6 Datenbanken

1. Motivation
2. Datenorganisation und Datenbankkonzept
3. Semantische Datenmodellierung
4. Umsetzung in Datenbanken
5. Datenbanknutzung mit SQL
6. **Transaktionsmanagement**
7. Datenbankentwicklung
8. Datenbanken und IT-Sicherheit
9. Systemarchitektur
10. Verteilte Datenbanken
11. NoSQL und Entwicklungstrends

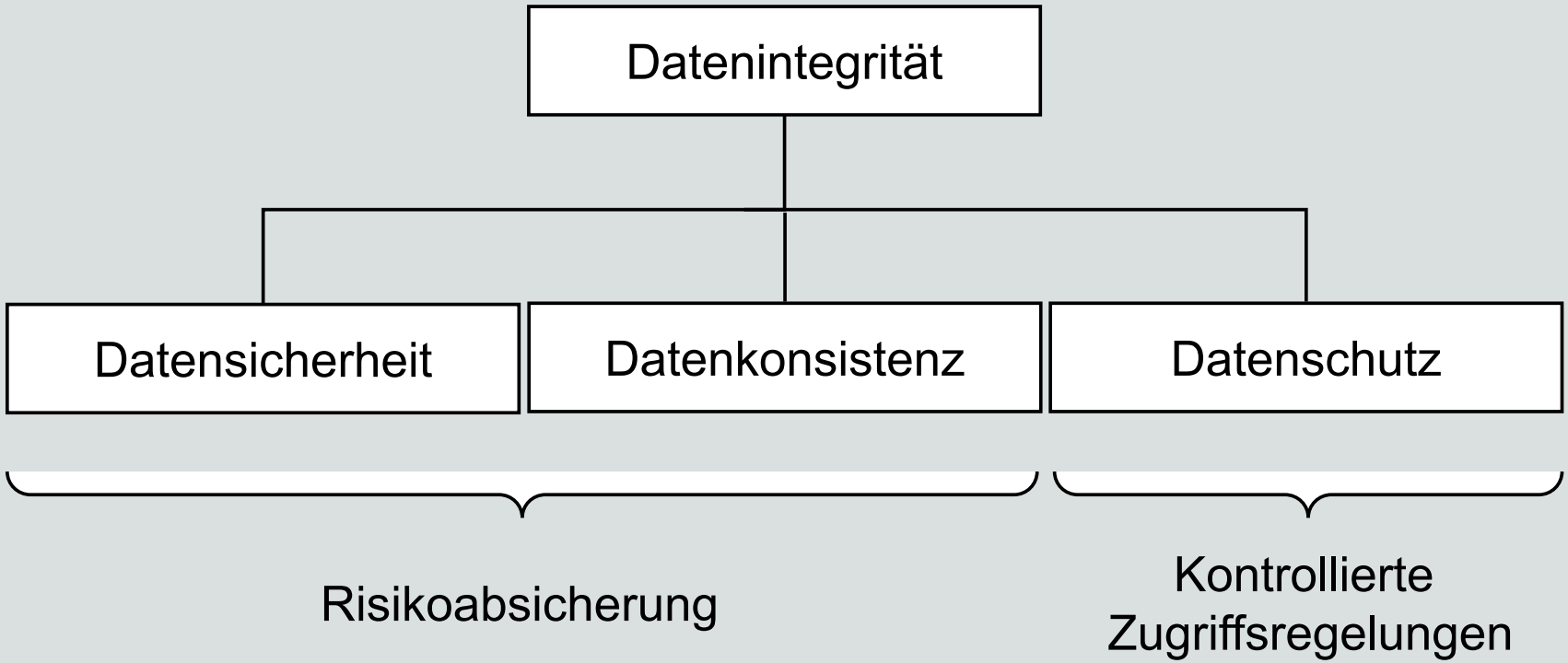
6 Lernziele

- Sie kennen die Probleme rund um Datenkonsistenz und wissen, wie eine Datenbank konsistent gehalten werden kann.
- Sie wissen, was Transaktionen sind und können diese sinnvoll einsetzen.
- Durch Transaktionen entstehen gewisse Probleme. Sie verstehen diese Probleme und können beschreiben, wie Datenbanksysteme damit umgehen (Sperrverfahren).

