



Kommunikation und Datenhaltung

Transaktionsverwaltung



Überblick über den Datenhaltungsteil

- Einleitung
 - Motivation und Grundlagen
 - Architektur von Datenbanksystemen
- Datenbankabfragen
 - Relationenmodell und Relationenalgebra
 - Relationale Datenbanksprachen (SQL)
- Datenbankentwurf
 - ER- und EER-Modell
 - Abbildung von ER-Modellen auf das Relationenmodell
 - Relationaler Entwurf
 - Sprachen zur Datenbankdefinition
- Anfrageoptimierung
- **Transaktionsverwaltung**
- Datenbankanwendungsentwicklung



Agenda

- **Einleitung / Probleme**
- Definitionen
 - Transaktionen
 - Konflikte
 - Histories
 - Äquivalenz
 - Serialisierbarkeit
 - Rücksetzbarkeit
- Locking

```
X Desktop
SQL> SELECT * FROM Bankkonten;

  KONTONR INHABER          STAND
-----
    1234 Klemens Boehm      500
    5678 Gunter Saake       500

SQL> UPDATE Bankkonten SET Stand=Stand-200 WHERE Kontonr=1234;
1 row updated.

SQL> SELECT * FROM Bankkonten;

  KONTONR INHABER          STAND
-----
    1234 Klemens Boehm      300
    5678 Gunter Saake       500

SQL> UPDATE Bankkonten SET Stand=Stand+200 WHERE Kontonr=5678;
1 row updated.

SQL> SELECT * FROM Bankkonten;

  KONTONR INHABER          STAND
-----
    1234 Klemens Boehm      300
    5678 Gunter Saake       700

SQL> █
```

Einleitung
Definitionen
Locking

Einleitung
Definitionen
Locking

Benutzer 1

```
X Desktop
SQL> SET AUTOCOMMIT OFF;
SQL> SELECT * FROM Bankkonten;
1
  KONTONR  INHABER                                STAND
-----
    1234  Klemens Boehm                            500
    5678  Gunter Saake                             500

SQL> UPDATE Bankkonten SET Stand=Stand-200 WHERE Kontonr=1234;
1 row updated.
2
SQL> SELECT * FROM Bankkonten;
  KONTONR  INHABER                                STAND
-----
    1234  Klemens Boehm                            300
    5678  Gunter Saake                             500

SQL> UPDATE Bankkonten SET Stand=Stand+200 WHERE Kontonr=5678;
1 row updated.
3
SQL> SELECT * FROM Bankkonten;
  KONTONR  INHABER                                STAND
-----
    1234  Klemens Boehm                            300
    5678  Gunter Saake                             700

SQL> COMMIT;
Commit complete.
4
SQL> SELECT * FROM Bankkonten;
  KONTONR  INHABER                                STAND
-----
    1234  Klemens Boehm                            300
    5678  Gunter Saake                             700

SQL>
```

- Einleitung
- Definitionen
- Locking

Benutzer 2

```
Terminal
SQL> SET AUTOCOMMIT OFF;
1 SQL> SELECT * FROM Bankkonten;
      KONTONR  INHABER                                STAND
-----
      1234  Klemens Boehm                             500
      5678  Gunter Saake                               500

SQL>
SQL>
SQL>
2 SQL> SELECT * FROM Bankkonten;
      KONTONR  INHABER                                STAND
-----
      1234  Klemens Boehm                             500
      5678  Gunter Saake                               500

SQL>
SQL>
3 SQL> SELECT * FROM Bankkonten;
      KONTONR  INHABER                                STAND
-----
      1234  Klemens Boehm                             500
      5678  Gunter Saake                               500

SQL>
SQL>
4 SQL> SELECT * FROM Bankkonten;
      KONTONR  INHABER                                STAND
-----
      1234  Klemens Boehm                             300
      5678  Gunter Saake                               700

SQL> █
```

Einleitung
Definitionen
Locking

DBMS führen
einzelne Operationen
meist per Default als
Transaktion aus.

```
X Desktop
SQL> SET AUTOCOMMIT OFF;
SQL> SELECT * FROM Bankkonten;

KONTONR  INHABER                                STAND
-----  -
1234  Klemens Boehm                                500
5678  Gunter Saake                                 500

SQL> UPDATE Bankkonten SET Stand=Stand-200 WHERE Kontonr=1234;
1 row updated.

SQL> SELECT * FROM Bankkonten;

KONTONR  INHABER                                STAND
-----  -
1234  Klemens Boehm                                300
5678  Gunter Saake                                 500

SQL> UPDATE Bankkonten SET Stand=Stand+200 WHERE Kontonr=1234;
1 row updated.

SQL> SELECT * FROM Bankkonten;

KONTONR  INHABER                                STAND
-----  -
1234  Klemens Boehm                                300
5678  Gunter Saake                                 700

SQL> ROLLBACK;
Rollback complete.

SQL> SELECT * FROM Bankkonten;

KONTONR  INHABER                                STAND
-----  -
1234  Klemens Boehm                                500
5678  Gunter Saake                                 500

SQL>
```

Rollback
als Gegensatz
zum **Commit**.

Transaktionseigenschaften I

- ACID-Eigenschaften:
 - **Atomicity**
 - ‚Alles oder nichts‘ (Auch bei Systemabsturz!)
 - **Consistency**
 - Überführung der DB von einem konsistenten Zustand in einen anderen (sonst: Abbruch).
 - **Isolation**
 - Jede Transaktion hat die DB ‚für sich allein‘.
 - **Durability**
 - Änderungen erfolgreicher Transaktionen dürfen nicht verloren gehen.

