serializable, but not conflict-serializable or recoverable
2. example: $w_1(X)$; $w_1(Y)$; $w_2(X)$; $w_2(Y)$; $r_3(Y)$; $w_3(X)$; c_3; c_2; c_1
   - This is conflict-serializable, but not recoverable
3. example: $w_2(X)$; $w_1(X)$; $w_1(Y)$; $w_2(Y)$; $r_3(Y)$; $w_3(X)$; c_2; c_3; c_1

Examples for serializable schedules

1. example: $\boxed{w_2(X);}$ $\boxed{w_1(X);}$ $\boxed{w_1(Y);}$ $\boxed{w_2(Y);}$ $r_3(Y); w_3(X); c_3; c_2; c_1$
   - This is serializable, but not conflict-serializable or recoverable
2. example: $w_1(X); w_1(Y); w_2(X); w_2(Y); r_3(Y); w_3(X); c_3; c_2; c_1$
   - This is conflict-serializable, but not recoverable
3. example: $w_2(X); w_1(X); w_1(Y); w_2(Y); r_3(Y); w_3(X); c_2; c_3; c_1$
   - This is recoverable and serializable, but not conflict-serializable nor cascadeless

Examples for serializable schedules

1. example: $\boxed{w_2(X);}$ $\boxed{w_1(X);}$ $\boxed{w_1(Y);}$ $\boxed{w_2(Y);}$ $r_3(Y)$; $w_3(X)$; c_3; c_2; c_1
   - This is serializable, but not conflict-serializable or recoverable
2. example: $w_1(X)$; $w_1(Y)$; $w_2(X)$; $w_2(Y)$; $r_3(Y)$; $w_3(X)$; c_3; c_2; c_1
   - This is conflict-serializable, but not recoverable
3. example: $\boxed{w_2(X);}$ $\boxed{w_1(X);}$ $\boxed{w_1(Y);}$ $\boxed{w_2(Y);}$ $r_3(Y)$; $w_3(X)$; c_2; c_3; c_1
   - This is recoverable and serializable, but not conflict-serializable nor cascadeless

Examples for serializable schedules

1. example: $\boxed{w_2(X);}$ $\boxed{w_1(X);}$ $\boxed{w_1(Y);}$ $\boxed{w_2(Y);}$ $r_3(Y); w_3(X); c_3; c_2; c_1$
   - This is serializable, but not conflict-serializable or recoverable
2. example: $w_1(X); w_1(Y); w_2(X); w_2(Y); r_3(Y); w_3(X); c_3; c_2; c_1$
   - This is conflict-serializable, but not recoverable
3. example: $\boxed{w_2(X);}$ $\boxed{w_1(X);}$ $\boxed{w_1(Y);}$ $\boxed{w_2(Y);}$ $\boxed{r_3(Y);}$ $w_3(X);$ $\boxed{c_2;}$ $c_3; c_1$
   - This is recoverable and serializable, but not conflict-serializable nor cascadeless

Examples for serializable schedules

1. example: $\boxed{w_2(X);}$ $\boxed{w_1(X);}$ $\boxed{w_1(Y);}$ $\boxed{w_2(Y);}$ $r_3(Y);\ w_3(X);\ c_3;\ c_2;\ c_1$
   - This is serializable, but not conflict-serializable or recoverable
2. example: $w_1(X);\ w_1(Y);\ w_2(X);\ w_2(Y);\ r_3(Y);\ w_3(X);\ c_3;\ c_2;\ c_1$
   - This is conflict-serializable, but not recoverable
3. example: $\boxed{w_2(X);}$ $\boxed{w_1(X);}$ $\boxed{w_1(Y);}$ $\boxed{w_2(Y);}$ $\boxed{r_3(Y);}$ $w_3(X);$ $\boxed{c_2;}$ $c_3;\ c_1$
   - This is recoverable and serializable, but not conflict-serializable nor cascadeless
4. example: $w_1(X);\ w_1(Y);\ w_2(X);\ w_2(Y);\ r_3(Y);\ w_3(X);\ c_2;\ c_3;\ c_1$

