

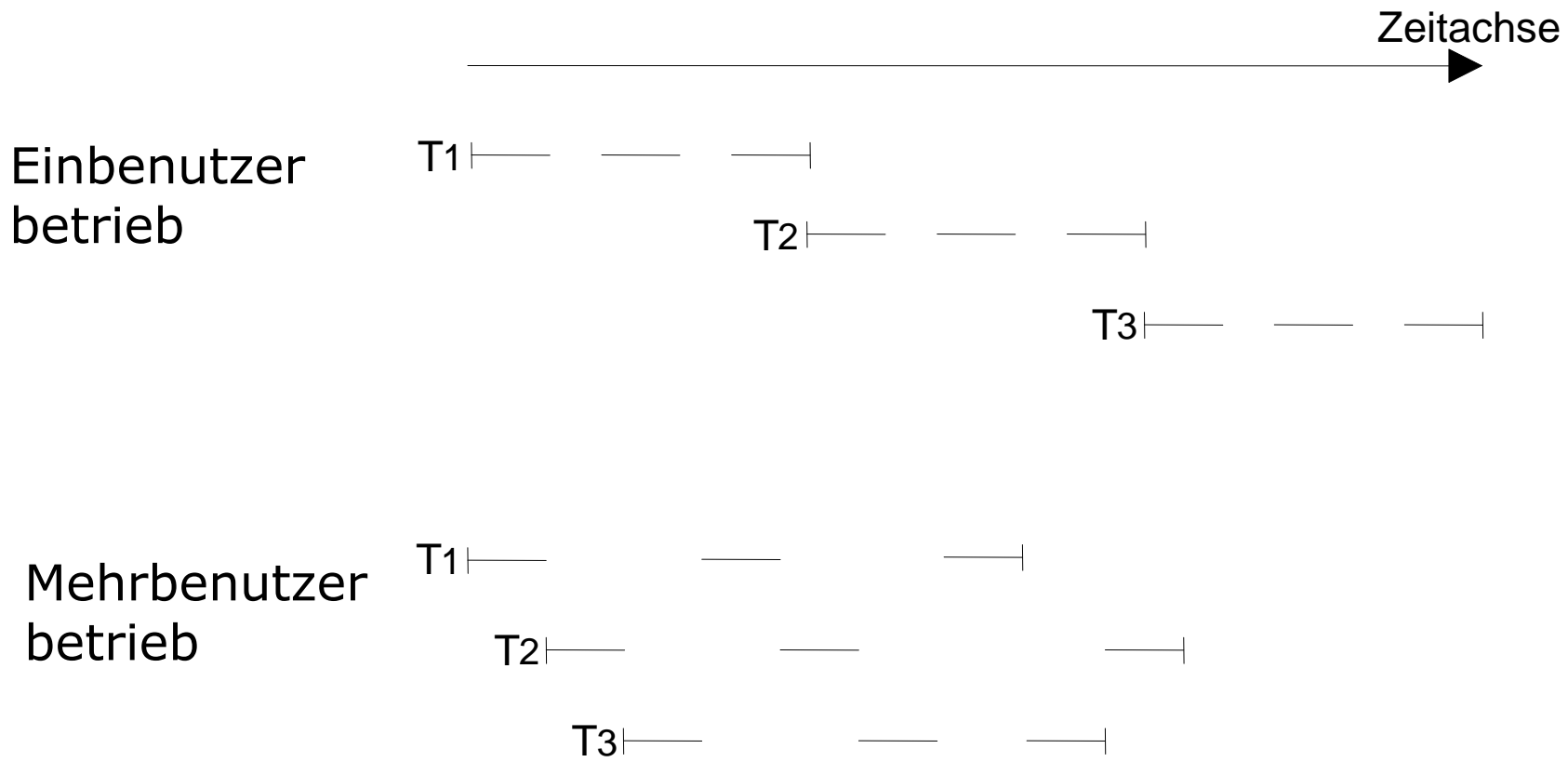
Datenbanksysteme 2011

Kapitel 15: Mehrbenutzersynchronisation

Oliver Vornberger

Institut für Informatik
Universität Osnabrück

Multiprogramming



Lost Update

T_1	T_2
<code>read(A, a₁)</code>	
<code>a₁ := a₁ - 300</code>	
	<code>read(A, a₂)</code>
	<code>a₂ := a₂ * 1.03</code>
	<code>write(A, a₂)</code>
<code>write(A, a₁)</code>	
<code>read(B, b₁)</code>	
<code>b₁ := b₁ + 300</code>	
<code>write(B, b₁)</code>	

