Datenbanksysteme
Übung 4: Transaktionen, Anfrageoptimierung und Join-Algorithmen

Sommersemester 2017

Jutta Mülle

Fakultät für Informatik

IPD, Lehrstuhl Prof. Klemens Böhm

muelle@kit.edu@kit.edu http://dbis.ipd.kit.edu/

Nebenläufigkeit & Transaktionen

Aus Vorlesung:

- Motivation: fehlerfreie Ausführung von nebenläufigen Zugriffen
- Definition von Transaktionen
- Definition von (vollständigen) Histories

Nebenläufigkeit & Transaktionen

- Histories aus Sicht des DBS:
  - Scheduler dient als Schnittstelle zwischen nebenläufigen Anfragen und DBS
  - Scheduler muss über Reihenfolge von Operationen entscheiden
  - Reihenfolge soll „korrekt“ sein

- Wie muss „korrekte Reihenfolge“ formal definiert werden?

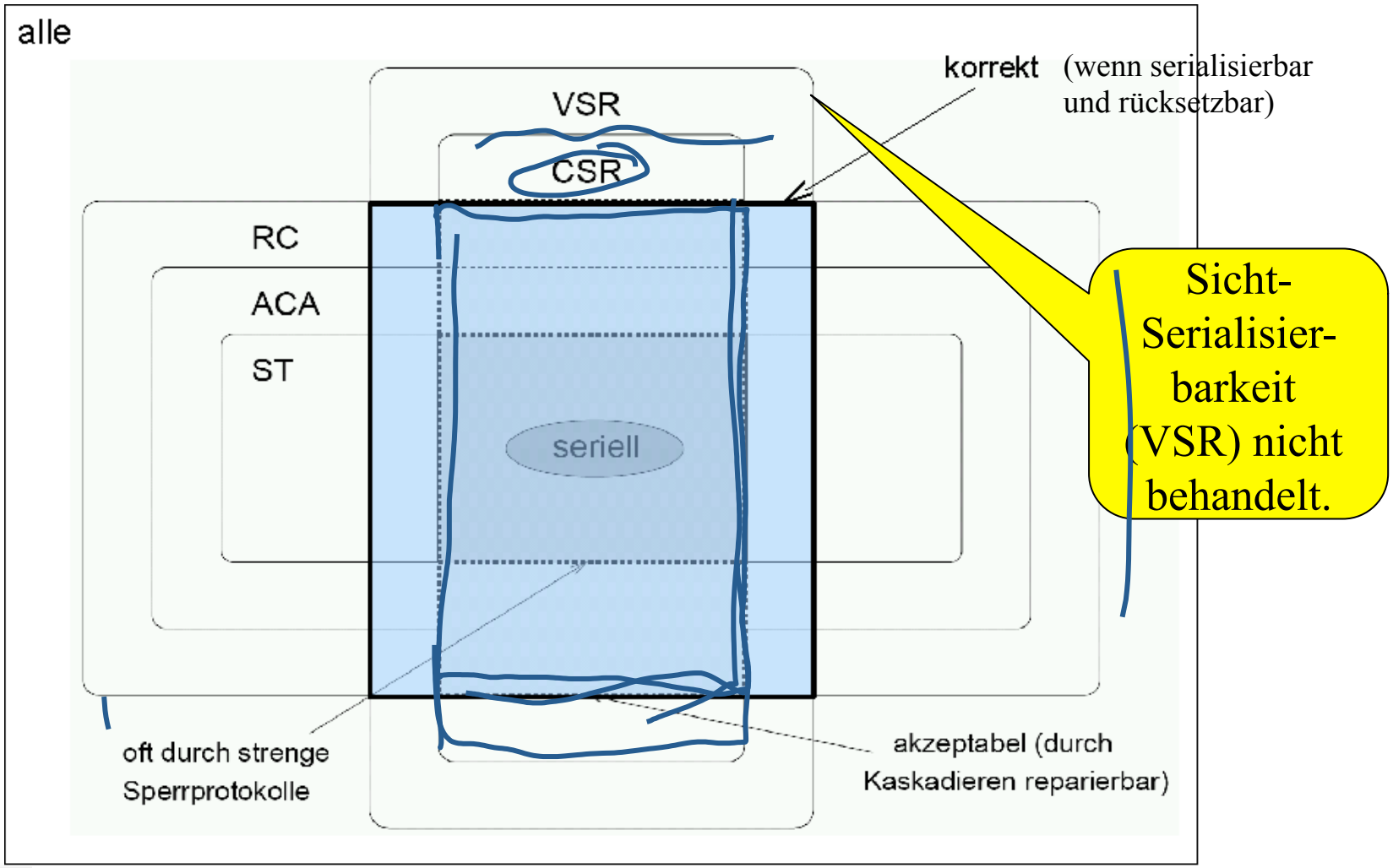
Eigenschaften von Histories

- Serialisierbarkeit (SR)

- Rücksetzbarkeitsklassen:
  - Recoverability (RC)
  - Avoids Cascading Aborts (ACA)
  - Strictness (ST)

- Korrektheit: SR + RC

Eigenschaften von Histories



Eigenschaften von Histories

- Die folgenden Eigenschaften von Histories sind alle *prefix commit-closed*

- *prefix commit-closed:*
  - „Eigenschaft von Eigenschaften“
  - Wenn Eigenschaft E für History H erfüllt

    $\Rightarrow$

    E auch für C(H‘) für jedes Prefix H‘ von H erfüllt

Serialisierbarkeit (SR)

- Ansatz:
  - Definiere Äquivalenz von Histories
  - Definiere idealisierte serielle Histories
  - Definiere Serialisierbarkeit über Äquivalenz zu einer seriellen History

Serialisierbarkeit (SR) – Konfliktäquivalenz (CSR)

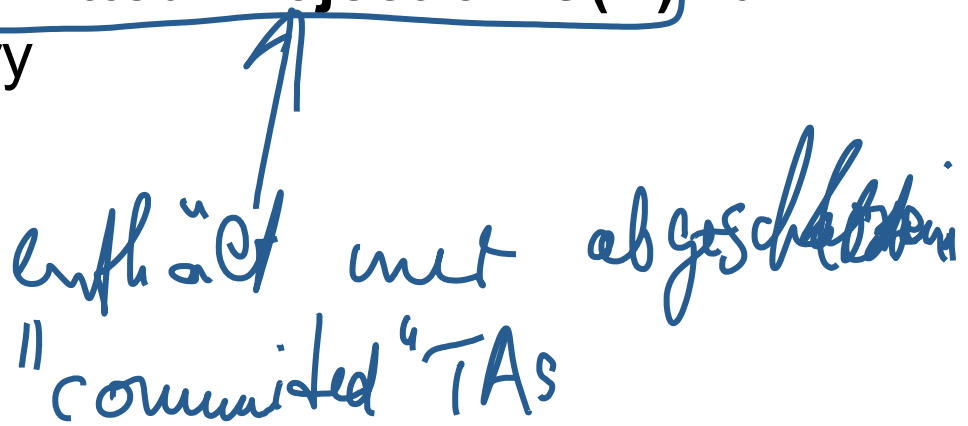
- Zwei Histories H und H' sind **konflikt-äquivalent**, wenn
  - H und H' die gleichen Transaktionen bzw. Operationen enthalten und
  - die Konfliktrelationen conf(H) und conf(H') identisch sind.
- **Zwei Operationen p, q konfligieren** :=
  p, q greifen auf das gleiche Datenobjekt zu, gehören zu verschiedenen Transaktionen und mindestens p oder q ist eine Schreiboperation.

Serialisierbarkeit (SR)

- Serielle History = vollständige History (d.h. nur committed Transaktionen) und

  $\forall T_i, T_j$ in der History: alle Operationen von T_i sind vor T_j oder umgekehrt

- Problem: Serielle History kann nicht äquivalent zu einer partiellen History sein
- Daher: Serialisierbarkeit von H definiert über Äquivalenz der **Committed Projection C(H)** zu einer seriellen History



Serialisierbarkeit (SR)

Serialisierbarkeit prüfen?