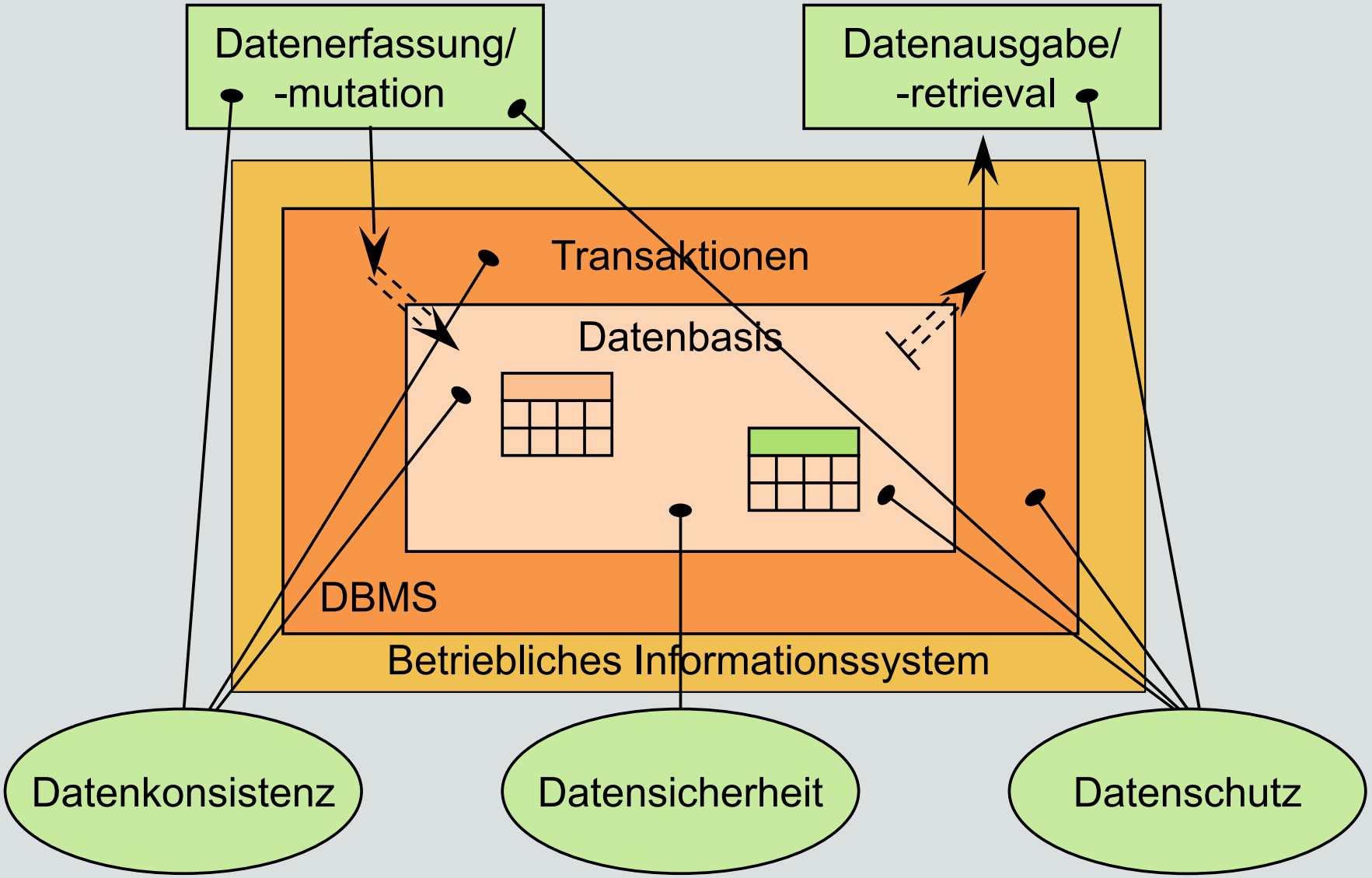
6 Integritätsaspekte

- Wert eines Informationssystems wird bestimmt durch Aktualität, Qualität und Korrektheit der Daten

