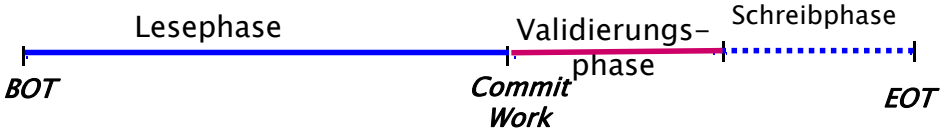
3. Synchronisation: Weitere Verfahren, Leistungsbewertung

- Optimistische Synchronisation
  - BOCC, FOCC
  - Kombination von OCC und Sperrverfahren
- Mehrversionen-Synchronisation
- Prädikatsperren
- Synchronisation bei "High Traffic"-Elementen
- Sperren von B*-Bäumen
- Leistungsbewertung, Lastkontrolle

Optimistische Synchronisation (OCC)

- Grundannahme: geringe Konfliktwahrscheinlichkeit
- Transaktionsverarbeitung in 3 Phasen



- Lesephase: eigentliche Transaktionsverarbeitung
  - Änderungen einer Transaktion erfolgen in einem privaten Puffer
- Validierungsphase: Serialisierbarkeitsprüfung
  - Test auf (RW-, WR-, WW-) Konflikte mit parallelen Transaktionen
  - Konfliktauflösung durch Zurücksetzen von Transaktionen
- Schreibphase
  - nur für Änderungstramsaktionen und nur bei positiver Validierung
  - Schreib-Transaktion schreibt hinreichende Log-Information und propagiert ihre Änderungen

Optimistische Synchronisation (2)

- Allgemeine Eigenschaften von OCC:
  - \+ einfache Transaktionsrücksetzung
  - \+ keine Deadlocks
  - \- mehr Rücksetzungen als bei Sperrverfahren
  - \- Gefahr des „Verhungerns“ (starvation) von Transaktionen

Validierung

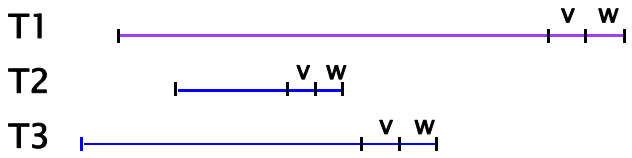
- zur Durchführung der Validierungen werden pro Transaktion der Read-Set (RS) und Write-Set (WS) geführt
- Forderung:
  - eine Transaktion kann nur erfolgreich validieren, wenn sie alle Änderungen von zuvor validierten Transaktionen gesehen hat *(Chronologieerhaltende Serialisierbarkeit)*
  - Validierungsreihenfolge bestimmt Serialisierungsreihenfolge
- generelle Validierungsstrategien:
  - **Backward Oriented (BOCC):** Validierung gegenüber bereits beendeten (Änderungs-) Transaktionen
  - **Forward Oriented (FOCC):** Validierung gegenüber laufenden Transaktionen

BOCC

- ursprünglich vorgeschlagenes Verfahren zur optimistischen Synchronisation [Kung&Robinson, ACM TODS 1981]
- Validierung von Transaktion T:

  BOCC-Test gegenüber allen Änderungstransaktionen Tj, die seit BOT von T erfolgreich validiert haben:

```
IF RS (T) ∩ WS (Tj) ≠ ∅
THEN  ROLLBACK (T)
ELSE  SCHREIBPHASE
```



v = Validierung
w = Schreibphase

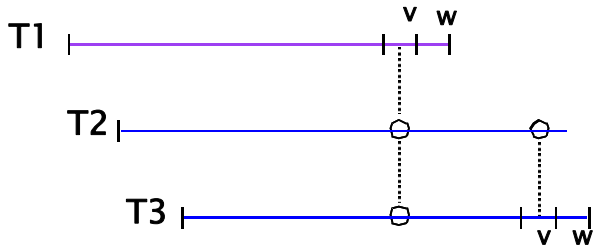
BOCC (2)

- Nachteile/Probleme:
  - unnötige Rücksetzungen wegen ungenauer Konfliktanalyse
  - Aufbewahren der Write-Sets beendeter Transaktionen erforderlich
  - hohe Anzahl von Vergleichen bei Validierung
  - Rücksetzung erst bei EOT => viel unnötige Arbeit
  - es kann nur die validierende Transaktion zurückgesetzt werden => Gefahr des „Verhungerns“
  - hohes Rücksetzrisiko für lange Transaktionen und bei Hot-Spots

