Concurrency Control

P.J. McBrien

Imperial College London

Topic 22: Serialisability and Recoverability

P.J. McBrien

Imperial College London

Transactions: ACID properties

ACID properties

database management systems (**DBMS**) implements indivisible tasks called **transactions**

Atomicity	all or nothing
Consistency	consistent before → consistent after
Isolation	independent of any other transaction
Durability	completed transaction are durable

Transactions: ACID properties

ACID properties

database management systems (DBMS) implements indivisible tasks called **transactions**

Atomicity	all or nothing
Consistency	consistent before → consistent after
Isolation	independent of any other transaction
Durability	completed transaction are durable

```
BEGIN TRANSACTION
   UPDATE branch
   SET    cash=cash-10000.00
   WHERE  sortcode=56

   UPDATE branch
   SET    cash=cash+10000.00
   WHERE  sortcode=34
COMMIT TRANSACTION
```

Transactions: ACID properties

ACID properties

database management systems (DBMS) implements indivisible tasks called **transactions**

Atomicity	all or nothing
Consistency	consistent before → consistent after
Isolation	independent of any other transaction
Durability	completed transaction are durable

```
BEGIN TRANSACTION
   UPDATE branch
   SET    cash=cash −10000.00
   WHERE  sortcode=56

   UPDATE branch
   SET    cash=cash +10000.00
   WHERE  sortcode=34
COMMIT TRANSACTION
```

Note that if total cash is £137,246.12 before the transaction, then it will be the same after the transaction.

Example Data

branch		
sortcode	bname	cash
56	'Wimbledon'	94340.45
34	'Goodge St'	8900.67
67	'Strand'	34005.00

movement			
mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

account				
no	type	cname	rate?	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
103	'current'	'Boyd, M.'	NULL	34
107	'current'	'Poulovassilis, A.'	NULL	56
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

key branch(sortcode)
key branch(bname)
key movement(mid)
key account(no)

movement(no) $\overset{fk}{\Rightarrow}$ account(no)

account(sortcode) $\overset{fk}{\Rightarrow}$ branch(sortcode)

Transaction Properties: Atomicity

```
BEGIN TRANSACTION
    UPDATE branch
    SET    cash=cash −10000.00
    WHERE  sortcode=56
```

CRASH

Suppose that the system crashes half way through processing a cash transfer, and the first part of the transfer has been written to disc

- The database on disc is left in an inconsistent state, with £10,000 ‘missing’
- A DBMS implementing **Atomicity** of transactions would on restart UNDO the change to branch 56

Transaction Properties: Consistency

```
BEGIN TRANSACTION
   DELETE FROM branch
   WHERE sortcode=56

   INSERT INTO account
   VALUES (100,'Smith, J','deposit',5.00,34)
END TRANSACTION
```

Suppose that a user deletes branch with sortcode 56, and inserts a deposit account number 100 for John Smith at branch sortcode 34

- The database is left in an inconsistent state for two reasons
  - it has three accounts recorded for a branch that appears not to exist, and
  - it has two records for account number 100, with different details for the account
- A DBMS implementing **Consistency** of transactions would forbid both of these changes to the database

Transaction Properties: Isolation

```
BEGIN TRANSACTION
   UPDATE branch
   SET    cash=cash -10000.00
   WHERE  sortcode=56
```

```
BEGIN TRANSACTION
   SELECT SUM(cash) AS net_cash
   FROM   branch
END TRANSACTION
```

```
   UPDATE branch
   SET    cash=cash +10000.00
   WHERE sortcode=34
END TRANSACTION
```

Suppose that the system sums the cash in the bank in one transaction, half way through processing a cash transfer in another transaction

- The result of the summation of cash in the bank erroneously reports that £10,000 is missing
- A DBMS implementing **Isolation** of transactions ensures that transactions always report results based on the values of committed transactions

Transaction Properties: Durability

```
BEGIN TRANSACTION
   UPDATE branch
   SET    cash=cash −10000.00
   WHERE  sortcode=56

   UPDATE branch
   SET cash=cash+10000.00
   WHERE sortcode=34
END TRANSACTION
```

CRASH

Suppose that the system crashes after informing the user that it has committed the transfer of cash, but has not yet written to disc the update to branch 34

- The database on disc is left in an inconsistent state, with £10,000 'missing'
- A DBMS implementing **Durability** of transactions would on restart complete the change to branch 34 (or alternatively never inform a user of commitment with writing the results to disc).

SQL Conversion to Histories

branch		
sortcode	bname	cash
56	'Wimbledon'	94340.45
34	'Goodge St'	8900.67
67	'Strand'	34005.00

```
BEGIN TRANSACTION T1
   UPDATE branch
   SET    cash=cash −10000.00
   WHERE  sortcode=56

   UPDATE branch
   SET    cash=cash +10000.00
   WHERE  sortcode=34
COMMIT TRANSACTION T1
```

⇒

$H_1 = r_1[b_{56}]$, cash=94340.45,
$w_1[b_{56}]$, cash=84340.45,
$r_1[b_{34}]$, cash=8900.67,
$w_1[b_{34}]$, cash=18900.67, c_1

history of transaction T_n

1. Begin transaction b_n (only given if necessary for discussion)
2. Various read operations on objects $r_n[o_j]$ and write operations $w_n[o_j]$
3. Either c_n for the commitment of the transaction, or a_n for the abort of the transaction

SQL Conversion to Histories

branch		
sortcode	bname	cash
56	'Wimbledon'	84340.45
34	'Goodge St'	18900.67
67	'Strand'	34005.00

```
BEGIN TRANSACTION
    UPDATE branch
    SET    cash=cash −2000.00
    WHERE  sortcode=34

    UPDATE branch
    SET    cash=cash +2000.00
    WHERE  sortcode=67
COMMIT TRANSACTION
```

⇨

$H_2 = r_2[b_{34}]$, cash=18900.67,
$w_2[b_{34}]$, cash=16900.67,
$r_2[b_{67}]$, cash=34005.00,
$w_2[b_{67}]$, cash=36005.00, c_2

history of transaction T_n

- Same pattern of transaction code gives same pattern of operations

Serial Execution

Serial Execution of Transactions

- Executing one transaction at a time
- Provided updates are recorded in stable storage at the time of c_i, must maintain the ACID properties

Serial Execution

Serial Execution of Transactions

- Executing one transaction at a time
- Provided updates are recorded in stable storage at the time of c_i, must maintain the ACID properties

Possible Serial Executions

$H_1 = r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1$

$H_2 = r_2[b_{34}], w_2[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2$

Serial Execution

Serial Execution of Transactions

- Executing one transaction at a time
- Provided updates are recorded in stable storage at the time of c_i, must maintain the ACID properties

Possible Serial Executions

$H_1 = r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1$

$H_2 = r_2[b_{34}], w_2[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2$

The only two possible serial executions are

$H_{s12} = r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1, r_2[b_{34}], w_2[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2$

$H_{s21} = r_2[b_{34}], w_2[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2, r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1$

Concurrent Execution

Concurrent Execution of Transactions

- Interleaving of several transaction histories
- Order of operations within each history preserved

Concurrent Execution

Concurrent Execution of Transactions

- Interleaving of several transaction histories
- Order of operations within each history preserved

$H_1 = r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1$

$H_2 = r_2[b_{34}], w_2[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2$

Concurrent Execution

Concurrent Execution of Transactions

- Interleaving of several transaction histories
- Order of operations within each history preserved

$H_1 = r_1[b56], w_1[b56], r_1[b34], w_1[b34], c_1$

$H_2 = r_2[b34], w_2[b34], r_2[b67], w_2[b67], c_2$

Some possible concurrent executions are

$H_x = r_2[b34], r_1[b56], w_1[b56], r_1[b34], w_1[b34], c_1, w_2[b34], r_2[b67], w_2[b67], c_2$

$H_y = r_2[b34], w_2[b34], r_1[b56], w_1[b56], r_1[b34], w_1[b34], r_2[b67], w_2[b67], c_2, c_1$

$H_z = r_2[b34], w_2[b34], r_1[b56], w_1[b56], r_1[b34], w_1[b34], c_1, r_2[b67], w_2[b67], c_2$

Which concurrent executions should be allowed?

Concurrency control $\rightarrow$ controlling interaction

Which concurrent executions should be allowed?

Concurrency control → controlling interaction

serialisability

A concurrent execution of transactions should always have the same final result as some serial execution of those same transactions

Which concurrent executions should be allowed?

Concurrency control → controlling interaction

serialisability

A concurrent execution of transactions should always have the same final result as some serial execution of those same transactions

recoverability

No transaction commits depending on data that has been produced by another transaction that has yet to commit

Which concurrent executions should be allowed?

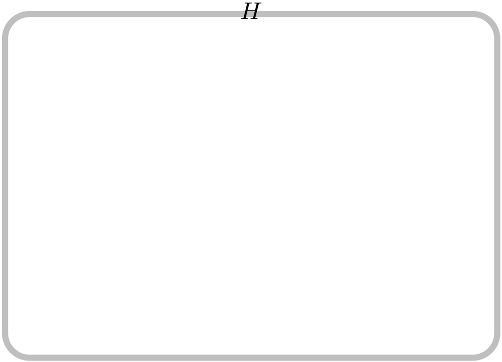
Concurrency control → controlling interaction

serialisability

A concurrent execution of transactions should always have the same final result as some serial execution of those same transactions

recoverability

No transaction commits depending on data that has been produced by another transaction that has yet to commit



H set of all possible histories

Which concurrent executions should be allowed?

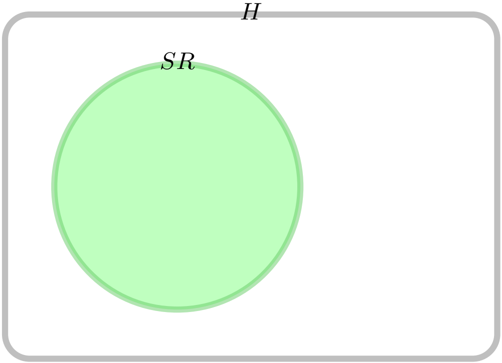
Concurrency control → controlling interaction

serialisability

A concurrent execution of transactions should always have the same final result as some serial execution of those same transactions

recoverability

No transaction commits depending on data that has been produced by another transaction that has yet to commit



H set of all possible histories

SR set of serialisable histories

Which concurrent executions should be allowed?