Dirty Read

T_1	T_2
<code>read(A, a₁)</code>	
<code>a₁ := a₁ - 300</code>	
<code>write(A, a₁)</code>	
	<code>read(A, a₂)</code>
	<code>a₂ := a₂ * 1.03</code>
	<code>write(A, a₂)</code>
<code>read(B, b₁)</code>	
<code>. . .</code>	
<code>abort</code>	

Phantomproblem

T1	T2
	<pre>select sum(KontoStand) from Konten;</pre>
<pre>insert into Konten values (C, 1000, . . .);</pre>	<pre>select sum(KontoStand) from Konten;</pre>

Historie, Schedule

Wichtig: Lese/Schreiboperationen

Unwichtig: lokale Variable

Historie =

Schedule = Festlegung für die Reihenfolge
sämtlicher relev. Datenbankoperationen.

seriell =

alle Schritte einer Transaktion
unmittelbar hintereinander

serialisierbar = es gibt äquiv. serielles Schedule

Schedule

nichtseriell

seriell

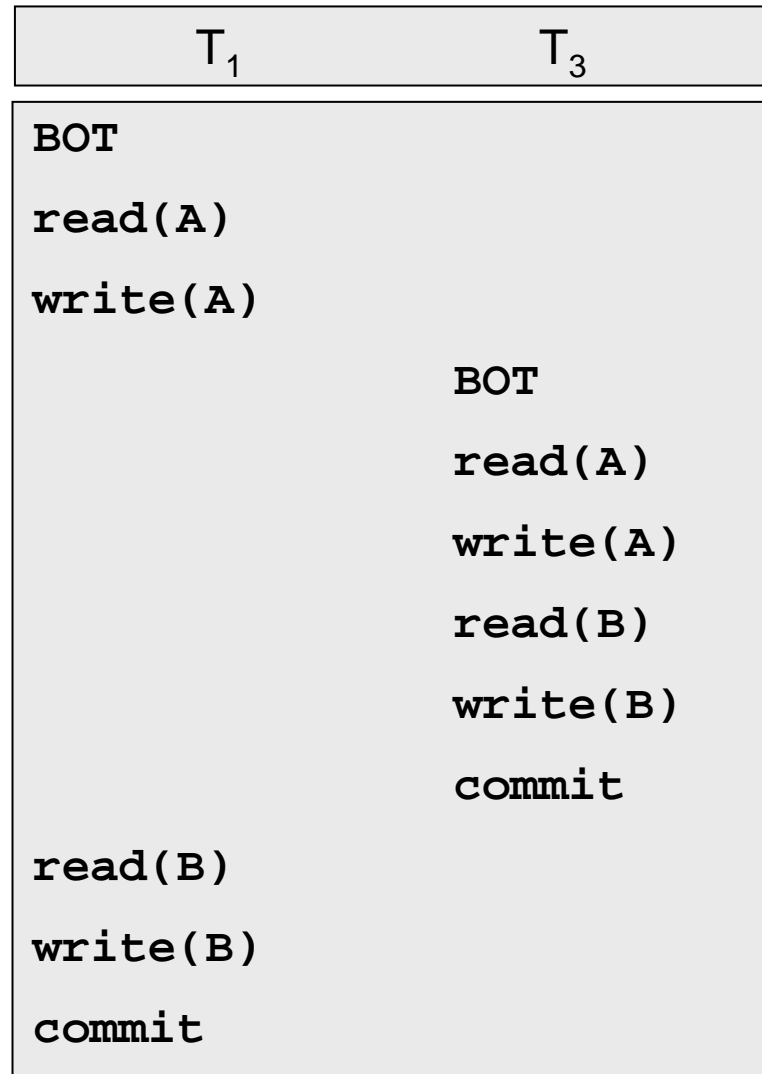
T ₁	T ₂
BOT	
read(A)	
	BOT
	read(C)
write(A)	
	write(C)
read(B)	
write(B)	
commit	
	read(A)
	write(A)
	commit



serialisierbar

T ₁	T ₂
BOT	
read(A)	
write(A)	
read(B)	
write(B)	
commit	
	BOT
	read(C)
	write(C)
	read(A)
	write(A)
	commit