FOCC

- nur Update-Transaktionen validieren gegenüber laufenden Transaktionen T_i

  Validierungstest:
  $WS(T) \cap RS(T_i) = \varnothing$



- Vorteile:
  - Wahlmöglichkeit des Opfers (Kill, Rollback, Prioritäten, ...)
  - keine unnötigen Rücksetzungen
  - frühzeitige Rücksetzung möglich => Einsparen unnötiger Arbeit
  - keine Aufbewahrung von Write-Sets nach Transaktionende
- Probleme:
  - Während Validierungs- und Schreibphase ist WS (T) zu 'sperren', damit sich die RS (Ti) nicht ändern (keine Deadlocks damit möglich)
  - immer noch hohe Rücksetzrate möglich

IDBS 2

BOCC+

- Verwendung von Zeitstempeln (Änderungszähler)

- erfolgreich validierte Transaktionen erhalten eindeutige, monoton wachsende *Transaktionsnummer*
- geänderte Objekte erhalten Transaktionsnummer der ändernden Transaktion als *Zeitstempel TS* zugeordnet
- beim Lesen eines Objektes wird Zeitstempel *ts* der gesehenen Version im Read-Set vermerkt

```
VALID := true
<< for all r in RS (T) do;
     if ts (r,T) < TS (r) then VALID := false;
   end;
   if VALID then do;
   TNC := TNC + 1; {ergibt Transaktionsnummer für T}
     for all w in WS (T) do;
         TS (w) := TNC;
       setze laufende Transaktionen mit w in RS zurück;
       end; >>
     Schreibphase für T;
   end;
   else (setze T zurück);
```

- Validierung überprüft durch Zeitstempelvergleich, ob gesehene Objektversionen zum Validierungszeitpunkt noch aktuell sind
  - Verwalten einer Objekttabelle mit TS-Zeitstempel zuletzt geänderter Objekte

IDBS 2

BOCC+ (2)

- zum Scheitern verurteilte Transaktionen können sofort zurückgesetzt werden
- Vorteile BOCC+
  - keine unnötigen Rücksetzungen
  - frühzeitiges Abbrechen zum Scheitern verurteilter Transaktionen
  - schnelle Validierung
- Probleme:
  - wie bei BOCC können einzelne Transaktionen verhungern
  - potentiell hohe Rücksetzrate
- Verbesserungsmöglichkeiten:
  - Kombination mit Sperrverfahren
  - Reduzierung der Konfliktwahrscheinlichkeit, z.B. durch
    - geringere Konsistenzebene (Lesetransaktionen werden bei Validierung nicht mehr berücksichtigt)
    - Mehrversionen-Konzept

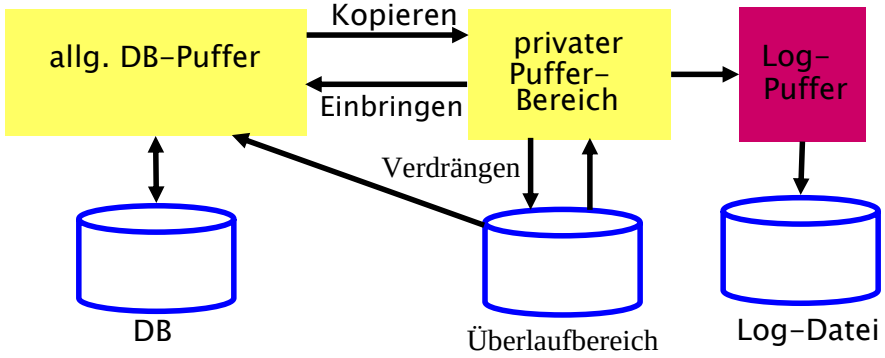
Kombination von OCC + Sperren

- Ziel: Vorteile beider Verfahrensklassen kombinieren
  - geringe Rücksetzhäufigkeit von Sperrverfahren
  - hohe Parallelität (weniger Sperrwartezeiten) von OCC
- Kombination auf Transaktionsebene
  - optimistisch und pessimistisch synchronisierte Transaktionen
  - für lange und bereits gescheiterte Transaktionen pessimistische Synchronisation => kein Verhungern
- Kombination auf Objekt-Ebene
  - optimistisch und pessimistisch synchronisierte Datenobjekte
  - pessimistische Synchronisation für Hot-Spot-Objekte
- Erhöhte Verfahrenskomplexität

Kombination von OCC + Sperren (2)

- auch bei pessimistischer Synchronisation Änderungen in privatem Transaktionspuffer
- erweiterte Validierung: Transaktion scheitert, falls unverträgliche Sperre gesetzt ist
- (teilweise) pessimistisch synchronisierte Transaktionen:
  - bei EOT optimistische Transaktionen zurücksetzen, die auf X-gesperrte Objekte zugegriffen haben
  - Schreibphase mit anschließender Sperrfreigabe