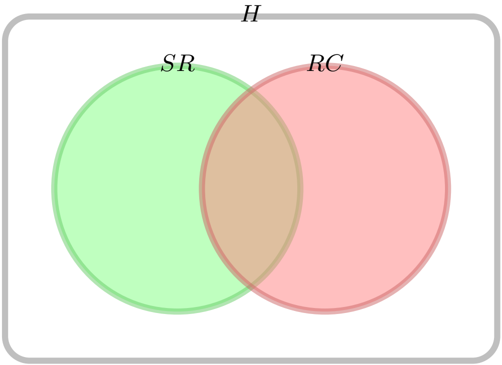
Concurrency control → controlling interaction

serialisability

A concurrent execution of transactions should always have the same final result as some serial execution of those same transactions

recoverability

No transaction commits depending on data that has been produced by another transaction that has yet to commit



H set of all possible histories

SR set of serialisable histories

RC set of recoverable histories

Quiz 22.1: Serialisability and Recoverability (1)

$H_x = r_2[b34], r_1[b56], w_1[b56], r_1[b34], w_1[b34], c_1, w_2[b34], r_2[b67], w_2[b67], c_2$

Is H_x

A

Not Serialisable, Not Recoverable

B

Not Serialisable, Recoverable

C

Serialisable, Not Recoverable

D

Serialisable, Recoverable

Quiz 22.2: Serialisability and Recoverability (2)

$H_y = r_2[b34], w_2[b34], r_1[b56], w_1[b56], r_1[b34], w_1[b34], r_2[b67], w_2[b67], c_2, c_1$

Is H_y

A

Not Serialisable, Not Recoverable

B

Not Serialisable, Recoverable

C

Serialisable, Not Recoverable

D

Serialisable, Recoverable

Quiz 22.3: Serialisability and Recoverability (3)

$H_z = r_2[b_{34}], w_2[b_{34}], r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1, r_2[b_{67}], w_2[b_{67}], c_2$

Is H_z

A

Not Serialisable, Not Recoverable

B

Not Serialisable, Recoverable

C

Serialisable, Not Recoverable

D

Serialisable, Recoverable

Topic 23: Anomalies in Transaction Execution

P.J. McBrien

Imperial College London

Anomaly 1: Lost Update

BEGIN TRANSACTION T1
EXEC move_cash(56,34,10000.00)
COMMIT TRANSACTION T1

BEGIN TRANSACTION T2
EXEC move_cash(34,67,2000.00)
COMMIT TRANSACTION T2

$r_1[b_{56}]$, $w_1[b_{56}]$, $r_1[b_{34}]$, $w_1[b_{34}]$, c_1

$r_2[b_{34}]$, $w_2[b_{34}]$, $r_2[b_{67}]$, $w_2[b_{67}]$, c_2

$r_1[b_{56}]$, cash=94340.45, $w_1[b_{56}]$, cash=84340.45, $r_1[b_{34}]$, cash=8900.67,
$r_2[b_{34}]$, cash=8900.67, $w_1[b_{34}]$, cash=18900.67, c_1, $w_2[b_{34}]$, cash=6900.67,
$r_2[b_{67}]$, cash=34005.00, $w_2[b_{67}]$, cash=36005.25, c_2

Anomaly 1: Lost Update

BEGIN TRANSACTION T1
EXEC move_cash(56,34,10000.00)
COMMIT TRANSACTION T1

BEGIN TRANSACTION T2
EXEC move_cash(34,67,2000.00)
COMMIT TRANSACTION T2

$r_1[b_{56}]$, $w_1[b_{56}]$, $r_1[b_{34}]$, $w_1[b_{34}]$, c_1

$r_2[b_{34}]$, $w_2[b_{34}]$, $r_2[b_{67}]$, $w_2[b_{67}]$, c_2

$r_1[b_{56}]$, cash=94340.45, $w_1[b_{56}]$, cash=84340.45, $r_1[b_{34}]$, cash=8900.67,
$r_2[b_{34}]$, cash=8900.67, lost update, c_1, $w_2[b_{34}]$, cash=6900.67,
$r_2[b_{67}]$, cash=34005.00, $w_2[b_{67}]$, cash=36005.25, c_2

− serialisable

+ recoverable

LU = set of histories with a lost update
$SR \cap LU = \emptyset$

Anomaly 2: Inconsistent analysis

BEGIN TRANSACTION T1
EXEC move_cash(56,34,10000.00)
COMMIT TRANSACTION T1

BEGIN TRANSACTION T4
SELECT SUM(cash) FROM branch
COMMIT TRANSACTION T4

$r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1$

$H_4 = r_4[b_{56}], r_4[b_{34}], r_4[b_{67}], c_4$

$r_1[b_{56}]$, cash=94340.45, $w_1[b_{56}]$, cash=84340.45, $r_4[b_{56}]$, cash=84340.45, $r_4[b_{34}]$, cash=8900.67, $r_4[b_{67}]$, cash=34005.00, $r_1[b_{34}]$, cash=8900.67, $w_1[b_{34}]$, cash=18900.67, c_1, c_4

− serialisable

+ recoverable

IA = set of histories with an inconsistent analysis
$SR \cap IA = \emptyset$

Anomaly 3: Dirty Reads

BEGIN TRANSACTION T1	BEGIN TRANSACTION T2
EXEC move_cash(56,34,10000.00)	EXEC move_cash(34,67,2000.00)
COMMIT TRANSACTION T1	COMMIT TRANSACTION T2

⇓

$r_1[b_{56}]$, $w_1[b_{56}]$, $r_1[b_{34}]$, $w_1[b_{34}]$, c_1

$r_2[b_{34}]$, $w_2[b_{34}]$, $r_2[b_{67}]$, $w_2[b_{67}]$, c_2

⇓

$r_1[b_{56}]$, cash=94340.45, $w_1[b_{56}]$, cash=84340.45, $r_2[b_{34}]$, cash=8900.67, $w_2[b_{34}]$, cash=6900.67, $r_1[b_{34}]$, cash=6900.67, $w_1[b_{34}]$, cash=16900.67, c_1, $r_2[b_{67}]$, cash=34005.00, $w_2[b_{67}]$, cash=36005.25, a_2

+ serialisable

− recoverable

DR = set of histories with a dirty read

$RC \cap DR \neq \emptyset$

Quiz 23.1: Anomalies (1)

$H_x = r_2[b_{34}], r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1, w_2[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2$

Which anomaly does H_x suffer?

A
None

B
Lost Update

C
Inconsistent Analysis

D
Dirty Read

Quiz 23.2: Anomalies (2)

$H_y = r_2[b34], w_2[b34], r_1[b56], w_1[b56], r_1[b34], w_1[b34], r_2[b67], w_2[b67], c_2, c_1$

Which anomaly does H_y suffer?

A
None

B
Lost Update

C
Inconsistent Analysis

D
Dirty Read

Quiz 23.3: Anomalies (3)

$H_z = r_2[b_{34}], w_2[b_{34}], r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1, r_2[b_{67}], w_2[b_{67}], c_2$

Which anomaly does H_z suffer?

A

None

B

Lost Update

C

Inconsistent Analysis

D

Dirty Read

Account Table

account				
no	type	cname	rate?	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
103	'current'	'Boyd, M.'	NULL	34
107	'current'	'Poulovassilis, A.'	NULL	56
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

Anomaly 3: Dirty Reads (Recoverable Example)

BEGIN TRANSACTION T1
EXEC move_cash(56,34,10000.00)
COMMIT TRANSACTION T1

BEGIN TRANSACTION T2
EXEC move_cash(34,67,2000.00)
COMMIT TRANSACTION T2

$r_1[b_{56}]$, $w_1[b_{56}]$, $r_1[b_{34}]$, $w_1[b_{34}]$, c_1

$r_2[b_{34}]$, $w_2[b_{34}]$, $r_2[b_{67}]$, $w_2[b_{67}]$, c_2

$r_1[b_{56}]$, cash=94340.45, $w_1[b_{56}]$, cash=84340.45, $r_2[b_{34}]$, cash=8900.67, $w_2[b_{34}]$, cash=6900.67, $r_1[b_{34}]$, cash=6900.67, $w_1[b_{34}]$, cash=16900.67, $r_2[b_{67}]$, cash=34005.00, $w_2[b_{67}]$, cash=36005.25, c_2, c_1

Anomaly 3: Dirty Reads (Recoverable Example)

BEGIN TRANSACTION T1
EXEC move_cash(56,34,10000.00)
COMMIT TRANSACTION T1

BEGIN TRANSACTION T2
EXEC move_cash(34,67,2000.00)
COMMIT TRANSACTION T2

$r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1$

$r_2[b_{34}], w_2[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2$

$r_1[b_{56}]$, cash=94340.45, $w_1[b_{56}]$, cash=84340.45, $r_2[b_{34}]$, cash=8900.67,
$w_2[b_{34}]$, cash=6900.67, $r_1[b_{34}]$, cash=6900.67, $w_1[b_{34}]$, cash=16900.67,
$r_2[b_{67}]$, cash=34005.00, $w_2[b_{67}]$, cash=36005.25, c_2, c_1

+ serialisable
+ recoverable

DR = set of histories with a dirty read
$RC \cap DR \neq \emptyset \wedge RC \cup DR = H$

Anomaly 4: Dirty Writes

```
BEGIN TRANSACTION T5
    UPDATE account
    SET rate=5.5
    WHERE type='deposit'
COMMIT TRANSACTION T5
```

```
BEGIN TRANSACTION T6
    UPDATE account
    SET rate=6.0
    WHERE type='deposit'
COMMIT TRANSACTION T6
```