Nicht serialisierbares Schedule



wegen A:
 T_1 , dann T_3

wegen B:
 T_3 , dann T_1

Semantik "Betrag überweisen"

T_1	T_3
<pre>BOT read(A, a₁) a₁ := a₁ - 50 write(A, a₁)</pre>	<pre>BOT read(A, a₂) a₂ := a₂ - 100 write(A, a₂) read(B, b₂) b₂ := b₂ + 100 write(B, b₂) commit</pre>
<pre>read(B, b₁) b₁ := b₁ + 50 write(B, b₁) commit</pre>	

Zufällig ist
Reihenfolge
unerheblich !

Semantik "Zinsen gutschreiben"

T_1	T_3
<pre>BOT read(A, a₁) a₁ := a₁ - 50 write(A, a₁)</pre>	<pre>BOT read(A, a₂) a₂ := a₂ * 1.03 write(A, a₂) read(B, b₂) b₂ := b₂ * 1.03 write(B, b₂) commit</pre>
<pre>read(B, b₁) b₁ := b₁ + 50 write(B, b₁) commit</pre>	

3 % Zinsen
fehlen !

Elementare Operationen

Bezogen auf Transaktion i :

- $r_i(A)$ zum Lesen von Datenobjekt A ,
- $w_i(A)$ zum Schreiben von Datenobjekt A ,
- a_i zur Durchführung eines **abort**,
- c_i zur Durchführung eines **commit**.

Vier Fälle

- $r_i(A)$ und $r_j(A)$: kein Konflikt, Reihenfolge unerheblich
- $r_i(A)$ und $w_j(A)$: Konflikt, Reihenfolge entscheidend
- $w_i(A)$ und $r_j(A)$: Konflikt, Reihenfolge entscheidend
- $w_i(A)$ und $w_j(A)$: Konflikt, Reihenfolge entscheidend

Äquivalenz

Zwei Historien H_1 und H_2 über der gleichen Menge von Transaktionen sind äquivalent (in Zeichen $H_1 \equiv H_2$), wenn sie die Konfliktoperationen der nicht abgebrochenen Transaktionen in derselben Reihenfolge ausführen.

Testen auf Serialisierbarkeit

Input: Eine Historie H für Transaktionen T_1, \dots, T_k .

Output: entweder: "nein, ist nicht serialisierbar"
oder "ja, ist serialisierbar" + serielles Schedule

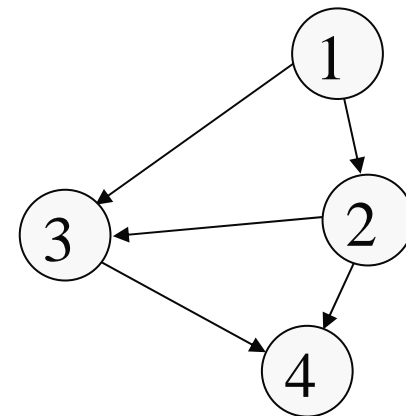
Idee: Bilde gerichteten Graph G , dessen Knoten den Transaktionen entsprechen.

Für zwei Konfliktoperationen p_i, q_j
aus der Historie H mit $p_i <_H q_j$
fügen wir die Kante $T_i \rightarrow T_j$ in den Graph ein.

