Datenbanken und SQL

Kapitel 8

Concurreny und Recovery

Concurrency und Recovery

- Transaktionen
- Recovery
  - Einführung in die Recovery
  - Logdateien
  - Checkpoints
- Conncurrency
- Sperrmechanismen
- Deadlocks
- SQL-Norm und Concurrency
- Concurrency in Oracle, SQL Server und MySQL

Transaktionen in Datenbanken

- Garantie an den Anwender:
  - Eine Transaktion wird im Fehlerfall komplett zurückgesetzt, wenn sie nicht mehr beendet werden kann
  - Alle Daten einer abgeschlossenen Transaktion sind persistent
  - Eine Transaktion läuft unabhängig von anderen parallel laufenden Transaktionen
  - Die Datenbank ist immer in sich schlüssig und korrekt
- Dies heißt:
  - Die Garantie gilt lebenslang und unter allen Umständen!
  - Die Garantie gilt auch bei Blitzeinschlag oder Attentaten!
  - Die Garantie gilt auch bei tausenden parallelen Anwendungen!

Atomarität

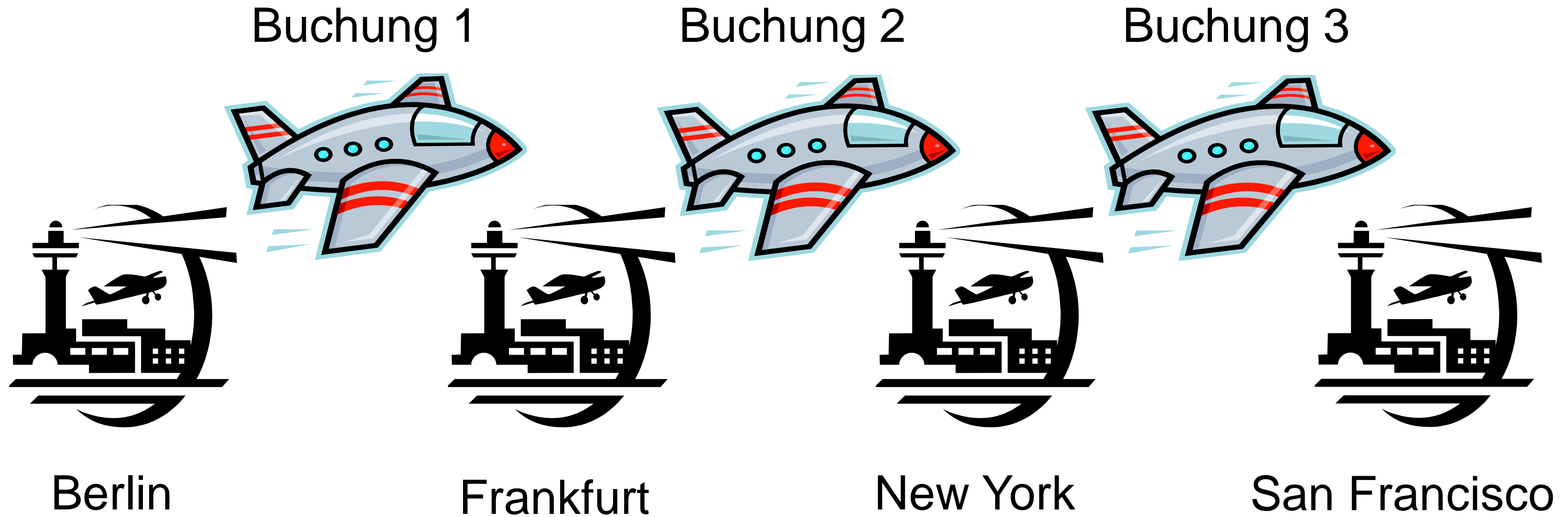
Dauerhaftigkeit

Isolation

Konsistenz

Beispiel: Flug Berlin – San Francisko

- Flug via Frankfurt und New York
- Drei Buchungen ergeben eine Transaktion



Flugbuchung

```
$conn->query( UPDATE Flugbuchung  SET ...  ) ;  // BER - FRA
$conn->query( UPDATE Flugbuchung  SET ...  ) ;  // FRA - NYC
$conn->query( UPDATE Flugbuchung  SET ...  ) ;  // NYC - SFO
$conn->commit() ;
```

Zusammen:
Eine Transaktion

- Konsistenzmodell ACID:
  - Buchung erfolgt nur komplett oder überhaupt nicht
  - Dies gilt auch bei Rechnerabsturz!

ACID:
- Atomarität
- Konsistenz
- Isolation
- Dauerhaftigkeit

Recovery

- Recovery
  - Wiederherstellung von Daten bei schweren Fehlern
- HW-Fehler:
  - Stromausfall, Wackelkontakt, Festplattenausfall, Arbeitsspeicherausfall, Brand (Feuer, Löschwasser)
- SW-Fehler:
  - Fehler in Datenbank-SW, Betriebssystem-SW, Netz-SW, Anwendungsprogramm

Auch: Anschlag, Terrorismus, Flugzeugabsturz

In der Verantwortung des Datenbankprogrammierers!

Vorsorge

- Hardware-Einkauf
  - Nur Geräte, die lange Standzeiten garantieren
  - Keine Geräte von der Stange
- Software-Einkauf
  - SW-Komponenten aufeinander abstimmen
  - Nur zuverlässige und zertifizierte Software einsetzen
- Sicherung
  - Tägliche Sicherung der Daten (Differenzsicherung)
  - Mindestens einmal pro Woche: Komplettsicherung
  - Sichere Aufbewahrung der Sicherungen

In der Regel: Nur zertifizierte Rechner

Updates vorher testen!

Oft viele GB pro Tag …

… und viele TB pro Woche

Sicherung des Datenbestands

Zeitlicher Rahmen	Sicherung
wöchentlich	Komplettsicherung des Datenbestandes
täglich	Differenzsicherung
im laufenden Betrieb	Protokollierung jeder Änderung in einer Logdatei

- Aufwand:
  - Nächtliche Sicherungen
    - Stören den Regelbetrieb nur geringfügig
  - Protokollierung
    - Erhebliche Störung des Regelbetriebs wegen der Erstellung des Protokolls und der Speicherung in Datei (Logdatei)

Performance-Problem!

Wie gegensteuern?

Sicherer Datenbankbetrieb

Medien sind direkt zugreifbar

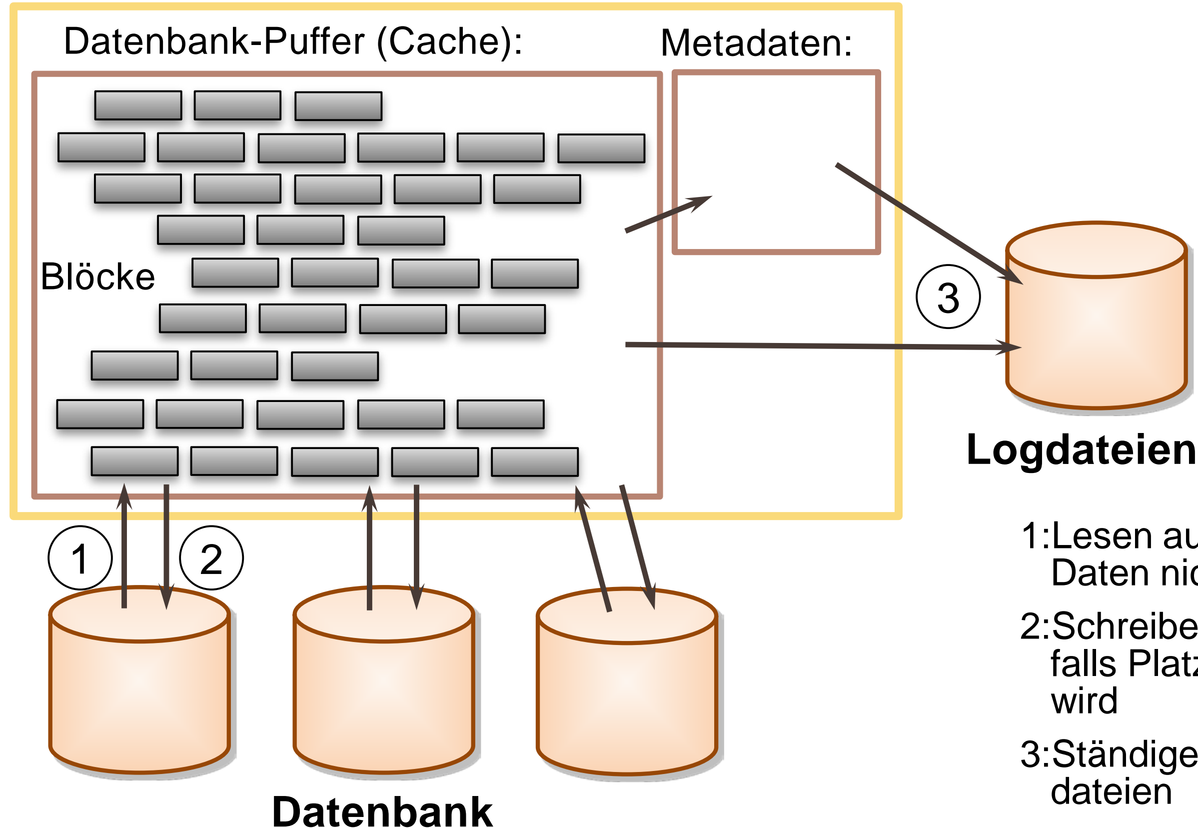
- Voraussetzungen
  - Datenbank befindet sich auf externen nichtflüchtigen Medien
  - Logdaten befinden sich auf externen nichtflüchtigen Medien
  - Datenbankdaten werden im Arbeitsspeicher zwischengespeichert (Performance)
    - Bezeichnung: Datenbank-Puffer, Datenbank-Cache
  - Daten werden vom Datenbank-Puffer gelesen
  - Sind Datum nicht im Puffer, so werden sie von der Datenbank geholt
  - Das Schreiben der Daten geschieht im Datenbank-Puffer
  - Aktualisierung der Datenbank erfolgt asynchron

Reduziert Anzahl der I/Os

Metadaten

- Metadaten sind Daten, die Informationen zu und Zustände über eine Datenbank merken
- Metadaten sind u.a.
  - Informationen zu laufenden Transaktionen
  - Zustände zu Synchronisationsmechanismen
  - Aktualität und Gültigkeit der Daten im Datenbankpuffer
  - Zustand der aktuellen Logdaten
- Aktuelle Metadaten werden vor allem im Arbeitsspeicher gehalten

Datenbankpuffer



1: Lesen aus der Datenbank, falls Daten nicht im Cache

2: Schreiben in die Datenbank, falls Platz im Cache benötigt wird

3: Ständiges Sichern in die Log-dateien

Datenbankpuffer - Erläuterung

- Datenbankpuffer
  - Nimmt einen großen Teil des Arbeitsspeichers in Anspruch
  - In großen Datenbanken auch mehr als 1 TB
  - Bereits gelesene Daten müssen nicht nochmals gelesen werden!
- Daten werden bei Bedarf von der Festplatte geholt und im Datenbankpuffer gehalten
- Daten werden ausschließlich im Datenbankpuffer bearbeitet
- Geänderte Daten werden bei Engpässen im Puffer in die Datenbank zurückgeschrieben
- Parallel werden Änderungen sofort in die Logdateien geschrieben und damit persistent gesichert

Die Idee der Pufferung!