$H_5 = w_5[a_{101}]$, rate=5.5, $w_5[a_{119}]$, rate=5.5, c_5

$H_6 = w_6[a_{101}]$, rate=6.0, $w_6[a_{119}]$, rate=6.0, c_6

$w_6[a_{101}]$, rate=6.0, $w_5[a_{101}]$, rate=5.5, $w_5[a_{119}]$, rate=5.5, $w_6[a_{119}]$, rate=6.0, c_5, c_6

Anomaly 4: Dirty Writes

BEGIN TRANSACTION T5	BEGIN TRANSACTION T6
UPDATE account	UPDATE account
SET rate=5.5	SET rate=6.0
WHERE type='deposit'	WHERE type='deposit'
COMMIT TRANSACTION T5	COMMIT TRANSACTION T6

$H_5 = w_5[a_{101}]$, rate=5.5, $w_5[a_{119}]$, rate=5.5, c_5

$H_6 = w_6[a_{101}]$, rate=6.0, $w_6[a_{119}]$, rate=6.0, c_6

$w_6[a_{101}]$, rate=6.0, $w_5[a_{101}]$, rate=5.5, $w_5[a_{119}]$, rate=5.5, $w_6[a_{119}]$, rate=6.0, c_5, c_6

− serialisable

\+ recoverable

WR = set of histories with a dirty write

$SR \cap WR \neq \emptyset$

Patterns of operations associated with Anomalies

Anomaly	Set	Pattern	Problem
Dirty Write	DW	$w_1[o] \prec w_2[o] \prec e_1$	Sometimes not SR
Dirty Read	DR	$w_1[o] \prec r_2[o] \prec e_1$	Sometimes not RC
Inconsistent Analysis	IA	$r_1[o_a] \prec w_2[o_a],\ w_2[o_b] \prec r_1[o_b]$	Not SR
Lost Update	LU	$r_1[o] \prec w_2[o] \prec w_1[o]$	Not SR

Notation

- e_i means either c_i or a_i occurring
- $op_a \prec op_b$ mean op_a occurs before op_b in a history

Worksheet: Anomalies

`rental_charge`

$H_1 = r_1[d_{1000}], w_1[d_{1000}], r_1[d_{1001}], w_1[d_{1001}], r_1[d_{1002}], w_1[d_{1002}]$

`transfer_charge`

$H_2 = r_2[d_{1000}], w_2[d_{1000}], r_2[d_{1002}], w_2[d_{1002}]$

`total_charge`

$H_3 = r_3[d_{1000}], r_3[d_{1001}], r_3[d_{1002}]$

Movement and Account Tables

movement			
mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

account				
no	type	cname	rate?	sor
100	'current'	'McBrien, P.'	NULL	
101	'deposit'	'McBrien, P.'	5.25	
103	'current'	'Boyd, M.'	NULL	
107	'current'	'Poulovassilis, A.'	NULL	
119	'deposit'	'Poulovassilis, A.'	5.50	
125	'current'	'Bailey, J.'	NULL	

Anomaly 5: Phantom Read

account				
no	type	cname	rate?	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
103	'current'	'Boyd, M.'	NULL	34
107	'current'	'Poulovassilis, A.'	NULL	56
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

```
BEGIN TRANSACTION T9
  INSERT INTO movement
  SELECT  1011,w_account.no,-7000.00,'21-jan-1999'
  FROM    movement NATURAL JOIN account
          JOIN account w_account
          ON account.cname=w_account.cname
  WHERE   w_account.no=101
  GROUP BY w_account.no
  HAVING SUM(amount)>=7000.00;
COMMIT TRANSACTION T9
```

```
BEGIN TRANSACTION T10
  INSERT INTO movement
  SELECT  1012,w_account.no,-2200.00,'21-jan-1999'
  FROM    movement NATURAL JOIN account
          JOIN account w_account
          ON account.cname=w_account.cname
  WHERE   w_account.no=100
  GROUP BY w_account.no
  HAVING SUM(amount)>=2200.00;
COMMIT TRANSACTION T10
```

$r_9[a_{101}]$, $r_9[a_{100}]$, $r_9[m_{1000}]$, $r_9[m_{1001}]$, $r_9[m_{1002}]$, $r_9[m_{1006}]$, $r_9[m_{1008}]$,
$r_{10}[a_{100}]$, $r_{10}[a_{101}]$, $r_{10}[m_{1000}]$, $r_{10}[m_{1001}]$, $r_{10}[m_{1002}]$, $r_{10}[m_{1006}]$, $r_{10}[m_{1008}]$,
$ins_9[m_{1011}]$, $ins_{10}[m_{1012}]$, c_9, c_{10}

Anomaly 5: Phantom Read

account				
no	type	cname	rate?	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
103	'current'	'Boyd, M.'	NULL	34
107	'current'	'Poulovassilis, A.'	NULL	56
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

```
BEGIN TRANSACTION T9
  INSERT INTO movement
  SELECT  1011,w_account.no,-7000.00,'21-jan-1999'
  FROM    movement NATURAL JOIN account
          JOIN account w_account
          ON account.cname=w_account.cname
  WHERE   w_account.no=101
  GROUP BY w_account.no
  HAVING SUM(amount)>=7000.00;
COMMIT TRANSACTION T9
```

```
BEGIN TRANSACTION T10
  INSERT INTO movement
  SELECT  1012,w_account.no,-2200.00,'21-jan-1999'
  FROM    movement NATURAL JOIN account
          JOIN account w_account
          ON account.cname=w_account.cname
  WHERE   w_account.no=100
  GROUP BY w_account.no
  HAVING SUM(amount)>=2200.00;
COMMIT TRANSACTION T10
```

$r_9[Match(101)], r_{10}[Match(100)], ins_9[m_{1011}], ins_{10}[m_{1012}], c_9, c_{10}$

Anomaly 6: Write Skew

account				
no	type	cname	rate?	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
103	'current'	'Boyd, M.'	NULL	34
107	'current'	'Poulovassilis, A.'	NULL	56
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

```
BEGIN TRANSACTION T11
   UPDATE account
   SET     rate=max_rate
   FROM    (SELECT MAX(rate) AS max_rate
            FROM account) AS max_data
   WHERE   rate<max_rate
COMMIT TRANSACTION T11
```

```
BEGIN TRANSACTION T12
   UPDATE account
   SET     rate=min_rate
   FROM    (SELECT MIN(rate) AS min_rate
            FROM account) AS min_data
   WHERE   rate>min_rate
COMMIT TRANSACTION T12
```

$r_{11}[a_{101}]$, $r_{11}[a_{119}]$, $r_{12}[a_{101}]$, $r_{12}[a_{119}]$, $w_{11}[a_{101}]$, $w_{12}[a_{119}]$, c_{11}, c_{12}

Topic 24: Serialisable Execution

P.J. McBrien

Imperial College London

Serialisable Transaction Execution

- Solve anomalies $\rightarrow H \equiv$ serial execution
- Only interested in the **committed projection**

$$H_c = r_1[b_{56}], r_2[b_{34}], w_2[b_{34}], r_3[m_{1000}], r_3[m_{1001}], r_3[m_{1002}], w_1[b_{56}], r_4[b_{56}], r_3[m_{1003}], r_3[m_{1004}], r_3[m_{1005}], r_1[b_{34}], a_3, w_1[b_{34}], c_1, r_4[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2, r_4[b_{67}], c_4$$

$\Rightarrow$

$$C(H_c) = r_1[b_{56}], r_2[b_{34}], w_2[b_{34}], w_1[b_{56}], r_4[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1, r_4[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2, r_4[b_{67}], c_4$$

Possible Serial Equivalents

$H_{cp} = r_1[b_{56}], r_2[b_{34}], w_2[b_{34}], w_1[b_{56}], r_4[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1, r_4[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2, r_4[b_{67}], c_4$

⇓ H_1, H_2, H_4 | ⇓ H_1, H_4, H_2 | ⇓ H_2, H_1, H_4 | ⇓ H_2, H_4, H_1 | ⇓ H_4, H_1, H_2 | ⇓ H_4, H_2, H_1

- how to determine that histories are equivalent?
- how to check this during execution?

View Equivalence & View Serialisable

Want database to end up in the same final state as if serial execution had occurred

View Equivalence

Two histories H_i, H_j will leave the database in the same final state if, at each commit

If the above are met, we say the histories are **view equivalent** $H_i \equiv_{VE} H_j$

View Equivalence & View Serialisable

Want database to end up in the same final state as if serial execution had occurred

View Equivalence

Two histories H_i, H_j will leave the database in the same final state if, at each commit

1. H_i, H_j have same set of operations

If the above are met, we say the histories are **view equivalent** $H_i \equiv_{VE} H_j$

View Equivalence & View Serialisable

Want database to end up in the same final state as if serial execution had occurred

View Equivalence

Two histories H_i, H_j will leave the database in the same final state if, at each commit

1. H_i, H_j have same set of operations
2. $w_t[o] \rightarrow r_s[o]$ in both H_i, H_j or in neither

If the above are met, we say the histories are **view equivalent** $H_i \equiv_{VE} H_j$

View Equivalence & View Serialisable

Want database to end up in the same final state as if serial execution had occurred

View Equivalence

Two histories H_i, H_j will leave the database in the same final state if, at each commit

1. H_i, H_j have same set of operations
2. $w_t[o] \rightarrow r_s[o]$ in both H_i, H_j or in neither
3. Last committed write on o is $w_t[o]$ in both H_i, H_j or in neither

If the above are met, we say the histories are **view equivalent** $H_i \equiv_{VE} H_j$

View Equivalence & View Serialisable

Want database to end up in the same final state as if serial execution had occurred

View Equivalence

Two histories H_i, H_j will leave the database in the same final state if, at each commit

1. H_i, H_j have same set of operations
2. $w_t[o] \rightarrow r_s[o]$ in both H_i, H_j or in neither
3. Last committed write on o is $w_t[o]$ in both H_i, H_j or in neither