⇒ Vertrauen der Nutzer



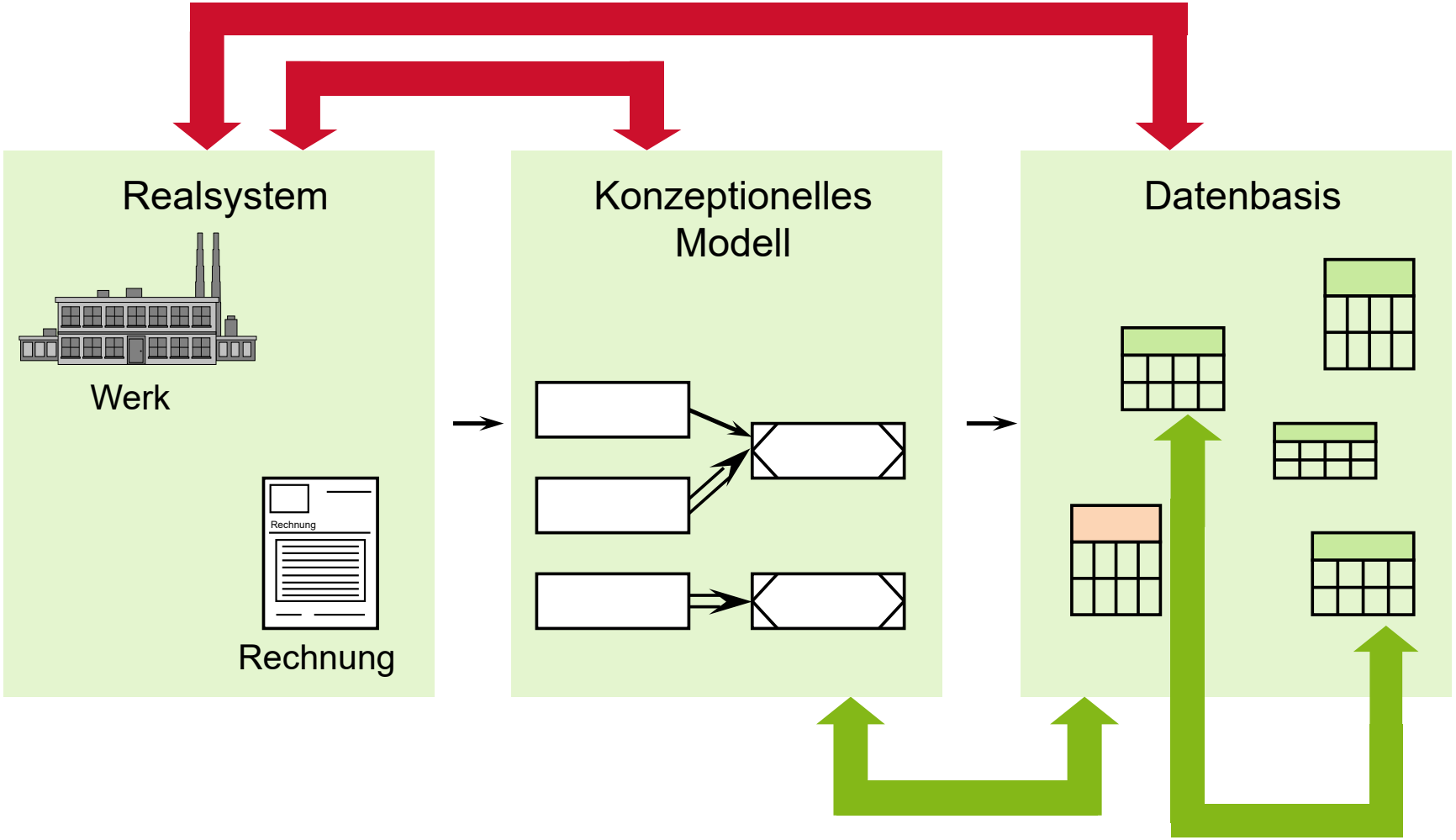
6 Datensicherheit, Datenkonsistenz und Datenschutz



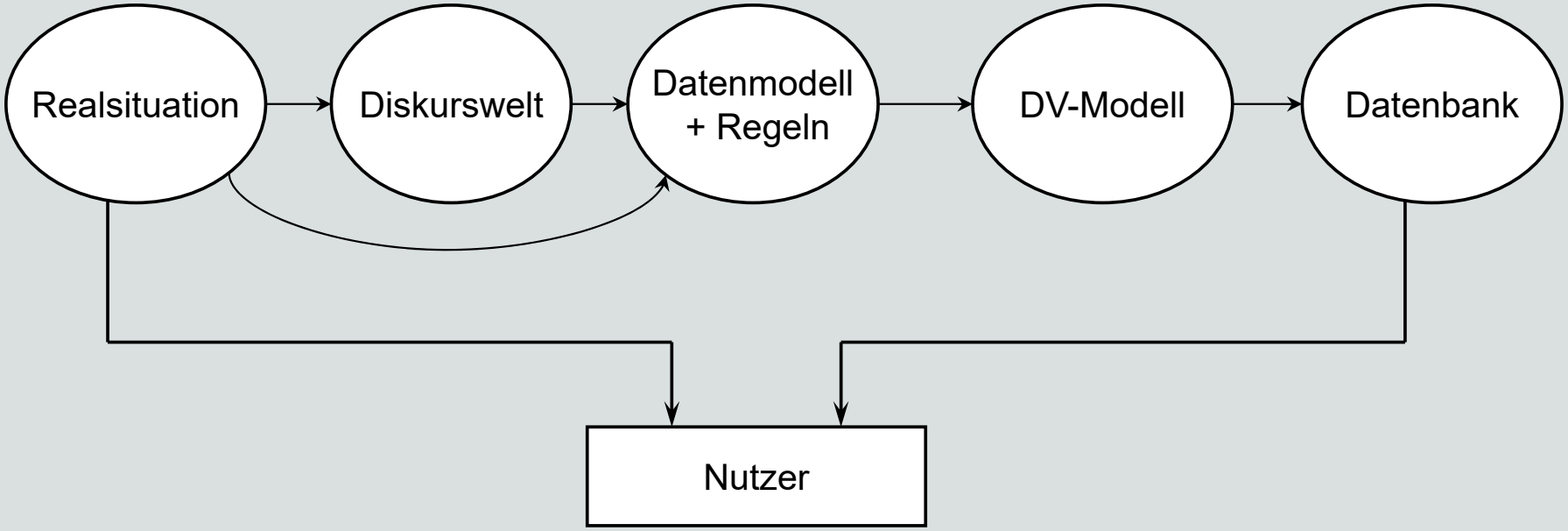
6 Datenkonsistenz

Datenkonsistenz
Widerspruchsfreiheit der Daten zu sich selbst und zu den Vereinbarungen des konzeptionellen Modells