Also: keine ständigen I/Os

Damit müssen geänderte Daten nicht sofort in die Datenbank geschrieben werden

Before- und After-Image

- Jedes Ändern, Löschen oder Einfügen führt dazu, dass Daten manipuliert werden. Wir unterscheiden:

- Before-Image:
  - Die zu ändernden Daten
- After-Image:
  - Die geänderten Daten

Das Abbild, bevor geändert wird

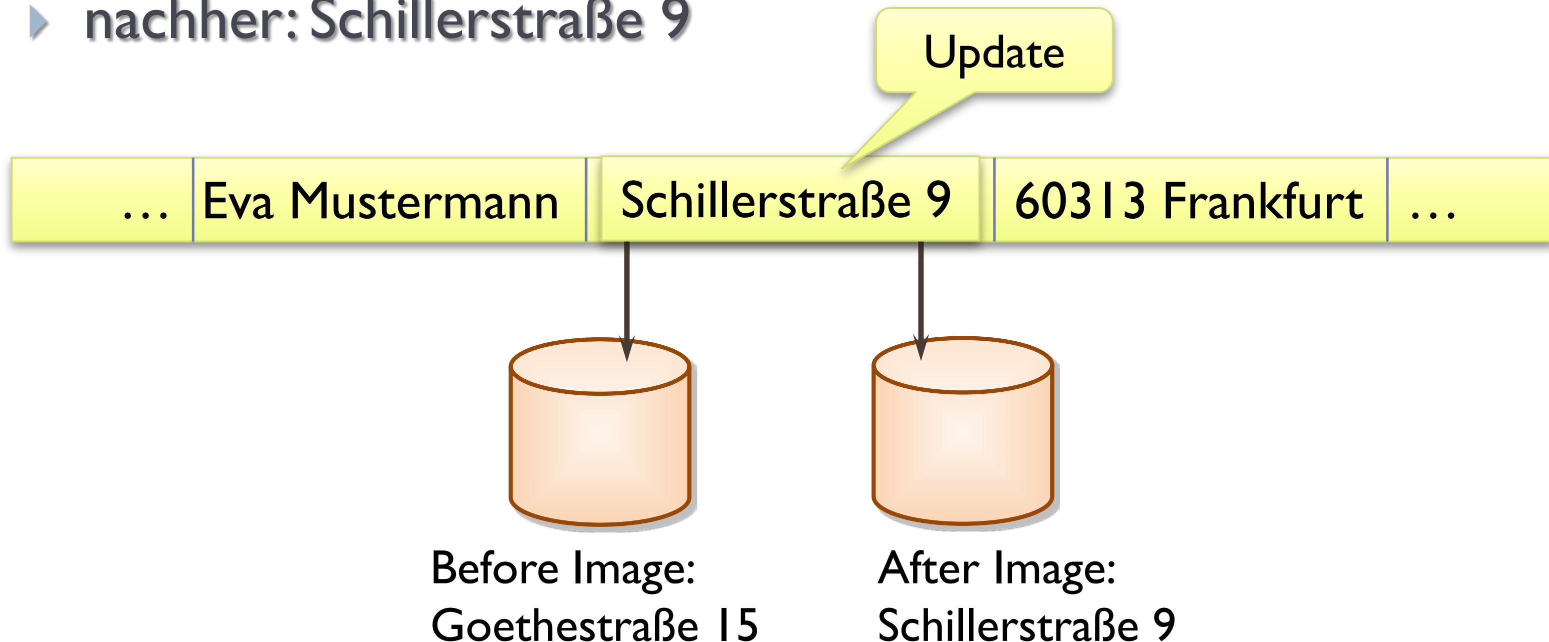
Das Abbild, nachdem geändert wurde

- Ein Before-Image wird bis Transaktionsende gespeichert
- Ein After-Image wird bis zur nächsten Sicherung gespeichert

Warum wohl?

Before- und Afterimage am Beispiel

- Wir wollen die Adresse eines Mitarbeiters ändern
  - vorher: Goethestraße 15
  - nachher: Schillerstraße 9



Einfacher Transaktionsbetrieb

Lesen der Daten	Die Daten werden von der Datenbank eingelesen, falls sie sich nicht bereits im Puffer des Arbeitsspeichers befinden.
Merken der bisherigen Daten	Die zu ändernden Daten werden in die Logdatei geschrieben (Before Image).
Ändern der Daten	Ändern (Update, Delete, Insert) der Daten im Arbeitsspeicher, Sperren dieser Einträge für andere Benutzer.
Merken der geänderten Daten	Die geänderten Daten werden in die Logdatei geschrieben (After Image).
. . .	Obige vier Schritte können sich innerhalb einer Transaktion mehrmals wiederholen.
Transaktionsende mit COMMIT	Transaktionsende in der Logdatei vermerken. Sperren freigeben. Schreiben aller Änderungen in die Datenbank.
Transaktionsende mit Rollback	Rücksetzen der Metadaten der Transaktion. Geänderte Daten im Arbeitsspeicher mittels Before-Images restaurieren. Sperren freigeben.

Schwächen des einfachen TA-Betriebs

- Daten von sehr lange laufenden Transaktionen werden im Arbeitsspeicher gehalten
- Absturz während des Schreibens der geänderten Daten am Transaktionsende:
  - Komplexe Recovery: Es muss überprüft werden, welche Daten schon geschrieben wurden und welche nicht
- Das Schreiben immer zu Transaktionsende kann zu punktuellen Überlasten führen
- Das Schreiben immer zu Transaktionsende ist inflexibel

Transaktionsende in der Praxis

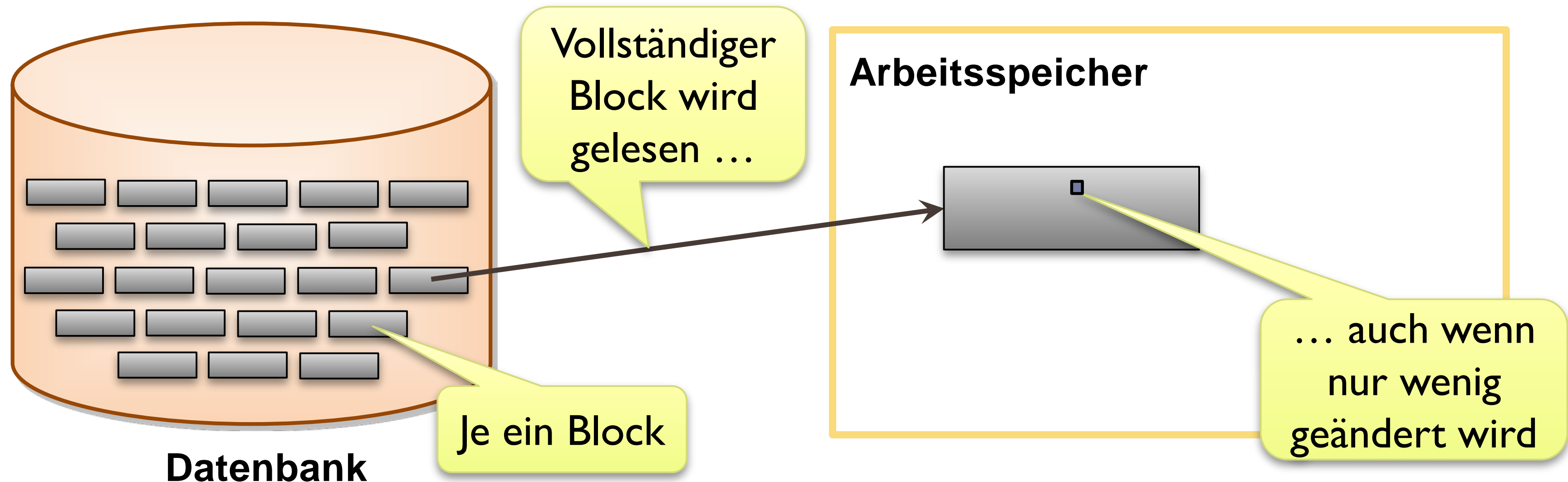
Erste vier Schritte	Wie bisher, auch wiederholt
Transaktionsende mit COMMIT	Transaktionsende in der Logdatei vermerken. Sperren freigeben.
Transaktionsende mit ROLLBACK	Rücksetzen der Metadaten der Transaktion. Geänderte Daten im Arbeitsspeicher mittels der Before-Images restaurieren. Alle geänderten Daten, die bereits in die Datenbank geschrieben wurden, werden für ungültig erklärt. Sperren freigeben.
Änderungen speichern	Die geänderten Daten werden asynchron (unabhängig vom Transaktionsbetrieb) in die Datenbank geschrieben.

Transaktionsbetrieb mit Pufferung

- Hochperformant
  - Der I/O-Verkehr wird minimiert
  - Werden Daten mehrmals geändert, so müssen diese nicht jedes Mal geschrieben werden
  - Werden gelesene Daten wieder gelesen, so stehen diese bereits zur Verfügung
- Die Konsistenz der Daten hängt wesentlich von den Logdateien ab
  - Dauerhaftigkeit nur dank Logdateien gesichert

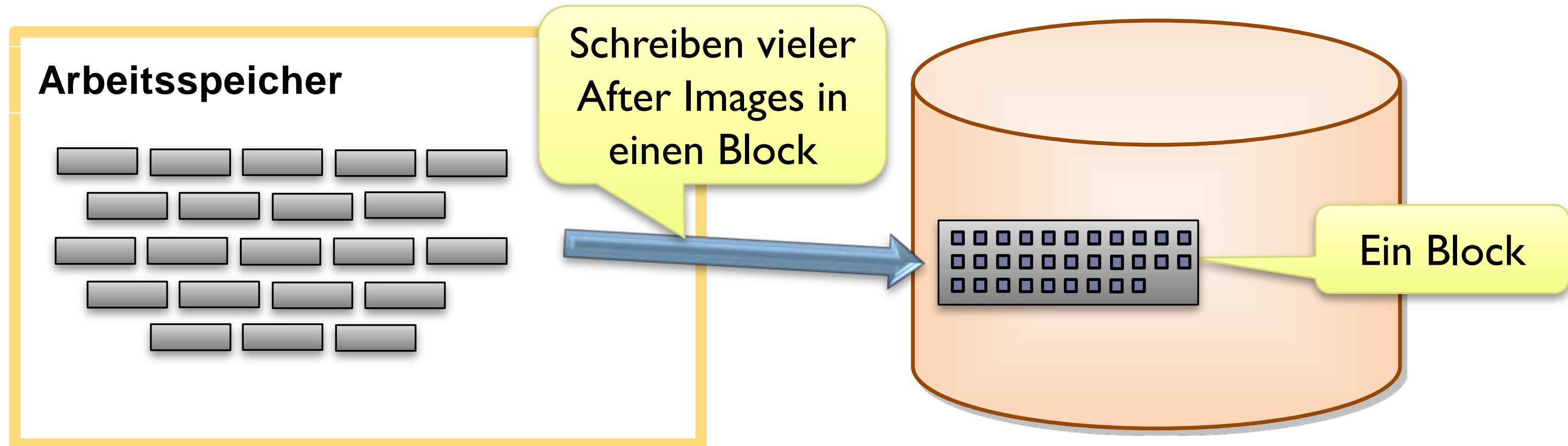
Wiederholung: Blockstruktur

- Alle Festplatten besitzen Blockstruktur
  - Formatiert in 2kB, 4kB, 8kB, 16kB, 32kB Blöcke
- Datenbank übernimmt die Struktur
  - Kann Blöcke noch zusammenfassen: bis zu 64kB Blöcke



Struktur der Before- und After-Images

- Logdateien besitzen ebenfalls Blockstruktur
- Aber: Before- und Afterimages enthalten nur die Änderungen!
- Viele Änderungen werden in einem Block zusammengefasst
- Daher: Geringer Schreibverkehr; zusätzlich: gestreamt



Inhalt der Logdateien

- Die Logdateien enthalten
  - alle Before-Images mit Transaktionsnummer und Zeitstempel
  - alle After-Images mit Transaktionsnummer und Zeitstempel
  - dazugehörige Metadaten
    - Transaktionsnummer,
    - Transaktionsende
    - gehaltene Sperren
    - weitere Infos
- Alle Logdaten müssen aber nicht gleich lange aufbewahrt werden!