Serialisierbarkeitsgraph (SG) zu einer History H konstruieren (z.B. Abhängigkeitsgraph):

- SG(H) = gerichteter Graph mit (commited) Transaktionen als Knoten
- Enthält alle gerichteten Kanten zwischen T_i und T_j, für die Fälle, dass eine konfligierende Operation von T_i vor T_j erfolgt

Serialisierbarkeitstheorem: H ist serialisierbar, falls SG(H) zyklenfrei

1. Aufgabe: Transaktionen, 2PL

Ein Großhandelsunternehmen verwendet zur Aufbewahrung der von ihm angebotenen Güter ein Warenlager. In einem Auslieferungslager werden die verkauften Güter zur Abholung durch Spediteure bereitgestellt. Die Datenbank zur Lagerverwaltung beinhalte die folgenden Relationen:

KUNDE (K)
WARENLAGER (W)
AUSLIEFERUNGSLAGER (A)
SPEDITION (S)

Auf die Datenbasis wird (nebenläufig) zugegriffen durch folgende Transaktionen:

- Kontrolle des Warenbestands

  T_1: $r_1[W]$ $r_1[A]$ c_1
- Verkauf der Ware, Transport in das Auslieferungslager und Benachrichtigung des Spediteurs

  T_2: $r_2[K]$ $w_2[K]$ $w_2[W]$ $w_2[A]$ $w_2[S]$ c_2
- Abholung der Ware, Übergabe der Ware aus dem Auslieferungslager an die Spedition

  T_3: $w_3[A]$ $w_3[S]$ c_3
- Abrechnung

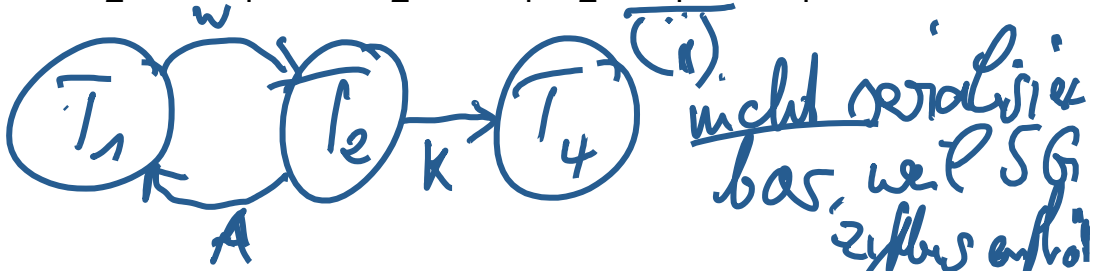
  T_4: $r_4[K]$ $w_4[K]$ c_4

1. Aufgabe: Transaktionen, 2PL

a) Untersuchen Sie die folgenden Schedules auf Konfliktserialisierbarkeit:

(i) $r_4[K]\ r_1[W]\ w_4[K]\ c_4\ r_2[K]\ r_1[A]\ w_2[K]\ c_1\ w_2[W]\ w_2[A]\ w_2[S]\ c_2$

(ii) $r_2[K]\ w_2[K]\ r_4[K]\ r_1[W]\ w_2[W]\ w_2[A]\ r_1[A]\ w_2[S]\ c_1\ c_2\ w_4[K]\ c_4$



(iii) $r_1[W]\ r_2[K]\ w_2[K]\ w_2[W]\ r_1[A]\ w_2[A]\ w_3[A]\ w_2[S]\ w_3[S]\ c_2\ c_3\ c_1$

1. Aufgabe: Transaktionen, 2PL

a) Untersuchen Sie die folgenden Schedules auf Konfliktserialisierbarkeit:

(i) (i) $r_4[K]\ r_1[W]\ w_4[K]\ c_4\ r_2[K]\ r_1[A]\ w_2[K]\ c_1\ w_2[W]\ w_2[A]\ w_2[S]\ c_2$

serialisierbar, da Abhängigkeitsgraph zyklenfrei

(ii) $r_2[K]\ w_2[K]\ r_4[K]\ r_1[W]\ w_2[W]\ w_2[A]\ r_1[A]\ w_2[S]\ c_1\ c_2\ w_4[K]\ c_4$

nicht serialisierbar, da Abhängigkeitsgraph Zyklen aufweist.

(iii) $r_1[W]\ r_2[K]\ w_2[K]\ w_2[W]\ r_1[A]\ w_2[A]\ w_3[A]\ w_2[S]\ w_3[S]\ c_2\ c_3\ c_1$

serialisierbar, da Abhängigkeitsgraph zyklenfrei

1. Aufgabe: Transaktionen, 2PL

(ii) $r_2[K]$ $w_2[K]$ $r_4[K]$ $r_1[W]$ $w_2[W]$ $w_2[A]$ $r_1[A]$ $w_2[S]$ c_1 c_2 $w_4[K]$ c_4 könnte zu folgendem Konflikt führen:

- Transaktion T_1 ermittelt den Bestand der Ware X im Warenlager (W), nämlich 1000 Einheiten.
- Transaktion T_2 verschiebt 100 Einheiten von Ware X vom Warenlager (W) ins Auslieferungslager (A).
- Es befinden sich jetzt 900 Einheiten der Ware X im Warenlager und 100 Einheiten im Auslieferungslager.
- Transaktion T_1 ermittelt den Bestand im Auslieferungslager: 100 Einheiten.
- Somit erhalten wir durch T_1 einen Gesamtbestand von 1000 + 100 = 1100 Einheiten und haben einen Bestandszuwachs von 10% zu vermelden (obwohl gesamthaft betrachtet keine 100 Einheiten zugeliefert wurden).

1. Aufgabe: Transaktionen, 2PL

b) Konstruieren Sie aus den Transaktionen T_1,..,T_4 einen Schedule, welcher eine „verlorene Änderung“ zur Folge hat. Es ist nicht zwingend notwendig, alle vier Transaktionen in den Schedule einzubringen.

1. Aufgabe: Transaktionen, 2PL

b) Konstruieren Sie aus den Transaktionen $T_1,..,T_4$ einen Schedule, welcher eine „verlorene Änderung“ zur Folge hat. Es ist nicht zwingend notwendig, alle vier Transaktionen in den Schedule einzubringen.

1. 2. 3.

r_4 [K] r_2 [K] w_4 [K] w_2 [K] w_2 [W] w_2 [A] w_2 [S] c_2 c_4

- Transaktion T_2 belastet das Konto mit 2000 EUR, die durch den Kauf des Kunden „Mayer“ mit Ware zum Preis von 2000 EUR erfolgte.
- Transaktion T_4 verbucht den Zahlungseingang des Kunden „Schmidt“ von 3000 EUR.
- Zu Beginn der Transaktion weise das Konto ein Soll von 1000 EUR auf.

1. T_4 und T_2 ermitteln den Kontostand: -1000 EUR.
2. Nun schreibt T_4 den Zahlungseingang gut. Der neue Kontostand ist jetzt 2000 EUR.
3. T_2 belastet aber das Konto, dessen Stand sie mit -1000 EUR vermutet, um weitere 2000 EUR und überschreibt den Kontostand mit -3000 EUR.
4. Das Unternehmen weist damit an Stelle des richtigen aktuellen Kontostandes (auch bzgl. des Warenbestandes, der ja entsprechend T2 durch die verkaufte Ware geändert ist) einen um 3000 EUR zu niedrigen Kontostand auf.