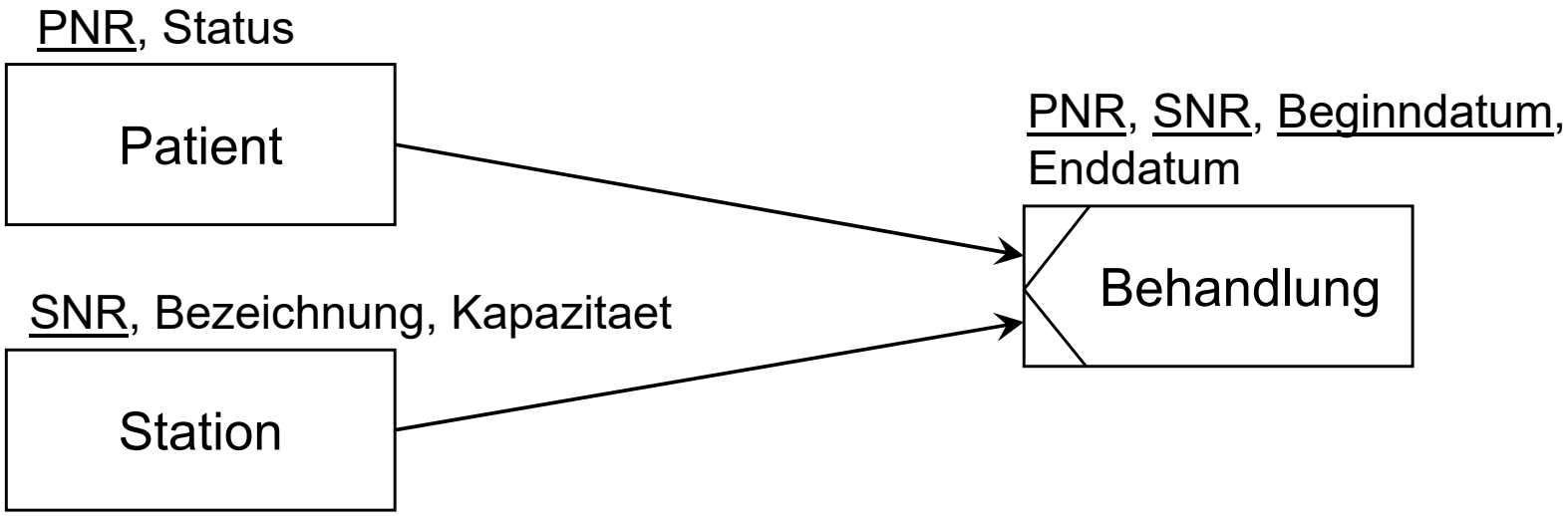
6 Datenkonsistenz



6 Entwicklungsprozess einer Datenbank



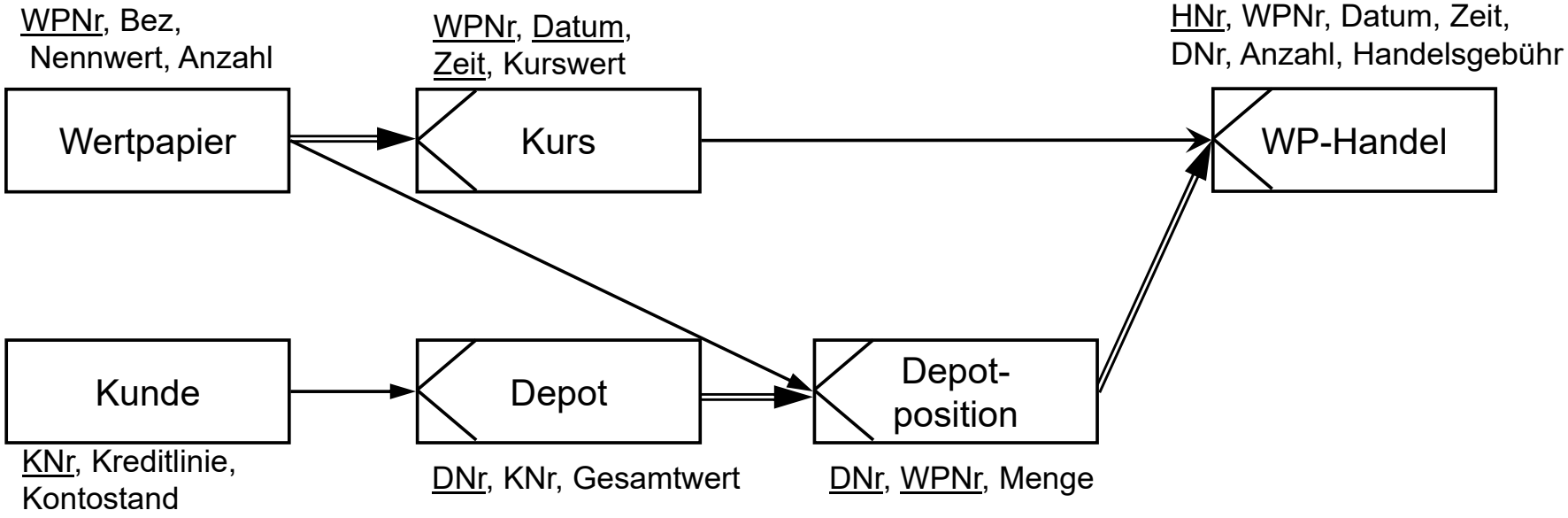
6 Beispiel zu Konsistenzregeln



Konsistenzregeln: Beschreibung von Eigenschaften zulässiger Datenbankzustände und -übergänge

