

# Kapitel 4: Transaktionen und ihre Realisierung

---

## ■ Lernziele und Überblick:

### Transaktionen, Eigenschaften von Transaktionen

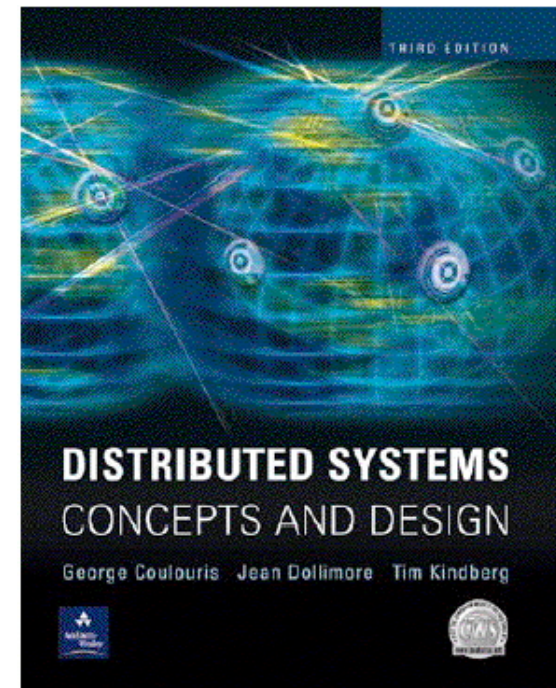
- Integritätsbedingungen
- Isolation von Transaktionen:
  - Terminologie und Formalisierung
  - Klassifikation der Probleme durch Verletzung der Isolation
  - Isolation durch Sperrprotokolle
- Diskussion: Vor- und Nachteile niedriger Isolationsgrade

# Danksagung

---

Diese Vorlesung baut auf einer Vorlesung "P3" von Bernd Neumann an der Universität Hamburg auf.

Für eine Vertiefung des Themas Transaktionen und Sperren (Locks) siehe auch Kapitel 12 aus:



# Beispiel Kontoführung (Lost Update)



Prozeß 1: Umbuchung eines Betrages von Konto A nach Konto B

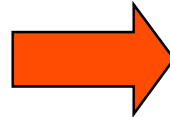
Prozeß 2: Zinsgutschrift für Konto A

## Umbuchung

```
read (A, a1)
a1 := a1 - 300
write (A, a1)
read (B, b1)
b1 := b1 + 300
write (B, b1)
```

## Zinsgutschrift

```
read (A, a2)
a2 := a2 * 1.03
write (A, a2)
```



## Möglicher verzahnter Ablauf:

### Umbuchung      Zinsgutschrift

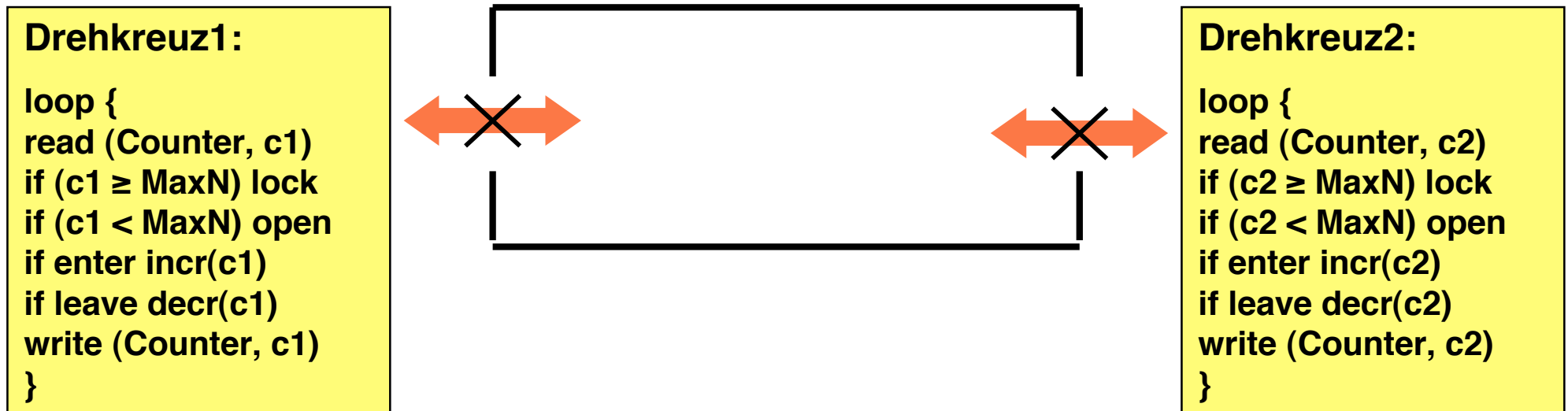
```
read (A, a1)
a1 := a1 - 300
```

```
read (A, a2)
a2 := a2 * 1.03
write (A, a2)
```

```
write (A, a1)
read (B, b1)
b1 := b1 + 300
write (B, b1)
```

Wo ist die Zinsgutschrift geblieben??