Integritäts-
bedingungen

Logging,
Recovery

Einleitung
Definitionen
Locking

Synchronisation in Datenbanken

- Zentrales Leistungsmerkmal von Datenbanken:
Viele Benutzer können die gleichen Daten gleichzeitig sowohl lesend als auch schreibend zugreifen.
- Konsistenz muss sichergestellt sein – Aufgabe der **Synchronisationskomponente**.
- Benutzer sollen vom Multi-User Betrieb so wenig wie möglich merken.
Nebenläufigkeit soll transparent sein.
,Illusion', dass man der einzige Nutzer ist.
- Engl. *concurrency, concurrency control*.

Einleitung
Definitionen
Locking

Synchronisationsprobleme

- Unkontrollierte **nicht-serielle Ausführung** führt zu Inkonsistenz:
 - **Lost Updates.**
 - **Dirty Reads**, Lesen von Updates, die noch nicht committet sind.
 - **Non-repeatable reads**, inkonsistente Lesezugriffe, unterschiedliche Ergebnisse bei identischer Anfrage.

(Die folgenden Beispiele beschreiben die verschränkte Ausführung von Programmen ohne Transaktionen.)

Einleitung
Definitionen
Locking

Lost Update

- Programm T_1 transferiert EUR 300,- von A auf B, Programm T_2 schreibt Konto A 3 % Zinsen gut.
- Zinsen aus Schritt 5 von Programm T_2 gehen verloren, weil T_1 den Wert Schritt 6 überschreibt.

Einleitung
Definitionen
Locking

Schritt	T_1	T_2
1	Read(A, a1)	
2	a1 := a1-300	
3		Read(A, a2)
4		a2 := a2 *1.03
5		Write(A, a2)
6	Write(A, a1)	
7	Read(B, b1)	
8	b1 := b1 + 300	
9	Write(B, b1)	

Dirty Read

- *Commit, Abort.*
- Programm T_2 berechnet Zinsen auf einem Wert aus einem inkonsistenten Zustand

Schritt	T_1	T_2
1	Read(A, a1)	
2	a1 := a1-300	
3	Write(A, a1)	
4		Read(A, a2)
5		a2 := a2 * 1.03
6		Write(A, a2)
7		commit
8	Read(B, b1)	
9	...	
10	abort	

Einleitung
Definitionen
Locking

Non-Repeatable Reads

- Programm liest Datenobjekt mehr als einmal und sieht Änderung, die eine andere Transaktion durchgeführt hat. (Auch: Phantom-Problem)

Einleitung
Definitionen
Locking

Schritt	T1	T2
1	Read(A, a1)	
2	a1 := a1-300	
3	Write(A, a1)	
4		Read(A, a2)
5		a2 := a2 *1.03
6		Write(A, a2)
7	Read(A, a3)	
8	...	

Lösbar ohne Synchronisation, aber...

- **Serielle Ausführung** von Transaktionen
 - neue Transaktion kann erst starten, wenn vorhergehende alle Daten geschrieben hat
 - Datenbank ist am Ende jeder Transaktion konsistent.
 - kein Aufwand durch Sperren und Scheduling
 - **Aber:** Extreme Wartezeiten und unbefriedigende Ausnutzung der Ressourcen.
 - während Festplatte arbeitet, steht der Rest des Rechners ungenutzt still

Einleitung
Definitionen
Locking

Maßnahmen zur Synchronisation

- Zusammenfassen von mehreren Zugriffen zu **Transaktionen**
 - **Commit/rollback bestätigt/verwirft alle Änderungen der Transaktion**
- Sperren von einzelnen Lese/Schreibzugriffen
- Umsortieren der Reihenfolge von parallelen Transaktionen
- Abbrechen von Transaktionen, die mit anderen in Konflikt stehen
 - DBMS kann Transaktion selbst zurücksetzen

Problem: Tradeoff zwischen Leistung der DB und Garantien zur Isolation!

Einleitung
Definitionen
Locking



Zielstellung dieses Kapitels

- Wie muss ein Ablaufplan (***History***) für mehrere Transaktionen aussehen, der
 - das gleiche Resultat zeigt wie eine serielle Ausführung, aber
 - die Ressourcen des Rechners durch parallele Ausführung besser ausnutzt?

Einleitung
Definitionen
Locking



Agenda

- Einleitung / Probleme
- **Definitionen**
 - Transaktionen
 - Konflikte
 - Histories
 - Äquivalenz
 - Serialisierbarkeit
 - Rücksetzbarkeit
- Locking

Transaktionen

- *Transaktion* := Ausführung eines Programms, das auf die Datenbank (lesend oder schreibend) zugreift.
- Programmausführung \neq Programm-Code.
- Genauer: Repräsentation der Ausführung, die folgende Charakteristika umfaßt:
 - Gelesene und geschriebene Datenobjekte,
 - Reihenfolge ihrer Ausführung,
 - kommt am Ende ein Commit (bzw. Abort) oder nicht?

Einleitung
Definitionen
Locking

Beispiel für Transaktionen

- Beispiel:

Procedure P begin

Start;

temp := Read(x);

temp := temp + 1;

Write(x, temp);

Commit

end

Operationen

- Repräsentation: $r_1[x] \rightarrow w_1[x] \rightarrow c_1$
- Transaktion ist partielle Ordnung $(\Sigma, <)$
(Σ wird im Folgenden meistens weggelassen.)

Einleitung
Definitionen
Locking

Transaktion – formale Definition

Transaktion ist partielle Ordnung

mit Ordnungsrelation $<$, so dass gilt

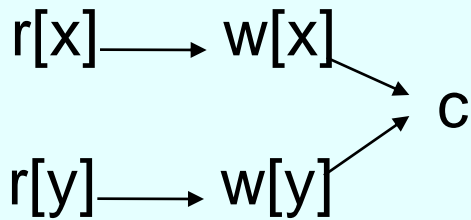
1. $T_i \subseteq \{r_i[x], w_i[x] \mid x \text{ ist ein Datenobjekt}\} \cup \{a_i, c_i\}$,
2. $a_i \in T_i \Leftrightarrow c_i \notin T_i$;
3. wenn es sich bei $t \in T_i$ um c_i oder a_i handelt,
dann gilt für jede andere Operation $p \in T_i$: $p <_i t$;
und
4. wenn $r_i[x], w_i[x] \in T_i$,
dann $r_i[x] <_i w_i[x]$ oder $w_i[x] <_i r_i[x]$.