If the above are met, we say the histories are **view equivalent** $H_i \equiv_{VE} H_j$

View Serialisable Histories

a history H is **view serialisable** (**VSR**) if $C(H) \equiv_{VE}$ a serial history

Testing for View Equivalence

$$H_{cp} = r_1[b_{56}], r_2[b_{34}], w_2[b_{34}], w_1[b_{56}], r_4[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1, r_4[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2, r_4[b_{67}], c_4$$

$$\equiv$$

$$H_2, H_1, H_4 = r_2[b_{34}], w_2[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2, r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1, r_4[b_{56}], r_4[b_{34}], r_4[b_{67}], c_4$$

1. H_{cp} and H_2, H_1, H_4 same set of operations
2. reads are $w_2[b_{34}] \rightarrow r_1[b_{34}]$, $w_2[b_{67}] \rightarrow r_4[b_{67}]$, $w_1[b_{34}] \rightarrow r_4[b_{34}]$, $w_1[b_{56}] \rightarrow r_4[b_{56}]$
3. last committed writes are $w_1[b_{34}]$, $w_1[b_{56}]$, and $w_2[b_{67}]$

$H_2, H_1, H_4 \equiv_{VE} H_{cp} \quad \rightarrow \quad H_{cp} \in VSR$

Conflicts: Potential For Problems

conflict

A **conflict** occurs when there is an interaction between two transactions

- $r_x[o]$ and $w_y[o]$ are in H where $x \neq y$ or
- $w_x[o]$ and $w_y[o]$ are in H where $x \neq y$

Only consider pairs where there is no third operation $rw_z[o]$ between the pair of operations that conflicts with both

conflicts

$H_x = r_2[b_{34}], r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1, w_2[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2$

$H_y = r_2[b_{34}], w_2[b_{34}], r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2, c_1$

$H_z = r_2[b_{34}], w_2[b_{34}], r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1, r_2[b_{67}], w_2[b_{67}], c_2$

Conflicts

Conflicts: Potential For Problems

conflict

A **conflict** occurs when there is an interaction between two transactions

- $r_x[o]$ and $w_y[o]$ are in H where $x \neq y$ or
- $w_x[o]$ and $w_y[o]$ are in H where $x \neq y$

Only consider pairs where there is no third operation $rw_z[o]$ between the pair of operations that conflicts with both

conflicts

$H_x = r_2[b_{34}], r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1, w_2[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2$

$H_y = r_2[b_{34}], w_2[b_{34}], r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2, c_1$

$H_z = r_2[b_{34}], w_2[b_{34}], r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1, r_2[b_{67}], w_2[b_{67}], c_2$

Conflicts

- $w_2[b_{34}] \rightarrow r_1[b_{34}]$ T1 reads from T2 in H_y, H_z

Conflicts: Potential For Problems

conflict

A **conflict** occurs when there is an interaction between two transactions

- $r_x[o]$ and $w_y[o]$ are in H where $x \neq y$ or
- $w_x[o]$ and $w_y[o]$ are in H where $x \neq y$

Only consider pairs where there is no third operation $rw_z[o]$ between the pair of operations that conflicts with both

conflicts

$H_x = r_2[b_{34}], r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1, w_2[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2$

$H_y = r_2[b_{34}], w_2[b_{34}], r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2, c_1$

$H_z = r_2[b_{34}], w_2[b_{34}], r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1, r_2[b_{67}], w_2[b_{67}], c_2$

Conflicts

- $w_2[b_{34}] \rightarrow r_1[b_{34}]$ T1 reads from T2 in H_y, H_z
- $w_1[b_{34}] \rightarrow w_2[b_{34}]$ T2 writes over T1 in H_x

Conflicts: Potential For Problems

conflict

A **conflict** occurs when there is an interaction between two transactions

- $r_x[o]$ and $w_y[o]$ are in H where $x \neq y$ or
- $w_x[o]$ and $w_y[o]$ are in H where $x \neq y$

Only consider pairs where there is no third operation $rw_z[o]$ between the pair of operations that conflicts with both

conflicts

$H_x = r_2[b_{34}], r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1, w_2[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2$

$H_y = r_2[b_{34}], w_2[b_{34}], r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2, c_1$

$H_z = r_2[b_{34}], w_2[b_{34}], r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1, r_2[b_{67}], w_2[b_{67}], c_2$

Conflicts

- $w_2[b_{34}] \rightarrow r_1[b_{34}]$ T1 reads from T2 in H_y, H_z
- $w_1[b_{34}] \rightarrow w_2[b_{34}]$ T2 writes over T1 in H_x
- $r_2[b_{34}] \rightarrow w_1[b_{34}]$ T1 writes after T2 reads in H_x

Quiz 24.1: Conflicts

$H_w =$
$r_2[a_{100}]$, $w_2[a_{100}]$, $r_2[a_{107}]$, $r_1[a_{119}]$, $w_1[a_{119}]$, $r_1[a_{107}]$, $w_1[a_{107}]$, c_1, $w_2[a_{107}]$, c_2

Which of the following is not a conflict in H_w?

A

$r_2[a_{107}] \rightarrow r_1[a_{107}]$

B

$r_2[a_{107}] \rightarrow w_1[a_{107}]$

C

$r_1[a_{107}] \rightarrow w_2[a_{107}]$

D

$w_1[a_{107}] \rightarrow w_2[a_{107}]$

Conflict Equivalence and Conflict Serialisable

Conflict Equivalence

Two histories H_i and H_j are **conflict equivalent** if:

1. Contain the same set of operations
2. Order conflicts (of non-aborted transactions) in the same way.

Conflict Serialisable

a history H is **conflict serialisable** (**CSR**) if $C(H) \equiv_{CE}$ a serial history

Failure to be conflict serialisable

$H_x = r_2[b34], r_1[b56], w_1[b56], r_1[b34], w_1[b34], c_1, w_2[b34], r_2[b67], w_2[b67], c_2$

Contains conflicts $r_2[b34] \rightarrow w_1[b34]$ and $w_1[b34] \rightarrow w_2[b34]$ and so is not conflict equivalence to H_1, H_2 nor H_2, H_1, and hence is not conflict serialisable.

Serialisation Graph

Serialisation Graph

A **serialisation graph** $SG(H)$ contains a node for each transaction in H, and an edge $T_i \rightarrow T_j$ if there is some object o for which a conflict $rw_i[o] \rightarrow rw_j[o]$ exists in H. If $SG(H)$ is acyclic, then H is conflict serialisable.

Demonstrating that a History is CSR

Given H_{cp}= $r_1[b_{56}]$, $r_2[b_{34}]$, $w_2[b_{34}]$, $w_1[b_{56}]$, $r_4[b_{56}]$, $r_1[b_{34}]$, $w_1[b_{34}]$, c_1, $r_4[b_{34}]$, $r_2[b_{67}]$, $w_2[b_{67}]$, c_2, $r_4[b_{67}]$, c_4

Serialisation Graph

Serialisation Graph

A **serialisation graph** $SG(H)$ contains a node for each transaction in H, and an edge $T_i \rightarrow T_j$ if there is some object o for which a conflict $rw_i[o] \rightarrow rw_j[o]$ exists in H. If $SG(H)$ is acyclic, then H is conflict serialisable.

Demonstrating that a History is CSR

Given H_{cp}= $r_1[b_{56}]$, $r_2[b_{34}]$, $w_2[b_{34}]$, $w_1[b_{56}]$, $r_4[b_{56}]$, $r_1[b_{34}]$, $w_1[b_{34}]$, c_1, $r_4[b_{34}]$, $r_2[b_{67}]$, $w_2[b_{67}]$, c_2, $r_4[b_{67}]$, c_4

Conflicts are $w_2[b_{34}]$ $\rightarrow$ $r_1[b_{34}]$,

$SG(H_{cp})$

Serialisation Graph

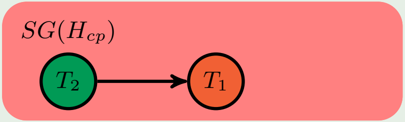
Serialisation Graph

A **serialisation graph** $SG(H)$ contains a node for each transaction in H, and an edge $T_i \rightarrow T_j$ if there is some object o for which a conflict $rw_i[o] \rightarrow rw_j[o]$ exists in H. If $SG(H)$ is acyclic, then H is conflict serialisable.

Demonstrating that a History is CSR

Given H_{cp}= $r_1[b_{56}]$, $r_2[b_{34}]$, $w_2[b_{34}]$, $w_1[b_{56}]$, $r_4[b_{56}]$, $r_1[b_{34}]$, $w_1[b_{34}]$, c_1, $r_4[b_{34}]$, $r_2[b_{67}]$, $w_2[b_{67}]$, c_2, $r_4[b_{67}]$, c_4

Conflicts are $w_2[b_{34}] \rightarrow r_1[b_{34}]$,



Serialisation Graph

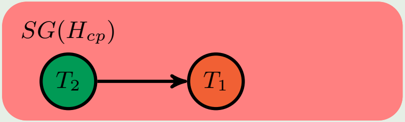
Serialisation Graph

A **serialisation graph** $SG(H)$ contains a node for each transaction in H, and an edge $T_i \rightarrow T_j$ if there is some object o for which a conflict $rw_i[o] \rightarrow rw_j[o]$ exists in H. If $SG(H)$ is acyclic, then H is conflict serialisable.