Undo-Log und Redo-Log

- Undo-Log:
  - Eine Logdatei, die alle Before-Images und dazugehörige Metadaten enthält
  - Ein Undo-Log-Eintrag muss nur bis Transaktionsende aufgehoben werden
- Redo-Log:
  - Eine Logdatei, die alle After-Images und dazugehörige Metadaten enthält
  - Ein Redo-Log muss bis zur nächste Sicherung aufgehoben werden

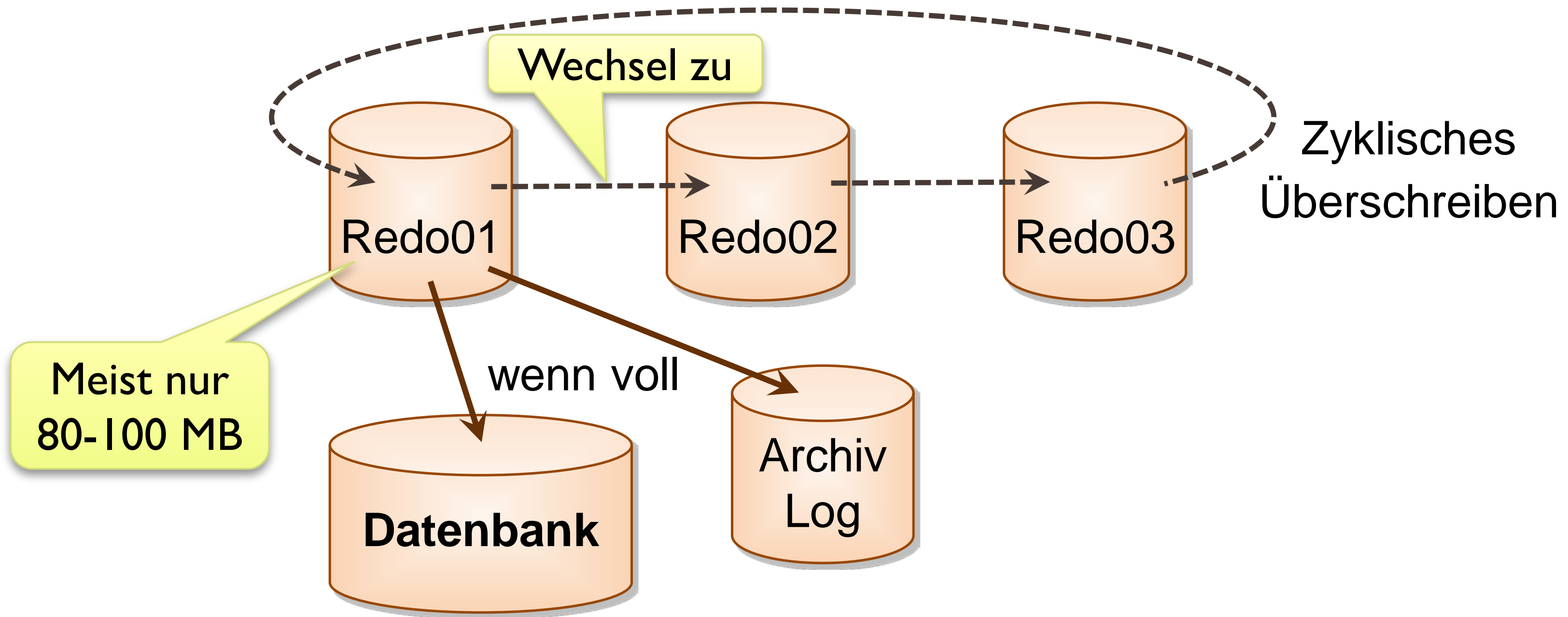
Undo-Log

- Undo-Log wird in speziellem Datenbankbereich gehalten
  - Oracle: Tablespace UNDOTBS1
- Undo-Log wird für Rollback benötigt
- Von einem Undo-Log wird auch gelesen
- Undo-Log wird in der Regel zyklisch überschrieben
- Undo-Log-Einträge müssen auf Festplatte stehen, bevor Daten nicht abgeschlossener Transaktionen in die Datenbank geschrieben werden!

Redo-Log

- Redo-Log ist „Lebensversicherung“ des aktuellen Datenbestands
- Redo-Logs werden ausschließlich sequentiell beschrieben
- Größe eines Redo-Logs:
  - in MySQL: max. 512 GB (ab V5.6)
  - in Oracle: zusätzlich Archive Log
  - in SQL Server: Log Backup
- Redo-Logs werden auf eigenem externen Medium angelegt
- Redo-Logs werden häufig gespiegelt (Raid 1) (Sicherheit)

Redo-Logs in Oracle

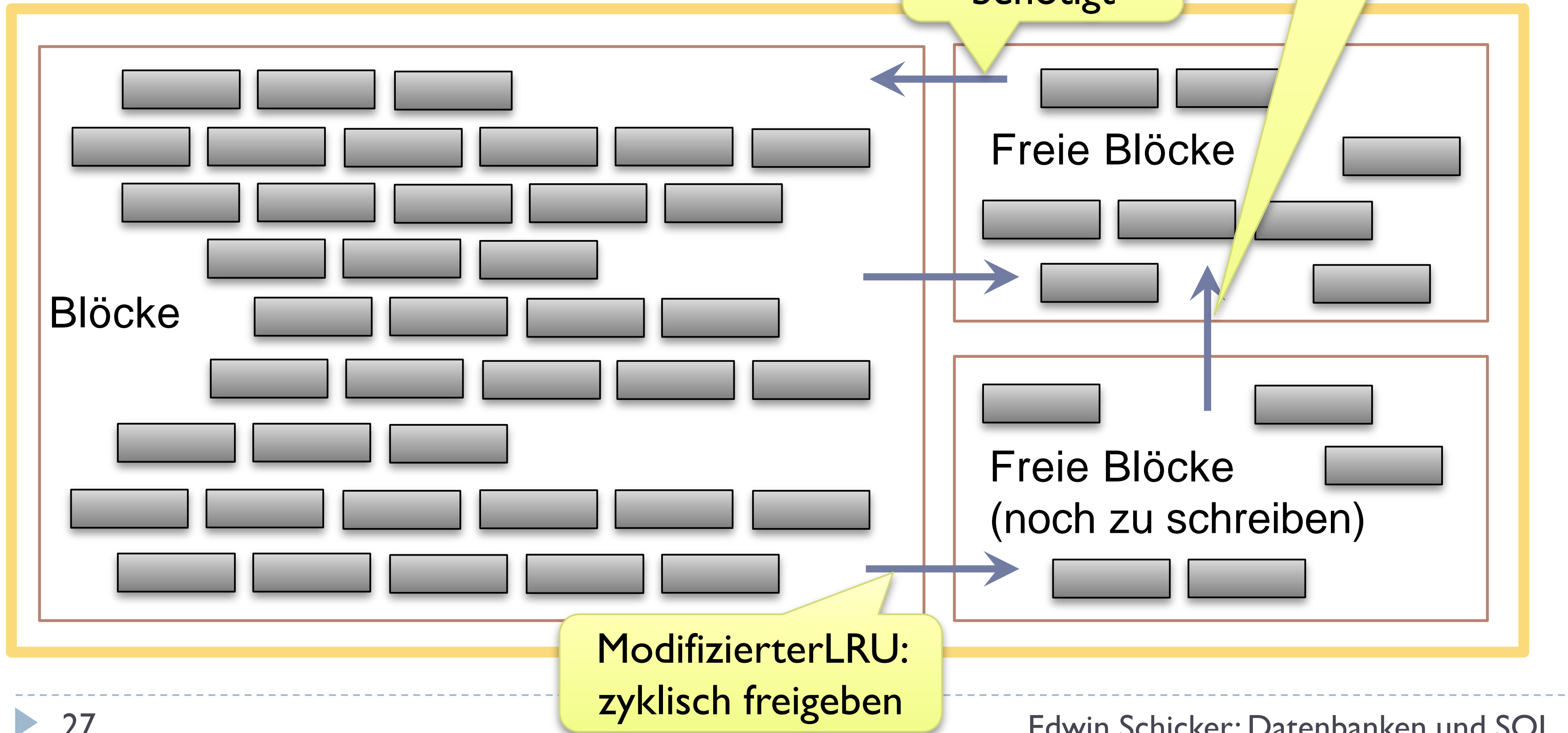


- Es existieren mindestens 2 Redo-Logs und 1 Archiv-Log
- Archiv-Log kann „billigeres" Medium sein

LRU Algorithmus

- Wenn Datenbankpuffer voll, müssen Seiten verdrängt werden → LRU Algorithmus
- LRU (Least Recently Used) Algorithmus:
  - Ausgewählt wird die Seite, die am längsten nicht mehr verwendet wurde
- Gute Erfahrung, da
  - ältere Seiten häufig nicht mehr benötigt
  - jüngere Seiten eventuell nochmals verwendet werden

Datenbankpuffer-Verwaltung



Hot Spots

- Hot Spots sind Daten,
  - auf die immer wieder zugegriffen wird
  - die deshalb nie verdrängt werden

- Sehr gute Performance
  - da weniger I/Os

- Alter Datenbestand auf Festplatte
  - Aufwändige Recovery
    - Alle Änderungen zu den Hot Spots sind nachzuvollziehen
  - Viele Metadaten
    - Kein zyklisches Überschreiben der Metadaten möglich