- Enddatum ≥ Beginndatum
- Status = ‘in’ ⇒ einen Datensatz in BEHANDLUNG eintragen
- Status = ‘out’ ⇒ Enddatum eintragen in entsprechenden Datensatz
- kein Patient kann auf mehreren Stationen gleichzeitig sein, d.h. falls neuer Datensatz in Behandlung‘ dann prüfen, ob alle Datensätze des gleichen Patienten auch ein gültiges Enddatum haben
- Kapazitätsprüfung: Summe aller zu einem Zeitpunkt auf der Station befindlichen Patienten ≤ Kapazität

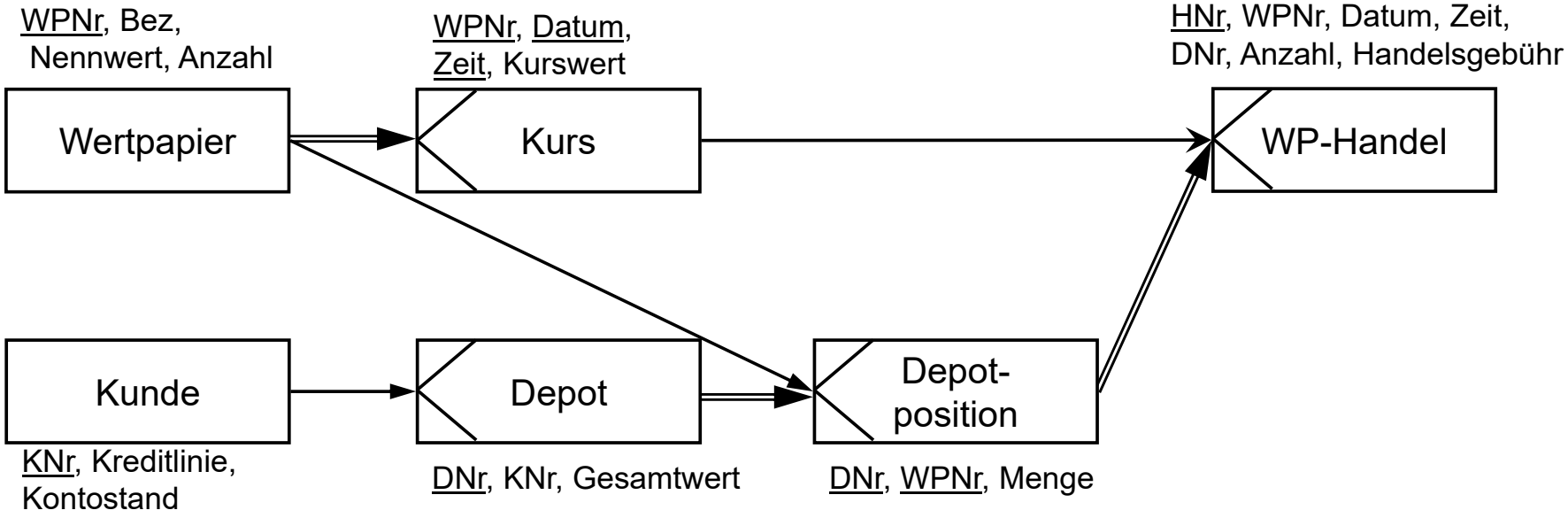
6 Beispiel zu Datenkonsistenz



Implizit unterstellte Konsistenzbedingungen:

- Jeder WP-Handel bezieht sich auf genau eine Depotposition (DNr und WPNr ausfüllen ist obligatorisch).
- Jede Depotposition muss mindestens einmal in WP-Handel vorkommen.
- Löschung des einzigen Datensatzes in WP-Handel impliziert Löschung der Depotposition.
- Löschung eines Depots nur bei Löschung aller Datensätze in Depotposition und WP-Handel.