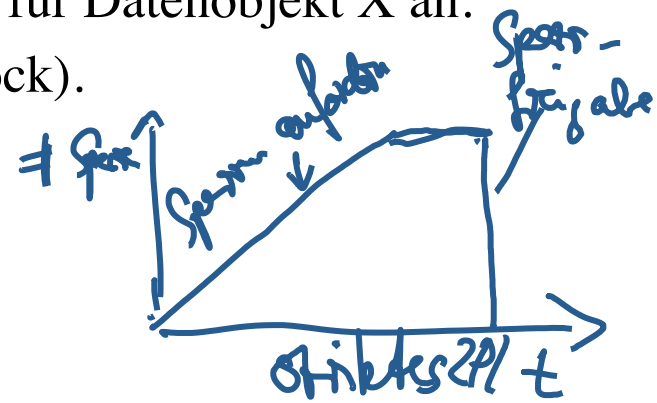
1. Aufgabe: Transaktionen, 2PL

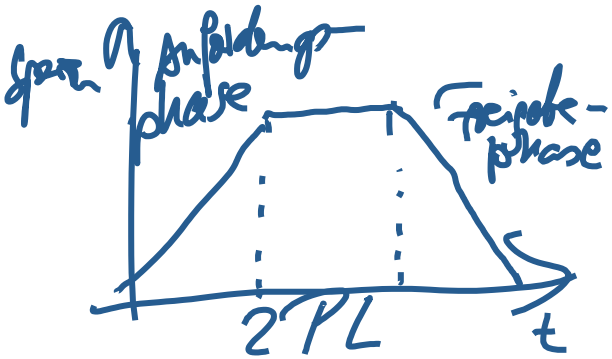
c) Zur Regelung der Nebenläufigkeit der Transaktionen in unserem System soll das **strikte 2-Phasen-Sperrprotokoll** (2PL) eingesetzt werden. Erweitern Sie die Transaktionen $T_1,...,T_4$ um die nötigen Aktionen zum Setzen und Rücksetzen von Sperren. Verwenden Sie die Notation aus der Vorlesung.

$ol_i[X]$: T_i fordert o-Sperre (o= r(ead) oder w(rite)) für Datenobjekt X an.

$ul_i[X]$: T_i gibt Sperre für Datenobjekt X frei (unlock).

T_1: $r_1[W]$ $r_1[A]$ c_1





1. Aufgabe: Transaktionen, 2PL

c) Zur Regelung der Nebenläufigkeit der Transaktionen in unserem System soll das strikte 2-Phasen-Sperrprotokoll (2PL) eingesetzt werden. Erweitern Sie die Transaktionen $T_1,...,T_4$ um die nötigen Aktionen zum Setzen und Rücksetzen von Sperren. Verwenden Sie die Notation aus der Vorlesung.

$ol_i[X]$: T_i fordert o-Sperre (o= r(ead) oder w(rite)) für Datenobjekt X an.

$ul_i[X]$: T_i gibt Sperre für Datenobjekt X frei (unlock)

T_1: $rl_1[W]$ $r_1[W]$ $rl_1[A]$ $r_1[A]$ $ul_1[W]$ $ul_1[A]$ c_1

T_2: $rl_2[K]$ $r_2[K]$ $wl_2[K]$ $w_2[K]$ $wl_2[W]$ $w_2[W]$ $wl_2[A]$ $w_2[A]$ $wl_2[S]$
$w_2[S]$ $ul_2[K]$ $ul_2[W]$ $ul_2[A]$ $ul_2S]$ c_2

Striktes 2PL

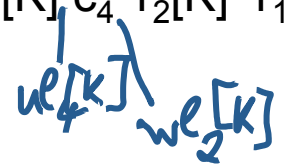
T_3: $wl_3[A]$ $w_3[A]$ $wl_3[S]$ $w_3[S]$ $ul_3[A]$ $ul_3[S]$ c_3

T_4: $rl_4[K]$ $r_4[K]$ $wl_4[K]$ $w_4[K]$ $ul_4[K]$ c_4

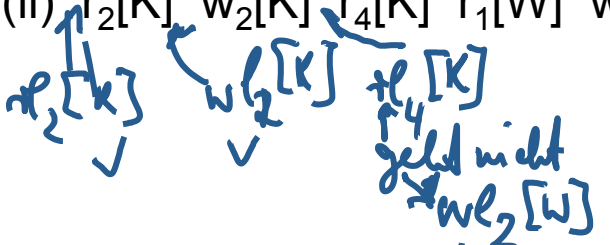
1. Aufgabe: Transaktionen, 2PL

d) Welche der Schedules (i) bis (iii) können unter Annahme des **strikten 2PL** (nutze Ergebnis aus c)) ausgeführt werden? Ändert sich diese Aussage bei Verwendung des schwachen (d.h. nicht strikten) 2-Phasen-Sperrprotokolls?

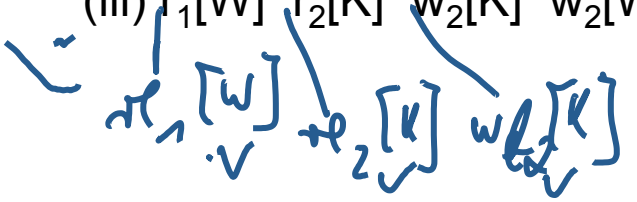
(i) $r_4[K]\ r_1[W]\ w_4[K]\ c_4\ r_2[K]\ r_1[A]\ w_2[K]\ c_1\ w_2[W]\ w_2[A]\ w_2[S]\ c_2$



(ii) $r_2[K]\ w_2[K]\ r_4[K]\ r_1[W]\ w_2[W]\ w_2[A]\ r_1[A]\ w_2[S]\ c_1\ c_2\ w_4[K]\ c_4$



(iii) $r_1[W]\ r_2[K]\ w_2[K]\ w_2[W]\ r_1[A]\ w_2[A]\ w_3[A]\ w_2[S]\ w_3[S]\ c_2\ c_3\ c_1$



1. Aufgabe: Transaktionen, 2PL

d) Welche der Schedules (i) bis (iii) können unter Annahme des strikten 2PL (nutze Ergebnis aus c)) ausgeführt werden? Ändert sich diese Aussage bei Verwendung des schwachen (d.h. nicht strikten) 2-Phasen-Sperrprotokolls?

(i) ergibt keine Probleme

(ii) beginne mit $rl_2[K]$ $r_2[K]$ $wl_2[K]$ $w_2[K]$
die nächste Aktion im Schedule wäre $rl_4[K]$. Diese Aktion kann aber wegen der Exklusivsperre von T_2 auf K nicht ausgeführt werden.

(iii) beginne mit $rl_1[W]$ $r_1[W]$ $rl_2[K]$ $r_2[K]$ $wl_2[K]$ $w_2[K]$
die folgende Anweisung $wl_2[W]$ wird wegen der Sperre von T_1 auf W blockiert.