Zyklisches Überschreiben erlaubt einfache Handhabung

Checkpoints

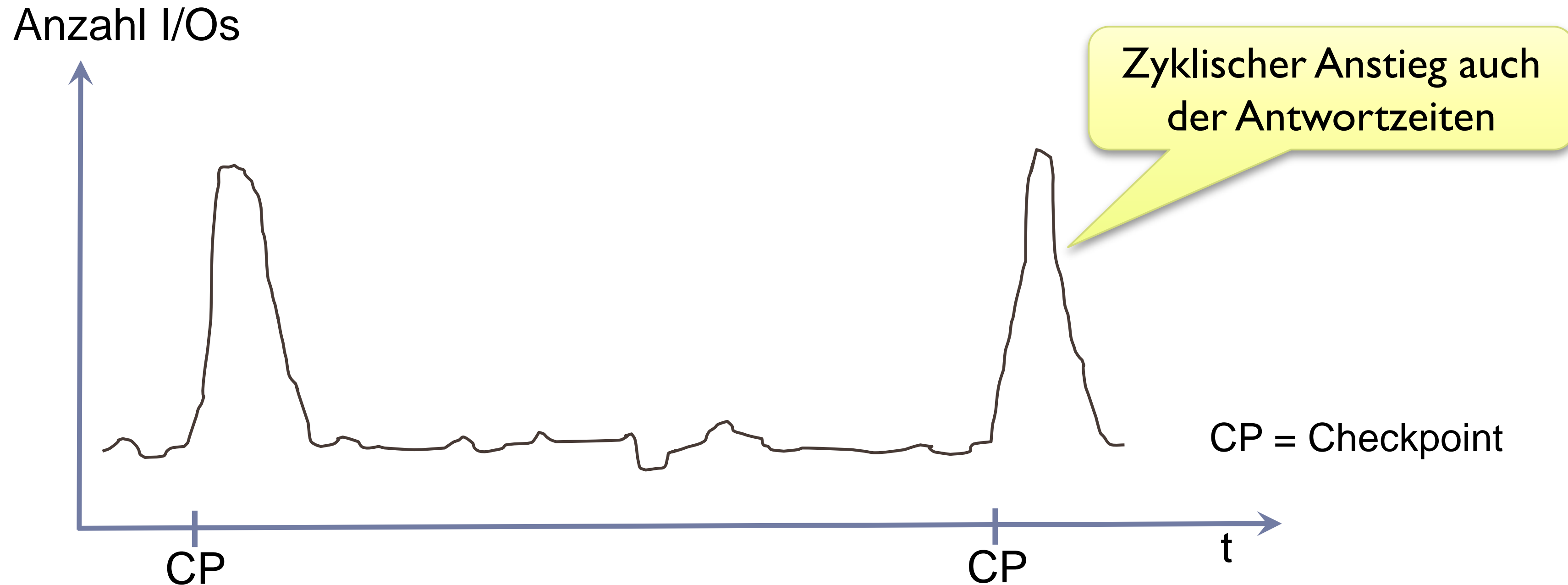
- Checkpoints sind Zeitpunkte, wo alle geänderten Daten zwangsweise in die Datenbank geschrieben werden

  - Nachteil:
    - Punktuell sehr viele I/Os
    - Dadurch auch Behinderung laufender Transaktionen

  - Vorteil:
    - Im Recoveryfall sind nur Redo-Daten seit dem letzten Checkpoints nachzuvollziehen
    - Viele Metadaten können gelöscht werden

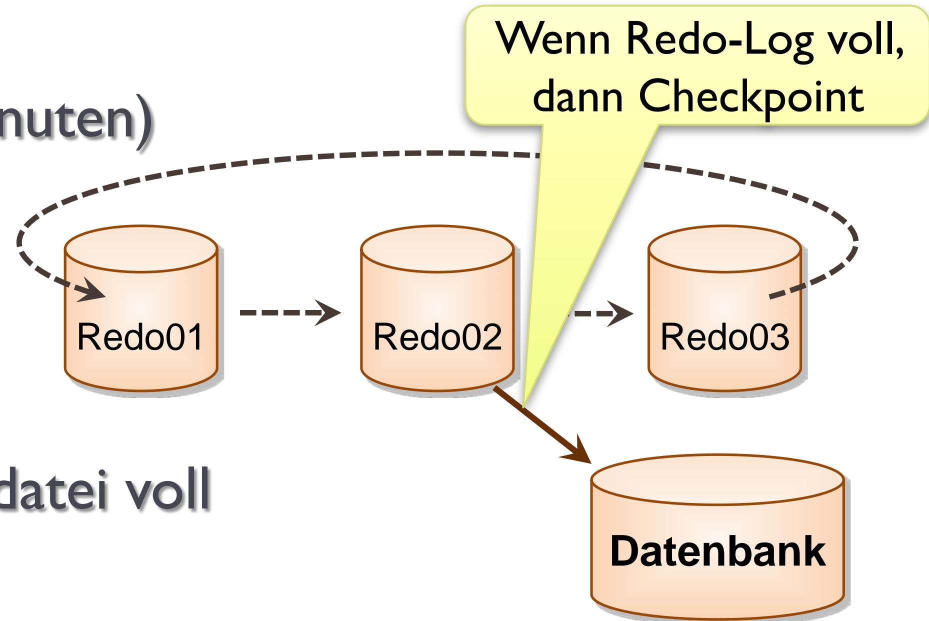
Nachteil von Checkpoints

- Die hohe I/O-Last behindert alle Transaktionen
- Bei jedem Checkpoint steigen die Antwortzeiten



Häufigkeit von Checkpoints

- Zwei Möglichkeiten
  - Zeitgesteuert (z.B. alle 15 Minuten)
  - Ereignisgesteuert
- In Oracle:
  - Checkpoint, wenn Redo-Logdatei voll
- Optimale Einstellung:
  - Parameter sind: Zeitintervall bzw. Größe des Redo-Logs
  - Werte hängen von Erfahrung ab



Transaktionen beim Crash

Bei Recovery außen vor

T1

90-99% aller TAs

T2

Redo

T3

Undo

T4

Letzte Sicherung

Redo

T5

t1
Checkpoint

Undo

t2
Softcrash

t

Sicherheit geht über alles!

- Normalfall:
  - Datenbank, Redo-Logs, Undo-Log, Checkpoints
  - Problem: Während einer Recovery darf keine weitere Komponente ausfallen
- Mehr Sicherheit:
  - Zusätzlich: Redo-Logs werden gespiegelt (z.B. Raid 1)
- Noch mehr Sicherheit:
  - Zusätzlich: Datenbank wird gespiegelt (räumlich getrennt)
- Extreme Sicherheit:
  - Zwei komplett autarke Rechenzentren im Parallelbetrieb

Concurrency

- Concurrency beschäftigt sich mit dem Parallelbetrieb in Datenbanken
- Grundregel der Concurrency:

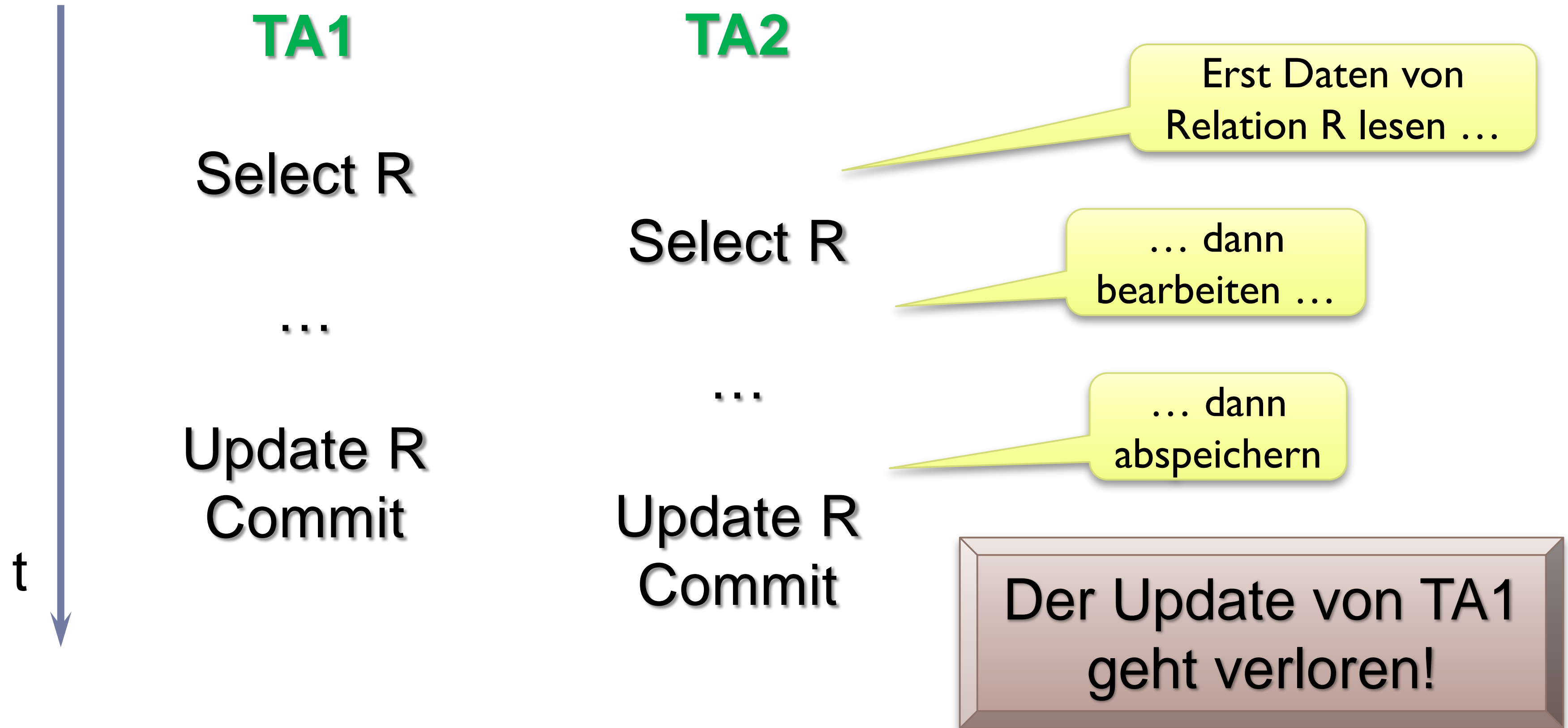
Jede Transaktion läuft so ab, als sei sie allein im System

- Insbesondere muss eine Transaktion Ergebnisse liefern, die unabhängig von anderen Transaktionen sind

Drei Concurrency Probleme

- Problem der verlorengegangenen Änderung
  - Zwei Transaktionen ändern (fast) gleichzeitig. Eine Änderung geht verloren
- Problem der Abhängigkeit von nicht abgeschlossenen Transaktionen
  - Daten werden gelesen, die mittels Rollback rückgesetzt werden
- Problem der Inkonsistenz der Daten
  - Fehlerhafte Daten werden gelesen, wenn andere Transaktionen gleichzeitig ändern