# Beispiel Besucherzählung



Verzahnte Ausführung der zwei Prozesse Drehkreuz1 und Drehkreuz2 mit Zugriff auf gemeinsamen Counter kann inkorrekte Besucherzahl ergeben!

=> Überfüllung, Panik, Katastrophen durch Studium der Nebenläufigkeit vermeiden

# Mehrbenutzersynchronisation

Die nebenläufige Ausführung mehrerer Prozesse auf einem Rechner kann grundsätzlich zu einer besseren Ausnutzung des Prozessors führen, weil Wartezeiten eines Prozesses (z.B. auf ein I/O-Gerät) durch Aktivitäten eines anderen Prozesses ausgefüllt werden können.



Prozesse synchronisieren = partielle zeitliche Ordnung herstellen

# Korrektheitskriterium Serialisierbarkeit

---

- Mehrbenutzerbetrieb führt zur Verschränkung der Ausführung der jeweiligen Kontrollflüsse
- Eine verschränkte Abarbeitung zweier Kontrollflüsse A und B ist korrekt, wenn der Ergebniszustand entweder einer Abarbeitungsreihenfolge „Erst A, dann B“ oder „Erst B, dann A“ entspricht
- Man sagt: die Abarbeitung der Einzelanweisungen ist serialisierbar

# Welche anderen Probleme gibt es?



- Dirty read
  - Abhängigkeit von nicht freigegebenen Änderungen
- Phantomproblem
  - Spätere Generierung eines Datums, das während der Abarbeitung eigentlich hätte berücksichtigt werden müssen
- ...

# Mehrbenutzerbetrieb von Datenbanksystemen



Um Probleme durch unerwünschte Verzahnung nebenläufiger Zugriffe (s. Beispiel Kontoführung) zu vermeiden, werden atomare Aktionen zu größeren Einheiten geklammert: *Transaktionen*.

Eine Transaktion ist eine Folge von Aktionen (Anweisungen), die ununterbrechbar ausgeführt werden soll.

Da Fehler während einer Transaktion auftreten können, muß eine Transaktionsverwaltung dafür sorgen, daß unvollständige Transaktionen ggf. zurückgenommen werden können.

Befehle für Transaktionsverwaltung:

- begin of transaction (BOT) *Beginn der Anweisungsfolge einer Transaktion*
- commit *Einleitung des Endes einer Transaktion, Änderungen der Datenbasis werden festgeschrieben*
- abort *Abbruch der Transaktion, Datenbasis wird in den Zustand vor der Transaktion zurückversetzt*

# Eigenschaften von Transaktionen

---



ACID-Paradigma steht für 4 Eigenschaften:

**A**tomicity (Atomarität)

Eine Transaktion wird als unteilbare Einheit behandelt ("alles-oder-nichts").

**C**onsistency (Konsistenz)

Eine Transaktion hinterläßt nach (erfolgreicher oder erfolgloser) Beendigung eine konsistente Datenbasis.

**I**solation

Nebenläufig ausgeführte Transaktionen beeinflussen sich nicht gegenseitig.

**D**urability (Dauerhaftigkeit)

Eine erfolgreich abgeschlossene Transaktion hat dauerhafte Wirkung auf die Datenbank, auch bei Hardware- und Software-Fehlern.

# Mehrbenutzerbetrieb in DBsystemen

---

## **Synchronisation mehrerer nebenläufiger Transaktionen:**

- **Bewahrung der intendierten Semantik einzelner Transaktionen**
- **Protokolle zur Sicherung der Serialisierbarkeit**
- **Sicherung von Rücksetzmöglichkeiten im Falle von Abbrüchen**
- **Vermeidung von Schneeballeffekten beim Rücksetzen**
- **Behandlung von Verklemmungen**

# Synchronisation bei Mehrbenutzerbetrieb

---

**Synchronisationsproblem = verzahnte sequentielle Ausführung nebenläufiger Transaktionen, so daß deren Wirkung der intendierten unverzahnten ("seriellen") Hintereinanderausführung der Transaktionen entspricht.**