1. Aufgabe: Transaktionen, 2PL

d) Unter dem schwachen 2-Phasen-Protokoll können die Transaktionen T_1 und T_2 folgendermaßen verändert werden:

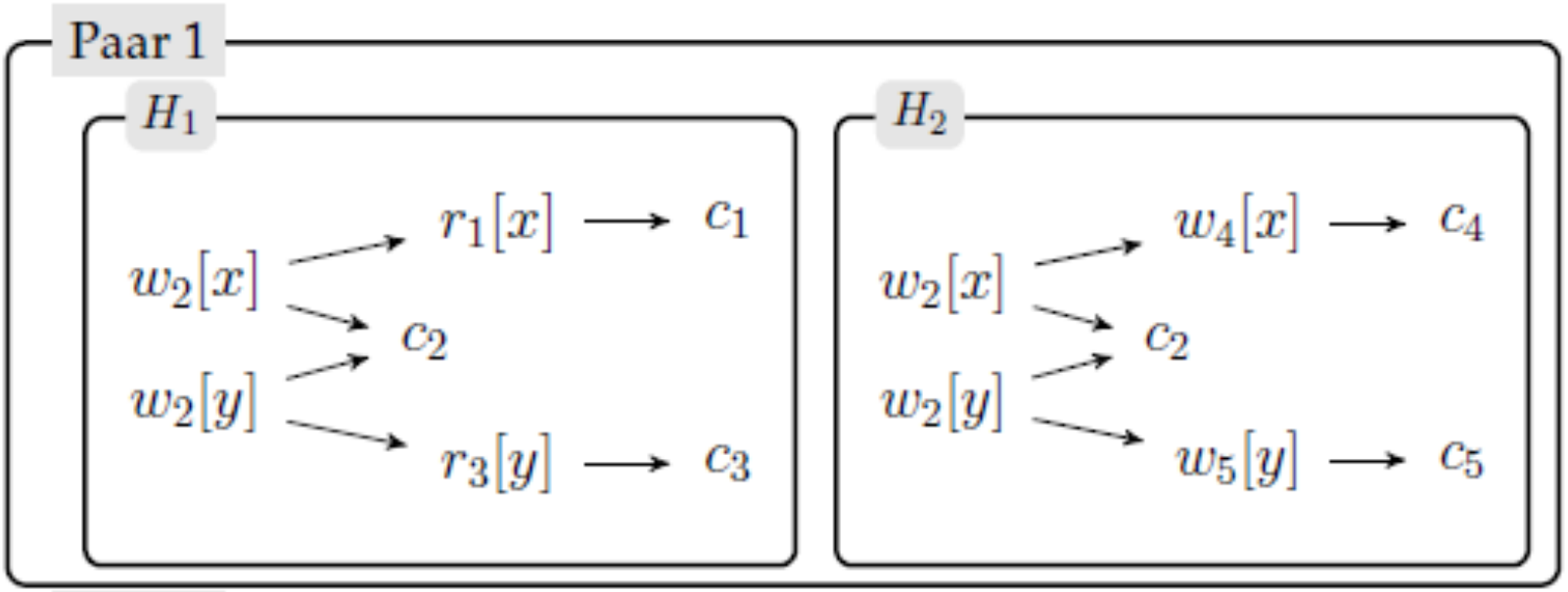
T_1: $rl_1[W]$ $r_1[W]$ $rl_1[A]$ $ul_1[W]$ $r_1[A]$ $ul_1[A]$ c_1

T_2: $rl_2[K]$ $r_2[K]$ $wl_2[K]$ $w_2[K]$ $wl_2[W]$ $w_2[W]$ $wl_2[A]$ $w_2[A]$ $wl_2[S]$
$ul_2[K]$ $ul_2[W]$ $ul_2[A]$ $w_2[S]$ $ul_2[S]$ c_2

Für Schedule (iii) ergibt sich dann
$rl_1[W]$ $r_1[W]$ $rl_1[A]$ $ul_1[W]$ $rl_2[K]$ $r_2[K]$ $wl_2[K]$ $w_2[K]$ $wl_2[W]$ $w_2[W]$
$r_1[A]$ $ul_1[A]$ $wl_2[A]$ $w_2[A]$ $wl_2[S]$ $ul_2[A]$ $wl_3[A]$ $w_3[A]$ $w_2[S]$ $ul_2[S]$
$wl_3[S]$ $w_3[S]$ c_2 $ul_3[A]$ $ul_3[S]$ c_3 c_1
Dies kann dann vom Scheduler bearbeitet werden.

Aufgabe 2 - Konfliktäquivalenz

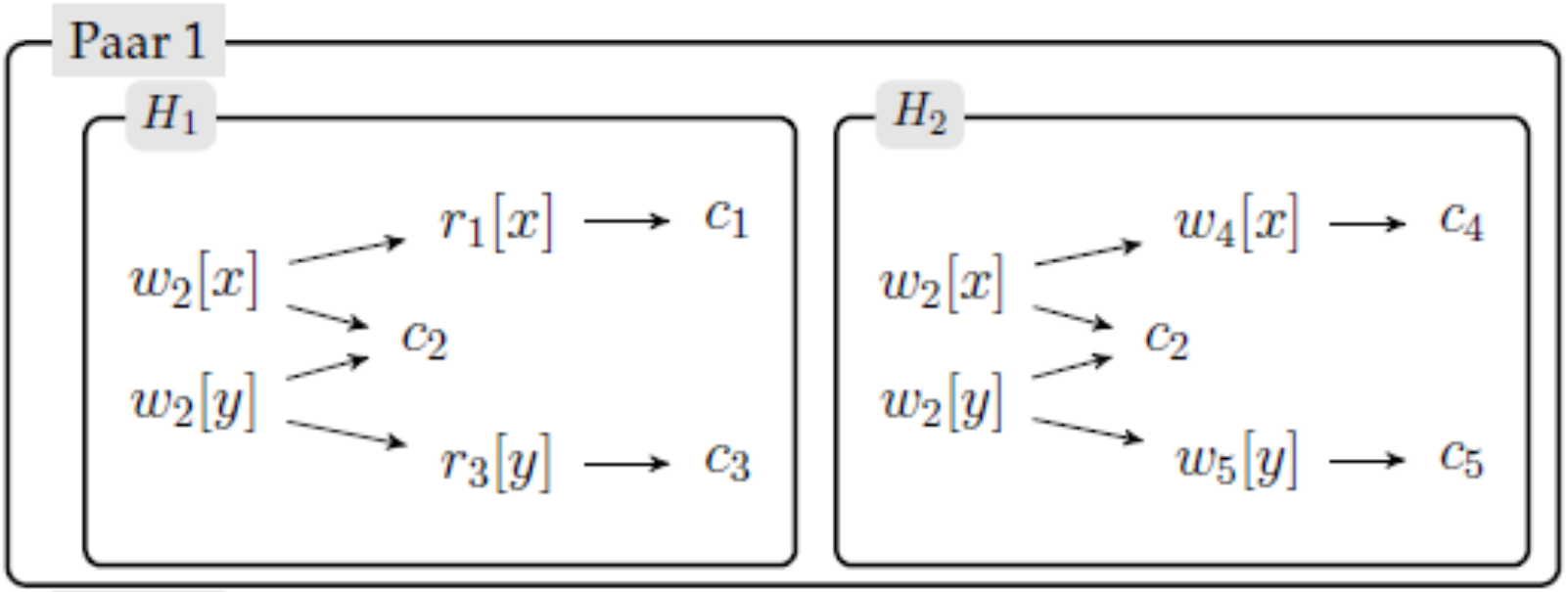
Geben Sie für jedes der folgenden History-Paare an, ob diese konflikt-äquivalent sind. Falls zwei Histories nicht konflikt-äquivalent sind, begründen Sie warum.