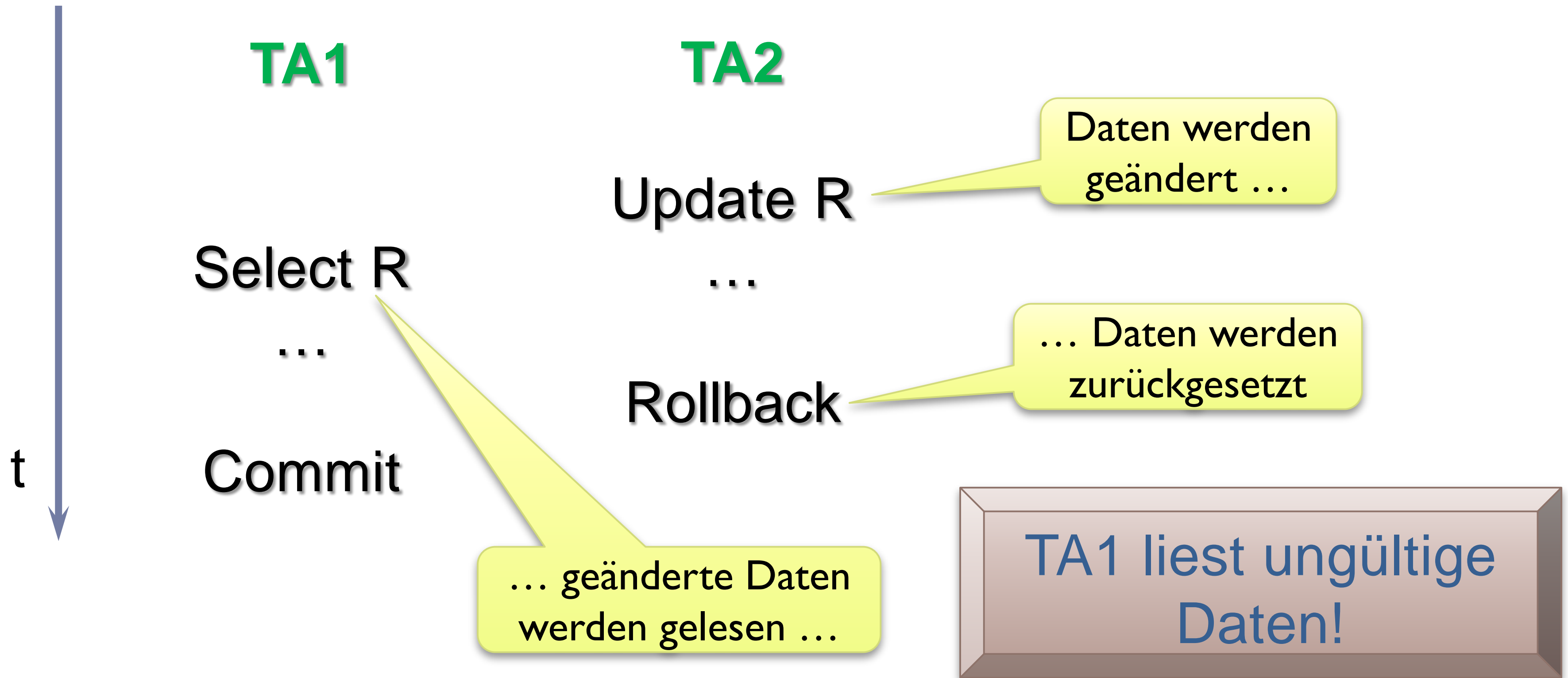
Verlorengegangene Änderung



Verlorengegangene Änderung

- Grundsätzlich nicht erlaubt
- Großes Problem in verteilten Systemen
- Beispiel:
  - Flugbuchung:
    - Die beiden letzten Plätze werden reserviert
    - Die Tickets werden ausgedruckt
    - Parallel dazu werden diese Plätze in anderem Reisebüro vergeben
    - Überbuchung trotz Sitzplatzbestätigung und Tickets!

Abhängigkeit von nicht abgeschl. TAen



Abhängigkeit von nicht abgeschl. TAs

- Problem sieht harmlos aus
- Es ist jedoch ein Problem bei konsequenter Ausnutzung dieser Lücke
- Beispiel:
  - Person hat Schulden, darf Konto nicht überziehen
  - Person muss 1000 Euro überweisen, Konto ist aber leer
  - Freund überweist 1000 Euro
  - Person kann überweisen
  - Freund führt einen Rollback durch!

Inkonsistenz der Daten

TA1	**TA2**
Summiere drei Kontoinhalte	*Überweise 600 € von Konto3 auf Konto1*
Select Konto1 (400 €)	Select Konto3 (700 €)
Select Konto2 (300 €)	Update Konto3 (700-600=100 €)
…	Select Konto1 (400 €)
Select Konto3 (100 €)	Update Konto1 (400+600=1000 €)
Ausgabe: Summe (**800 €**)	Commit
Commit	

t

In Wahrheit: 1400€

Inkonsistenz der Daten

- Problem sieht harmlos aus
- Aber:
  - Mit diesen falschen Werten könnte jetzt intern weiter gearbeitet werden!

- Konsequenz:
- Alle drei Probleme sind zu vermeiden
  - Problem 1 ist grundsätzlich sehr kritisch
  - Probleme 2 und 3 bei „harmlosen" Transaktionen vorstellbar

z.B. Sammeln von einfachen Statistiken

Concurrency Strategien: optimistisch

- **Optimistische Strategie:**
  - Jede Transaktion darf beliebig lesen und ändern
  - Die drei Concurrency Probleme werden in Kauf genommen
- Aber:
  - Bei Transaktionsende wird auf parallele Zugriffe überprüft
  - Wenn keine parallelen Zugriffe: Alles OK
  - Wenn doch: Rücksetzen der Transaktion und Neustart
- Realisierung:
  - Zugriffszähler
- **Nachteil:**
  - Nur bei extrem niedriger Kollisionswahrscheinlichkeit einsetzbar
  - Gegenseitiges Aufschaukeln ist möglich: Immer wieder Neustarts

Concurrency Strategien: pessimistisch

- Pessimistische Strategie:
  - Alle von einer Transaktion angefassten Daten sind für andere Transaktionen gesperrt
  - Freigabe der Sperre am Transaktionsende
- Folgerung:
  - Andere Transaktionen müssen gegebenenfalls warten
- Realisierung:
  - Mit Sperrmechanismen (Locks)
- Nachteil:
  - Einschränkung der Parallelität
  - Hoher Verwaltungsaufwand für die Sperrmechanismen

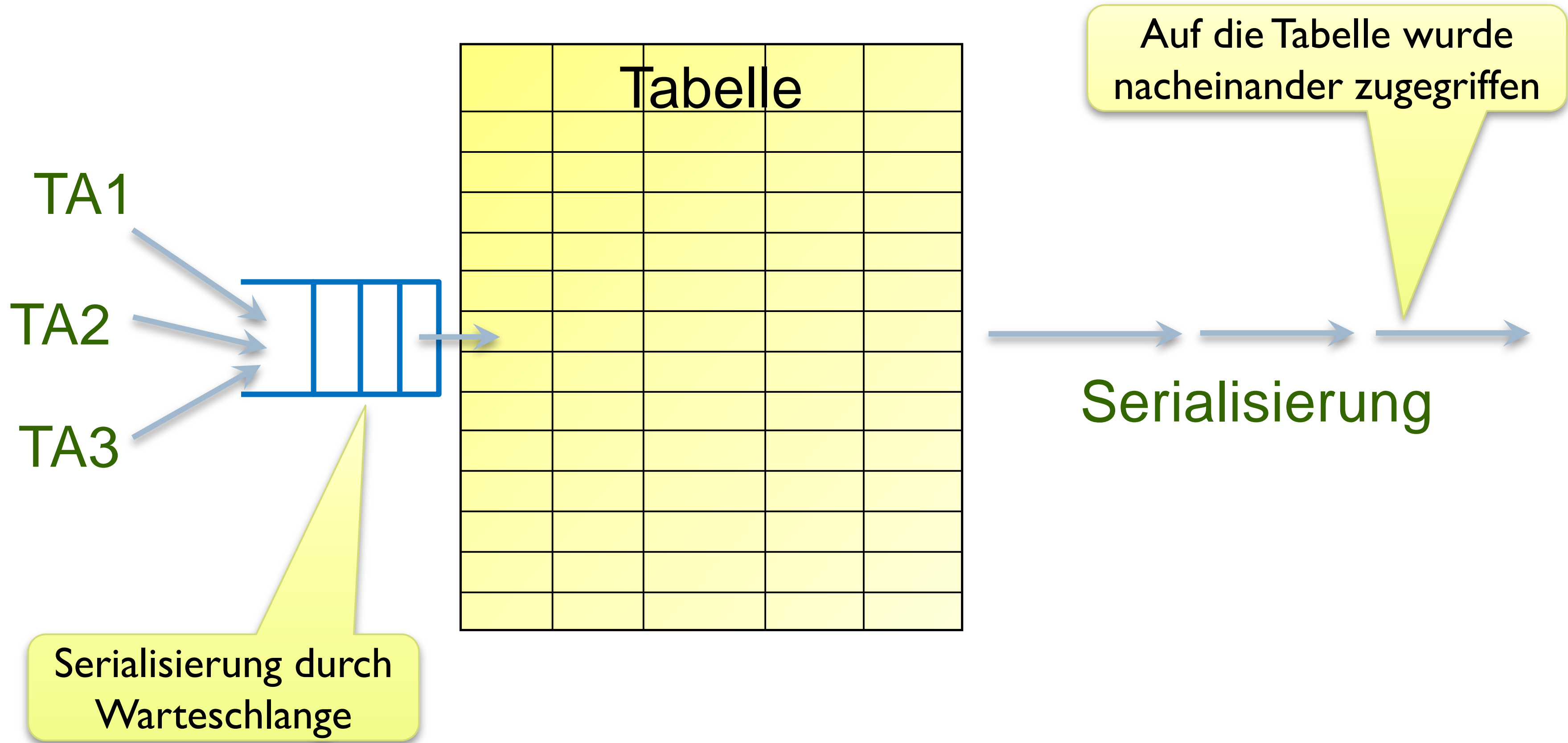
Vergleich der Concurrency Strategien

	Optimistische Strategie	Pessimistische Strategie
Vorteile	Einfache Implementierung Gute Performance Grundregel der Concurrency kann garantiert werden	Universell einsetzbar Grundregel der Concurrency kann garantiert werden
Nachteile	Kann sich aufschaukeln: daher nur in Spezialfällen einsetzbar	Aufwändige Implementierung Provoziert Wartezeiten anderer Transaktionen