Einleitung
Definitionen
Locking

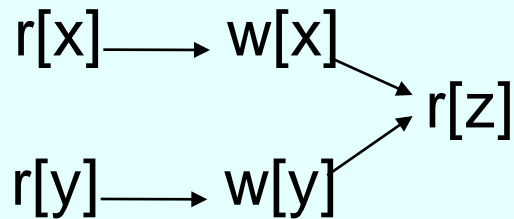
Transaktion – weitere Beispiele

- Gegeben: Halbordnung von Operationen

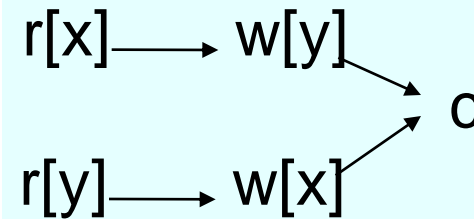
Einleitung
Definitionen
Locking



ist Transaktion.



ist keine TA.



ist keine TA.

Beziehungen zwischen Transaktionen

- *Im DBMS viele parallele Transaktionen möglich*

Einleitung
Definitionen
Locking

- ***Reads-from-Beziehung:***
Transaktion T_i liest von Transaktion T_j wenn

1. T_i liest x , nachdem T_j x geschrieben hat;
2. T_j abortet nicht, bevor T_i x liest; und
3. Jede Transaktion, die x schreibt, bevor T_i x liest, und nachdem T_j x überschreibt, abortet, bevor T_i x liest.

Beispiele für Beziehungen

- $w_1[x] w_2[x] r_3[x] r_4[x] c_1 c_2 c_3 c_4$
reads-from Beziehungen:
 T_3 von T_2 , T_4 von T_2 .
- $w_1[x] w_2[x] r_3[x] r_4[x] c_1 a_2 c_3 c_4$
reads-from Beziehungen:
 T_3 von T_2 , T_4 von T_2 .
- $w_1[x] w_2[x] a_2 r_3[x] r_4[x] c_1 c_3 c_4$
reads-from Beziehungen:
 T_3 von T_1 , T_4 von T_1 .

Einleitung
Definitionen
Locking

Konflikte

- **Zwei Operationen p, q konfliktieren** :=
p, q greifen auf das gleiche Datenobjekt zu,
und p oder q ist eine Schreiboperation.
- Visualisierung als Kompatibilitätsmatrix:

	Read	Write
Read	y	n
Write	n	n



Histories

- Ausführung der Operationen mehrerer Transaktionen, die miteinander ‘verzahnt’ sind, d.h. nebenläufig ablaufen.
- Formal: $H = \{T_1, T_2, \dots, T_n\}$ ist eine Menge von Transaktionen.

- Anm.: in der Literatur werden *Histories* auch häufig als *Schedules* bezeichnet

Einleitung
Definitionen
Locking

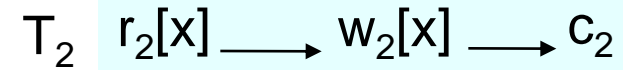
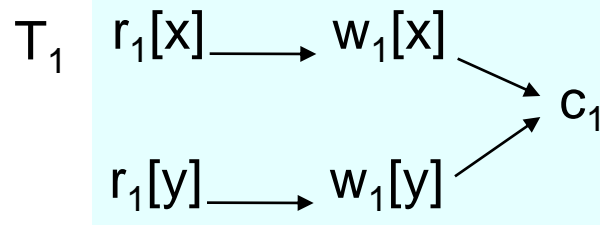
Definition von Histories

- *Vollständige History* H über $\{T_1, T_2, \dots, T_n\} :=$ partielle Ordnung mit Ordnungsbeziehung $<_H$, so dass
 1. $H = \bigcup_i T_i$
 2. $<_H \supseteq \bigcup_i <_i$
 3. p, q konfligieren ($p \in T_p; q \in T_q; T_p, T_q \in H$)
 $\rightarrow p <_H q$ oder $q <_H p$

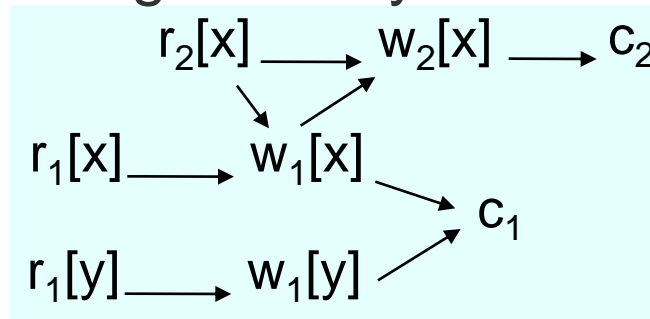
Einleitung
Definitionen
Locking

Histories – Beispiele I

- Gegeben zwei Transaktionen:



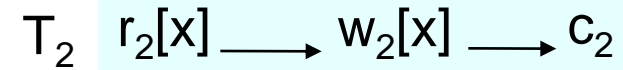
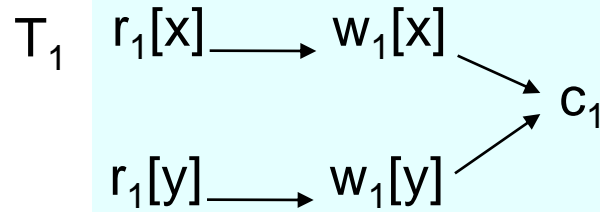
- Vollständige History:



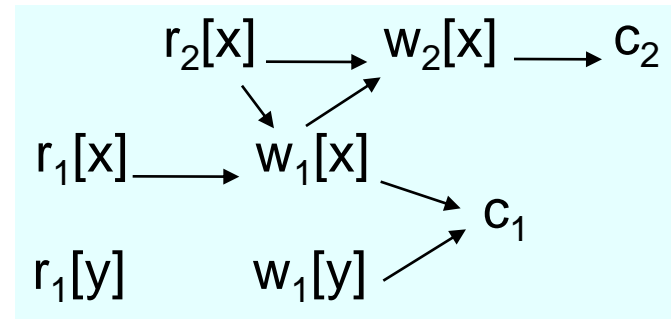
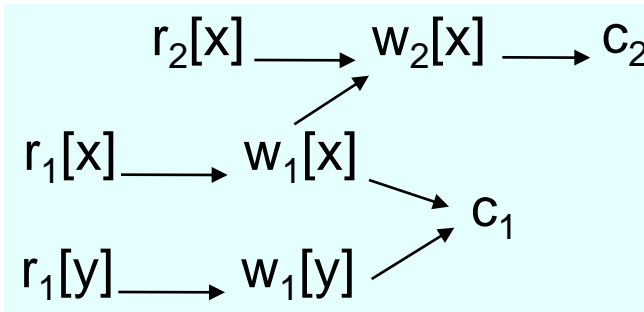
Einleitung
Definitionen
Locking

Histories – Beispiele II

- Gegeben zwei Transaktionen:



- Keine Histories:



Einleitung
Definitionen
Locking

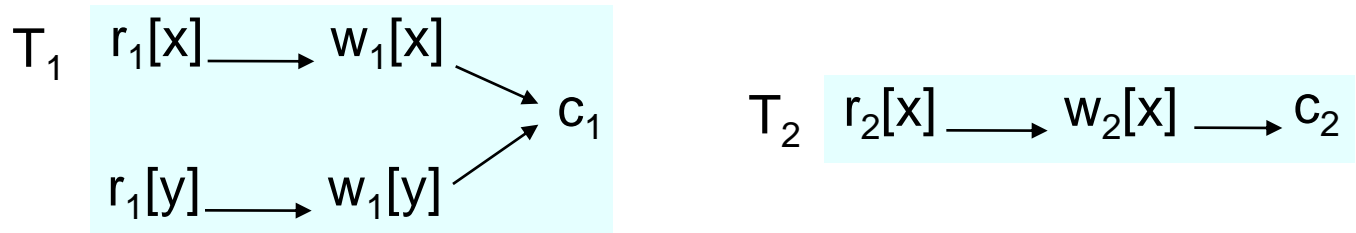
Äquivalenz von Histories

- Mehrere Definitionen möglich:
 - (Konflikt-)Äquivalenz
 - (Sicht-)Äquivalenz
- Im Folgenden wird nur Konflikt-Äquivalenz betrachtet.
- Definition **Konflikt-Äquivalenz**:
Histories H, H' sind Konfliktäquivalent, wenn
 1. gleiche Transaktionen, gleiche Operationen;
 2. gleiche Ordnung konfligierender Operationen.
z.B. gehören p_i und q_j zu T_i bzw T_j .
 $a_i, a_j \notin H$. Wenn $p_i <_H q_j$, dann $p_i <_{H'} q_j$.

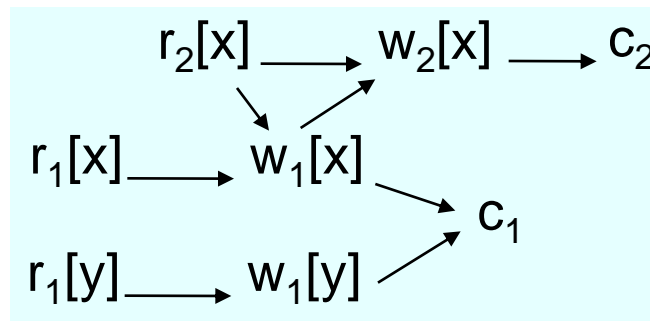
Einleitung
Definitionen
Locking

Äquivalenz von Histories – Beispiele I

- Gegeben zwei Transaktionen:



- Eine vollständige History:



- Beispiel für eine Konflikt-äquivalente History, die nicht identisch ist?

Äquivalenz von Histories – Beispiele II

- History 1:

Schritt	T1	T2	T3
1	Read(A)		
2		Write(A)	
3	Write(A)		
4			Write(A)

Alle
Transaktionen
committen

- History 2:

Schritt	T1	T2	T3
1	Read(A)		
2	Write(A)		
3		Write(A)	
4			Write(A)

- Sind diese Histories Konflikt-äquivalent?

Einleitung
Definitionen
Locking



Überlegungen zur Korrektheit

- History muß nicht korrekt sein, kann z.B. Lost Updates etc. enthalten.
- Ziel im folgenden:
Suche eine Definition 'Korrektheit' für Histories.

Einleitung
Definitionen
Locking

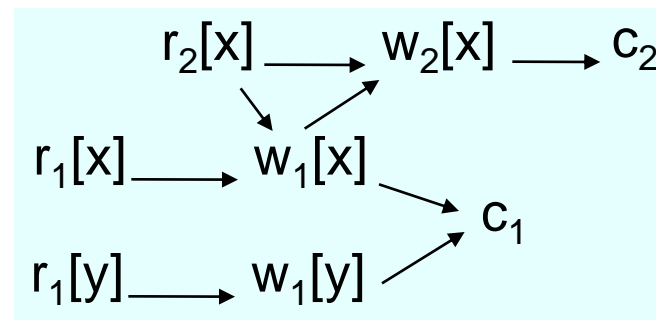
Eigenschaften von Histories

- **Conflict-Serialisability (CSR)**
 - **Konflikt-Serialisierbarkeit**
- **Recoverability (RC)**
 - Rücksetzbarkeit
- **Avoids Cascading Aborts (ACA)**
 - keine kaskadierenden Abbrüche
- **Strictness (ST)**
 - keine Zwischenergebnisse von uncommitted Transaktionen lesen

Einleitung
Definitionen
Locking

Konflikt-Serialisierbarkeit I

- H ist serialisierbar, wenn H äquivalent zu einer seriellen History H_S ist.
- Ist diese History serialisierbar?
Wenn ja, wie sieht äquivalente serielle History aus?