nein

Aufgabe 2 - Konfliktäquivalenz

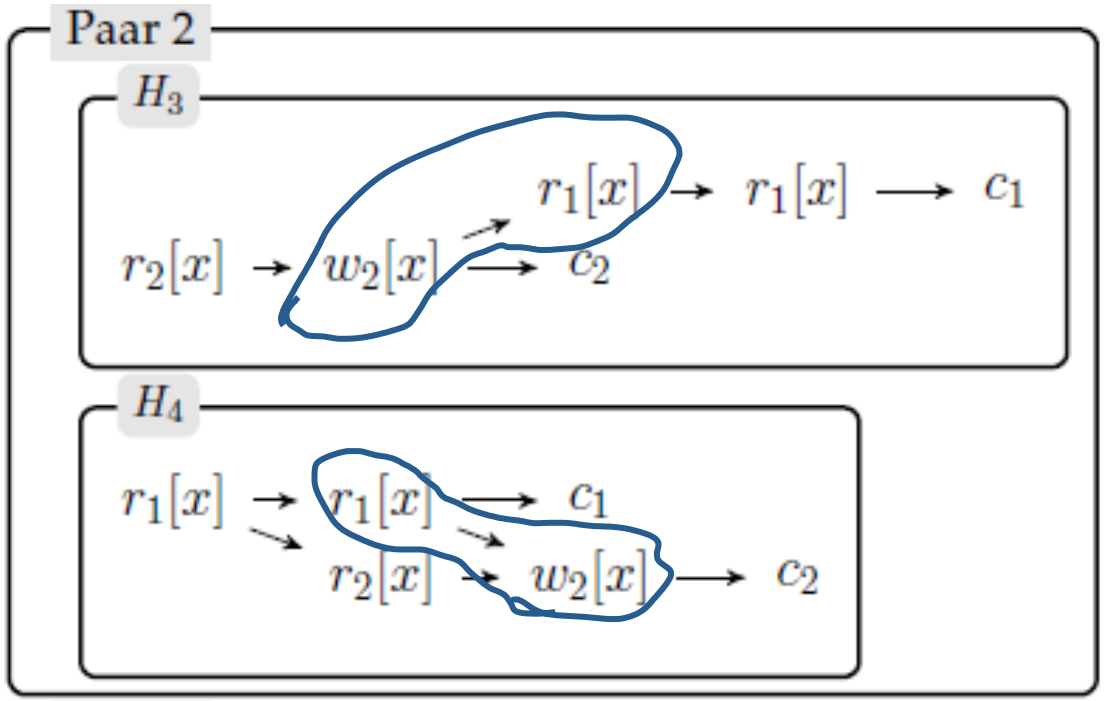
Geben Sie für jedes der folgenden History-Paare an, ob diese konflikt-äquivalent sind. Falls zwei Histories nicht konflikt-äquivalent sind, begründen Sie warum.



H1 und H2 sind nicht konflikt-äquivalent.
Die Histories bestehen aus unterschiedlichen Transaktionen

Aufgabe 2 - Konfliktäquivalenz

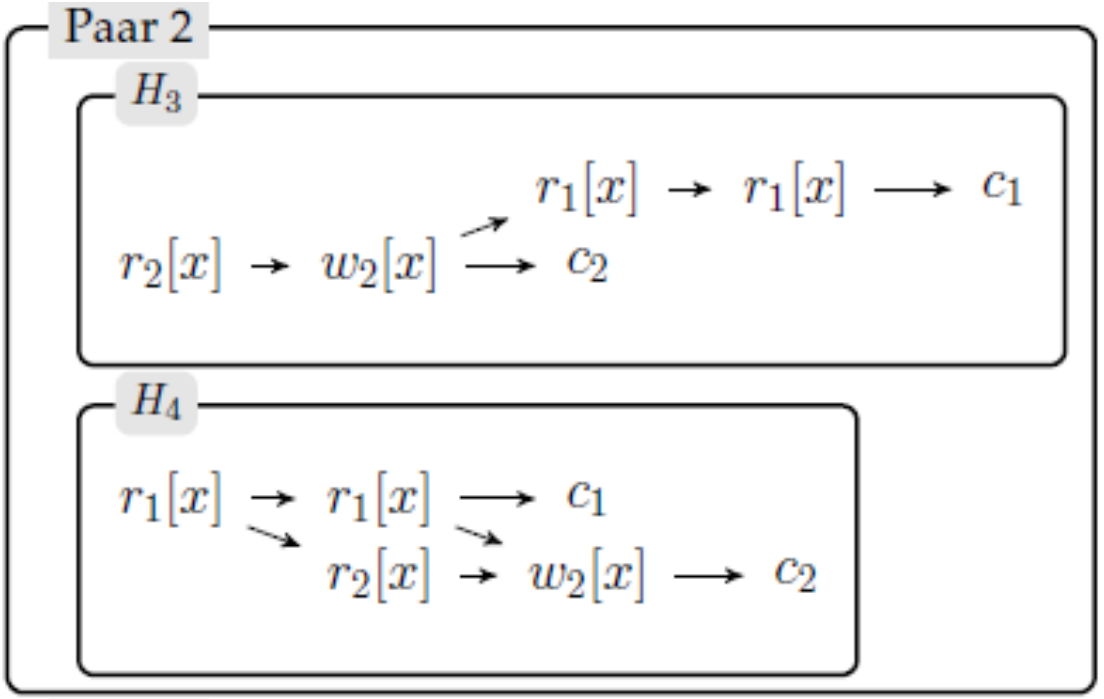
Geben Sie für jedes der folgenden History-Paare an, ob diese konflikt-äquivalent sind. Falls zwei Histories nicht konflikt-äquivalent sind, begründen Sie warum.



Konfliktpaare sind unterschiedlich

Aufgabe 2 - Konfliktäquivalenz

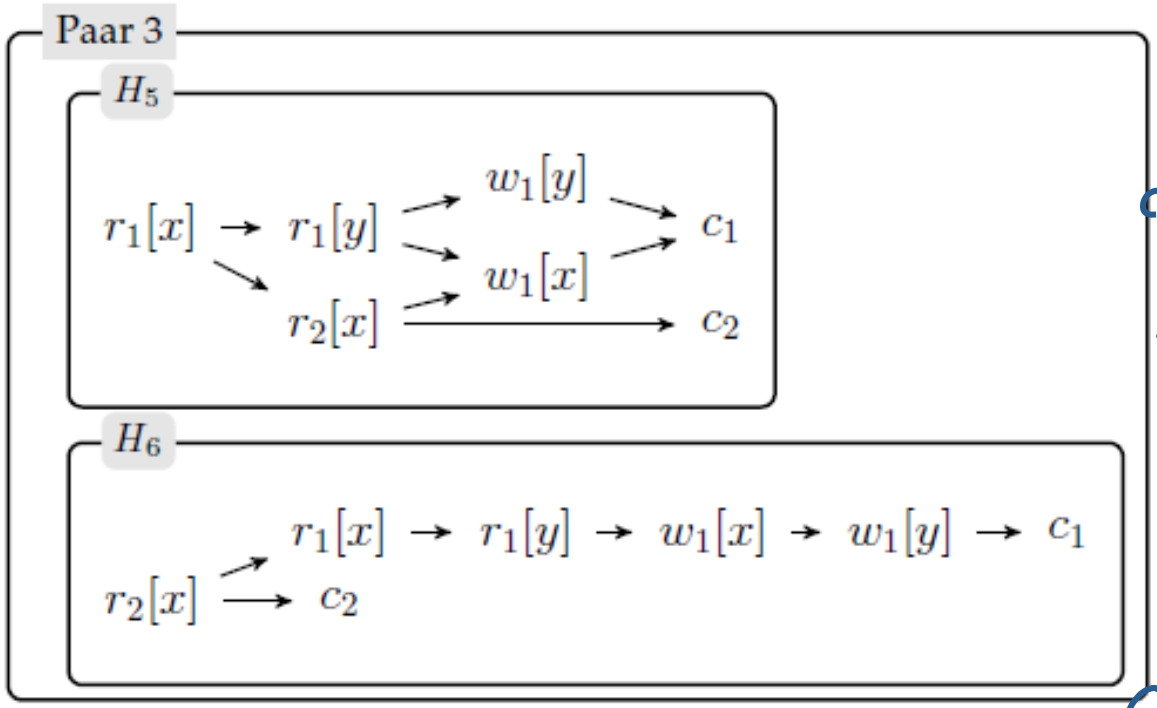
Geben Sie für jedes der folgenden History-Paare an, ob diese konflikt-äquivalent sind. Falls zwei Histories nicht konflikt-äquivalent sind, begründen Sie warum.



H3 und H4 sind nicht konflikt-äquivalent.
Konfligierende Operationen haben unterschiedliche Ordnung:
H3 : w2[x] < r1[x] vs. H4 : r1[x] < w2[x])

Aufgabe 2 - Konfliktäquivalenz

Geben Sie für jedes der folgenden History-Paare an, ob diese konflikt-äquivalent sind. Falls zwei Histories nicht konflikt-äquivalent sind, begründen Sie warum.