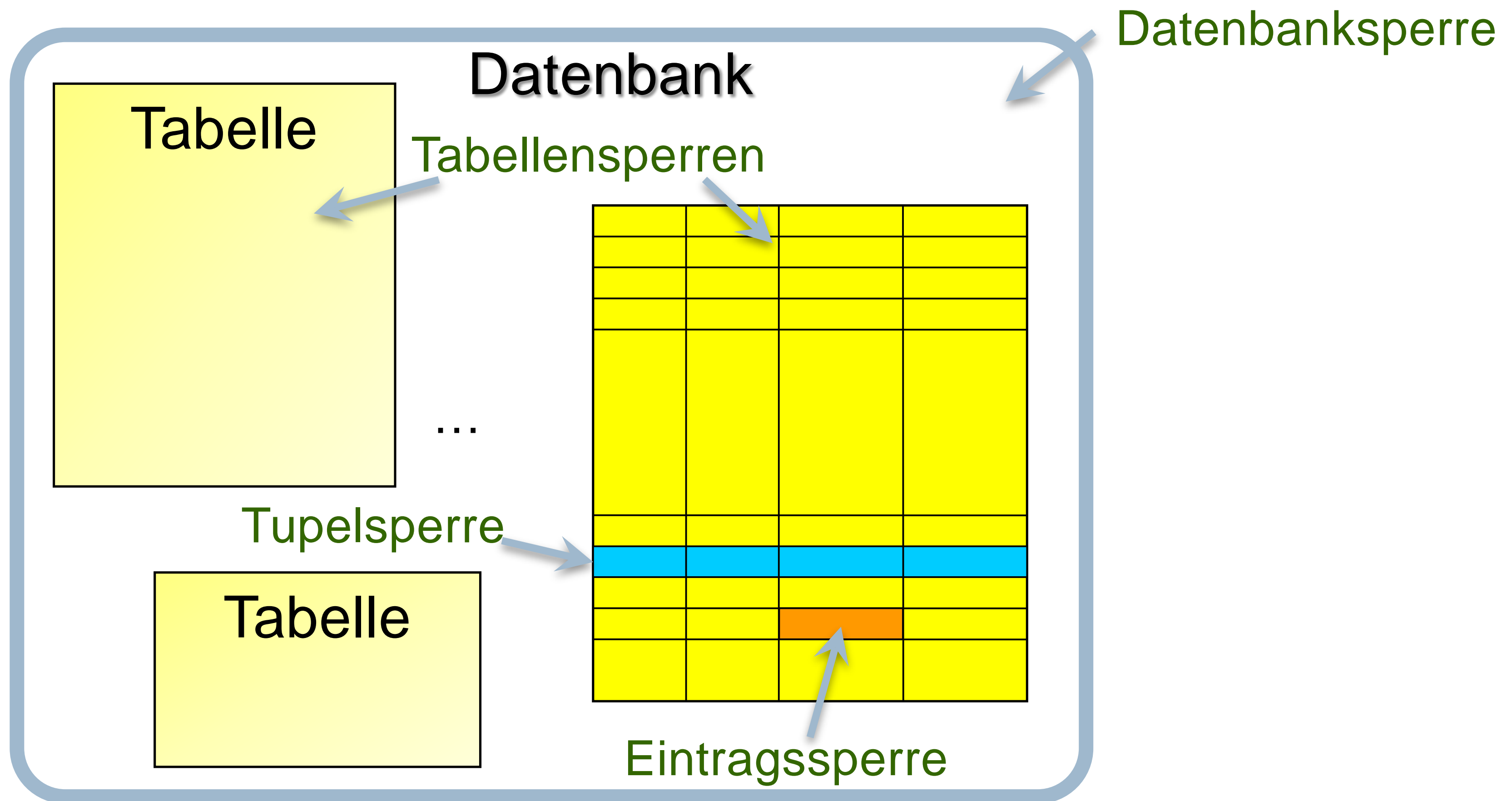
Sperrmechanismen

- Sperrmechanismen werden mit Locks realisiert
- Grundidee zu Locks in Datenbanken:
  - Zu jeder Relation existiert ein Lock
  - Eine Transaktion holt vor jedem Zugriff auf eine Relation automatisch den Lock dieser Relation
  - Ist der Lock von einer anderen Transaktion belegt, so wartet die Transaktion in einer Warteschlange, bis sie nach der Freigabe des Locks an der Reihe ist
  - Bei Transaktionsende werden alle gehaltenen Locks freigegeben

Lockmechanismus



Sperrgranulat in Datenbanken



Sperrgranulat in der Praxis (1)

- Datenbanksperre
  - Erlaubt keinen Parallelbetrieb
  - Nur im Einzelplatzbetrieb vorstellbar
- Tabellensperren
  - Relativ flexibel
  - Problem: Auf einzelne Tabellen wird intensiv zugegriffen
  - Bei geringer Parallelität gut einsetzbar
- Eintragssperren
  - Sehr sehr aufwändig
  - Daher kaum implementiert

Sperrgranulat in der Praxis (2)

- Tupelsperren
  - Alle wichtigen Datenbanken unterstützen Tupelsperren
  - Ist Standard in allen größeren Datenbanken
- Problem bei großen Datenbanken:
  - Beispiel: 1000 Tabellen mit je 100.000 Zeilen
  - Also: 100 Millionen unterschiedliche Locks!
- Implementierung:
  - Lockpool mit Poolverwaltung
  - Locks werden bei Bedarf eingerichtet

Lesende Zugriffe und Concurrency

- In der Praxis:
  - 80-90% Lesezugriffe
- Lesezugriffe sollten sich nicht gegenseitig behindern
- Aber:
  - Lesende dürfen Änderungen nicht lesen
  - Schreibende dürfen Daten nicht ändern, wenn vorher von anderen gelesen

vor dem Commit der anderen Transaktion!

Inkonsistenz der Daten!

Exklusiv- und Share-Lock

- **Definition (Exklusiv-Lock, Share-Lock)**
  - Ein Exklusiv-Lock auf ein Objekt weist alle weiteren Exklusiv- und Share-Lockanforderungen auf dieses Objekt zurück.
  - Ein Share-Lock auf ein Objekt gestattet weitere Share-Lockzugriffe auf dieses Objekt, weist aber exklusive Lockanforderungen zurück.
- Bei Zurückweisung wird bis zur Lockfreigabe gewartet

Locks in Datenbanken

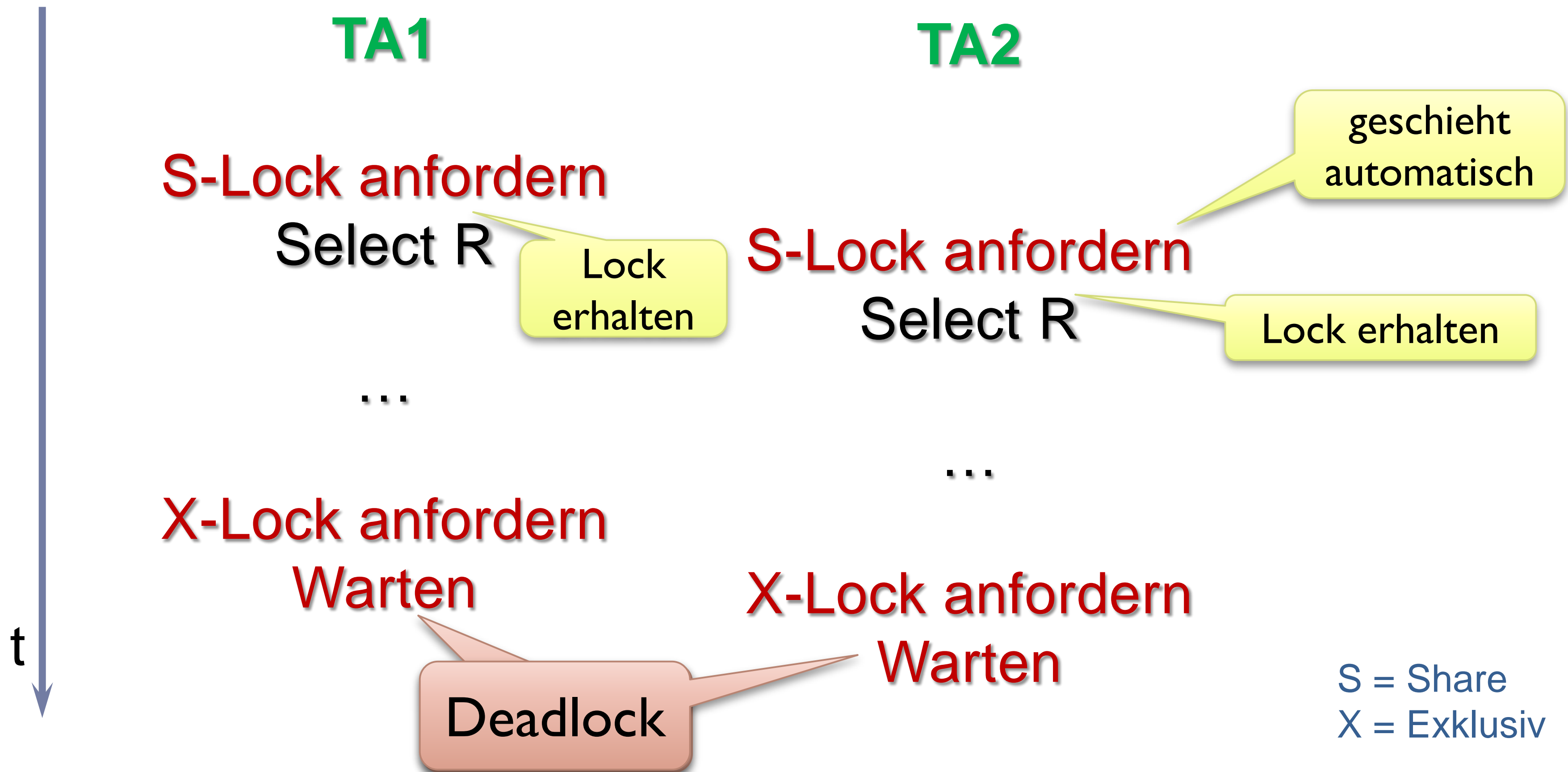
- Vor dem lesenden Zugriff auf eine Zeile:
  - Share-Lock für diese Zeile wird geholt
- Vor dem schreibenden Zugriff auf eine Zeile:
  - Exklusiv-Lock für diese Zeile wird geholt

- Gegebenenfalls wird so lange gewartet, bis der Lock verfügbar ist

Sperren in Datenbanken

- Vor Lesezugriff:
  - Share-Lock wird automatisch angefordert
  - Transaktion erhält Share-Lock, wenn es keine anderen Exklusiv-Lock-Anforderungen anderer Transaktionen gibt
- Vor Schreibzugriff:
  - Exklusiv-Lock wird automatisch angefordert
  - Transaktion erhält Exklusiv-Lock, wenn es keine anderen Share- oder Exklusiv-Lock-Anforderungen gibt
  - Hält Transaktion bereits den Share-Lock, so wird dieser in Exklusiv-Lock umgewandelt, sobald verfügbar
- Misslingt Lock-Anforderung, so wird bis Lockfreigabe gewartet
- Bei Transaktionsende
  - Freigabe aller gehaltenen Locks

Verlorengegangene Änderung (2)