Konflikt-Serialisierbarkeit – Beispiel

- **Serielle Ausführung: Variante 1**
 - $r_2(x)w_2(x)c_2r_1(x)w_1(x)r_1(y)w_1(y)c_1$
 - Konflikte: $(w_2(x), r_1(x))$
- **Serielle Ausführung: Variante 2**
 - $r_1(x)w_1(x)r_1(y)w_1(y)c_1r_2(x)w_2(x)c_2$
 - Konflikte: $(w_1(x), r_2(x))$
- **Beispiel: Zu untersuchende History**
 - $r_1(x)r_2(x)w_1(x)r_1(y)w_1(y)w_2(x)c_1c_2$
 - Konflikte: $(r_2(x), w_1(x)), (w_1(x), w_2(x))$
- **Hinweis: Operationen sind immer gleich**

Serialisierbarkeitsgraph I

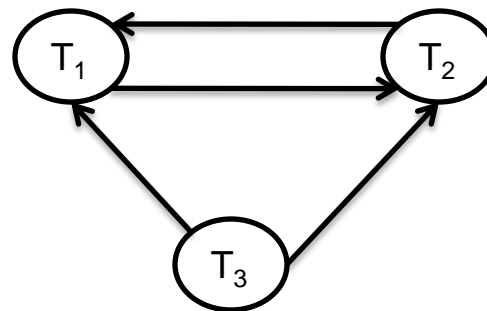
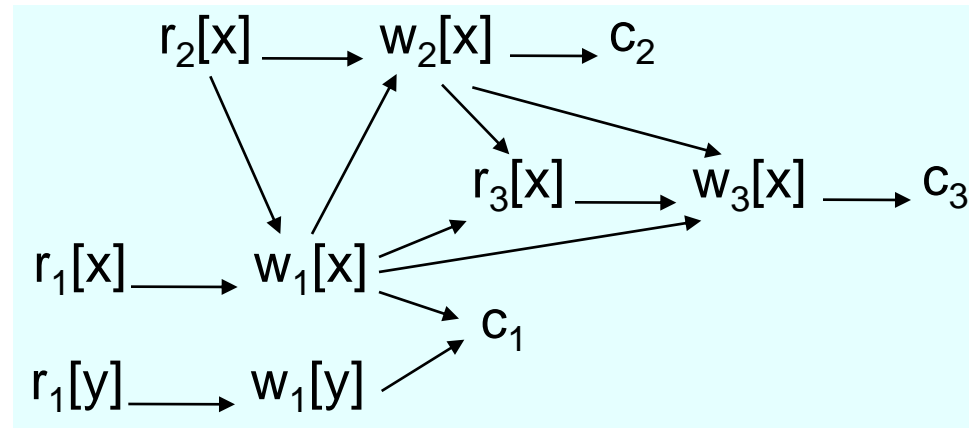
- Test, ob eine History (= Transaktionen, zusammen mit ihren Operationen) serialisierbar ist:
 - Erzeuge Serialisierbarkeitsgraphen bzw. **Abhängigkeitsgraphen**.
 - Knoten = Transaktionen,
 - (gerichtete) Kante = Abhängigkeit zwischen zwei Transaktionen:
Transaktionen greifen auf das gleiche Datenobjekt zu, und Operationen konfliktieren.

Einleitung
Definitionen
Locking

Serialisierbarkeitsgraph II

- Wie sieht der Serialisierbarkeitsgraph aus?

Einleitung
Definitionen
Locking





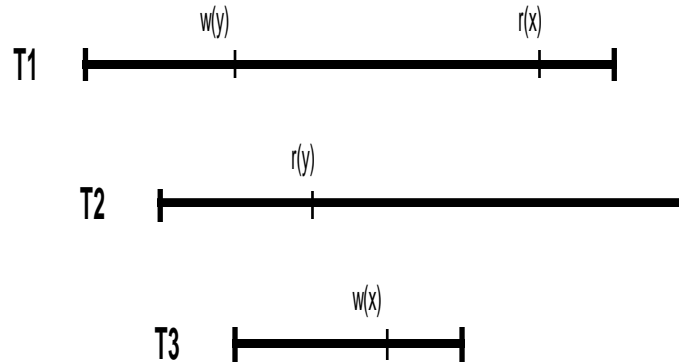
Serialisierbarkeitsgraph III

- Theorem: Schedule ist serialisierbar, wenn entsprechender Abhängigkeitsgraph *zyklenfrei* ist.
- Denn: Partielle Ordnung, zu totaler Ordnung erweiterbar – äquivalenter serieller Schedule.

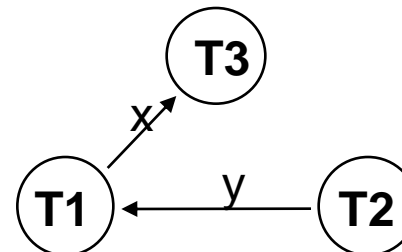
Einleitung
Definitionen
Locking

Serialisierbarkeitsgraph – Beispiel

- $r(x)/w(x)$ – Lese-/Schreibzugriff auf Datenobjekt x .
- Schedule:



- Abhängigkeitsgraph:



- azyklisch \rightarrow Schedule ist serialisierbar.
- Serialisierungsreihenfolge: $T3 < T1 < T2$



Serialisierbarkeitsgraph – Diskussion

Erstellen und Überprüfen von
Serialisierbarkeitsgraphen zur Laufzeit ist
kaum praktikabel.