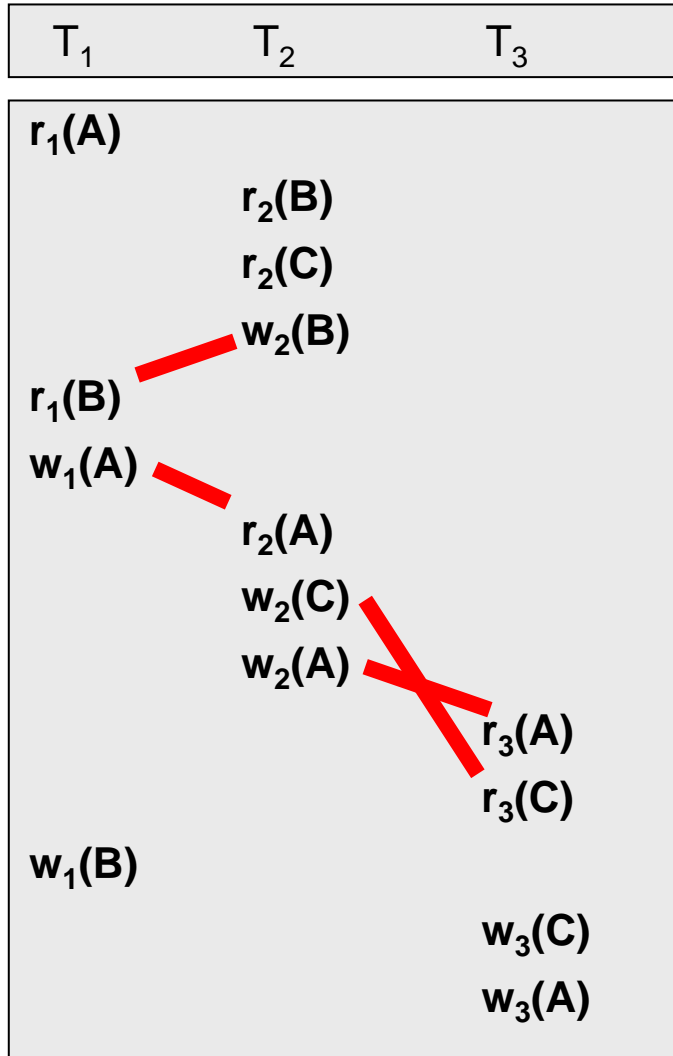
Serialisierbarkeitstheorem

Eine Historie H ist genau dann serialisierbar, wenn der zugehörige Serialisierbarkeitsgraph azyklisch ist.

Im Falle der Kreisfreiheit läßt sich die äquivalente serielle Historie aus der topologischen Sortierung des Serialisierbarkeitsgraphen bestimmen.



Beispiel



Konflikte:

$$w_2(B) < r_1(B)$$

$$w_1(A) < r_2(A)$$

$$w_2(C) < r_3(C)$$

$$w_2(A) < r_3(A)$$

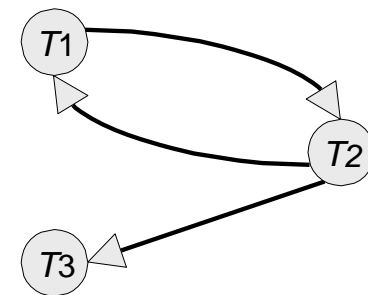
Kanten:

$$T_2 \rightarrow T_1$$

$$T_1 \rightarrow T_2$$

$$T_2 \rightarrow T_3$$

$$T_2 \rightarrow T_3$$



Sperrbasierte Synchronisation

Stelle durch Sperren die Serialisierbarkeit sicher:

S (shared, read lock, Lesesperre):

Wenn Transaktion T_i eine S-Sperre für Datum A besitzt, kann T_i **read**(A) ausführen. Mehrere Transaktionen können gleichzeitig eine S-Sperre auf demselben Objekt A besitzen

X (exclusive, write lock, Schreibsperre):

Ein **write**(A) darf nur die eine Transaktion ausführen, die eine X-Sperre auf A besitzt.

Kompatibilitätsmatrix

(vorhanden)