Examples for serializable schedules

1. example: $w_2(X)$; $w_1(X)$; $w_1(Y)$; $w_2(Y)$; $r_3(Y)$; $w_3(X)$; c_3; c_2; c_1
   - This is serializable, but not conflict-serializable or recoverable
2. example: $w_1(X)$; $w_1(Y)$; $w_2(X)$; $w_2(Y)$; $r_3(Y)$; $w_3(X)$; c_3; c_2; c_1
   - This is conflict-serializable, but not recoverable
3. example: $w_2(X)$; $w_1(X)$; $w_1(Y)$; $w_2(Y)$; $r_3(Y)$; $w_3(X)$; c_2; c_3; c_1
   - This is recoverable and serializable, but not conflict-serializable nor cascadeless
4. example: $w_1(X)$; $w_1(Y)$; $w_2(X)$; $w_2(Y)$; $r_3(Y)$; $w_3(X)$; c_2; c_3; c_1
   - This is recoverable and conflict-serializable, but not cascadeless

Examples for serializable schedules

1. example: $\boxed{w_2(X);}$ $\boxed{w_1(X);}$ $\boxed{w_1(Y);}$ $\boxed{w_2(Y);}$ $r_3(Y); w_3(X); c_3; c_2; c_1$
   - This is serializable, but not conflict-serializable or recoverable
2. example: $w_1(X); w_1(Y); w_2(X); w_2(Y); r_3(Y); w_3(X); c_3; c_2; c_1$
   - This is conflict-serializable, but not recoverable
3. example: $\boxed{w_2(X);}$ $\boxed{w_1(X);}$ $\boxed{w_1(Y);}$ $\boxed{w_2(Y);}$ $\boxed{r_3(Y);}$ $w_3(X);$ $\boxed{c_2;}$ $c_3; c_1$
   - This is recoverable and serializable, but not conflict-serializable nor cascadeless
4. example: $w_1(X); w_1(Y); w_2(X); w_2(Y);$ $\boxed{r_3(Y);}$ $w_3(X);$ $\boxed{c_2;}$ $c_3; c_1$
   - This is recoverable and conflict-serializable, but not cascadeless

Examples for serializable schedules

1. example: $w_2(X); w_1(X); w_1(Y); w_2(Y); r_3(Y); w_3(X); c_3; c_2; c_1$
   - This is serializable, but not conflict-serializable or recoverable
2. example: $w_1(X); w_1(Y); w_2(X); w_2(Y); r_3(Y); w_3(X); c_3; c_2; c_1$
   - This is conflict-serializable, but not recoverable
3. example: $w_2(X); w_1(X); w_1(Y); w_2(Y); r_3(Y); w_3(X); c_2; c_3; c_1$
   - This is recoverable and serializable, but not conflict-serializable nor cascadeless
4. example: $w_1(X); w_1(Y); w_2(X); w_2(Y); r_3(Y); w_3(X); c_2; c_3; c_1$
   - This is recoverable and conflict-serializable, but not cascadeless

5: example: $w_2(X); w_1(X); w_1(Y); w_2(Y); c_2; r_3(Y); w_3(X); c_3; c_1$

Examples for serializable schedules

1. example: $w_2(X); w_1(X); w_1(Y); w_2(Y); r_3(Y); w_3(X); c_3; c_2; c_1$
   - This is serializable, but not conflict-serializable or recoverable
2. example: $w_1(X); w_1(Y); w_2(X); w_2(Y); r_3(Y); w_3(X); c_3; c_2; c_1$
   - This is conflict-serializable, but not recoverable
3. example: $w_2(X); w_1(X); w_1(Y); w_2(Y); r_3(Y); w_3(X); c_2; c_3; c_1$
   - This is recoverable and serializable, but not conflict-serializable nor cascadeless
4. example: $w_1(X); w_1(Y); w_2(X); w_2(Y); r_3(Y); w_3(X); c_2; c_3; c_1$
   - This is recoverable and conflict-serializable, but not cascadeless

5: example: $w_2(X); w_1(X); w_1(Y); w_2(Y); c_2; r_3(Y); w_3(X); c_3; c_1$
   - This is cascadeless and serializable, but not conflict-serializable

Examples for serializable schedules

1. example: $\boxed{w_2(X);}$ $\boxed{w_1(X);}$ $\boxed{w_1(Y);}$ $\boxed{w_2(Y);}$ $r_3(Y)$; $w_3(X)$; c_3; c_2; c_1
   - This is serializable, but not conflict-serializable or recoverable
2. example: $w_1(X)$; $w_1(Y)$; $w_2(X)$; $w_2(Y)$; $r_3(Y)$; $w_3(X)$; c_3; c_2; c_1
   - This is conflict-serializable, but not recoverable
3. example: $\boxed{w_2(X);}$ $\boxed{w_1(X);}$ $\boxed{w_1(Y);}$ $\boxed{w_2(Y);}$ $\boxed{r_3(Y);}$ $w_3(X)$; $\boxed{c_2;}$ c_3; c_1
   - This is recoverable and serializable, but not conflict-serializable nor cascadeless
4. example: $w_1(X)$; $w_1(Y)$; $w_2(X)$; $w_2(Y)$; $\boxed{r_3(Y);}$ $w_3(X)$; $\boxed{c_2;}$ c_3; c_1
   - This is recoverable and conflict-serializable, but not cascadeless