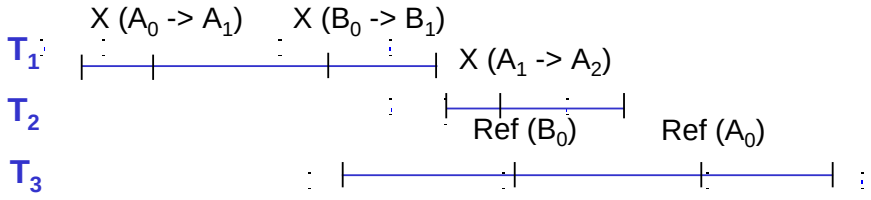
Pufferverwaltung bei OCC



- Änderungen laufender Transaktionen werden nicht in DB verdrängt
  - separater Überlaufbereich auf Externspeicher
  - Transaktionsabbruch erfordert keine Undo-Recovery auf DB
- Einbringen vom Überlaufbereich aufwändig
- Einbringen verlangt i.a. Synchronisation auf Seitenebene

Mehrversionen-Konzept

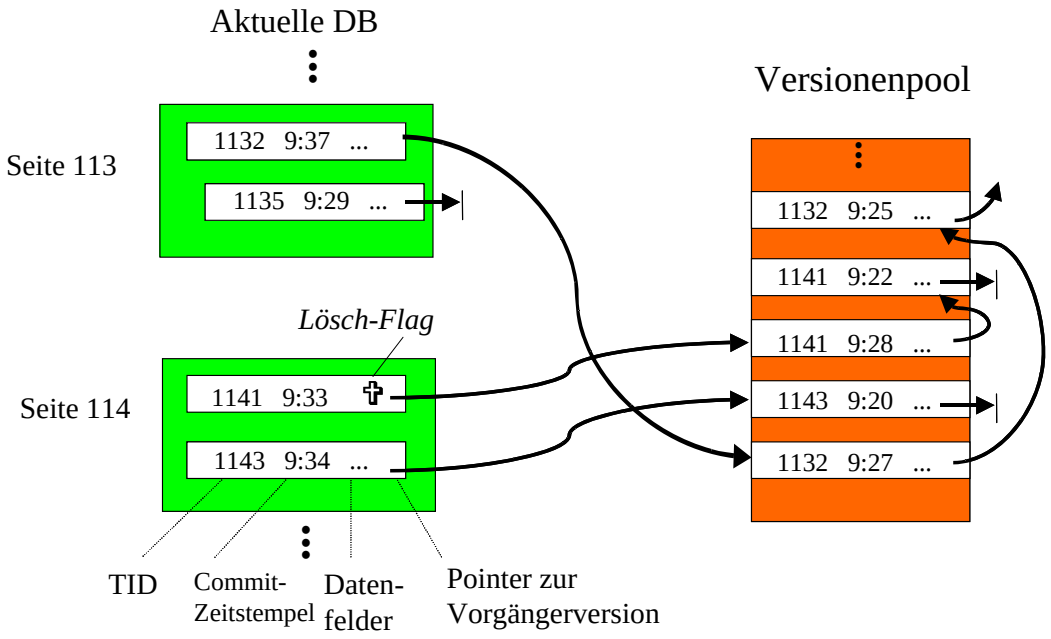
- jede Änderung erzeugt neue Objektversion
- Lesetransaktionen sehen den bei ihrem BOT gültigen DB-Zustand => reihenfolgeerhaltende Serialisierbarkeit
- Keine Synchronisation mehr für Lese-Transaktionen !
  - keine Blockierungen und Rücksetzungen für Leser, dafür ggf. Zugriff auf veraltete Objektversionen
  - Änderungstransaktionen werden untereinander über ein allgemeines Verfahren (Sperren, OCC, ...) synchronisiert
  - erheblich weniger Synchronisationskonflikte
- Nutzung in kommerziellen DBS (z.B. Oracle)



Versionenpool-Verwaltung

- zusätzlicher Speicher- und Wartungsaufwand
  - Versionenpool-Verwaltung
  - Auffinden von Versionen
  - Garbage Collection
- Realisierungsmöglichkeit
  - Lesetransaktionen erhalten BOT-Zeitstempel
  - Änderungstransaktionen kennzeichnen geänderte Objekte mit Commit-Zeitstempel
  - alte Objektversionen kommen in Versionenpool; Verkettung der Versionen eines Objektes
  - Reihenfolge der Objektversionen im Pool entspricht den Zeitstempeln der Nachfolgerversionen
- Lesetransaktionen greifen auf jüngste Objektversionen mit Zeitstempel kleiner dem Transaktionsbeginn zu
- Objektversion V kann freigegeben werden, sobald eine jüngere Version Vj existiert mit Zeitstempel kleiner dem *Beginn der ältesten Lesetransaktion* im System