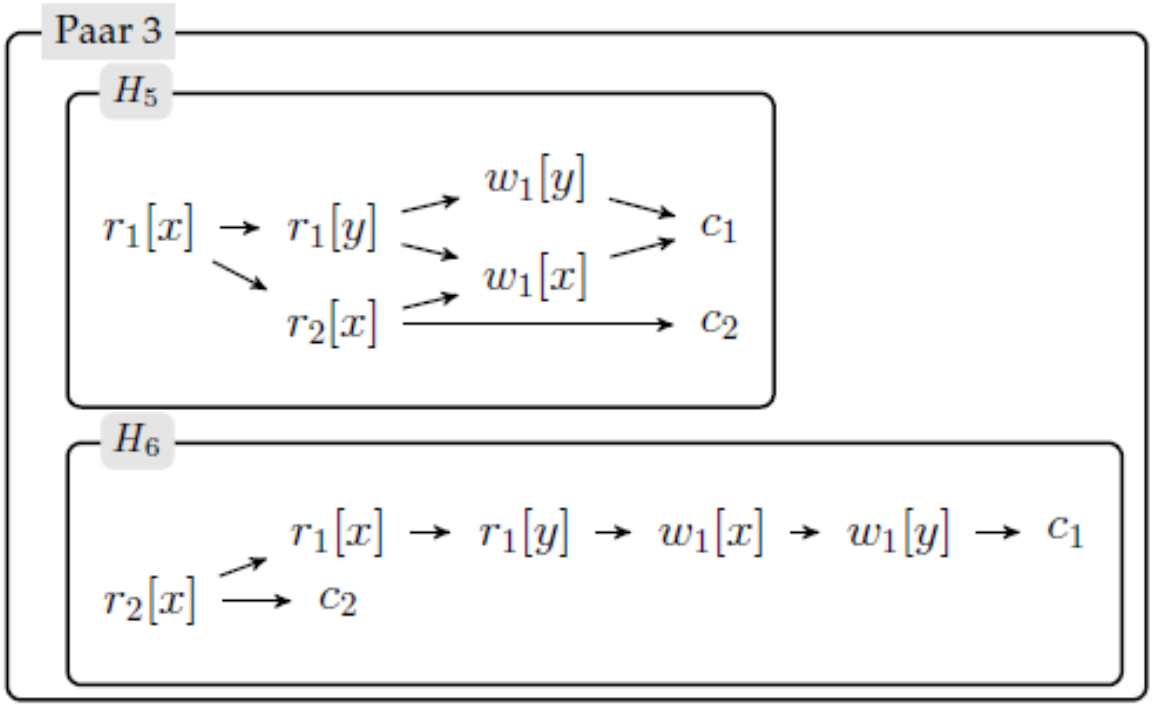


$conf(H_5) = \{ (\quad) \}$

$conf(H_6) = \{ \quad \}$

Aufgabe 2 - Konfliktäquivalenz

Geben Sie für jedes der folgenden History-Paare an, ob diese konflikt-äquivalent sind. Falls zwei Histories nicht konflikt-äquivalent sind, begründen Sie warum.



H5 und H6 sind konflikt-äquivalent.

Einschub: Rücksetzbarkeitsklassen (für Aufgabe 3)

Ziel: Datenbanksystem soll sich (auch im Fehlerfall) so verhalten, dass

- es alle Effekte von committeten Transaktionen
- keine Effekte von nicht committeten Transaktionen

enthält.

Rücksetzbarkeitsklassen

Wichtige formale Definition *“reads-from”-Beziehung*

T_i *liest x von* T_j falls:

- $w_j(x) < r_i(x)$
- $a_j \ \neg< \ r_i(x)$
- falls es ein w_k gibt mit $w_j(x) < w_k(x) < r_i(x)$, dann $a_k < r_i(x)$

“… lesen von der letzten Transaktion, die eine Änderung an x gemacht hat.”

1. Rücksetzbar

Ein Abbruch (Abort) darf die Semantik einer committeten Transaktion nicht verändern!

$w_1(x,2)\ \ r_2(x)\ \ w_2(y,3)\ c_2\ a_1$

Lösung: Commits für T_i erst dann erlauben, wenn **alle** Transaktionen T_j, von denen **T_i liest**, committet haben.

Formale Definition RC:

Für alle T_i, T_j ($i \neq j$), für die gilt T_i liest von T_j und

$c_i \in H$: $\quad c_j < c_i$

2. ACA

Rücksetzbarkeit kann immer noch zu Problemen führen:

$w_1(x,2)$ $r_2(x)$ $w_2(y,3)$ a_1

$\Rightarrow$ Cascading Abort

Lösung: nur **Lesen von bereits committeten** Transaktionen

Formale Definition ACA:

Für alle T_i, T_j $(i \neq j)$ für die gilt, T_i liest x von T_j, gilt:

$c_j < r_i(x)$

3. Strict

"Rückgängig machen" soll einfach implementierbar sein.

Ansatz: "*Before Images*" (BI)

$w_1(x,1)$ $w_1(y,3)$ $w_2(y,1)$ c_1 $r_2(x)$ a_2

BI: y=3

3. Strict

x = 1 $w_1(x,2)$ $w_2(x,3)$ a_1 a_2

BI: x=1 BI: x=2

Problem: Before Images basieren hier auf abgebrochenen Transaktionen

Lösung: Auch beim Schreiben von x darauf warten, dass alle Transaktionen, die x verändert haben, committed oder aborted sind.

3. Strict

Formale Definition Strict:

Wenn $w_j(x) < o_i(x)$ $(i \neq j)$, dann gilt

- entweder: $a_j < o_i(x)$
- oder: $c_j < o_i(x)$

wobei $o_i(x)$ für $r_i(x)$ oder für $w_i(x)$ steht.

Anmerkung: Für strict wird die Betrachtung der ‘reads-from’-Paare auf ‘overwrite’-Paare erweitert (o=w).

Beispiele

- $T_1 = w_1[x]\ w_1[y]\ w_1[z]\ c_1$
- $T_2 = r_2[u]\ w_2[x]\ r_2[y]\ w_2[y]\ c_2$

- $H_1 = w_1[x]\ w_1[y]\ r_2[u]\ w_2[x]\ r_2[y]\ w_2[y]\ c_2\ w_1[z]\ c_1$
- $H_2 = w_1[x]\ w_1[y]\ r_2[u]\ w_2[x]\ r_2[y]\ w_2[y]\ w_1[z]\ c_1\ c_2$
- $H_3 = w_1[x]\ w_1[y]\ r_2[u]\ w_2[x]\ w_1[z]\ c_1\ r_2[y]\ w_2[y]\ c_2$
- $H_4 = w_1[x]\ w_1[y]\ r_2[u]\ w_1[z]\ c_1\ w_2[x]\ r_2[y]\ w_2[y]\ c_2$

- Welche Histories sind RC, ACA und ST?

Beispiele

$T_1 = w_1[x]\ w_1[y]\ w_1[z]\ c_1$

$T_2 = r_2[u]\ w_2[x]\ r_2[y]\ w_2[y]\ c_2$

- $H_1 = w_1[x]\ w_1[y]\ r_2[u]\ w_2[x]\ r_2[y]\ w_2[y]\ c_2\ w_1[z]\ c_1$

  Ist nicht RC, da reads-from-Konfliktpaar (w1[y], r2[y]) aber c2 vor c1.
- $H_2 = w_1[x]\ w_1[y]\ r_2[u]\ w_2[x]\ r_2[y]\ w_2[y]\ w_1[z]\ c_1\ c_2$

  Ist RC, aber nicht ACA, da reads-from-Paar (w1[y],r2[y]) aber c1 nach r2[y].
- $H_3 = w_1[x]\ w_1[y]\ r_2[u]\ w_2[x]\ w_1[z]\ c_1\ r_2[y]\ w_2[y]\ c_2$

  Ist RC und ACA, aber nicht Strict, da Overwrites-Konfliktpaar (w1[x],w2[x]) aber c1 nach w2[x].
- $H_4 = w_1[x]\ w_1[y]\ r_2[u]\ w_1[z]\ c_1\ w_2[x]\ r_2[y]\ w_2[y]\ c_2$

  ist Strict und damit auch ACA und RC.