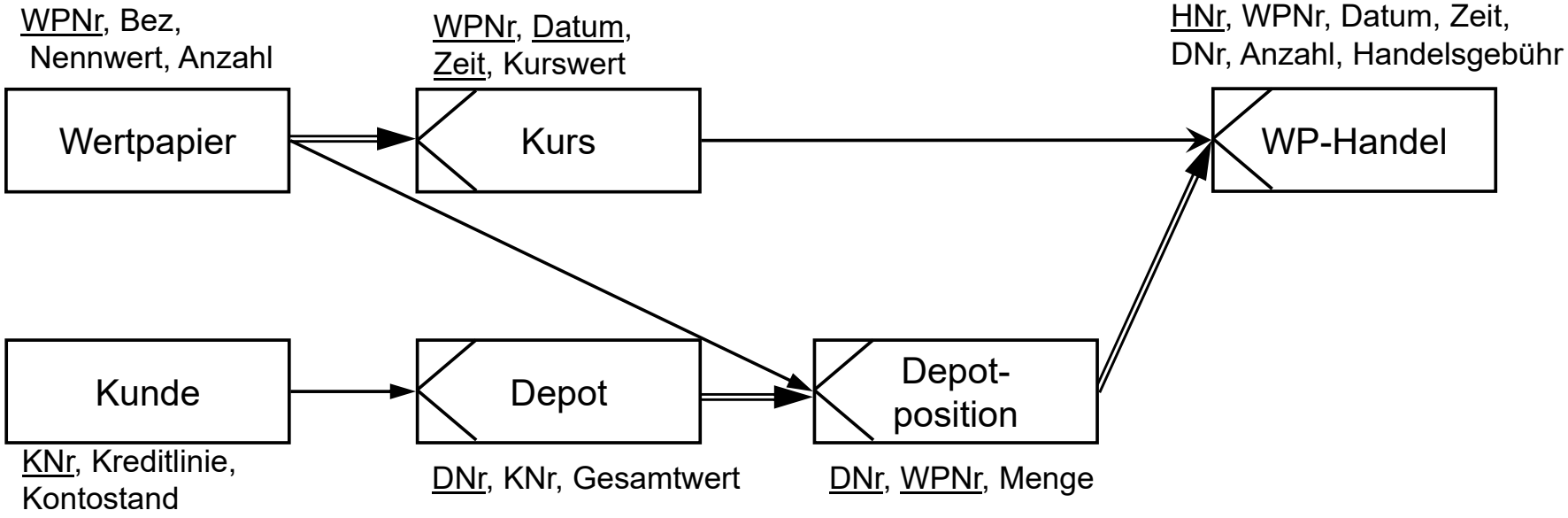
6 Beispiel zu Datenkonsistenz



Konsistenzbedingungen bei Neuaufnahme eines WP-Handel-Datensatzes:

- KB1: HNr muss korrekt angegeben werden (Bildungsgesetz).
- KB2: HNr muss eindeutig sein.
- KB3: WPNr, Datum und Zeit müssen gültig sein.
- KB4: Anzahl ist numerisch.

6 Beispiel zu Datenkonsistenz

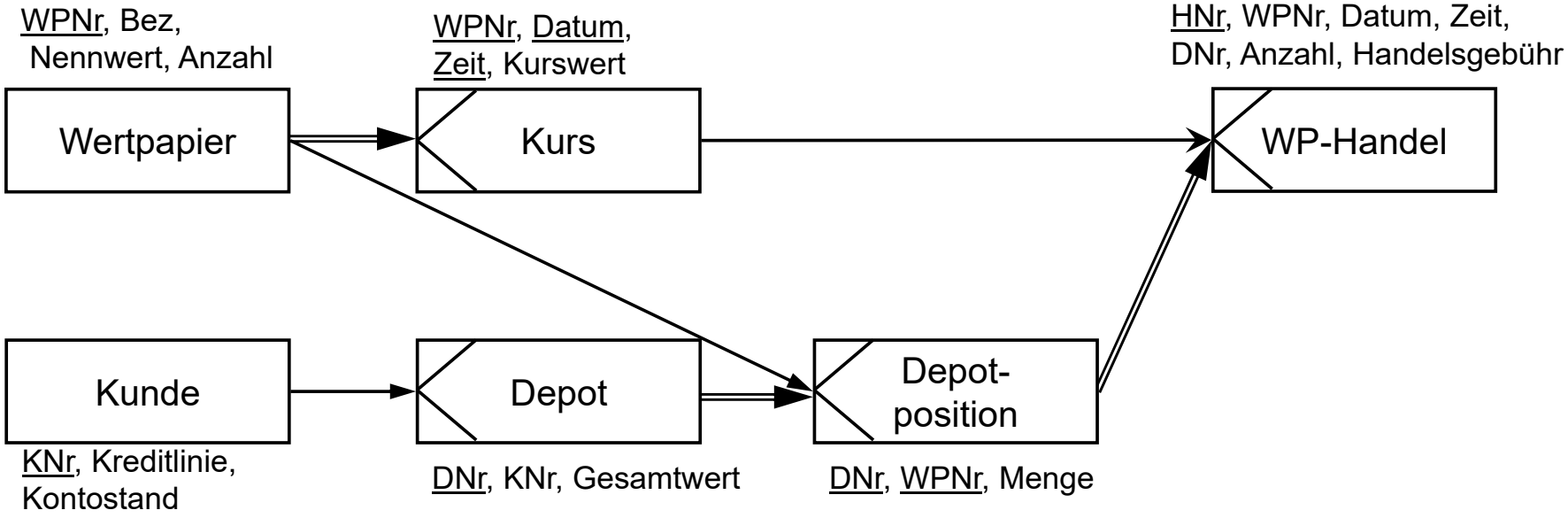


Konsistenzbedingungen bei Neuaufnahme eines WP-Handel-Datensatzes:

- KB5: Handelsgebühr bei Käufen: 0,10 % vom Handelswert
  bei Verkäufen: 0,12 % vom Handelswert

```
CONSTRAINT HG CHECK  (Handelsgebühr=0.10 AND Anzahl>0) OR
                     (Handelsgebühr=0.12 AND Anzahl<0)
```

6 Beispiel zu Datenkonsistenz



Konsistenzbedingungen bei Anlegen eines Datensatzes in WP-Handel:

- KB6: Kreditlinie + Kontostand >= Kurswert x Anzahl
  + Kurswert x |Anzahl| x Handelsgebühr
- KB7: Menge in Depotposition u. Gesamtwert in Depot sind zu aktualisieren
- KB8: Falls Anzahl < 0 muss |Anzahl| <= Menge des betreffenden Wertpapiers und Depots sein
- KB9: WPNr aus Kurs und Depotposition müssen übereinstimmen

6 Transaktion

Transaktion
Folge zusammenhängender DB-Operationen, bei deren Ausführung auf einer konsistenten DB die Konsistenz der DB erhalten bleibt

- Eine Transaktion überführt eine Datenbank von einem konsistenten Zustand in einen anderen konsistenten Zustand.