Versionenpool-Verwaltung: Beispiel

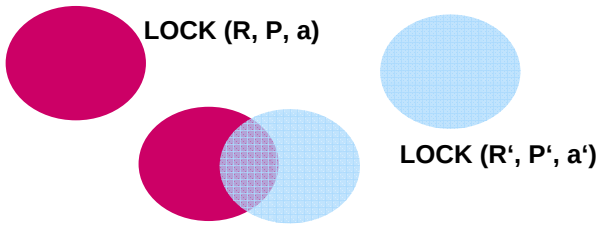


Logische Sperren (Prädikate)

- Verhütung des Phantomproblems ⇒ Prädikatssperren
- Form: LOCK(R, P, a), UNLOCK (R, P)

  R Relationenname, P Prädikat, a ∈ {read, write}
- Wie kann Konflikt zwischen Prädikaten festgestellt werden?

  1. Wenn R ≠ R', kein Konflikt
  2. Wenn a = read und a' = read, kein Konflikt
  3. Wenn P(t) ∧ P'(t) = TRUE für irgendein t, dann besteht ein Konflikt



- i.a. unentscheidbar, selbst mit arithmetischen Prädikaten

  Bsp.: T1: LOCK (PERS, ALTER > 50, read)
  T2: LOCK (PERS, PNR = 4711, write)
- pessimistische Entscheidungen => Einschränkung d. Parallelität
  - Sonderfall: P = TRUE entspricht einer Relationensperre

Prädikatsperren (2)

- effizientere Implementierung durch sog. *Präzisionssperren*
- nur gelesene Daten werden durch Prädikat gesperrt, Schreibsperren werden für Sätze gesetzt
  - kein Disjunktheitstest für Prädikate mehr, sondern lediglich Test, ob Satz ein Prädikat erfüllt
- Datenstrukturen:
  - Prädikatliste: pro Relation werden Lesesperren laufender Transaktionen durch Prädikate beschrieben
  - Update-Liste: geänderte Sätze laufender Transaktionen
- Leseanforderung (Prädikat P):
  - für jeden Satz der Update-Liste ist zu prüfen, ob es P erfüllt
  - wenn ja -> Sperrkonflikt
- Schreibanforderung (Satz T):
  - Schreibsperre wird gewährt, wenn T keines der Leseprädikate erfüllt

Synchronisation von High-Traffic-Objekten

- High-Traffic-Objekte: meist numerische Felder mit aggregierten Informationen
  - z.B. Anzahl freier Plätze, Summe aller Kontostände
- einfachste Lösung der Sperrprobleme: Vermeidung solcher Felder beim DB-Entwurf
- Alternative: Nutzung von semantischem Wissen zur Synchronisation wie Kommutativität von Änderungsoperationen auf solchen Feldern
- Beispiel: Inkrement-/Dekrement-Operation

	R	**X**	**Inc/Dec**
R			
X			
Inc/Dec			

- Problem: Inkrementieren/Dekrementierung erfordert i.a. vorheriges Lesen des Objektes (R-Sperre jedoch inkompatibel mit Inc/Dec-Sperre)

IMS Fast Path-Ansatz

- Spezielle Operationen für High-Traffic-Objecte:

  V E R I F Y #Plätze >= Anforderung

  M O D I F Y #Plätze := #Plätze - Anforderung

- quasi-optimistische Synchronisation:
  - zunächst werden keine Sperren gesetzt
  - Änderungen werden nicht direkt vorgenommen, sondern nur in 'work-to-do-list' vermerkt
  - bei EOT Validierung- und Schreibphase:
    Überprüfung, ob VERIFY-Prädikate noch erfüllt sind (geringe Rücksetzwahrscheinlichkeit) sowie
    Durchführung von Inkrement/Dekrement
- Sperren werden nur für Dauer der EOT-Behandlung gehalten
- weit geringere Konfliktgefahr als mit normalen Schreibsperren

Escrow-Ansatz

- Deklaration von High-Traffic-Attributen als Escrow-Felder
- Benutzung spezieller Operationen auf Escrow-Feldern
  - Reservierung einer bestimmten Wertemenge :
    *IF **ESCROW** **(field=F1, quantity=C1, test=(condition))***
    THEN 'continue with normal processing'
    ELSE 'perform exception handling'
  - Benutzung der reservierten Wertmengen: ***USE** **(field=F1, quantity=C2)***
- optionale Spezifizierung eines *Bereichstests* bei Escrow-Anforderung
- es wird garantiert, dass Prädikat nachträglich nicht mehr invalidiert wird (keine spätere Validierung/Zurücksetzung)
- Gleichzeitiger Objektzugriff durch mehrere Änderer, solange Prädikate erfüllt bleiben
- Nachteil: spezielle Programmierschnittstelle

Escrow-Ansatz (2)