Demonstrating that a History is CSR

Given H_{cp}= $r_1[b_{56}]$, $r_2[b_{34}]$, $w_2[b_{34}]$, $w_1[b_{56}]$, $r_4[b_{56}]$, $r_1[b_{34}]$, $w_1[b_{34}]$, c_1, $r_4[b_{34}]$, $r_2[b_{67}]$, $w_2[b_{67}]$, c_2, $r_4[b_{67}]$, c_4

Conflicts are $w_2[b_{34}] \rightarrow r_1[b_{34}]$, $w_1[b_{56}] \rightarrow r_4[b_{56}]$,



Serialisation Graph

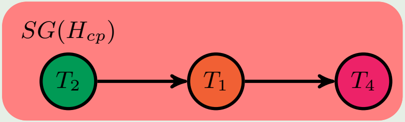
Serialisation Graph

A **serialisation graph** $SG(H)$ contains a node for each transaction in H, and an edge $T_i \rightarrow T_j$ if there is some object o for which a conflict $rw_i[o] \rightarrow rw_j[o]$ exists in H. If $SG(H)$ is acyclic, then H is conflict serialisable.

Demonstrating that a History is CSR

Given H_{cp}= $r_1[b56]$, $r_2[b34]$, $w_2[b34]$, $w_1[b56]$, $r_4[b56]$, $r_1[b34]$, $w_1[b34]$, c_1, $r_4[b34]$, $r_2[b67]$, $w_2[b67]$, c_2, $r_4[b67]$, c_4

Conflicts are $w_2[b34] \rightarrow r_1[b34]$, $w_1[b56] \rightarrow r_4[b56]$,



Serialisation Graph

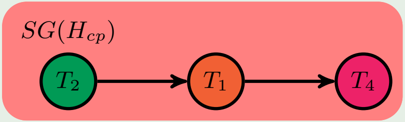
Serialisation Graph

A **serialisation graph** $SG(H)$ contains a node for each transaction in H, and an edge $T_i \rightarrow T_j$ if there is some object o for which a conflict $rw_i[o] \rightarrow rw_j[o]$ exists in H. If $SG(H)$ is acyclic, then H is conflict serialisable.

Demonstrating that a History is CSR

Given H_{cp}= $r_1[b_{56}]$, $r_2[b_{34}]$, $w_2[b_{34}]$, $w_1[b_{56}]$, $r_4[b_{56}]$, $r_1[b_{34}]$, $w_1[b_{34}]$, c_1, $r_4[b_{34}]$, $r_2[b_{67}]$, $w_2[b_{67}]$, c_2, $r_4[b_{67}]$, c_4

Conflicts are $w_2[b_{34}] \rightarrow r_1[b_{34}]$, $w_1[b_{56}] \rightarrow r_4[b_{56}]$, $w_1[b_{34}] \rightarrow r_4[b_{34}]$,



Serialisation Graph

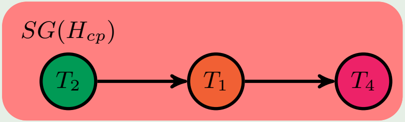
Serialisation Graph

A **serialisation graph** $SG(H)$ contains a node for each transaction in H, and an edge $T_i \rightarrow T_j$ if there is some object o for which a conflict $rw_i[o] \rightarrow rw_j[o]$ exists in H. If $SG(H)$ is acyclic, then H is conflict serialisable.

Demonstrating that a History is CSR

Given H_{cp}= $r_1[b56]$, $r_2[b34]$, $w_2[b34]$, $w_1[b56]$, $r_4[b56]$, $r_1[b34]$, $w_1[b34]$, c_1, $r_4[b34]$, $r_2[b67]$, $w_2[b67]$, c_2, $r_4[b67]$, c_4

Conflicts are $w_2[b34] \rightarrow r_1[b34]$, $w_1[b56] \rightarrow r_4[b56]$, $w_1[b34] \rightarrow r_4[b34]$, $w_2[b67] \rightarrow r_4[b67]$



Serialisation Graph

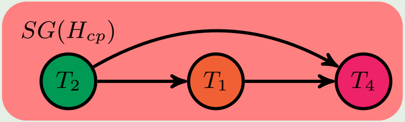
Serialisation Graph

A **serialisation graph** $SG(H)$ contains a node for each transaction in H, and an edge $T_i \rightarrow T_j$ if there is some object o for which a conflict $rw_i[o] \rightarrow rw_j[o]$ exists in H. If $SG(H)$ is acyclic, then H is conflict serialisable.

Demonstrating that a History is CSR

Given $H_{cp}=$ $r_1[b_{56}]$, $r_2[b_{34}]$, $w_2[b_{34}]$, $w_1[b_{56}]$, $r_4[b_{56}]$, $r_1[b_{34}]$, $w_1[b_{34}]$, c_1, $r_4[b_{34}]$, $r_2[b_{67}]$, $w_2[b_{67}]$, c_2, $r_4[b_{67}]$, c_4

Conflicts are $w_2[b_{34}] \rightarrow r_1[b_{34}]$, $w_1[b_{56}] \rightarrow r_4[b_{56}]$, $w_1[b_{34}] \rightarrow r_4[b_{34}]$, $w_2[b_{67}] \rightarrow r_4[b_{67}]$



Serialisation Graph

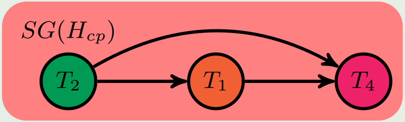
Serialisation Graph

A **serialisation graph** $SG(H)$ contains a node for each transaction in H, and an edge $T_i \rightarrow T_j$ if there is some object o for which a conflict $rw_i[o] \rightarrow rw_j[o]$ exists in H. If $SG(H)$ is acyclic, then H is conflict serialisable.

Demonstrating that a History is CSR

Given H_{cp}= $r_1[b_{56}]$, $r_2[b_{34}]$, $w_2[b_{34}]$, $w_1[b_{56}]$, $r_4[b_{56}]$, $r_1[b_{34}]$, $w_1[b_{34}]$, c_1, $r_4[b_{34}]$, $r_2[b_{67}]$, $w_2[b_{67}]$, c_2, $r_4[b_{67}]$, c_4

Conflicts are $w_2[b_{34}] \rightarrow r_1[b_{34}]$, $w_1[b_{56}] \rightarrow r_4[b_{56}]$, $w_1[b_{34}] \rightarrow r_4[b_{34}]$, $w_2[b_{67}] \rightarrow r_4[b_{67}]$



$SG(H_{cp})$ is acyclic, therefore H_{cp} is CSR. Serialisation order T_2, T_1, T_4

Worksheet: Serialisability

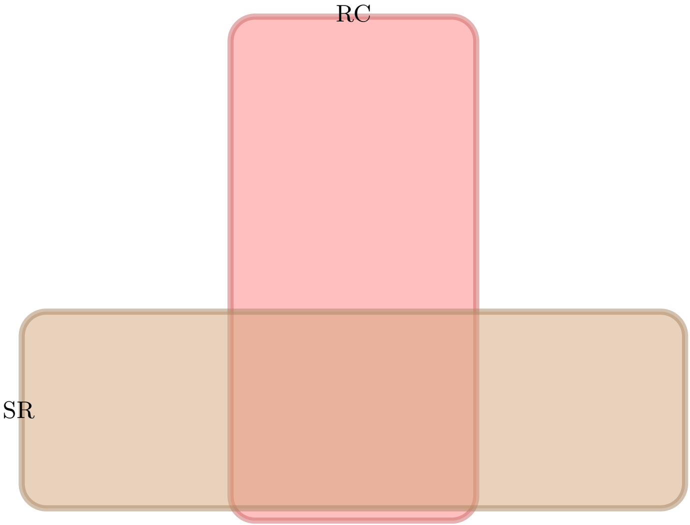
$H_1 = r_1[o_1], w_1[o_1], w_1[o_2], w_1[o_3], c_1$

$H_2 = r_2[o_2], w_2[o_2], w_2[o_1], c_2$

$H_3 = r_3[o_1], w_3[o_1], w_3[o_2], c_3$

$H = r_1[o_1], w_1[o_1], r_2[o_2], w_2[o_2], w_2[o_1], c_2, w_1[o_2], r_3[o_1], w_3[o_1], w_3[o_2], c_3, w_1[o_3], c_1$