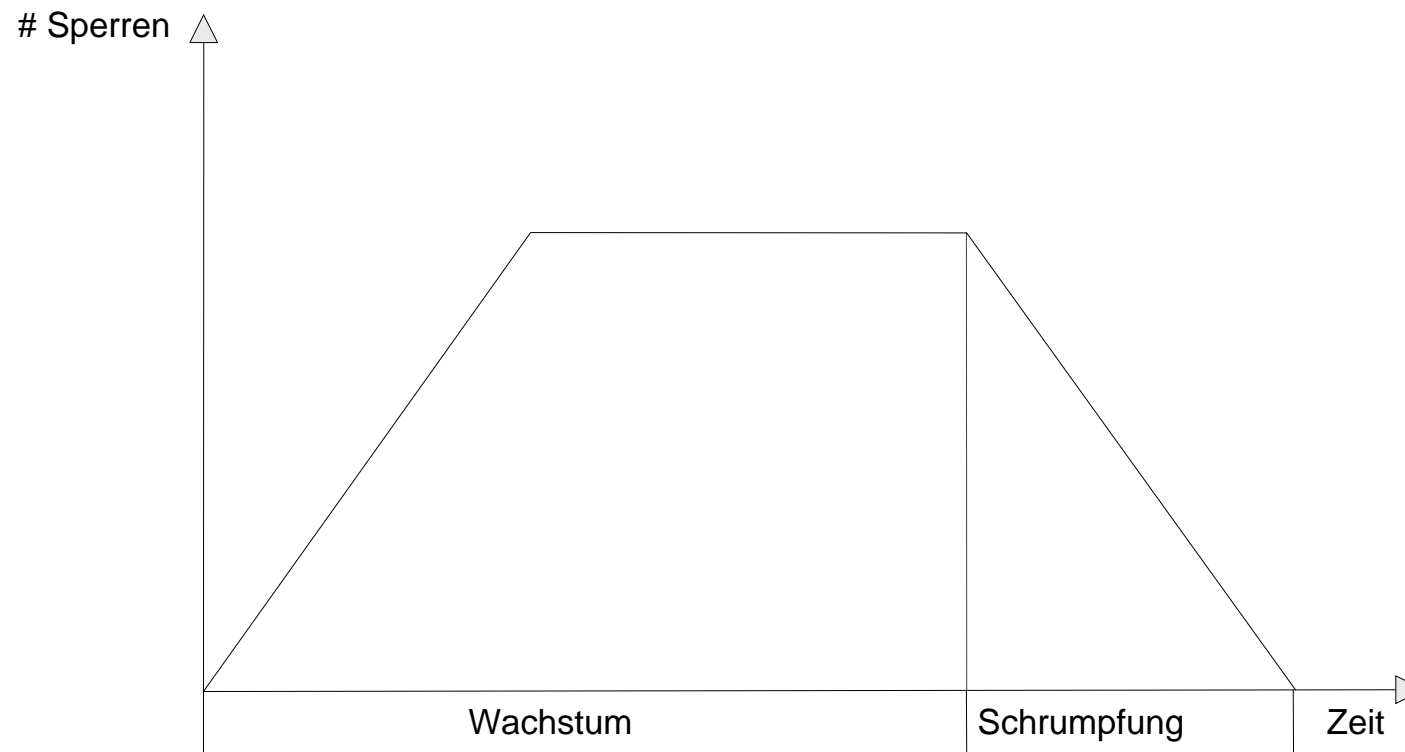
	NL	S	X
S	✓	✓	-
X	✓	-	-

Zwei-Phasen-Sperrprotokoll

- Jedes Objekt muß vor der Benutzung gesperrt werden.
- Eine Transaktion fordert eine Sperre, die sie schon besitzt, nicht erneut an.
- Eine Transaktion respektiert vorhandene Sperren gemäß der Verträglichkeitsmatrix und wird ggf. in eine Warteschlange eingereiht.
- Jede Transaktion durchläuft eine *Wachstumsphase* (nur Sperren anfordern) und dann eine *Schrumpfungsphase* (nur Sperren freigeben).
- Bei Transaktionsende muß eine Transaktion alle ihre Sperren zurückgeben

2-Phasen-Sperrprotokoll

liefert nur serialisierbare Schedules



2-Phasen-Sperrprotokoll

T_1	T_2	
BOT		
lockX(A)		
read(A)		
write(A)		
	BOT	
	lockS(A)	T_2 muß warten
lockX(B)		
read(B)		
unlockX(A)		T_2 wecken
	read(A)	
	lockS(B)	T_2 muß warten
write(B)		
unlockX(B)		T_2 wecken
	read(B)	
commit		
	unlockS(A)	
	unlockS(B)	
	commit	

Verklemmungen

T_1	T_2	
BOT		
lockX(A)		
	BOT	
	lockS(B)	
	read(B)	
read(A)		
write(A)		
lockX(B)		T_1 muß warten auf T_2
	lockS(A)	T_2 muß warten auf T_1
...	...	\Rightarrow Deadlock