- Serialisierbarkeit von Schedules nur im nachhinein überprüfbar.
- Administrativer Overhead ist hoch: Abhängigkeiten zu bereits terminierten Transaktionen müssen ebenfalls berücksichtigt werden.

Einleitung
Definitionen
Locking

Eigenschaften von Histories

- **Conflict-Serialisability (CSR)**
 - Konflikt-Serialisierbarkeit
- **Recoverability (RC)**
 - Rücksetzbarkeit
- **Avoids Cascading Aborts (ACA)**
 - keine kaskadierenden Abbrüche
- **Strictness (ST)**
 - keine Zwischenergebnisse von uncommitted Transaktionen lesen

Einleitung
Definitionen
Locking

Recoverability I

- ‘Commit einer Transaktion erfolgreich durchgeführt’ – kein Abort mehr möglich.
- Commit nur, wenn alle Änderungen an Datenobjekten, die T gelesen hat, committet sind.
- Gegenbeispiel: Dirty Read.

Schritt	T1	T2
1	Read(A, a1)	
2	a1 := a1 - 300	
3	Write(A, a1)	
4		Read(A, a2)
5		a2 := a2 * 1.03
6		Write(A, a2)
7		commit
8	Read(B, b1)	
9	...	
10	abort	



Recoverability II

- *History ist recoverable* :=
Commit von T nach Commit aller Transaktionen,
von denen T gelesen hat.

- folgendes **darf nie** passieren:
 $r_1[x] w_1[x] r_2[x] w_2[x] c_2 a_1$

Einleitung
Definitionen
Locking



Cascading Aborts I

- Transaktion T abortet,
wenn sie nicht korrekt beenden kann.
- T muß zurückgesetzt werden:
 - Writes von T,
 - dto. alle anderen Transaktionen,
die solche Änderungen gelesen haben.
Undo kann zu *cascading abort* führen.

Einleitung
Definitionen
Locking



Cascading Aborts II

- Beispiel für cascading abort:
 - Zwei Transaktionen T_1 und T_2
 $r_1[x] w_1[x] r_2[x] a_1 \dots$
 - Abort von T_1 , undo $w_1[x]$
 $\Rightarrow T_2$ muß ebenfalls aborten.
- Cascading aborts sind möglich,
auch wenn die History recoverable ist.

Einleitung
Definitionen
Locking



Cascadelessness

- Cascading aborts sind unerwünscht:
 - Sie ziehen 'Buchhaltung' nach sich,
 - Zahl der Transaktionen, die aborten müssen, ist nicht beschränkt.
- History ist *cascadeless* :=
Jede Transaktion liest Datenobjekte nur von committeten Transaktionen.

Einleitung
Definitionen
Locking

Strictness

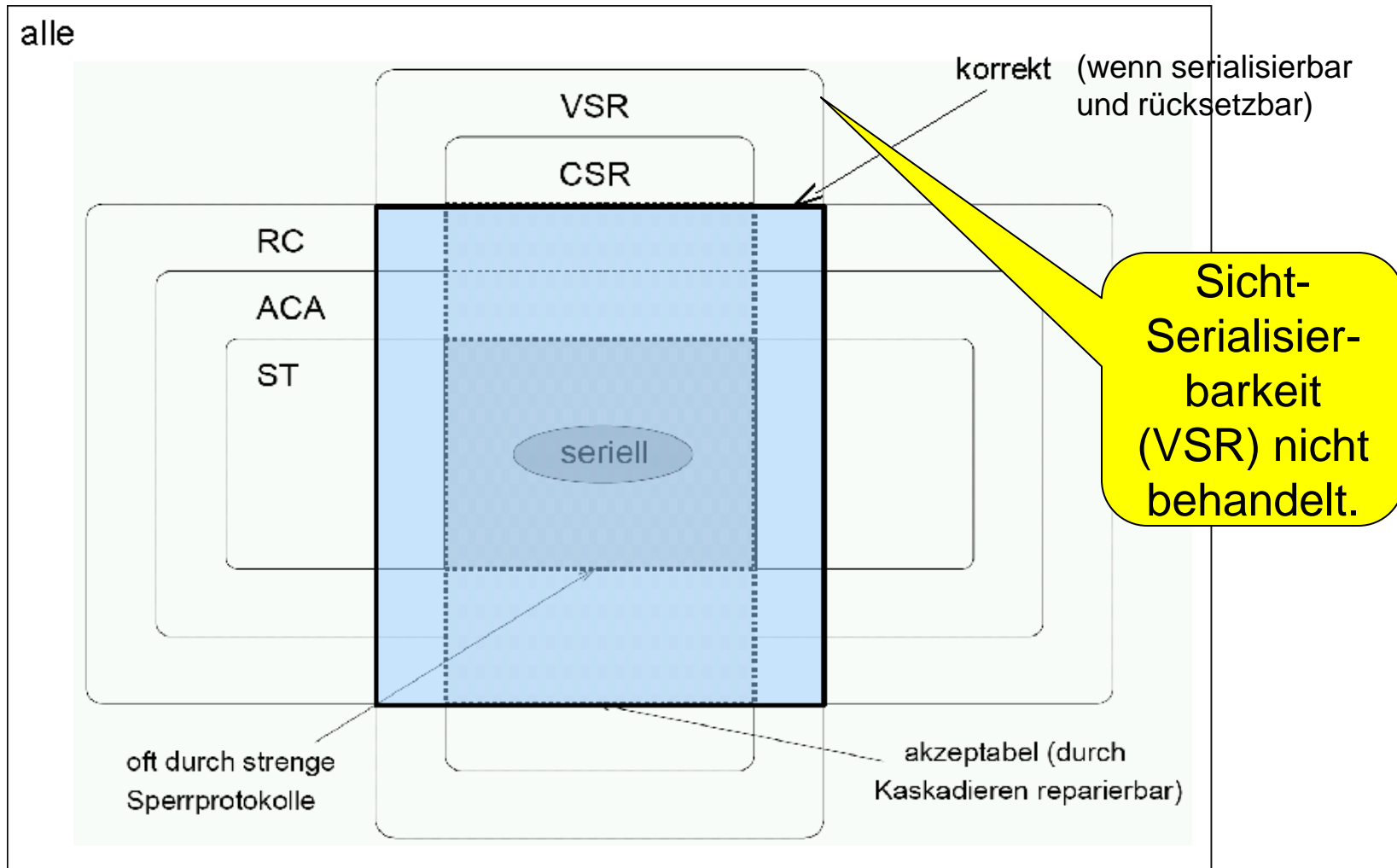
- Problem: Undo basiert auf *Before-Images* BF