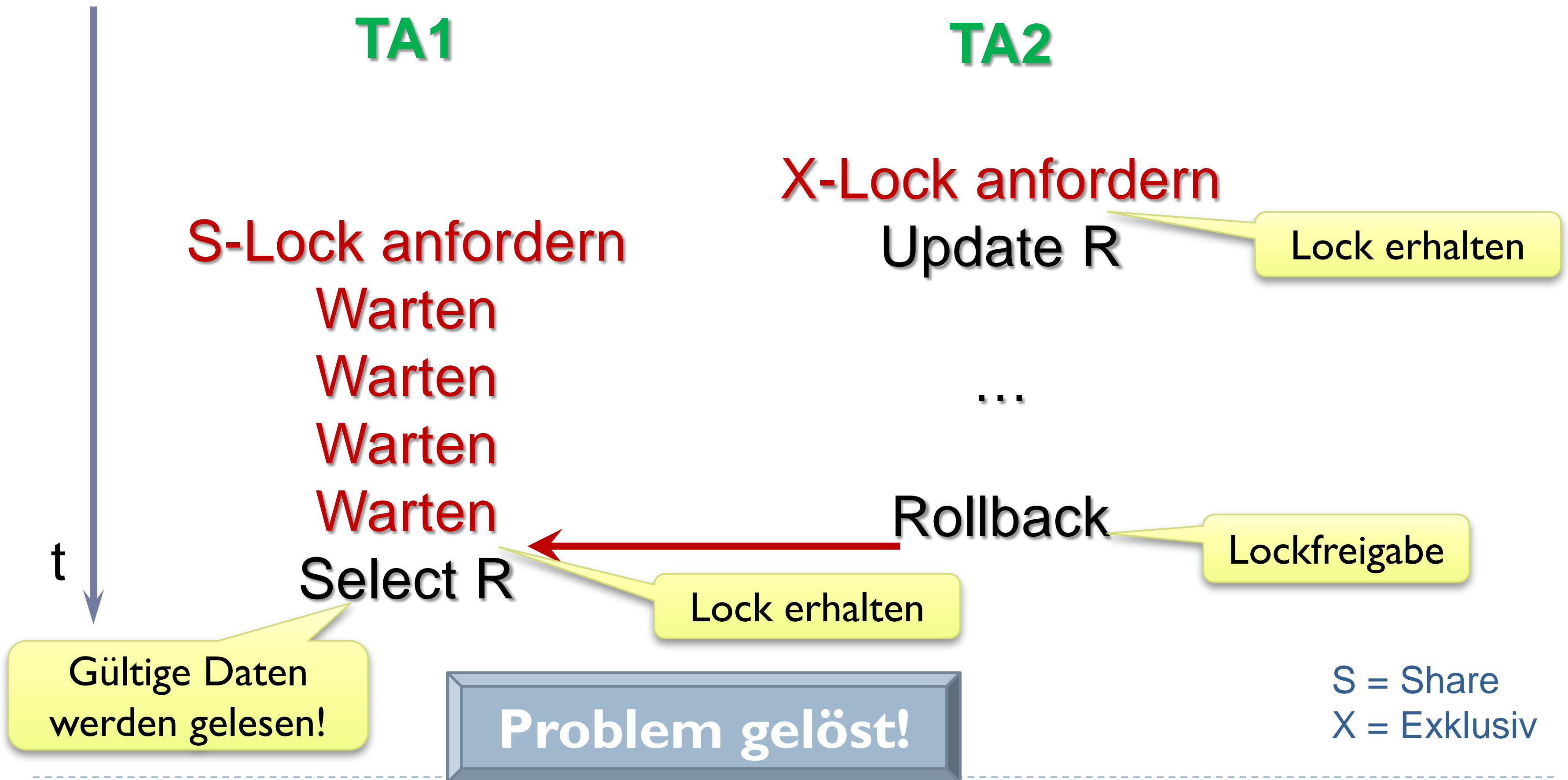


Deadlock

- **Definition (Deadlock)**
  - Eine Verklemmung, bei der mindestens zwei Transaktionen gegenseitig auf die Freigabe eines oder mehrerer Locks warten, heißt Deadlock.

Wir stellen Deadlocks zunächst zurück und betrachten erst die beiden anderen Concurrency-Probleme

Abhängigkeit von nicht abgeschl. TAs (2)



Inkonsistenz der Daten (2)

TA1	TA2
Summiere drei Kontoinhalte	*Überweise 600 € von Konto3 auf Konto1*
	S-Lock Konto3 anfordern
	Select Konto3 (700 €)
S-Lock Konto1 anfordern	
Select Konto1 (400 €)	
	X-Lock Konto3 anfordern
	Update Konto3 (700-600=100 €)
S-Lock Konto2 anfordern	
Select Konto2 (300 €)	
	S-Lock Konto1 anfordern
	Select Konto1 (400 €)
S-Lock Konto3 anfordern	
Warten	
	X-Lock Konto1 anfordern
	Warten

t

Deadlock

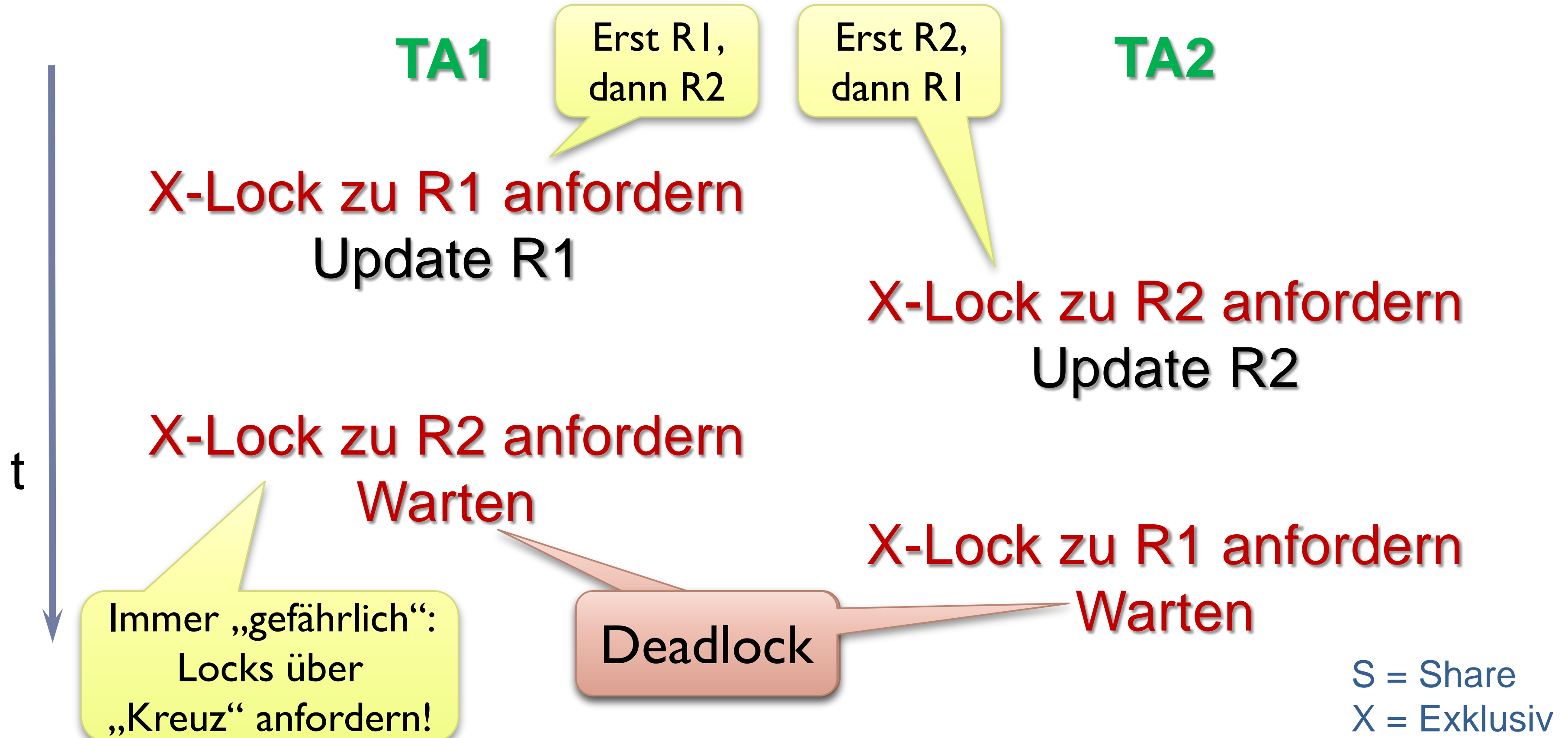
S = Share
X = Exklusiv

Ergebnis

- Einsatz von Exklusiv- und Share-Locks
  - löst ein Concurrency-Problem
  - führt zweimal zu Deadlock

- Folgerung:
  - Können wir das Deadlockproblem lösen, so sind auch die Concurrency-Probleme gelöst
  - Lesende behindern Schreibende und umgekehrt
  - Deadlocks treten auch zwischen Lesenden und Schreibenden auf

Deadlocks bei Schreibzugriffen



Deadlockvermeidung

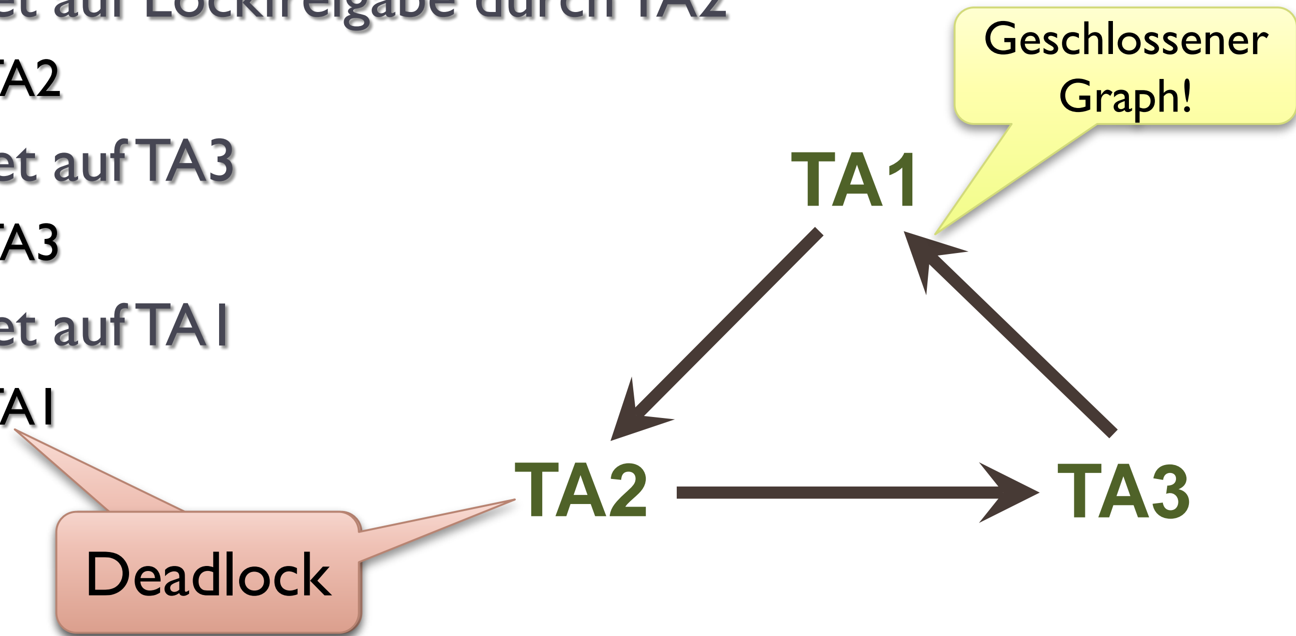
- Es entstehen keine Deadlocks, wenn
  - Locks in einer vorgegebenen Reihenfolge angefordert werden
- Beispiel:
  - Relationen werden alphabetisch geordnet
  - Tupel werden nach Primärschlüssel geordnet
  - Auf alle während einer Transaktion verwendeten Tupel wird in der Reihenfolge gemäß obiger Ordnung zugegriffen
- Aber:
  - Nicht immer ist zu Beginn einer Transaktion bekannt, auf welche Tupel zugegriffen wird
  - Eventuell muss Transaktion zurückgesetzt und neu gestartet werden
  - In der Praxis zu unflexibel