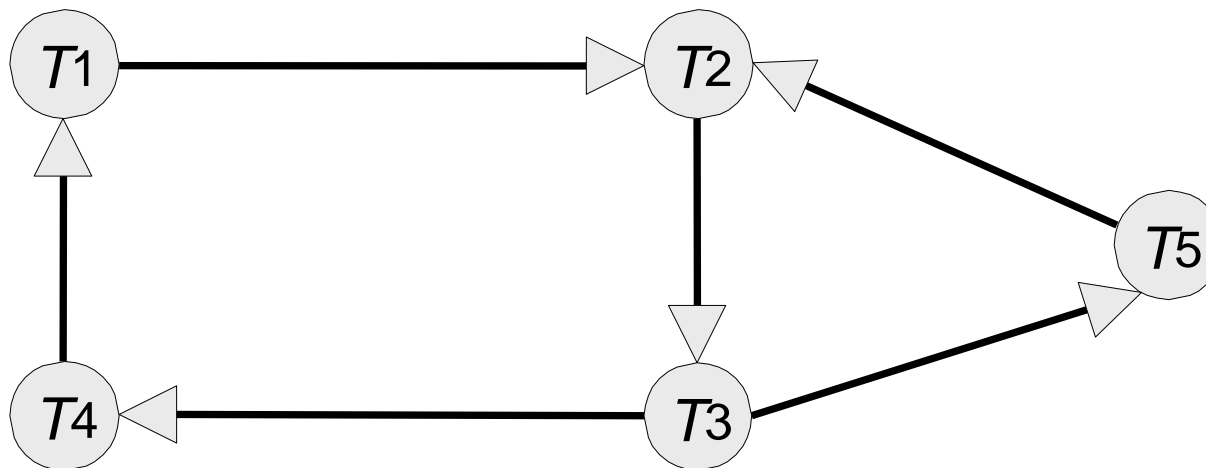
Wartegraphen

Knoten

Transaktionen

Kante von T_i nach T_j

T_i wartet auf T_j

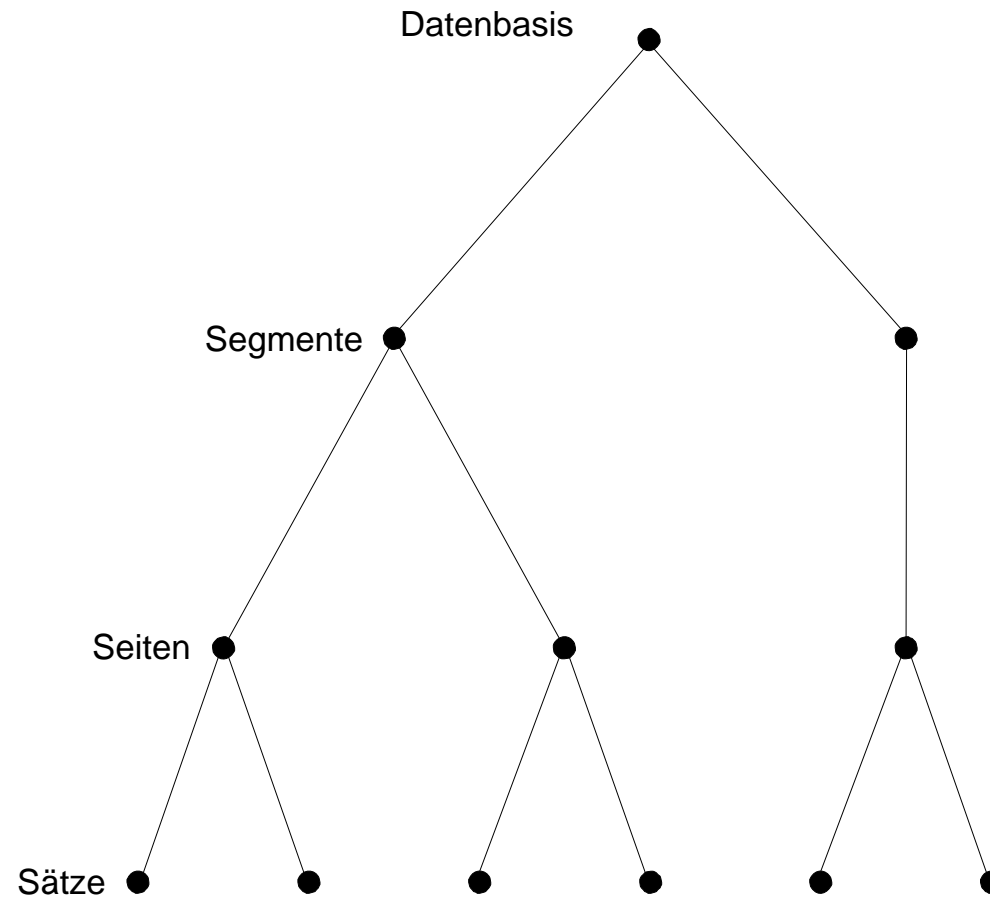


Satz: Deadlock \Leftrightarrow Zyklus im Wartegraph

Zurücksetzen einer Transaktion

- Minimierung des Rücksetzaufwandes:
Wähle jüngste beteiligte Transaktion.
- Maximierung der freigegebenen Ressourcen:
Wähle Transaktion mit den meisten Sperren.
- Vermeidung von Verhungern (engl. Starvation): Wähle nicht diejenige Transaktion, die schon oft zurückgesetzt wurde.
- Mehrfache Zyklen: Wähle Transaktion, die an mehreren Zyklen beteiligt ist.

Hierarchie der Sperrgranulate



Granularität

- Bei zu kleiner Granularität werden Transaktionen mit hohem Datenzugriff stark belastet.
- Bei zu großer Granularität wird der Parallelitätsgrad unnötig eingeschränkt.

Multiple granularity locking (MGL)

NL keine Sperrung (no lock)

S Sperrung durch Leser

X Sperrung durch Schreiber

IS Lesesperre (S) weiter unten beabsichtigt

IX Schreibsperre (X) weiter unten beabsichtigt

	NL	S	X	IS	IX
S	✓	✓	-	✓	-
X	✓	-	-	-	-
IS	✓	✓	-	✓	✓
IX	✓	-	-	✓	✓

(vorhanden)

Protokoll

Idee: vor dem Sperren erst geeignete Sperren in allen übergeordneten Knoten von oben nach unten anfordern.

Bevor ein Knoten mit S oder IS gesperrt wird, müssen alle Vorgänger vom Sperrer im IX - oder IS -Modus gehalten werden.

Bevor ein Knoten mit X oder IX gesperrt wird, müssen alle Vorgänger vom Sperrer im IX -Modus gehalten werden.

Sperren von unten nach oben freigeben.