Einleitung
Definitionen
Locking

DB-Inhalt	Operation
$x = 1$ (Anfangswert)	
$x = 2$	$w_1(x \leftarrow 2)$ [$BF_{x,T_1} = 1$]
$x = 3$	$w_2(x \leftarrow 3)$ [$BF_{x,T_2} = 2$] a_1 Rücksetzen von $w_1(x \leftarrow 2)$ mit $BF_x := 1$. Überschreiben durch T_2 muß erhalten bleiben!
$x = 3$	a_2 Rücksetzen von $w_2(x \leftarrow 3)$ mit $BF_x := ??$

Strictness := kein Wert einer nicht abgeschlossenen Transaktion darf gelesen oder überschrieben werden.

Zusammenfassung



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?
- Welche Histories sind korrekt?

Einleitung
Definitionen
Locking



Agenda

- Einleitung / Probleme
- Definitionen
 - Transaktionen
 - Konflikte
 - Histories
 - Äquivalenz
 - Serialisierbarkeit
 - Rücksetzbarkeit
- **Locking**

Verzögern und Zurücksetzen

- Verzögern und Zurücksetzen mittels Locking sind übliche Techniken in der Transaktionsverwaltung.
- Locking:
 - Datenobjekte werden zum Schreiben und/oder Lesen gesperrt
 - Verzögerte Transaktionen müssen warten bis Lock aufgehoben
 - Zahlreiche Strategien zum Locking möglich, im folgenden: 2-Phase-Locking

Im Allgemeinen immer noch schneller als rein serielle Ausführung der Transaktionen

Einleitung
Definitionen
Locking



Locking

- Transaktion T_i setzt Lock für Operation o auf Datenobjekt x – Notation: $o|_i[x]$
- Einfachster Fall: Nur Read Locks und Write Locks („RX Locking Scheme“).
 - read lock (rl): Lesesperre
 - write lock (wl): Schreibsperre
 - read/write unlock (ru/wu): Sperre aufheben

Einleitung
Definitionen
Locking



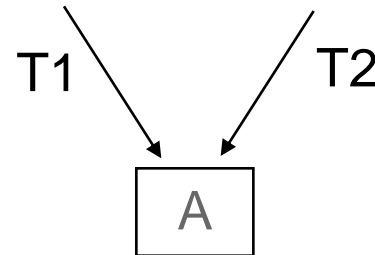
Sperrdisziplin

- Schreibzugriff $w[x]$ nur nach $wl[x]$
- Lesezugriff $r[x]$ nur nach $rl[x]$ oder $wl[x]$
- Transaktion darf nur Objekte sperren, die keine *konfligierenden* Locks von anderen Transaktionen tragen, sonst: verzögert bis ru/wu
- ein $rl[x]$ darf zum $wl[x]$ verschärft werden
- nach dem Entsperren darf eine Transaktion das selbe Objekt nicht erneut sperren
- vor Commit alle Sperren aufheben

Einleitung
Definitionen
Locking

Konflikte beim Locking

- Locks können konfliktieren.



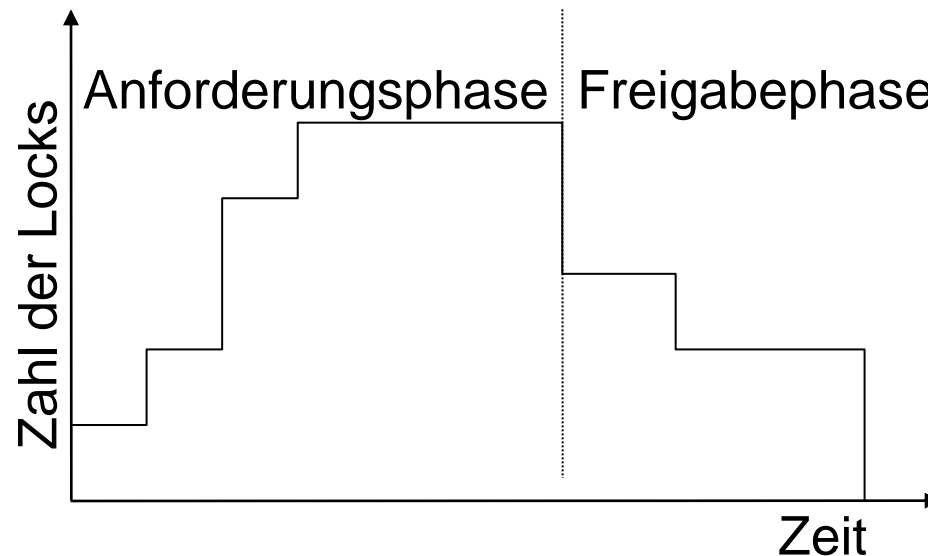
- Locks konfliktieren in gleicher Weise wie entsprechende Operationen.