- aktueller Wert eines Escrow-Feldes ist unbekannt, solange Reservierungen bestehen
  - Führen eines Werteintervalls für Escrow-Felder, das alle möglichen Werte nach Abschluss der laufenden Transaktionen umfasst
  - für Wert Q_k des Escrow-Feldes k gilt: $LO_k <= INF_k <= Q_k <= SUP_k <= HI_k$
  - Anpassung von INF, Q, SUP bei Anforderung, Commit und Rollback einer Transaktion
  - Minimal-/Maximalwerte (LO, HI) sind einzuhalten
  - Durchführung von Bereichstests bezüglich des Werteintervalls
- Beispiel: Zugriffe auf Feld mit LO=0, HI=500 (# freier Plätze)

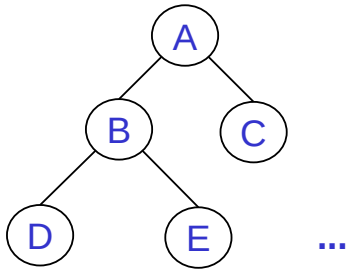
Anforderung/Rückgaben			Werteintervall		
T1	T2	T3	INF	Q	SUP
			15	15	15
-5					
	-8				
		+4			
Commit					
		Commit			
	Rollback				

Escrow-Ansatz (3)

- Escrow-Operationen führen zur Vergrößerung des Intervalls
  - Anforderungen (C1 < 0): INF wird um C1 dekrementiert, falls LO nicht unterschritten wird
  - Rückgabe (C1 > 0): SUP wird um C1 inkrementiert, falls HI nicht überschritten wird
- Commit/Rollback führen zur Reduzierung des Intervalls
  - Dekrementierung von SUP: Commit für Anforderung (C1<0) bzw. Rollback für Rückgabe
  - Inkrementierung von INF: Commit für Rückgabe (C1>0) bzw. Rollback für Anforderung
- nach Intervallanpassung ggf. wartende Escrow-Operationen bedienen

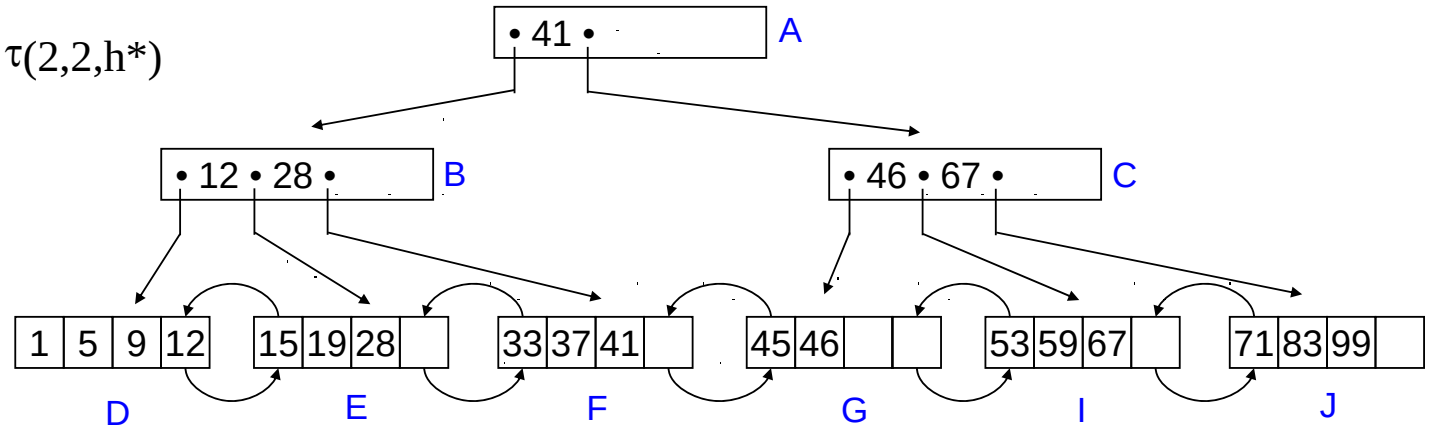
Sperren in B*-Bäumen

- (lange) Seitensperren führen zu inakzeptablen Behinderungen
  - v.a. Wurzelknoten und Zwischenknoten werden zu Engpässen
  - zahlreiche Einträge auch pro Blatt



- Nutzung der Baum-Charakteristika
  - alle Operationen starten bei Wurzel
  - Zwischenknoten im Baum steuern nur die Suche nach Blattknoten
  - nur Inserts/Deletes, die Baumstruktur ändern, erfordern exklusive Sperren des betroffenen Pfades von der Wurzel
- Baumsperrprotokoll (Lock Coupling)
  - Seite (außer der Wurzel) kann erst gesperrt werden, wenn Vorgänger gesperrt ist („lock coupling“)
  - Sperre auf Vorgänger kann freigegeben werden, wenn seine spätere Änderung - z.B. wegen Split-Vorgängen - ausgeschlossen werden kann
  - garantiert Serialisierbarkeit trotz fehlender Zweiphasigkeit