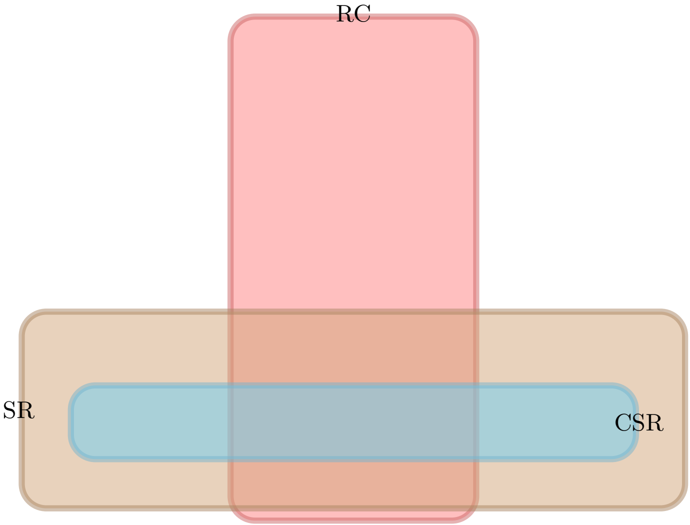
Review of Serialisable Histories



RC recoverable

SR serialisable

Review of Serialisable Histories

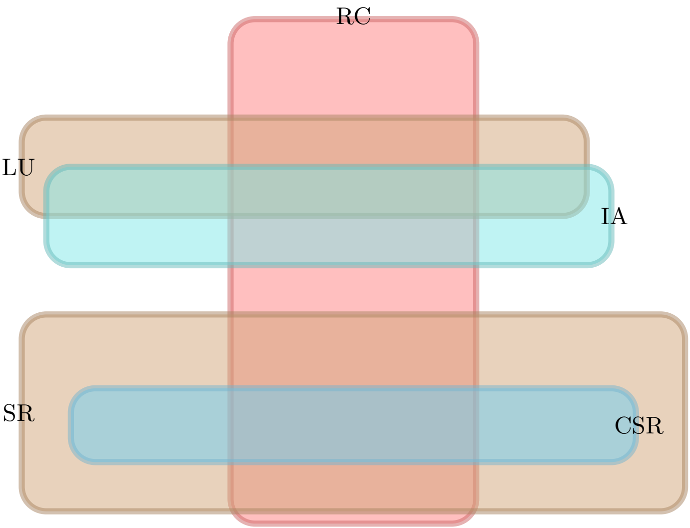


RC recoverable

SR serialisable

CSR conflict serialisable

Review of Serialisable Histories

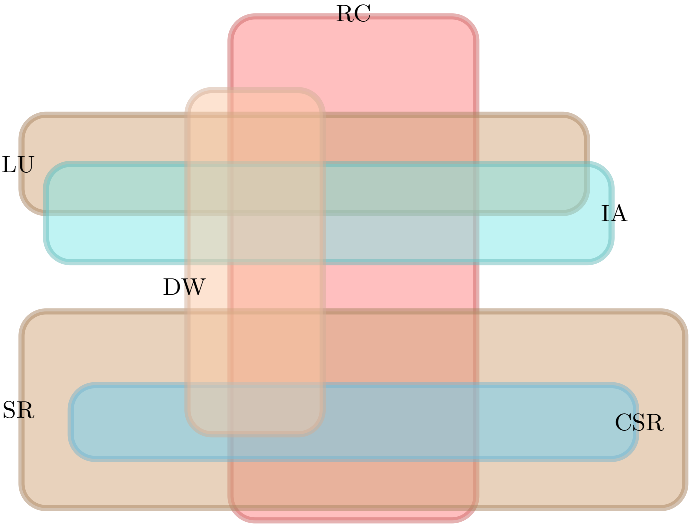


RC recoverable

SR serialisable
CSR conflict serialisable

LU lost update
IA inconsistent analysis

Review of Serialisable Histories



RC recoverable

DW dirty write

SR serialisable
CSR conflict serialisable

LU lost update
IA inconsistent analysis

Topic 25: Recoverable Execution

P.J. McBrien

Imperial College London

Recoverability

- Serialisability necessary for isolation and consistency of committed transactions
- Recoverability necessary for isolation and consistency when there are also aborted transactions

Recoverable execution

A **recoverable** (**RC**) history H has no transaction committing before another transaction from which it read

Execution avoiding cascading aborts

A history which **avoids cascading aborts** (**ACA**) does not read from a non-committed transaction

Strict execution

A **strict** (**ST**) history does not read from a non-committed transaction nor write over a non-committed transaction

$ST \subset ACA \subset RC$

Non-recoverable executions

```
BEGIN TRANSACTION T1
    UPDATE branch
    SET cash=cash-10000.00
    WHERE sortcode=56
    UPDATE branch
    SET cash=cash+10000.00
    WHERE sortcode=34
COMMIT TRANSACTION T1
```

```
BEGIN TRANSACTION T4
    SELECT SUM(cash) FROM branch
COMMIT TRANSACTION T4
```

$H_1 = r_1[b_{56}], w_1[b_{56}], a_1$

$H_4 = r_4[b_{56}], r_4[b_{34}], r_4[b_{67}], c_4$

$H_c = r_1[b_{56}]$, cash=94340.45, $w_1[b_{56}]$, cash=84340.45, $r_4[b_{56}]$, cash=84340.45, $r_4[b_{34}]$, cash=8900.67, $r_4[b_{67}]$, cash=34005.00, c_4, a_1

$H_c \notin RC$

Cascading Aborts

```
BEGIN TRANSACTION T1
    UPDATE branch
    SET cash=cash-10000.00
    WHERE sortcode=56
    UPDATE branch
    SET cash=cash+10000.00
    WHERE sortcode=34
COMMIT TRANSACTION T1
```

```
BEGIN TRANSACTION T4
    SELECT SUM(cash) FROM branch
COMMIT TRANSACTION T4
```

$H_1 = r_1[b_{56}], w_1[b_{56}], a_1$

$H_4 = r_4[b_{56}], r_4[b_{34}], r_4[b_{67}], c_4$

$H_c =$ $r_1[b_{56}]$, cash=94340.45, $w_1[b_{56}]$, cash=84340.45, $r_4[b_{56}]$, cash=84340.45, $r_4[b_{34}]$, cash=8900.67, $r_4[b_{67}]$, cash=34005.00, a_1, a_4

$H_c \in RC$
$H_c \notin ACA$

Strict Execution

```
BEGIN TRANSACTION T5
    UPDATE account
    SET rate=5.5
    WHERE type='deposit'
COMMIT TRANSACTION T5
```

```
BEGIN TRANSACTION T6
    UPDATE account
    SET rate=6.0
    WHERE type='deposit'
COMMIT TRANSACTION T6
```

$H_5 = w_5[a_{101}]$, rate=5.5, $w_5[a_{119}]$, rate=5.5, a_5

$H_6 = w_6[a_{101}]$, rate=6.0, $w_6[a_{119}]$, rate=6.0, c_6

$H_c = w_6[a_{101}]$, rate=6.0, $w_5[a_{101}]$, rate=5.5, $w_5[a_{119}]$, rate=5.5, $w_6[a_{119}]$, rate=6.0, a_5, c_6

$H_c \in ACA$

$H_c \notin ST$

Quiz 25.1: Recoverability (1)

$H_x = r_2[b34], r_1[b56], w_1[b56], r_1[b34], w_1[b34], c_1, w_2[b34], r_2[b67], w_2[b67], c_2$

Which describes the recoverability of H_x?

A
Non-recoverable

B
Recoverable

C
Avoids Cascading Aborts

D
Strict

Quiz 25.2: Recoverability (2)

$H_y = r_2[b34], w_2[b34], r_1[b56], w_1[b56], r_1[b34], w_1[b34], r_2[b67], w_2[b67], c_2, c_1$

Which describes the recoverability of H_y?

A

Non-recoverable

B

Recoverable

C

Avoids Cascading Aborts

D

Strict

Quiz 25.3: Recoverability (3)

$H_z = r_2[b_{34}], w_2[b_{34}], r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1, r_2[b_{67}], w_2[b_{67}], c_2$

Which describes the recoverability of H_z?

A Non-recoverable	B Recoverable
C Avoids Cascading Aborts	D Strict

Quiz 25.4: Recoverability (4)

$H_w = r_2[b_{34}], r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], w_2[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2, c_1$

Which describes the recoverability of H_w?

A

Non-recoverable

B

Recoverable

C

Avoids Cascading Aborts

D

Strict

Worksheet: Recoverability

$H_w = r_2[o_1], r_2[o_2], w_2[o_2], r_1[o_2], w_2[o_1], r_2[o_3], c_2, c_1$

$H_x = r_2[o_1], r_2[o_2], w_2[o_1], w_2[o_2], w_1[o_1], w_1[o_2], c_1, r_2[o_3], c_2$

$H_y = r_2[o_1], r_2[o_2], w_2[o_2], r_1[o_2], w_2[o_1], c_1, r_2[o_3], c_2$

$H_z = r_2[o_1], w_1[o_1], r_2[o_2], w_2[o_2], r_2[o_3], c_2, r_1[o_2], w_1[o_2], w_1[o_3], c_1$

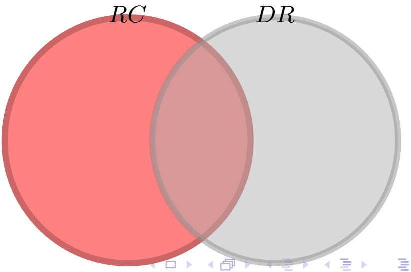
Review of Recoverable Histories

Non-recoverable → Dirty Read

For a history to be non-recoverable, it must contain a dirty read DR

Thus $H = RC \cup DR$

However, a dirty read does not imply a history is non-recoverable



RC recov

DR set o

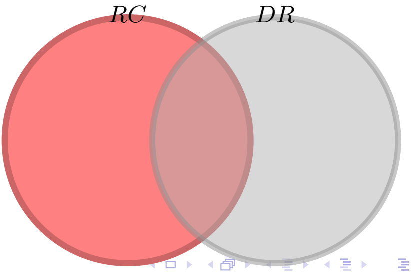
Review of Recoverable Histories

Non-recoverable → Dirty Read

For a history to be non-recoverable, it must contain a dirty read DR
Thus $H = RC \cup DR$
However, a dirty read does not imply a history is non-recoverable

No Dirty Read → Recoverable

A history that contains no dirty read must be recoverable, and **avoids cascading aborts** (**ACA**) at the commit of a transaction.
Thus $ACA = RC - DR$ and $ACA \subset RC$



RC recov

DR set o

Review of Recoverable Histories

Non-recoverable $\rightarrow$ Dirty Read

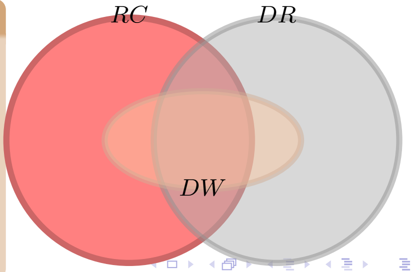
For a history to be non-recoverable, it must contain a dirty read DR
Thus $H = RC \cup DR$
However, a dirty read does not imply a history is non-recoverable

No Dirty Read $\rightarrow$ Recoverable

A history that contains no dirty read must be recoverable, and **avoids cascading aborts** (**ACA**) at the commit of a transaction.
Thus $ACA = RC - DR$ and $ACA \subset RC$

Dirty Write $\not\leftrightarrow$ Recoverable

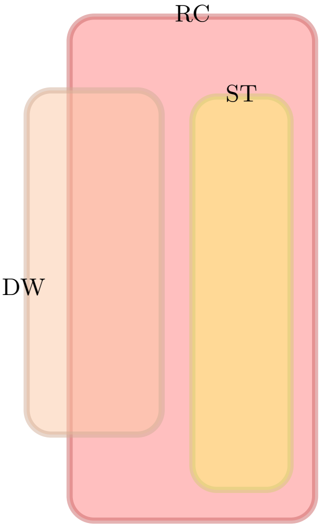
A dirty writes and recoverabilty do not imply anything about each other
However, dirty writes make executing recovery complex, and can lead to non-serialisable executions. A **strict** (**ST**) history has no dirty reads or dirty writes.
Thus $ST = ACA - DW$ and $ST \subset ACA$