**Konfliktursache im DB-Kontext ist read und write von zwei Prozessen i und k auf dasselbe Datum A:**

$read_i(A)$ $read_k(A)$	Reihenfolge irrelevant, kein Konflikt
$read_i(A)$ $write_k(A)$	Reihenfolge muß spezifiziert werden, Konflikt
$write_i(A)$ $read_k(A)$	analog
$write_i(A)$ $write_k(A)$	Reihenfolge muß spezifiziert werden, Konflikt

**Serialisierbarkeitsgraph:**

**Knoten = atomare Operationen (read, write)**

**Kanten = Ordnungsbeziehung (Operation i vor Operation k)**

**Serialisierbarkeitstheorem:**

**Eine partiell geordnete Menge nebenläufiger Operationen ist genau dann serialisierbar, wenn der Serialisierungsgraph zyklensfrei ist.**

# Beispiel für nicht serialisierbare Historie

T1	T2	T1	T2	T1	T2
<b>BOT</b> read(A) write(A)	<b>BOT</b> read(A) write(A) read(B) write(B) commit	<b>BOT</b> read(A) write(A) read(B) write(B) commit	<b>BOT</b> read(A) write(A) read(B) write(B) commit	<b>BOT</b> read(A) write(A) read(B) write(B) commit	<b>BOT</b> read(A) write(A) read(B) write(B) commit
read(B) write(B) commit					
verzahnte Historie		Serialisierung 1		Serialisierung 2	

Der Effekt dieser Verzahnung entspricht keiner der 2 möglichen Serialisierungen T1 vor T2 oder T2 vor T1: Die Historie ist nicht serialisierbar

# Sperrsynchronisation

Viele Datenbank-Systeme verwenden Sperranweisungen zur Erzeugung konfliktfreier Abläufe:

- Sperrmodus S (shared, read lock, Lesesperre)

Wenn Transaktion  $T_i$  eine S-Sperre für ein Datum A besitzt, kann  $T_i$   $\text{read}(A)$  ausführen. Mehrere Transaktionen können gleichzeitig eine S-Sperre für dasselbe Objekt A besitzen.

- Sperrmodus X (exclusive, write lock, Schreibsperre)

Nur eine einzige Transaktion, die eine X-Sperre für A besitzt, darf  $\text{write}(A)$  ausführen.

Verträglichkeit der Sperren untereinander:

(NL = no lock, keine Sperrung)

	NL	S	X
S	ok	ok	-
X	ok	-	-

# Zwei-Phasen-Sperrprotokoll

(Englisch: two-phase locking, 2PL)

Protokoll gewährleistet die Serialisierbarkeit von Transaktionen.  
Für jede individuelle Transaktion muß gelten:

1. Jedes von einer Transaktion betroffene Objekt muß vorher entsprechend gesperrt werden.
2. Eine Transaktion fordert eine Sperre, die sie besitzt, nicht erneut an.
3. Eine Transaktion muß solange warten, bis es eine erforderliche Sperre entsprechend der Verträglichkeitstabelle erhalten kann.
4. Jede Transaktion durchläuft 2 Phasen:
  - in Wachstumsphase werden Sperren angefordert, aber nicht freigegeben
  - in Schrumpfungsphase werden Sperren freigegeben, aber nicht angefordert
5. Bei EOT (Transaktionsende) muß eine Transaktion alle ihre Sperren zurückgeben.