5: example: $\boxed{w_2(X);}$ $\boxed{w_1(X);}$ $\boxed{w_1(Y);}$ $\boxed{w_2(Y);}$ c_2; $r_3(Y)$; $w_3(X)$; c_3; c_1
   - This is cascadeless and serializable, but not conflict-serializable

Examples for serializable schedules

1. example: $\boxed{w_2(X);}$ $\boxed{w_1(X);}$ $\boxed{w_1(Y);}$ $\boxed{w_2(Y);}$ $r_3(Y); w_3(X); c_3; c_2; c_1$
   - This is serializable, but not conflict-serializable or recoverable

2. example: $w_1(X); w_1(Y); w_2(X); w_2(Y); r_3(Y); w_3(X); c_3; c_2; c_1$
   - This is conflict-serializable, but not recoverable

3. example: $\boxed{w_2(X);}$ $\boxed{w_1(X);}$ $\boxed{w_1(Y);}$ $\boxed{w_2(Y);}$ $\boxed{r_3(Y);}$ $w_3(X);$ $\boxed{c_2;}$ $c_3; c_1$
   - This is recoverable and serializable, but not conflict-serializable nor cascadeless

4. example: $w_1(X); w_1(Y); w_2(X); w_2(Y);$ $\boxed{r_3(Y);}$ $w_3(X);$ $\boxed{c_2;}$ $c_3; c_1$
   - This is recoverable and conflict-serializable, but not cascadeless

5: example: $\boxed{w_2(X);}$ $\boxed{w_1(X);}$ $\boxed{w_1(Y);}$ $\boxed{w_2(Y);}$ $c_2; r_3(Y); w_3(X); c_3; c_1$
   - This is cascadeless and serializable, but not conflict-serializable

6: example: $w_1(X); w_1(Y); w_2(X); w_2(Y); c_2; r_3(Y); w_3(X); c_3; c_1$

Examples for serializable schedules

1. example: $\boxed{w_2(X);}$ $\boxed{w_1(X);}$ $\boxed{w_1(Y);}$ $\boxed{w_2(Y);}$ $r_3(Y);\ w_3(X);\ c_3;\ c_2;\ c_1$
   - This is serializable, but not conflict-serializable or recoverable
2. example: $w_1(X);\ w_1(Y);\ w_2(X);\ w_2(Y);\ r_3(Y);\ w_3(X);\ c_3;\ c_2;\ c_1$
   - This is conflict-serializable, but not recoverable
3. example: $\boxed{w_2(X);}$ $\boxed{w_1(X);}$ $\boxed{w_1(Y);}$ $\boxed{w_2(Y);}$ $\boxed{r_3(Y);}$ $w_3(X);$ $\boxed{c_2;}$ $c_3;\ c_1$
   - This is recoverable and serializable, but not conflict-serializable nor cascadeless
4. example: $w_1(X);\ w_1(Y);\ w_2(X);\ w_2(Y);$ $\boxed{r_3(Y);}$ $w_3(X);$ $\boxed{c_2;}$ $c_3;\ c_1$
   - This is recoverable and conflict-serializable, but not cascadeless

5: example: $\boxed{w_2(X);}$ $\boxed{w_1(X);}$ $\boxed{w_1(Y);}$ $\boxed{w_2(Y);}$ $c_2;\ r_3(Y);\ w_3(X);\ c_3;\ c_1$
   - This is cascadeless and serializable, but not conflict-serializable

6: example: $w_1(X);\ w_1(Y);\ w_2(X);\ w_2(Y);\ c_2;\ r_3(Y);\ w_3(X);\ c_3;\ c_1$

Conflict-serializable and cascadeless but not strict

Examples for serializable schedules

1. example: $w_2(X)$; $w_1(X)$; $w_1(Y)$; $w_2(Y)$; $r_3(Y)$; $w_3(X)$; c_3; c_2; c_1
   - This is serializable, but not conflict-serializable or recoverable
2. example: $w_1(X)$; $w_1(Y)$; $w_2(X)$; $w_2(Y)$; $r_3(Y)$; $w_3(X)$; c_3; c_2; c_1
   - This is conflict-serializable, but not recoverable
3. example: $w_2(X)$; $w_1(X)$; $w_1(Y)$; $w_2(Y)$; $r_3(Y)$; $w_3(X)$; c_2; c_3; c_1
   - This is recoverable and serializable, but not conflict-serializable nor cascadeless
4. example: $w_1(X)$; $w_1(Y)$; $w_2(X)$; $w_2(Y)$; $r_3(Y)$; $w_3(X)$; c_2; c_3; c_1
   - This is recoverable and conflict-serializable, but not cascadeless