Sperren in B*-Bäumen : Beispiel



1. Suche 37

2. Insert 27

3. Insert 10

Methoden zur Leistungsbewertung von DBS

1.) Kostenformeln
- nur für grobe Abschätzungen brauchbar („back of the envelope“)

2.) Analytische Modelle
- z.B. Warteschlangenmodelle
- oft starke Vereinfachungen erforderlich

3.) Simulationen
- sehr geeignet zum Vergleich verschiedener Realisierungsalternativen (Verfahren können direkt implementiert werden)
- synthetische Lasten vs. Trace-getrieben (reale Lasten)

4.) Benchmarks
- Messungen mit DBS-Prototyp bzw. realem DBS
- Bewertung / Vergleich bestimmter Systeme

Wahrscheinlichkeit von Sperrkonflikten und Deadlocks

- Annahmen
  - m Datenbankobjekte
  - n parallele Transaktionen (Multiprogramming Level)
  - k Objektzugriffe pro Transaktion
  - nur exklusive Sperren
  - Gleichverteilung der DB-Zugriffe
- mittlere Anzahl von Sperren pro Transaktion: $\bar{L} \approx \frac{k}{2}$
- Wahrscheinlichkeit eines Sperrkonfliktes für den i-ten Objektzugriff einer Transaktion T:

$$P_c = \frac{\text{\#Sperren anderer Transaktionen}}{\text{\#DB-Objekte, die nicht von T gesperrt}}$$

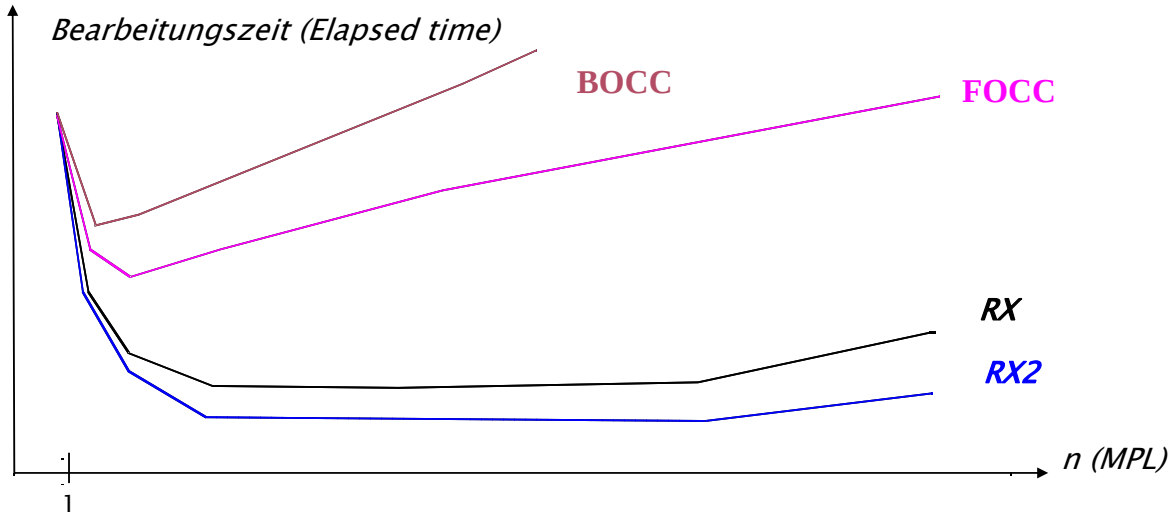
- Wahrscheinlichkeit, dass eine Transaktion einen Sperrkonflikt erfährt:

$P_w \approx k \cdot P_c \approx$

- Deadlock-Wahrscheinlichkeit zwischen zwei Transaktionen T1 und T2:

Pr [T1 -> T2) • Pr [T2 -> T1] • #Kandidaten für T2 =

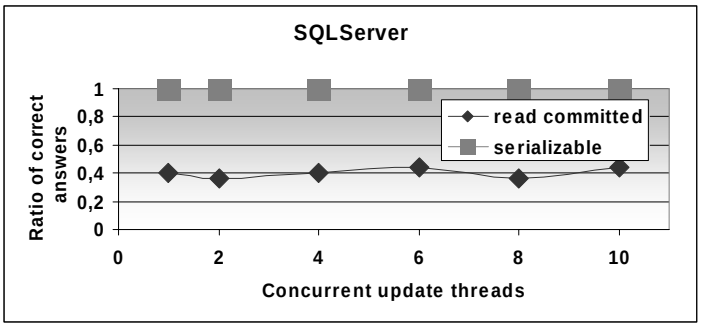
Bewertung von Synchronisationsverfahren mit Trace-basierter Simulation (Peinl, 1987)