RC recov

DR set o
DW dirty

Review of Serialisable and Recoverable Histories

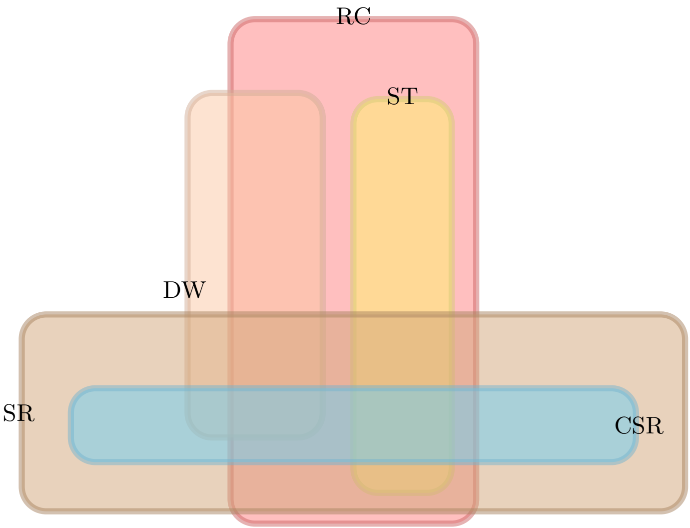


RC recoverable

ST strict

DW dirty write

Review of Serialisable and Recoverable Histories



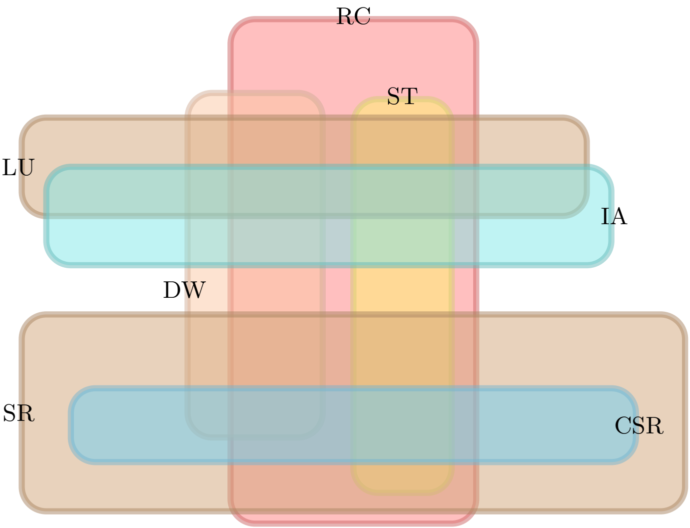
RC recoverable

ST strict

DW dirty write

SR serialisable
CSR conflict serialisable

Review of Serialisable and Recoverable Histories



RC recoverable

ST strict

DW dirty write

SR serialisable
CSR conflict serialisable

LU lost update
IA inconsistent analysis

Topic 26: Concurrency Control

P.J. McBrien

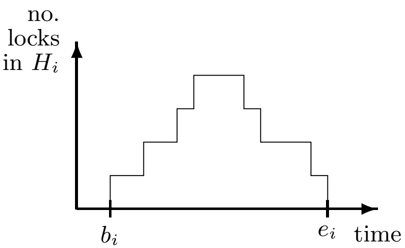
Imperial College London

Maintaining Serialisability and Recoverability

- **two-phase locking (2PL)**
  - conflict based
  - uses **locks** to prevent problems
  - common technique
- **time-stamping**
  - add a timestamp to each object
  - write sets timestamp to that of transaction
  - may only read or write objects with earlier timestamp
  - abort when object has new timestamp
  - common technique
- **optimistic concurrency control**
  - do nothing until commit
  - at commit, inspect history for problems
  - good if few conflicts

The 2PL Protocol

1. read locks $rl[o], \ldots, r[o], \ldots, ru[o]$
2. write locks $wl[o], \ldots, w[o], \ldots, wu[o]$
3. Two phases
   1. **growing phase**
   2. **shrinking phase**
4. refuse $rl_i[o]$ if $wl_j[o]$ already held
   refuse $wl_i[o]$ if $rl_j[o]$ or $wl_j[o]$ already held
5. $rl_i[o]$ or $wl_i[o]$ refused $\rightarrow$ delay T_i



Quiz 26.1: Two Phase Locking (2PL)

Which history is not valid in 2PL?

A

$rl_1[a_{107}]$, $r_1[a_{107}]$, $wl_1[a_{107}]$, $w_1[a_{107}]$, $wu_1[a_{107}]$, $ru_1[a_{107}]$

B

$wl_1[a_{107}]$, $wl_1[a_{100}]$, $r_1[a_{107}]$, $w_1[a_{107}]$, $r_1[a_{100}]$, $w_1[a_{100}]$, $wu_1[a_{100}]$, $wu_1[a_{107}]$

C

$wl_1[a_{107}]$, $r_1[a_{107}]$, $w_1[a_{107}]$, $wu_1[a_{107}]$, $wl_1[a_{100}]$, $r_1[a_{100}]$, $w_1[a_{100}]$, $wu_1[a_{100}]$

D

$wl_1[a_{107}]$, $r_1[a_{107}]$, $w_1[a_{107}]$, $wl_1[a_{100}]$, $r_1[a_{100}]$, $wu_1[a_{107}]$, $w_1[a_{100}]$, $wu_1[a_{100}]$

Lost Update Anomaly

BEGIN TRANSACTION T1
EXEC move_cash(56,34,10000.00)
COMMIT TRANSACTION T1

BEGIN TRANSACTION T2
EXEC move_cash(34,67,2000.00)
COMMIT TRANSACTION T2

$r_1[b56]$, $w_1[b56]$, $r_1[b34]$, $w_1[b34]$, c_1

$r_2[b34]$, $w_2[b34]$, $r_2[b67]$, $w_2[b67]$, c_2

$r_1[b56]$, cash=94340.45, $w_1[b56]$, cash=84340.45, $r_1[b34]$, cash=8900.67,
$r_2[b34]$, cash=8900.67, $w_1[b34]$, cash=18900.67, c_1, $w_2[b34]$, cash=6900.67,
$r_2[b67]$, cash=34005.00, $w_2[b67]$, cash=36005.25, c_2

Lost Update Anomaly

BEGIN TRANSACTION T1
EXEC move_cash(56,34,10000.00)
COMMIT TRANSACTION T1

BEGIN TRANSACTION T2
EXEC move_cash(34,67,2000.00)
COMMIT TRANSACTION T2

$r_1[b_{56}]$, $w_1[b_{56}]$, $r_1[b_{34}]$, $w_1[b_{34}]$, c_1

$r_2[b_{34}]$, $w_2[b_{34}]$, $r_2[b_{67}]$, $w_2[b_{67}]$, c_2

$r_1[b_{56}]$, cash=94340.45, $w_1[b_{56}]$, cash=84340.45, $r_1[b_{34}]$, cash=8900.67,
$r_2[b_{34}]$, cash=8900.67, lost update, c_1, $w_2[b_{34}]$, cash=6900.67,
$r_2[b_{67}]$, cash=34005.00, $w_2[b_{67}]$, cash=36005.25, c_2

Lost Update Anomaly with 2PL

BEGIN TRANSACTION T1
EXEC move_cash(56,34,10000.00)
COMMIT TRANSACTION T1

BEGIN TRANSACTION T2
EXEC move_cash(34,67,2000.00)
COMMIT TRANSACTION T2

b_1, $wl_1[b56]$, $r_1[b56]$, $w_1[b56]$, $wl_1[b34]$, $r_1[b34]$, $w_1[b34]$, c_1, $wu_1[b56]$, $wu_1[b34]$

b_2, $wl_2[b34]$, $r_2[b34]$, $w_2[b34]$, $wl_2[b67]$, $r_2[b67]$, $w_2[b67]$, c_2, $wu_2[b34]$, $wu_2[b67]$

b_1, $wl_1[b56]$, $r_1[b56]$, $w_1[b56]$, $wl_1[b34]$, $r_1[b34]$, b_2, $wl_2[b34]$, $r_2[b34]$, $w_1[b34]$, c_1, $wu_1[b56]$, $wu_1[b34]$, $w_2[b34]$, $wl_2[b67]$, $r_2[b67]$, $w_2[b67]$, c_2, $wu_2[b34]$, $wu_2[b67]$

Lost Update history not permitted by 2PL, since $wl_2[b34]$ not granted

Lost Update Anomaly with 2PL

BEGIN TRANSACTION T1
EXEC move_cash(56,34,10000.00)
COMMIT TRANSACTION T1

BEGIN TRANSACTION T2
EXEC move_cash(34,67,2000.00)
COMMIT TRANSACTION T2

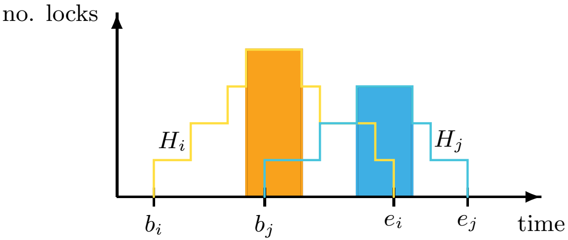
$b_1, wl_1[b_{56}], r_1[b_{56}], w_1[b_{56}], wl_1[b_{34}], r_1[b_{34}], w_1[b_{34}], c_1, wu_1[b_{56}], wu_1[b_{34}]$

$b_2, wl_2[b_{34}], r_2[b_{34}], w_2[b_{34}], wl_2[b_{67}], r_2[b_{67}], w_2[b_{67}], c_2, wu_2[b_{34}], wu_2[b_{67}]$

$b_1, wl_1[b_{56}], r_1[b_{56}], w_1[b_{56}], wl_1[b_{34}], r_1[b_{34}], b_2, w_1[b_{34}], c_1, wu_1[b_{56}], wu_1[b_{34}], wl_2[b_{34}], r_2[b_{34}], w_2[b_{34}], wl_2[b_{67}], r_2[b_{67}], w_2[b_{67}], c_2, wu_2[b_{34}], wu_2[b_{67}]$

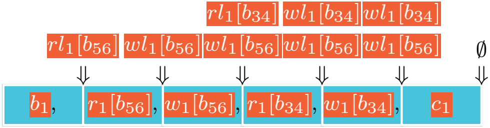
2PL causes T2 to be delayed

Why does 2PL Work?