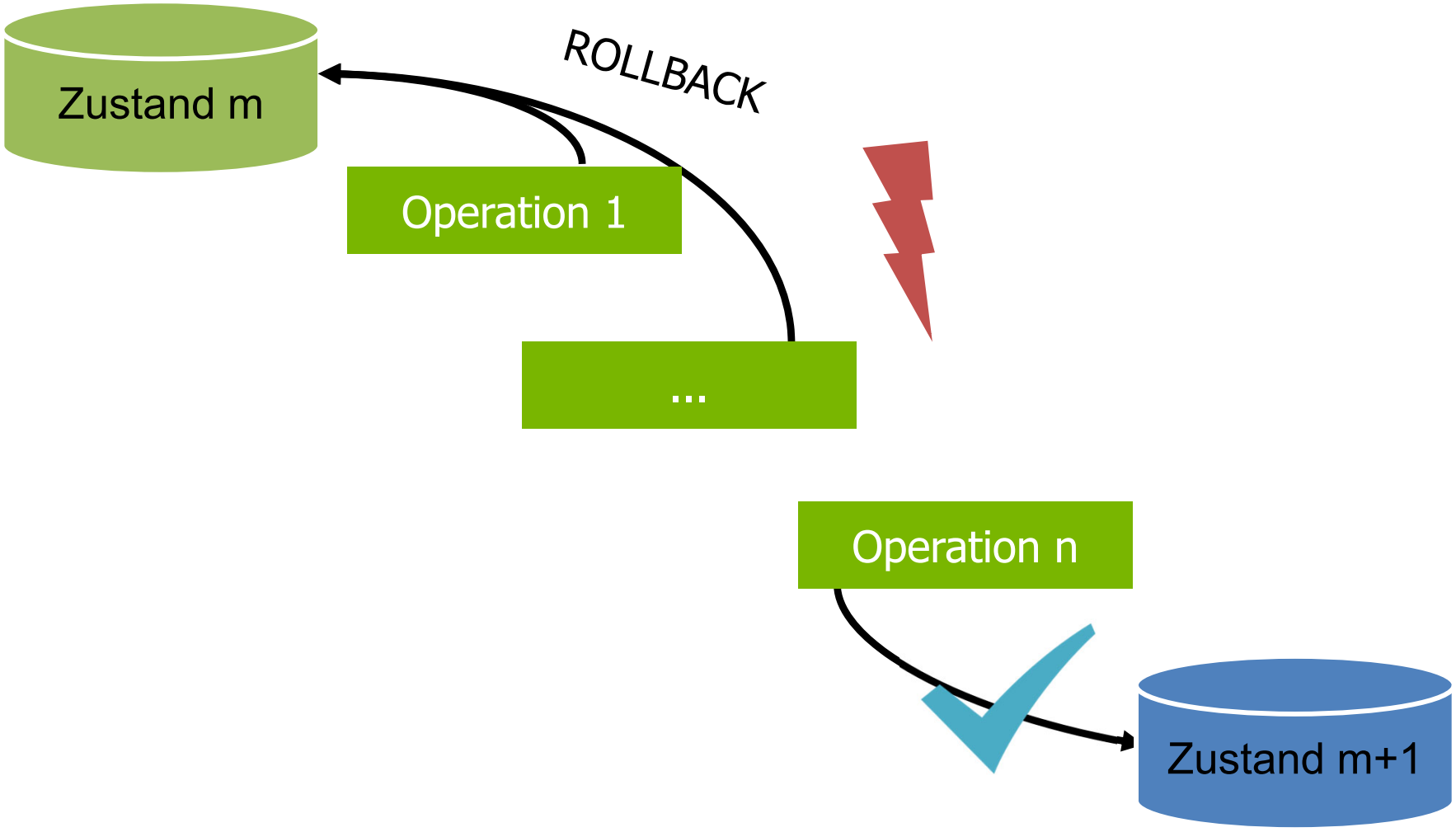
6 Transaktion – Bestandteile

- START TRANSACTION SQL-Befehlssequenz starten
  - SQL-Befehle

Transaktion beenden mit:

- COMMIT zur Übernahme der Änderungen in die DB
- ROLLBACK zum Abbruch der Transaktion

Transaktion – Ablauf



6 ACID-Prinzip zur Transaktionsverwaltung

- **A**tomicity
  Transaktionen sind elementar, nicht unterbrechbar, d.h., sie werden vollständig oder gar nicht ausgeführt.
  („Alles-oder-Nichts-Prinzip“)
- **C**onsistency
  Transaktionen sind konsistenzerhaltend.
- **I**solation
  Benötigte Datenobjekte sind vor anderen Operationen geschützt. Das Ergebnis einer Transaktion wird nicht durch parallel ablaufende Transaktionen beeinflusst
- **D**urability
  Wirkung einer Transaktion ist dauerhaft.