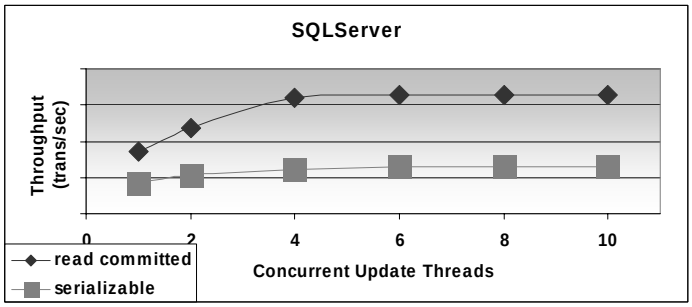


- sehr geringe Parallelität → keine effektive Nutzung der Ressourcen
- geringe Parallelität → bester Durchsatz, nicht notwendigerweise kürzeste Antwortzeiten
- pessimistische Methoden gewinnen: Blockierung vermeidet häufig Deadlocks
- optimistische Methoden geraten leicht in ein Thrashing-Verhalten
- RX2 (kurze Lesesperren) reduziert effektiv Konfliktrate

Benchmark-Ergebnisse (Shasha, 2002)

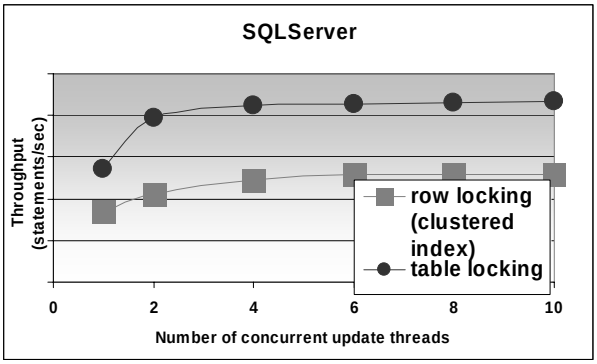
- Serializable vs. Read Committed (RX2)

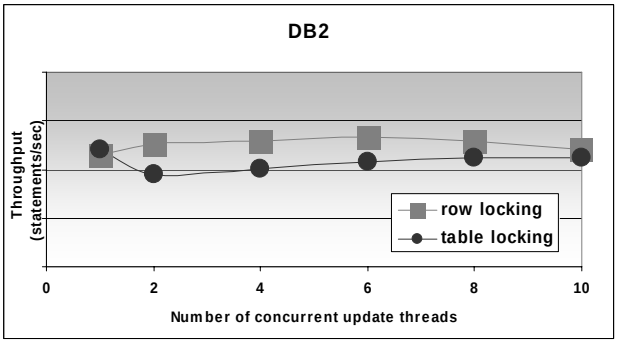


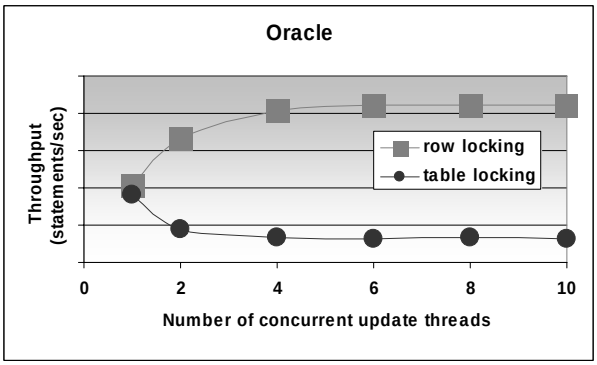


Benchmark-Ergebnisse (2)