5: example: $w_2(X)$; $w_1(X)$; $w_1(Y)$; $w_2(Y)$; c_2; $r_3(Y)$; $w_3(X)$; c_3; c_1
   - This is cascadeless and serializable, but not conflict-serializable

6: example: $w_1(X)$; $w_1(Y)$; $w_2(X)$; $w_2(Y)$; c_2; $r_3(Y)$; $w_3(X)$; c_3; c_1

Conflict-serializable and cascadeless but not strict

Still… Risk of Deadlocks

T_1
lock(X)
read_item(X)
X := X + 100
write_item(X)
lock(Y)
read_item(Y)
Y := Y + 100
write_item(Y)
commit
unlock(X)
unlock(Y)

T_2
lock(Y)
read_item(Y)
Y := Y + 100
write_item(Y)
lock(X)
read_item(X)
X := X + 100
write_item(X)
commit
unlock(X)
unlock(Y)

Still… Risk of Deadlocks

T_1
lock(X)
read_item(X)
X := X + 100
write_item(X)
lock(Y)
read_item(Y)
Y := Y + 100
write_item(Y)
commit
unlock(X)
unlock(Y)

T_2
lock(Y)
read_item(Y)
Y := Y + 100
write_item(Y)
lock(X)
read_item(X)
X := X + 100
write_item(X)
commit
unlock(X)
unlock(Y)

$l_1(X); r_1(X); w_1(X);$

Still… Risk of Deadlocks

T_1
lock(X)
read_item(X)
X := X + 100
write_item(X)
lock(Y)
read_item(Y)
Y := Y + 100
write_item(Y)
commit
unlock(X)
unlock(Y)

T_2
lock(Y)
read_item(Y)
Y := Y + 100
write_item(Y)
lock(X)
read_item(X)
X := X + 100
write_item(X)
commit
unlock(X)
unlock(Y)

$l_1(X); r_1(X); w_1(X);$ $l_2(Y); r_2(Y); w_2(Y);$

Still… Risk of Deadlocks

T_1
lock(X)
read_item(X)
X := X + 100
write_item(X)
lock(Y)
read_item(Y)
Y := Y + 100
write_item(Y)
commit
unlock(X)
unlock(Y)

T_2
lock(Y)
read_item(Y)
Y := Y + 100
write_item(Y)
lock(X)
read_item(X)
X := X + 100
write_item(X)
commit
unlock(X)
unlock(Y)

$l_1(X); r_1(X); w_1(X);$ $l_2(Y); r_2(Y); w_2(Y);$ __?__

Still… Risk of Deadlocks

T_1		T_2
lock(X)		lock(Y)
read_item(X)		read_item(Y)
X := X + 100		Y := Y + 100
write_item(X)		write_item(Y)
lock(Y)		lock(X)
read_item(Y)		read_item(X)
Y := Y + 100		X := X + 100
write_item(Y)		write_item(X)
commit		commit
unlock(X)		unlock(X)
unlock(Y)		unlock(Y)

T_2's request for lock on X denied

$l_1(X); r_1(X); w_1(X);\ l_2(Y); r_2(Y); w_2(Y);\ \underline{\ ?\ }$

Still… Risk of Deadlocks

T_1	T_2
lock(X)	lock(Y)
read_item(X)	read_item(Y)
X := X + 100	Y := Y + 100
write_item(X)	write_item(Y)
lock(Y)	lock(X)
read_item(Y)	read_item(X)
Y := Y + 100	X := X + 100
write_item(Y)	write_item(X)
commit	commit
unlock(X)	unlock(X)
unlock(Y)	unlock(Y)

T_2's request for lock on X denied

$l_1(X); r_1(X); w_1(X);$ $l_2(Y); r_2(Y); w_2(Y);$ _?_

T_1's request for lock on Y denied

Strict 2PL and Deadlocks

Strict 2PL yields **conflict-serialisable**, **strict schedules**

Strict 2PL and Deadlocks

Strict 2PL yields **conflict-serialisable**, **strict schedules**

Problem: **deadlocks**

Strict 2PL and Deadlocks

Strict 2PL yields **conflict-serialisable**, **strict schedules**

Problem: **deadlocks**

T_1	T_2
lock(X)	lock(Y)
read_item(X)	read_item(Y)
X := X + 100	Y := Y + 100
write_item(X)	write_item(Y)
lock(Y)	**lock(X)**
…	…