Aufgabe 3 - Histories

Gegeben seien die folgenden Transaktionen:

- $T_1 = r_1[y]\ w_1[x]\ w_1[z]\ c_1$
- $T_2 = r_2[z]\ r_2[x]\ w_2[y]\ w_2[z]\ c_2$
- $T_3 = r_3[x]\ w_3[y]\ r_3[y]\ c_3$

Basierend auf diesen Transaktionen seien zwei unterschiedliche Histories definiert:

- $H_1 = r_1[y]\ w_1[x]\ r_3[x]\ w_1[z]\ r_2[z]\ w_3[y]\ r_2[x]\ w_2[y]\ c_1\ r_3[y]\ c_3\ w_2[z]\ c_2$
- $H_2 = r_3[x]\ r_2[z]\ r_2[x]\ w_2[y]\ w_2[z]\ w_3[y]\ r_1[y]\ r_3[y]\ c_3\ w_1[x]\ c_2\ w_1[z]\ c_1$

Aufgabe 3 “Histories” – a)

a) Konstruieren Sie für beide Histories den Serialisierbarkeitsgraphen und geben Sie an, ob H1 bzw. H2 serialisierbar sind.

$H_1 = r_1[y]\ w_1[x]\ r_3[x]\ w_1[z]\ r_2[z]\ w_3[y]\ r_2[x]\ w_2[y]\ c_1\ r_3[y]\ c_3\ w_2[z]\ c_2$

Konfligierende Operationen?

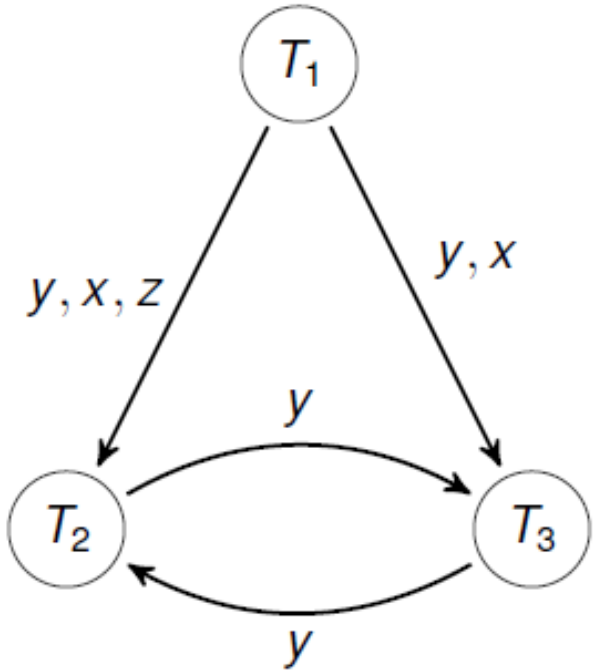
Aufgabe 3 “Histories” – a)

a) Konstruieren Sie für beide Histories den Serialisierbarkeitsgraphen und geben Sie an, ob H1 bzw. H2 serialisierbar sind.

$H_1 = r_1[y]\ w_1[x]\ r_3[x]\ w_1[z]\ r_2[z]\ w_3[y]\ r_2[x]\ w_2[y]\ c_1\ r_3[y]\ c_3\ w_2[z]\ c_2$

- T3 vor T2 (über Variable y)
- T2 vor T3 (über Variable y)

Serialisierbarkeitsgraph hat Zyklus, damit ist H1 nicht serialisierbar.



Aufgabe 3 “Histories” – a)

a) Konstruieren Sie für beide Histories den Serialisierbarkeitsgraphen und geben Sie an, ob H1 bzw. H2 serialisierbar sind.

$H_2 = r_3[x]\ r_2[z]\ r_2[x]\ w_2[y]\ w_2[z]\ w_3[y]\ r_1[y]\ r_3[y]\ c_3\ w_1[x]\ c_2\ w_1[z]\ c_1$

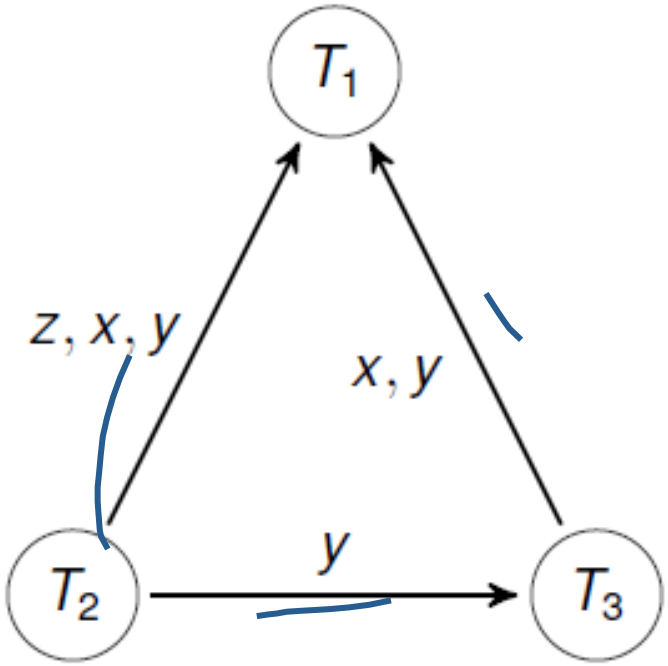
Konfligierende Operationen?

Aufgabe 3 “Histories” – a)

a) Konstruieren Sie für beide Histories den Serialisierbarkeitsgraphen und geben Sie an, ob H1 bzw. H2 serialisierbar sind.

$H_2 = r_3[x]\ r_2[z]\ r_2[x]\ w_2[y]\ w_2[z]\ w_3[y]\ r_1[y]\ r_3[y]\ c_3\ w_1[x]\ c_2\ w_1[z]\ c_1$

Serialisierbarkeitsgraph zyklenfrei, damit ist H2 serialisierbar.



Aufgabe 3 “Histories” – b) und c)

b) Ermitteln Sie für beide Histories sämtliche reads-from Beziehungen. Halten Sie sich bei der Angabe der Lösung an folgende Notation:

T_i reads <Variable> from T_j .

c) Analysieren Sie für jede reads-from Beziehung aus Aufgabe b), ob die Rücksetzbarkeitseigenschaften rücksetzbar (RC) und kaskadenfrei rücksetzbar (ACA) erfüllt sind. Bestimmen Sie für beide Histories die maximale Rücksetzbarkeitsklasse.