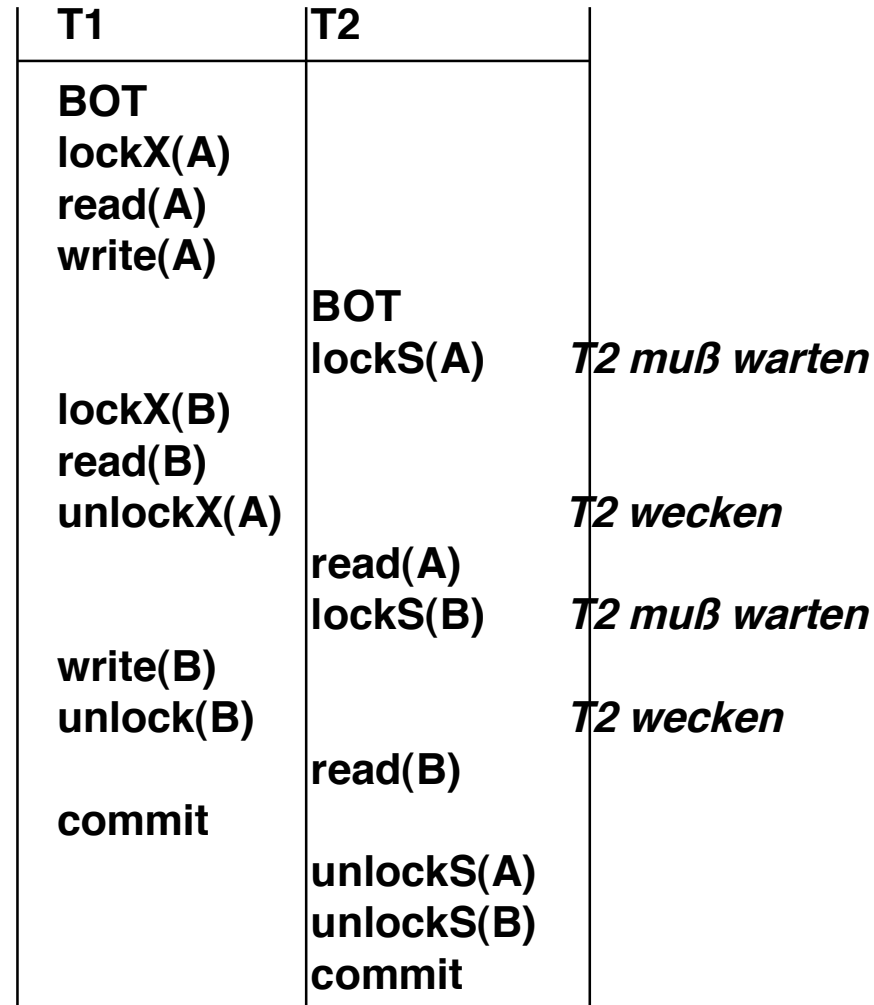
**Verschärfung zum "Strengen 2PL-Protokoll" zur Vermeidung von kaskadierten Rücksetzungen:**

**Keine Schrumpfungsphase, alle Sperren werden bei EOT freigegeben.**

# Beispiel für 2PL-Verzahnung

**T1: Modifikation von A und B**  
(z.B. Umbuchung)

**T2: Lesen von A und B**  
(z.B. Addieren der Salden)



# Verklemmungen (Deadlocks)

Sperrbasierte Synchronisationsmethoden können (unvermeidbar) zu Verklemmungen führen:

Gegenseitiges Warten auf Freigabe von Sperrern

Transaktionen leicht modifiziert:

T1: Modifikation von A und B  
(z.B. Umbuchung)

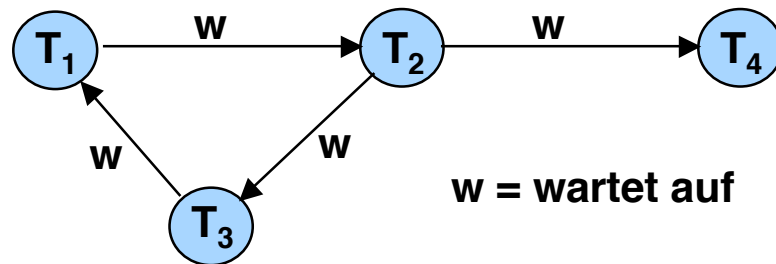
T2: Lesen von B und A  
(z.B. Addieren der Salden)

T1	T2
BOT lockX(A)	
	BOT lockS(B) read(B)
read(A) write(A) lockX(B)	
	lockS(A)

*T1 muß auf T2 warten  
T2 muß auf T1 warten  
=> **Deadlock***

# Strategien zur Erkennung und Vermeidung von Verklemmungen

## 1. Wartegraph hat Zyklen



w = wartet auf

Nach Erkennen eines Zyklus muß Verklemmung durch Zurücksetzen einer geeigneten Transaktion beseitigt werden.

## 2. Preclaiming - Vorabforderung aller Sperren

Beginn einer Transaktion erst, nachdem die für diese Transaktion insgesamt erforderlichen Sperren erfolgt sind.

Problem: Vorab die erforderlichen Sperren erkennen

## 3. Zeitstempel

Transaktionen werden durch Zeitstempel priorisiert. Zurücksetzen statt Warten, wenn  $T_1$  Sperre fordert,  $T_2$  aber Sperre erst freigeben muß:

- Strategie Wound-wait: Abbruch von  $T_2$ , falls  $T_2$  jünger als  $T_1$ , sonst warten
- Strategie Wait-die: Abbruch von  $T_1$ , wenn  $T_1$  jünger als  $T_2$ , sonst warten

# Automatisches Rücksetzen

---

- Bei Verklemmungen muß zur Behebung eine automatische Rücksetzung erfolgen
- Durch ACID-Eigenschaften der Transaktionen ist dieses potentiell möglich
- Aber: Neuaufsetzen der Transaktionen erforderlich (-> Laufzeitproblematik)