Datenbasis-Hierarchie mit Sperren

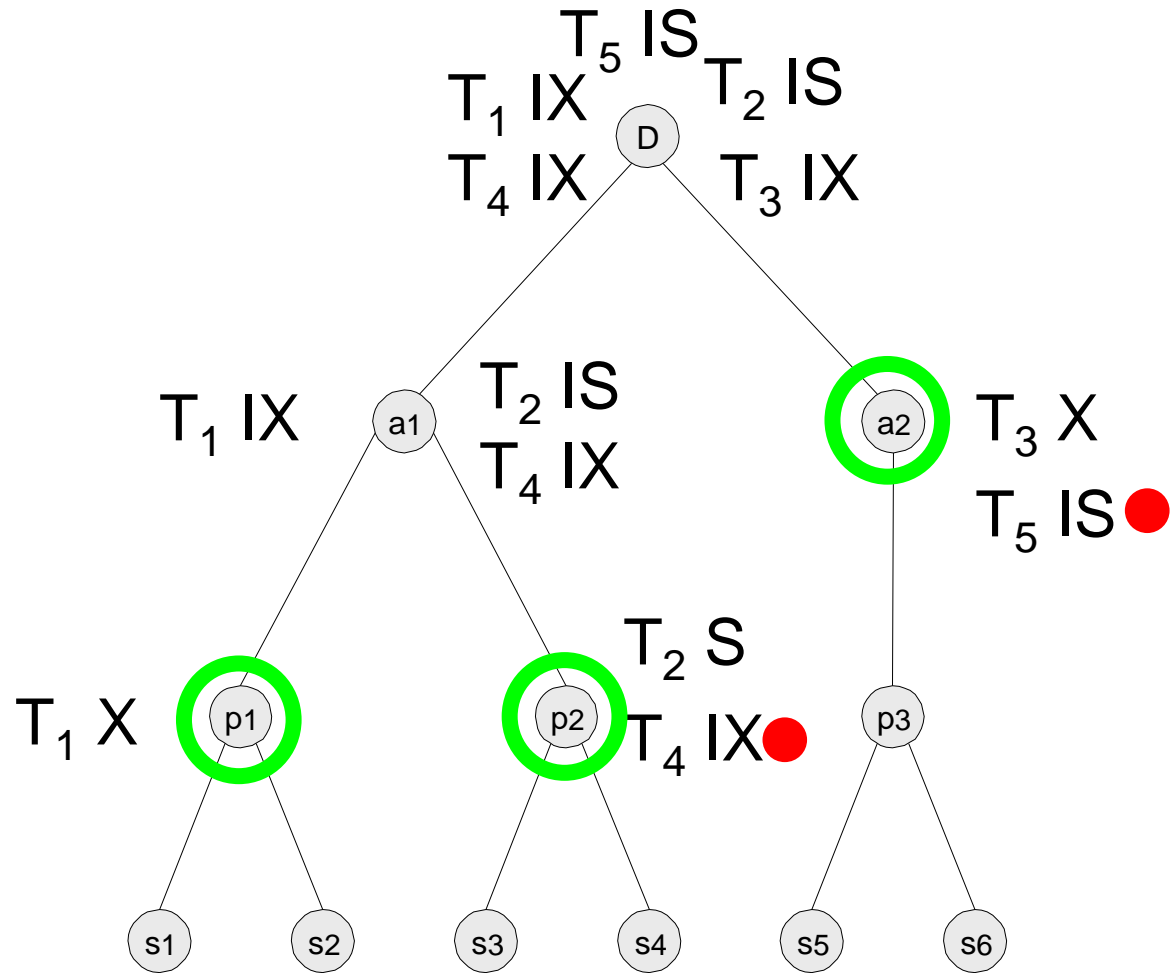
T₁ will X p₁

T₂ will S p₂

T₃ will X a₂

T₄ will X s₃

T₅ will S s₅



Zeitstempelverfahren

Äquivalenz zu serieller Schedule gemäß Eintrittszeit

Jede Transaktion

- erhält Zeitstempel bei Eintritt ins System
- drückt einem Item seinen Zeitstempel auf

Jedes Item hat

- Lesestempel = höchster Zeitstempel durch Leseoperation
- Schreibstempel = höchster Zeitstempel durch Schreiboperation

Regeln

Transaktion mit Zeitstempel t darf kein Item lesen mit Schreibstempel $t_w > t$ (denn der alte Item-Wert ist weg).
Zurücksetzen !

Transaktion mit Zeitstempel t darf kein Item schreiben mit Lesestempel $t_r > t$ (denn der neue Wert kommt zu spät).
Zurücksetzen !

Zwei Transaktionen können dasselbe Item zu beliebigen Zeitpunkten lesen.

OK !

Transaktion mit Zeitstempel t darf kein Item beschreiben mit Schreibstempel $t_w > t$

ignorieren !

Regeln

Transaktion X mit Zeitstempel t bei Zugriff auf Item mit Lesestempel t_r und Schreibstempel t_w :

```
if (X = read) and ( $t \geq t_w$ )
    führe X aus und setze  $t_r := \max\{t_r, t\}$ 
if (X = write) and ( $t \geq t_r$ ) and ( $t \geq t_w$ ) then
    führe X aus und setze  $t_w := t$ 
if (X = write) and ( $t_r \leq t < t_w$ )
    tue nichts
If (X = read and  $t < t_w$ ) or (X = write and  $t < t_r$ )
    setze Transaktion zurück
```


Beispiel für Zeitstempelverfahren

T_1	T_2	a
150	160	$t_r = 0$ $t_w = 0$
read(a)		$t_r = 150$
	read(a)	$t_r = 160$
a:=a-1		
	a:=a-1	
	write(a)	
	ok	da $t_2 = 160 \geq t_r(a) = 160$ und $t_2 = 160 \geq t_w(a) = 0$
write(a)		
zurückgesetzt		da $t_1 = 150 < t_r(a) = 160$

Beispiel für Zeitstempelverfahren

T_1	T_2	T_3	a	b	c
200	150	175	$t_r = 0$	$t_r = 0$	$t_r = 0$
			$t_w = 0$	$t_w = 0$	$t_w = 0$
read(b)				$t_r = 200$	
	read(a)		$t_r = 150$		
		read(c)			$t_r = 175$
write(b)				$t_w = 200$	
write(a)			$t_w = 200$		
	write(c)				
	Abbruch				da $t_2=150 < t_r(c)=175$
		write(a)			
		ignoriert			da $t_3=175 < t_w(a)=200$