Deadlock-Erkennung (1)

- Einfache Strategie:
  - Beobachten von Wartezeiten
  - Bei langen Wartezeiten: Transaktion mit Fehler abbrechen
- Nachteile:
  - Eine lange wartende Transaktion muss nicht im Deadlock sein
  - Die optimale Wartezeit bis zum Abbruch ist nicht bekannt:
    - Ein zu kurzes Warten bricht auch „unschuldige" Transaktionen ab
    - Ein zu langes Warten verlängert die Antwortzeit dieser Transaktionen

Deadlockerkennung (2)

- Sicheres Erkennen mittels Wartegraphen
- Beispiel (3 Transaktionen TA1, TA2, TA3):
  - TA1 wartet auf Lockfreigabe durch TA2
    - TA1 → TA2
  - TA2 wartet auf TA3
    - TA2 → TA3
  - TA3 wartet auf TA1
    - TA3 → TA1



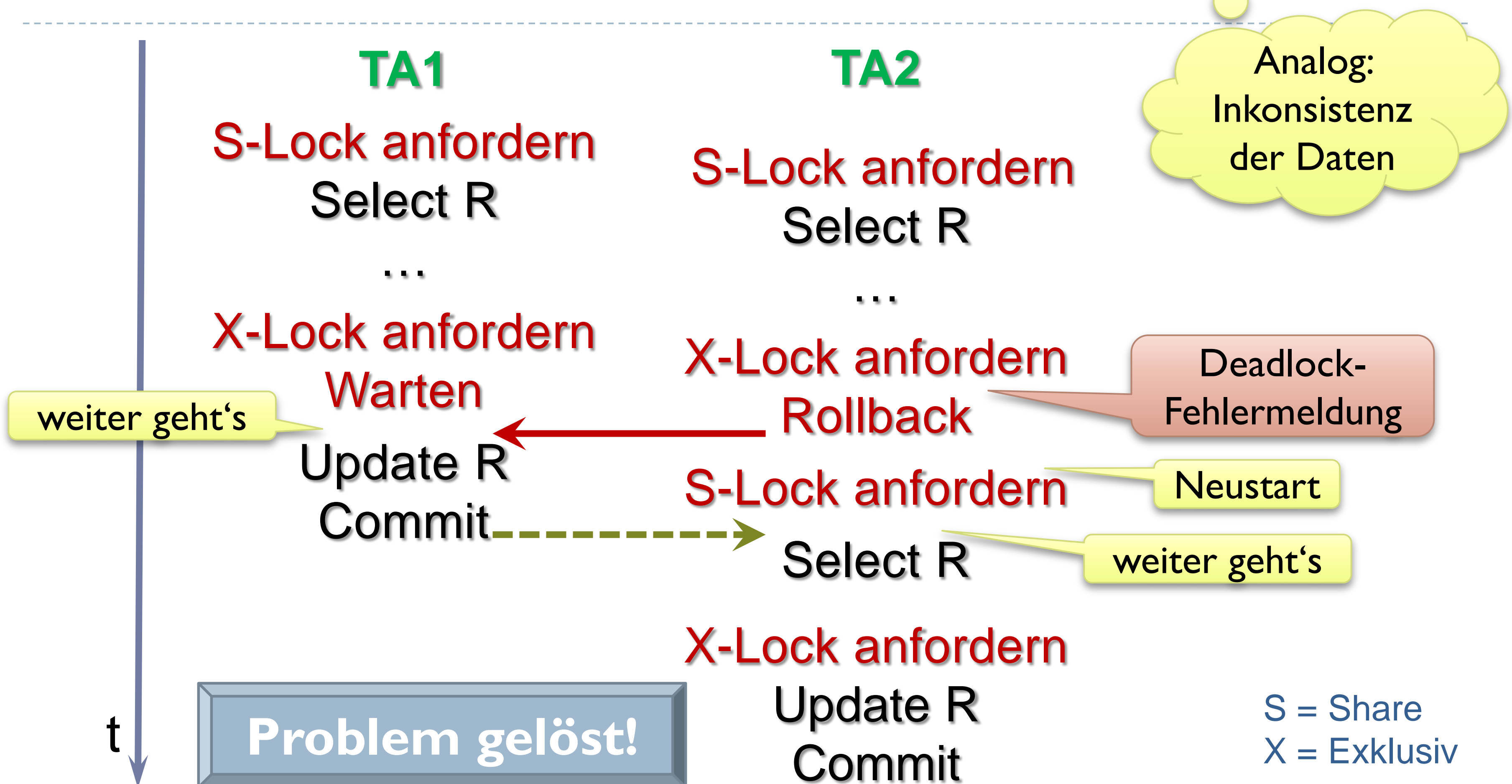
Deadlockerkennung in der Praxis

- Meist warten nur wenige Transaktionen auf Lockfreigabe
- Muss eine Transaktion warten,
  - wird ein gerichteter Graph hinzugefügt (Pfeil)
  - wird überprüft, ob dadurch ein geschlossener Zyklus entsteht
- Der Aufwand dieser Implementierung ist nicht allzu hoch
- Heute: Standard in modernen Datenbanken

Deadlockauflösung

- Datenbank:
  - Zuständig für Deadlockerkennung
  - DML-Befehl führt zu Fehlermeldung
    - Fehlervariable SQLSTATE: 40001 (SQL-Norm)
- Datenbankanwendung:
  - Reagiert auf Fehlermeldung
  - Fast immer die einzige sinnvolle Reaktion:
    - Zurücksetzen der Transaktion
    - Neustart der Transaktion
  - Warum Transaktion zurücksetzen?
    - Deadlock entstand, weil Transaktion Locks hält
    - Nur die Freigabe aller Locks löst Verklemmung sicher auf

Verlorengegangene Änderung (3)



Endergebnis

- Die Probleme der Concurrency lassen sich lösen
  - mit Exklusiv- und Share-Locks
  - und mit Lockerkennung
  - und mit Rücksetzen und Neustart einer Transaktion
- Merke:
  - Bei Deadlockfehlermeldung: Transaktion zurücksetzen
  - Gegebenenfalls Neustart dieser Transaktion

Concurrency in der SQL-Norm

- Concurrency Probleme nach SQL-Norm:
  - Dirty Write
    - (Verlorengegangene Änderung)
  - Dirty Read
    - (Abhängigkeit von nicht abgeschlossenen Transaktionen)
  - Non Repeatable Read
    - (Wiederholtes Lesen führt zu anderen Ergebnissen)
  - Phantom
    - (Eine bisher nicht vorhandene Zeile erscheint)

trotz Sperren möglich, da andere Transaktion neue Zeile einfügen kann

Isolationslevel gemäß SQL-Norm

Isolationslevel	Dirty Write erlaubt?	Dirty Read erlaubt?	Non Repeatable Read erlaubt?	Phantom erlaubt?
Read Uncommitted	Nein	Ja	Ja	Ja
Read Committed	Nein	Nein	Ja	Ja
Repeatable Read	Nein	Nein	Nein	Ja
Serializable	Nein	Nein	Nein	Nein

Isolationslevel setzen

- Befehl SET TRANSACTION

SET TRANSACTION ISOLATION LEVEL Level

- Level:
  - Read Uncommitted
  - Read Committed
  - Repeatable Read
  - Serializable
- Meist erster Befehl einer Transaktion

Mögliche Realisierung der Level

z.B. S-Locks auf Tabellen; Insert benötigt X-Lock

Isolationslevel	Exklusiv-Locks zum Schreiben	Schreiben auf Kopie	Share-Locks zum Lesen	Range-Locks
Read Uncommitted	Ja	Nein	Nein	Nein
Read Committed	Ja	Ja	Nein	Nein
Repeatable Read	Ja	Ja	Ja	Nein
Serializable	Ja	Ja	Ja	Ja

Concurrency und Oracle

Set Transaction Isolation Level Serializable;
Set Transaction Isolation Level Read Committed;
Set Session Isolation Level Serializable;
Set Session Isolation Level Read Committed;

Erster Befehl einer Transaktion!

Standard

Ab jetzt für gesamte Session

Read Uncommitted und Repeatable Read nicht implementiert

- Oracle verwendet keine Lesesperren!
- Im Level Serializable:
  - Optimistische Lesestrategie!
  - Gegebenenfalls Serialisierungsfehler ORA-08177
- Ausnahme:
  - Share-Lock bei SELECT … FOR UPDATE

Concurrency und SQL Server

- Alle vier Isolationslevel implementiert
- Intern:
  - Exklusiv- und Share-Locks
  - Phantomvermeidung durch Key-Range-Locks
- Notwendig:
  - Transaktion explizit starten: BEGIN TRANSACTION;
- Überprüfen des Isolationslevels:

  dbcc useroptions;

Sonst keine Transaktion und keine Concurrency

Concurrency und MySQL

SET TRANSACTION ISOLATION LEVEL Level;

SET GLOBAL TRANSACTION ISOLATION LEVEL Level;

Ab der nächsten Transaktion für gesamte Session

- Standard: Repeatable Read
- Voraussetzung:
  - Engine INNODB
  - Transaktionsmodus
    - Entweder: SET AUTOCOMMIT=0;
    - Oder: START TRANSACTION;
- Intern:
  - Exklusiv- und Share-Locks
  - Phantomvermeidung durch Next-Key-Locking-Algorithmus

Ab jetzt: wie in Oracle

Wie in SQL Server expliziter Start

Zusammenfassung

- Ohne Transaktionen:
  - Keine sichere Recovery
  - Keine Concurrency
- Recovery:
  - Datenbank-Pufferung, Redo-Logs, Undo-Log, Checkpoints
- Concurrency:
  - Sperren (Exklusiv, Share), Deadlock, Deadlockerkennung, Rücksetzen der Transaktion und Neustart, Isolationslevel