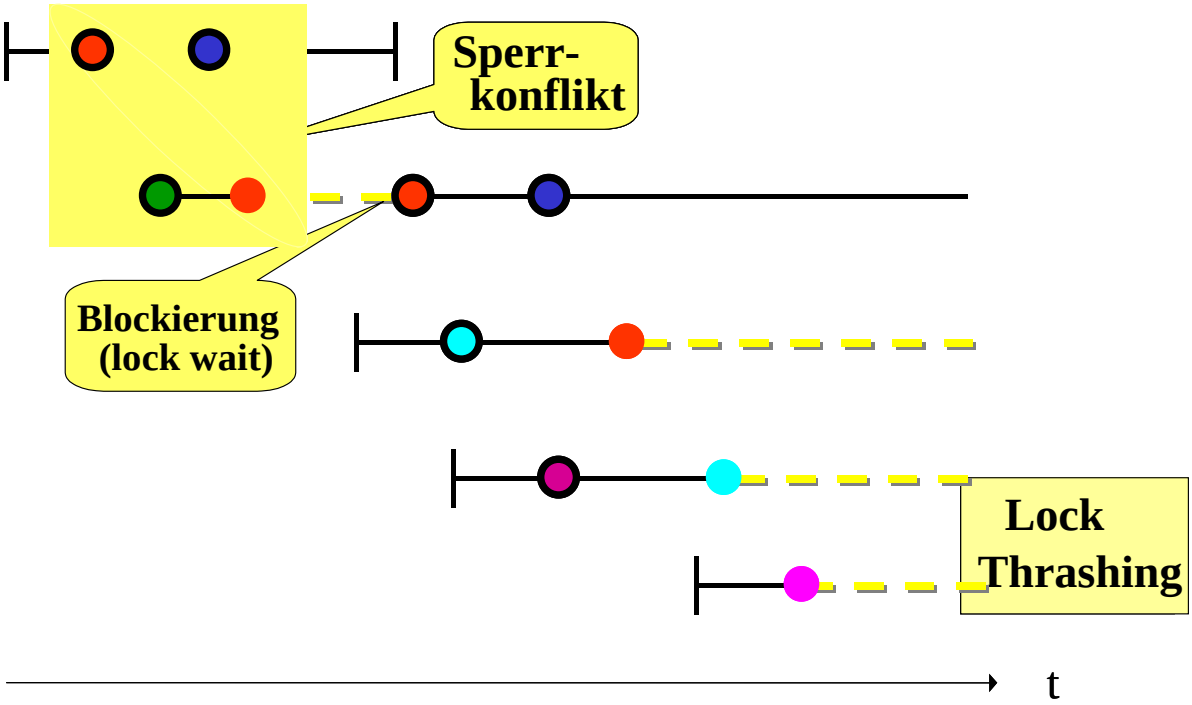
- Table vs. row (record) locking





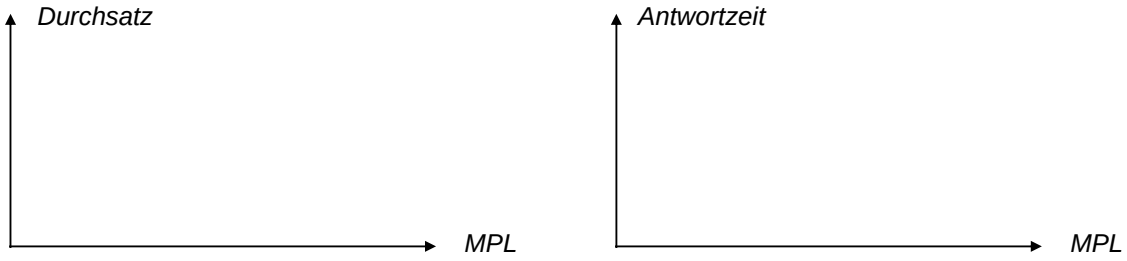


Lock Thrashing



Dynamische MPL-Kontrolle

- Parallelitätsgrad (MPL) bestimmt wesentlich Leistungsverhalten, insbesondere Umfang an Konflikten bzw. Rücksetzungen
  - Gefahr von "lock thrashing" bei Überschreiten eines kritischen MPL-Wertes



- statische MPL-Einstellung problematisch: wechselnde Lastsituationen, mehrere Transaktionstypen
- Idee: dynamische MPL-Einstellung zur Vermeidung von "lock thrashing"

Dynamische MPL-Kontrolle (2)

- Nutzung einer aktuellen, systemweiten Konfliktrate:

Konfliktrate = *# gehaltener Sperren aller Transaktionen / #Sperren nicht-blockierter Transaktionen*

- kritischer Wert: ca. 1,3 (experimentell bestimmt)
- Zulassung neuer Transaktionen nur, wenn kritische Wert noch nicht erreicht ist
- bei Überschreiten erfolgt Abbrechen von Transaktionen

Zusammenfassung

- Sperrverfahren am universellsten einsetzbar
  - reine OCC- und Zeitstempelverfahren erzeugen i.a. zu viele Rücksetzungen
  - Hierarchische Sperrverfahren erlauben Begrenzung des Verwaltungs-Overheads
- generelle Optimierungen:
  - reduzierte Konsistenzebene
  - Mehrversionen-Ansatz
- „harte“ Synchronisationsprobleme durch Hot Spots und lange (Änderungs-) Transaktionen
  - möglichst Vermeidungsstrategie anwenden
  - Spezialprotokolle nutzen semantisches Wissen über Operationen/Objekte zur Reduzierung von Synchronisationskonfliken
  - allerdings: ggf. Erweiterung der Programmierschnittstelle, begrenzte Einsetzbarkeit, Zusatzaufwand
- dynamische MPL-Kontrolle kann Lock Thrashing vermeiden