Strict 2PL and Deadlocks

Strict 2PL yields **conflict-serialisable**, **strict schedules**

Problem: **deadlocks**

T_1	T_2
lock(X)	lock(Y)
read_item(X)	read_item(Y)
X := X + 100	Y := Y + 100
write_item(X)	write_item(Y)
lock(Y)	**lock(X)**
…	…

$l_1(X); r_1(X); w_1(X);$

Strict 2PL and Deadlocks

Strict 2PL yields **conflict-serialisable**, **strict schedules**

Problem: **deadlocks**

T_1	T_2
lock(X)	lock(Y)
read_item(X)	read_item(Y)
X := X + 100	Y := Y + 100
write_item(X)	write_item(Y)
lock(Y)	**lock(X)**
…	…

$l_1(X); r_1(X); w_1(X);$

$l_2(Y); r_2(Y); w_2(Y);$

Strict 2PL and Deadlocks

Strict 2PL yields **conflict-serialisable**, **strict schedules**

Problem: **deadlocks**

T_1	T_2
lock(X)	lock(Y)
read_item(X)	read_item(Y)
X := X + 100	Y := Y + 100
write_item(X)	write_item(Y)
lock(Y)	**lock(X)**
…	…

$l_1(X); r_1(X); w_1(X);$

$l_2(Y); r_2(Y); w_2(Y);$?

Strict 2PL and Deadlocks

Strict 2PL yields **conflict-serialisable**, **strict schedules**

Problem: **deadlocks**

T_1	T_2
lock(X)	lock(Y)
read_item(X)	read_item(Y)
X := X + 100	Y := Y + 100
write_item(X)	write_item(Y)
lock(Y)	**lock(X)**
…	…

$l_1(X); r_1(X); w_1(X);$

$l_2(Y); r_2(Y); w_2(Y);$ _?_

Strict 2PL and Deadlocks

Strict 2PL yields **conflict-serialisable**, **strict schedules**

Problem: **deadlocks**

T_1	T_2
lock(X)	lock(Y)
read_item(X)	read_item(Y)
X := X + 100	Y := Y + 100
write_item(X)	write_item(Y)
lock(Y)	**lock(X)**
…	…

$l_1(X); r_1(X); w_1(X);$

$l_2(Y); r_2(Y); w_2(Y);$ _?_

Roll back (and restart) one of the transactions

Strict 2PL and Deadlocks

Strict 2PL yields **conflict-serialisable**, **strict schedules**

Problem: **deadlocks**

T_1	T_2
lock(X)	lock(Y)
read_item(X)	read_item(Y)
X := X + 100	Y := Y + 100
write_item(X)	write_item(Y)
lock(Y)	**lock(X)**
…	…

$l_1(X); r_1(X); w_1(X);$

$l_2(Y); r_2(Y); w_2(Y);$ _?_

Roll back (and restart) one of the transactions

Two approaches for deadlock prevention:

- **Detect deadlocks & fix them**
- **Enforce deadlock-free schedules**

Strict 2PL and Deadlocks

Strict 2PL yields **conflict-serialisable**, **strict schedules**

Problem: **deadlocks**

T_1	T_2
lock(X)	lock(Y)
read_item(X)	read_item(Y)
X := X + 100	Y := Y + 100
write_item(X)	write_item(Y)
lock(Y)	**lock(X)**
…	…

$l_1(X); r_1(X); w_1(X);$

$l_2(Y); r_2(Y); w_2(Y);$?

Roll back (and restart) one of the transactions

Two approaches for deadlock prevention:

- **Detect deadlocks & fix them**
- **Enforce deadlock-free schedules**

Not based on (strict) 2PL

Summary

Cascadeless schedules are a more restrictive form of schedules

- You can only read things that are already committed

Summary

Cascadeless schedules are a more restrictive form of schedules

- You can only read things that are already committed

Instead of implementing that, we use Strict

- You can only read or overwrite things that are already committed

Summary

Cascadeless schedules are a more restrictive form of schedules

- You can only read things that are already committed

Instead of implementing that, we use Strict

- You can only read or overwrite things that are already committed

We then finally use Strict 2PL

- Like 2PL, but where you can only unlock (locks that could write) after commit