	rl_i	wl_i
rl_j	y	n
wl_j	n	n

→ mehrere Lesesperren auf dem gleichen Objekt möglich

Zwei-Phasen-Sperrprotokoll

- **Zwei-Phasen-Sperrprotokoll** (two-phase locking, 2PL) stellt Serialisierbarkeit (CSR) sicher (kein RC!).
 - Zwei Phasen:
 - Locks hinzunehmen
 - Locks freigeben



Einleitung
Definitionen
Locking

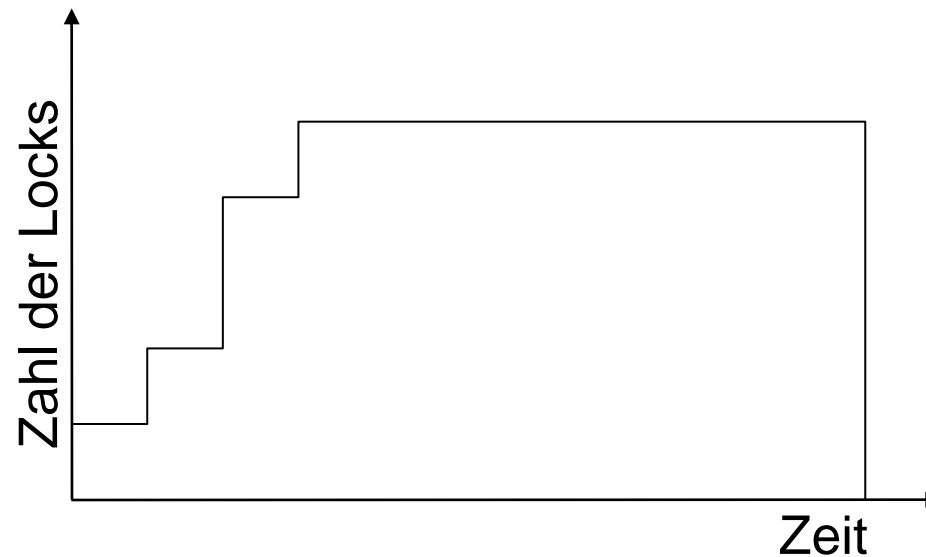
Beispiel für Deadlock mit 2PL

- $T_1: r_1[x] \rightarrow w_1[y] \rightarrow c_1; T_2: w_2[y] \rightarrow w_2[x] \rightarrow c_2$
- Abfolge:
 1. Beide Transaktionen starten ohne Locks.
 2. TM sendet $r_1[x]$ an Scheduler.
 $rl_1[x]$, Scheduler sendet $r_1[x]$ an DM.
 3. TM sendet $w_2[y]$ an Scheduler.
 $wl_2[y]$, Scheduler sendet $w_2[y]$ an DM.
 4. TM sendet $w_2[x]$ an Scheduler.
 $wl_2[x]$ nicht möglich. Verzögerung.
 5. TM sendet $w_1[y]$ an Scheduler.
 $wl_1[y]$ nicht möglich. Verzögerung.

Deadlock!
Muss extern
zurückgesetzt
werden

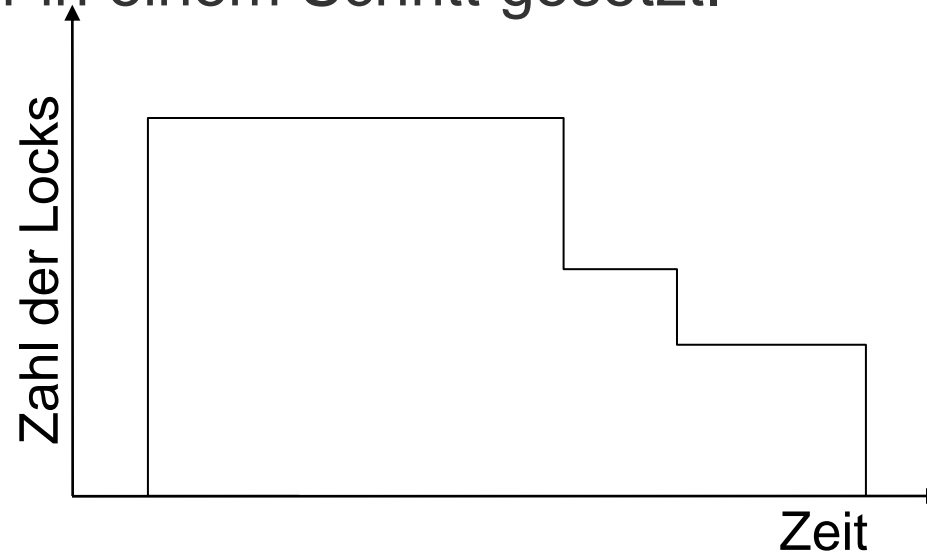
Strenges 2PL

- Stellt zudem Striktheit (ST) sicher (so auch ACA und RC) :
Freigabe der Locks erst am Ende der Transaktion.



Konservatives 2PL

- Stellt Deadlock-Freiheit sicher:
Alle Locks werden am Anfang der Transaktion atomar in einem Schritt gesetzt.



Kombination möglich: **Konservatives Strenges Zwei-Phasen-Sperrprotokoll**



Zusammenfassung

- Nebenläufiger Zugriff
 - fundamental wichtiges Anliegen.
- Serielle Ausführung wäre *immer* korrekt, ist wegen mangelhafter Performance aber nicht akzeptabel.
- Korrektheitskriterium:
 - Äquivalenz zu serieller Ausführung und Rücksetzbarkeit!
- Two-Phase Locking stellt Serialisierbarkeit sicher.

Einleitung
Definitionen
Locking