Aufgabe 3 “Histories” – b) und c)

$H_1 = r_1[y]\ w_1[x]\ r_3[x]\ w_1[z]\ r_2[z]\ w_3[y]\ r_2[x]\ w_2[y]\ c_1\ r_3[y]\ c_3\ w_2[z]\ c_2$

Reads-from-Beziehung:	RC	ACA

Aufgabe 3 “Histories” – b) und c)

$H_1 = r_1[y]\ w_1[x]\ r_3[x]\ w_1[z]\ r_2[z]\ w_3[y]\ r_2[x]\ w_2[y]\ c_1\ r_3[y]\ c_3\ w_2[z]\ c_2$

Reads-from-Beziehung:	RC	ACA
T_3 reads y from T_2	nein, da $c_2 < c_3$	nein
T_2 reads x from T_1	ja, da $c_2 > c_1$	nein, da $c_1 > r_2[x]$
T_2 reads z from T_1	ja, da $c_2 > c_1$	nein, da $c_1 > r_2[z]$
T_3 reads x from T_1	ja, da $c_3 > c_1$	nein, da $c_1 > r_3[x]$

Damit ist H_1 ¬ RC, ¬ACA, ¬ STRICT

Aufgabe 3 “Histories” – b) und c)

$H_2 = r_3[x]\ r_2[z]\ r_2[x]\ w_2[y]\ w_2[z]\ w_3[y]\ r_1[y]\ r_3[y]\ c_3\ w_1[x]\ c_2\ w_1[z]\ c_1$

Reads-from-Beziehung:	RC	ACA

Aufgabe 3 “Histories” – b) und c)

$H_2 = r_3[x]\ r_2[z]\ r_2[x]\ w_2[y]\ w_2[z]\ w_3[y]\ r_1[y]\ r_3[y]\ c_3\ w_1[x]\ c_2\ w_1[z]\ c_1$

Reads-from Beziehung:	RC	ACA
T_1 reads y from T_3	ja, da $c_1 > c_3$	nein, da c_3 nach $r_1[y]$

Damit ist H_2 RC, ¬ ACA, ¬ STRICT

Hinweis: T_3 reads y from T_2 ist nicht gültig, da y vor dem Lesen von T_3 selbst zuletzt geschrieben wurde.

Aufgabe 4 - Anfrageoptimierung

Künstler (kId, name, geburtstag, todestag, heimatland)
Bild (bildId, name, kId, wert)
Museum (mName, stadt, land)
Ausgestellt (bildId, mName, von, bis

a) Anfrage in relationaler Algebra formulieren:

- in welchen Museen (Museumsname und Stadt, in der sich das Museum befindet) sind Bilder des Künstlers ‚Dali' ausgestellt?

Aufgabe 4 - Anfrageoptimierung

Künstler (kId, name, geburtstag, todestag, heimatland)
Bild (bildId, name, kId, wert)
Museum (mName, stadt, land)
Ausgestellt (bildId, mName, von, bis)

a) Anfrage in relationaler Algebra formulieren:

- in welchen Museen (Museumsname und Stadt, in der sich das Museum befindet) sind Bilder des Künstlers ‚Dali' ausgestellt?

$$\Pi_{\text{mName, stadt}} (\text{Museum} \bowtie_{\text{natVerbund}}$$
$$(\Pi_{\text{mName, bildId}} (\text{Ausgestellt}) \bowtie_{\text{natVerbund}}$$
$$(\Pi_{\text{bildId}} (\text{Bild}) \bowtie_{\text{natVerbund}}$$
$$\Pi_{\text{kId}} (\sigma_{\text{name='Dali'}} (\text{Künstler})))))$$

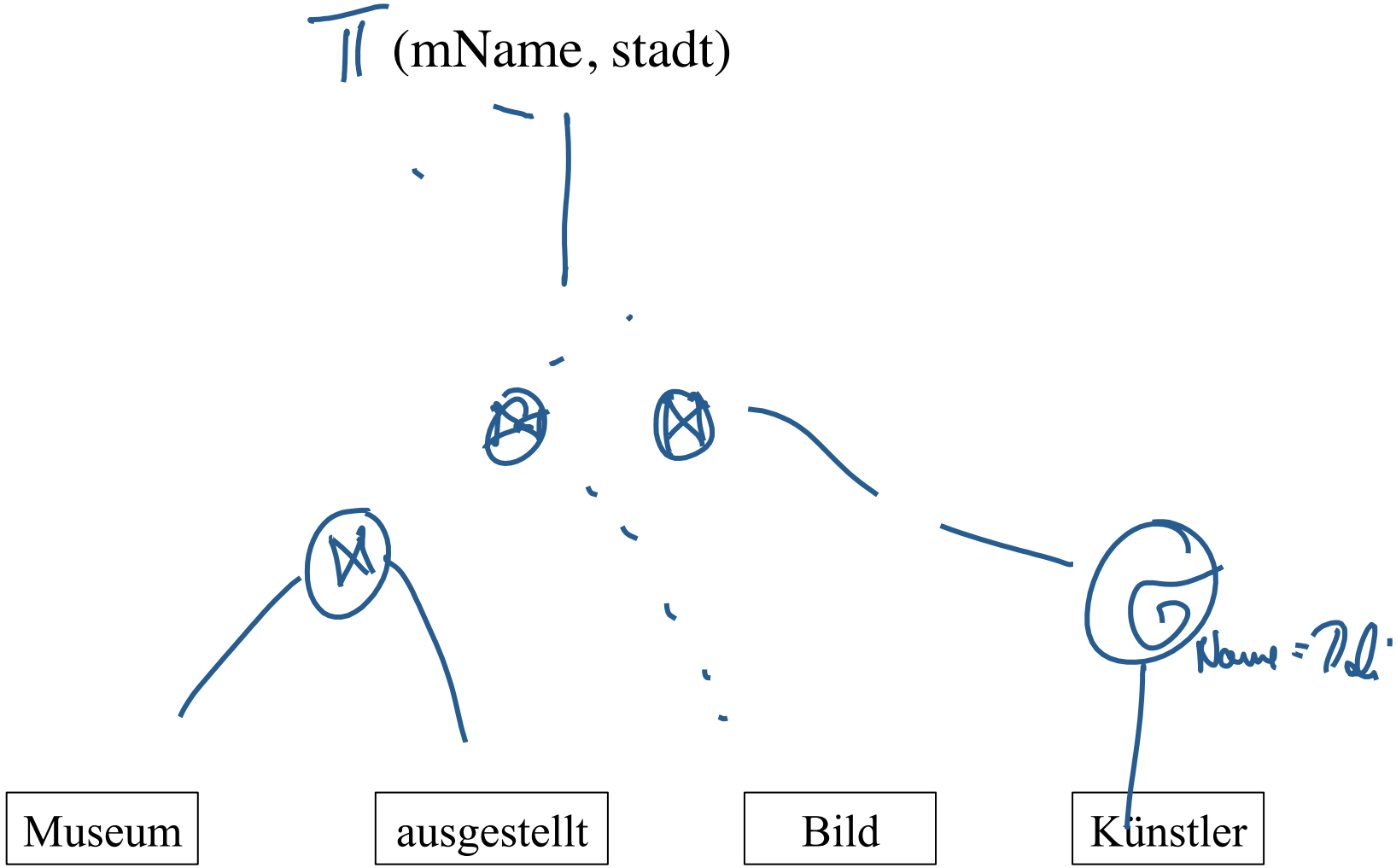
Hinweis: <natVerbund> steht für natural Join.

Aufgabe 4 - Anfrageoptimierung

b) Anfrageplan

Aufgabe 4 - Anfrageoptimierung

b) Anfrageplan



Aufgabe 4 - Anfrageoptimierung

c) Auswahl der Implementierung des Join-Operators (Nested-Loop-Join, Block-Nested-Loop Join oder Merge-Join)

Aufgabe 4 - Anfrageoptimierung

c) Auswahl der Implementierung des Join-Operators (Nested-Loop-Join, Block-Nested-Loop Join oder Merge-Join)

Kosten:

- Nested-Loop Join $O(n_s * n_r)$
- block-Nested-Loop Join: $O(b_r * b_s)$
- Merge-basierter Join:
  - alle Tupel gleichen x-Wert (worst case):

    $O(n_r * n_s)$
  - X Schlüssel in R oder S: $O(n_r \log n_r + n_s \log n_s)$
  - Falls Relationen bereits sortiert und nicht alle Tupel gleichen x-Wert haben: $O(n_r + n_s)$

Aufgabe 4 - Anfrageoptimierung

d) Nutzung von Indexen? (evtl. schon in b) diskutiert)

- Nur bei direktem Zugriff auf die Relationen,
- nicht für Zugriffe auf Zwischenrelationen nutzbar