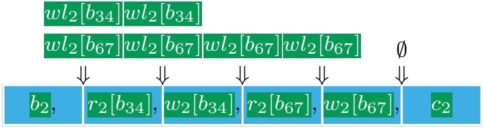
- two-phase rule → maximum lock period
- can re-time history so all operations take place during maximum lock period
- CSR since *all* conflicts prevented during maximum lock period

When to lock: Aggressive Scheduler



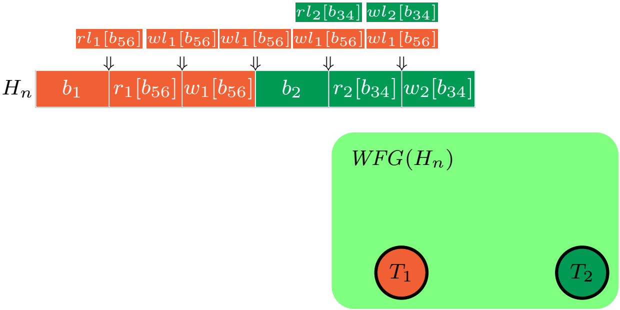
- delay taking locks as long as possible
- maximises concurrency
- might suffer delays later on

When to lock: Conservative Scheduler

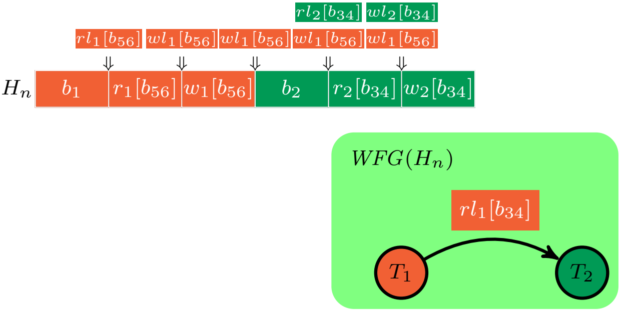


- take locks as soon as possible
- removes risks of delays later on
- might refuse to start

Deadlock Detection: WFG with No Cycle = No Deadlock



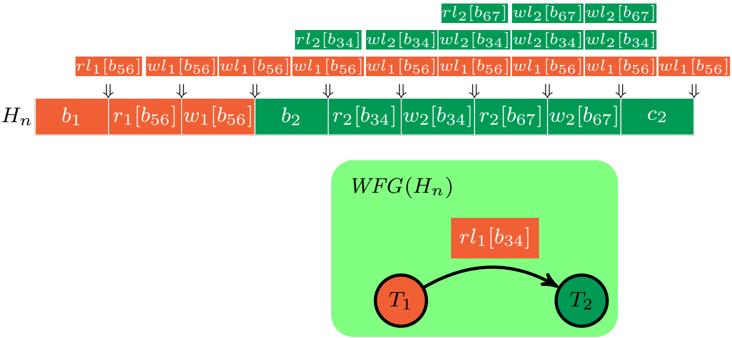
Deadlock Detection: WFG with No Cycle = No Deadlock



H_1 attempts $r_1[b_{34}]$, but is refused since H_2 has a write-lock, and so is put on WFG

- **waits-for graph** (**WFG**)
- describes which transactions waits for others

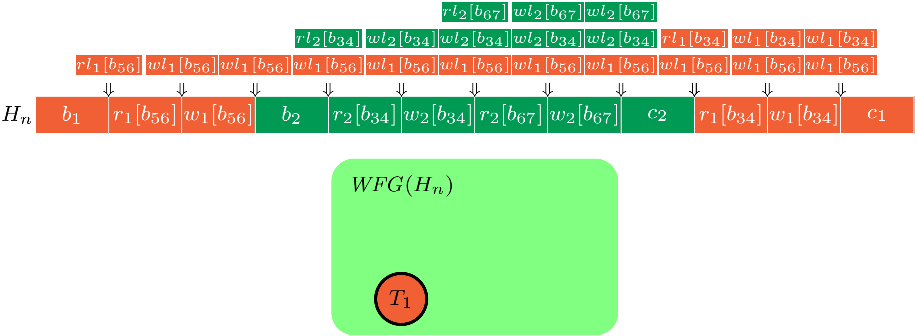
Deadlock Detection: WFG with No Cycle = No Deadlock



H_2 can proceed to complete its execution, after which it will have released all its locks

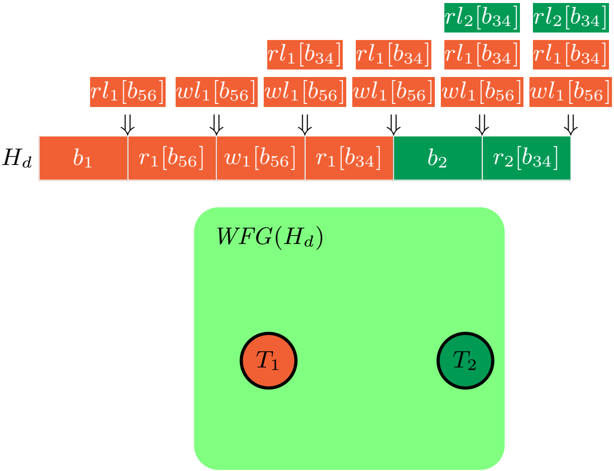
- **waits-for graph** (**WFG**)
- describes which transactions waits for others

Deadlock Detection: WFG with No Cycle = No Deadlock

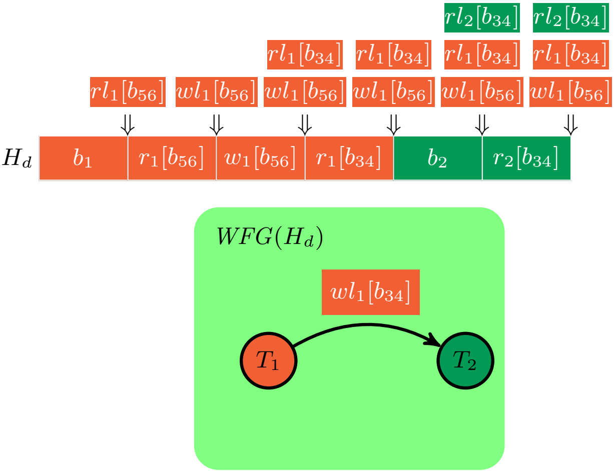


H_1 may now proceed to completion

Deadlock Detection: WFG with Cycle = Deadlock

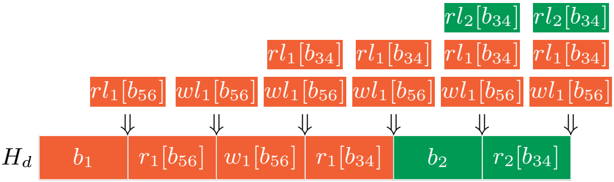


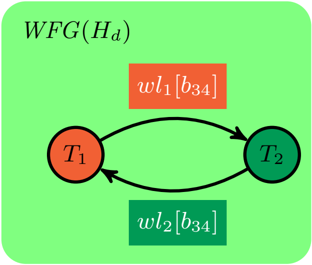
Deadlock Detection: WFG with Cycle = Deadlock



H_1 attempts $w_1[b_{34}]$, but is refused since H_2 has a read-lock, and so is put on WFG

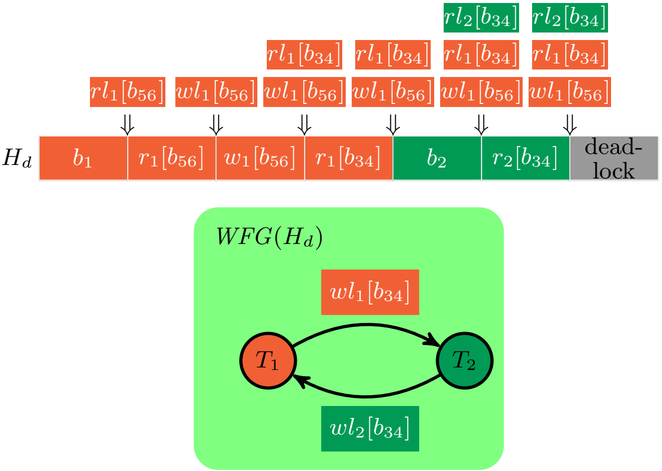
Deadlock Detection: WFG with Cycle = Deadlock





H_2 attempts $w_2[b_{34}]$, but is refused since H_1 has a read-lock, and so is put on WFG

Deadlock Detection: WFG with Cycle = Deadlock



Cycle in WFG means DB in a deadlock state, must abort either H_1 or H_2

Worksheet: Deadlocks

$H_1 = w_1[o_1], r_1[o_2], r_1[o_4]$

$H_2 = r_2[o_3], r_2[o_2], r_2[o_1]$

$H_3 = r_3[o_4], w_3[o_4], r_3[o_3], w_3[o_3]$

Transaction Isolation Levels

- Do we always need ACID properties?

```
BEGIN TRANSACTION T3
      SELECT DISTINCT no
      FROM movement
      WHERE amount>=1000.00
COMMIT TRANSACTION T3
```

- Some transactions only need 'approximate' results
  e.g. Management overview
  e.g. Estimates
- May execute these transactions at a 'lower' level of concurrency control
  SQL allows you to vary the level of concurrency control