6 Probleme der Datenbanknutzung

- Gleichzeitige Nutzung durch verschiedene Nutzer
- Viele parallele Datenbankoperationen

⇒ Hohe Wahrscheinlichkeit, dass

⇒ Nutzer parallel auf gleiche Daten zugreifen wollen

⇒ Daten schnell Änderungen unterworfen sind

⇒ Ohne weitere Gegenmaßnahmen führt das zu Problemen

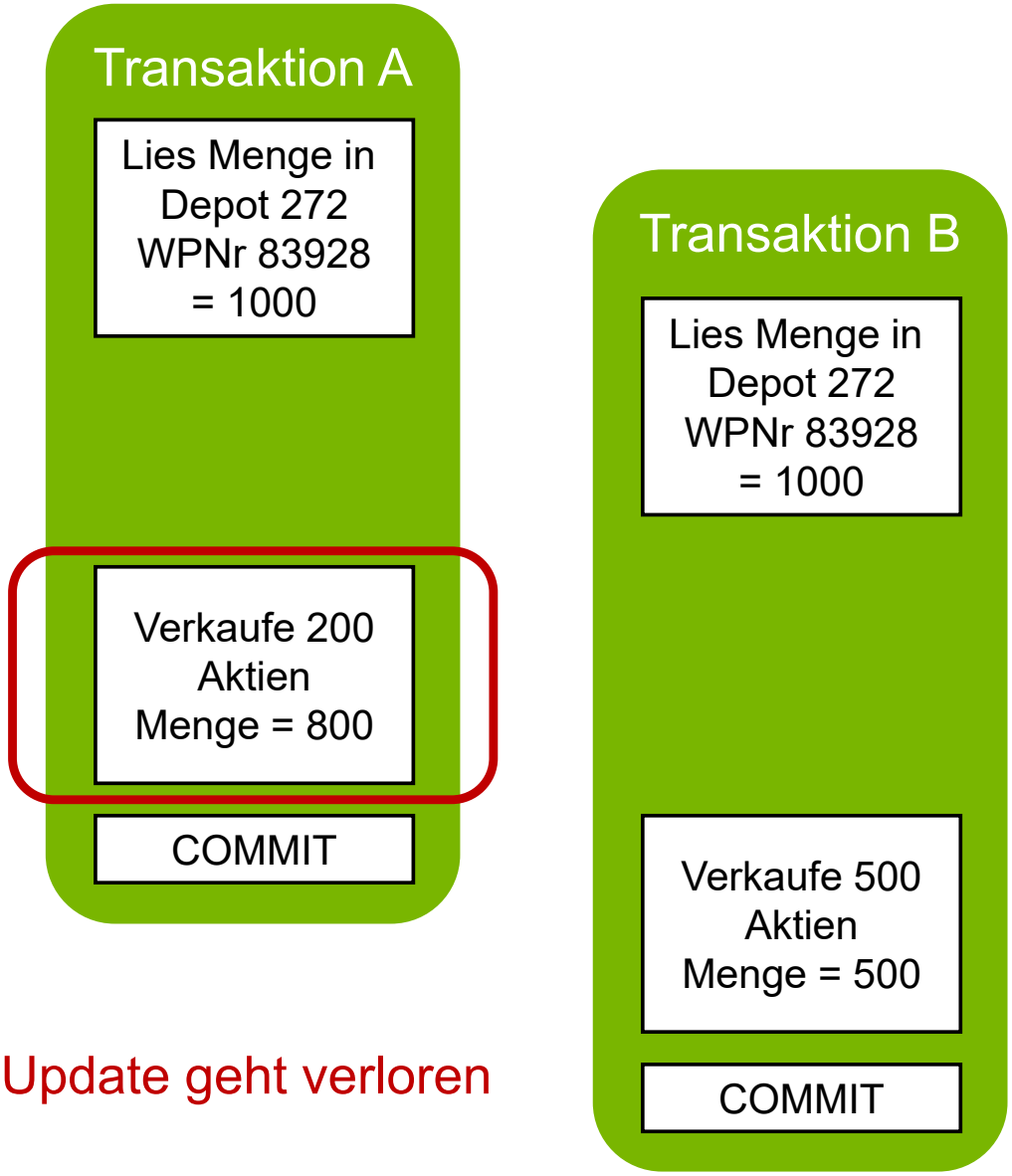
⇒ Lost Update
Transaktion B überschreibt von Transaktion A aktualisierten Wert

⇒ Phantom Read

⇒ Dirty Read
Nutzung eines ungültigen/veralteten Wertes

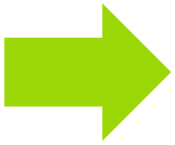
⇒ Non-Repeatable Read
Mehrfaches Auslesen führt zu unterschiedlichen Ergebnissen

6 Lost Update



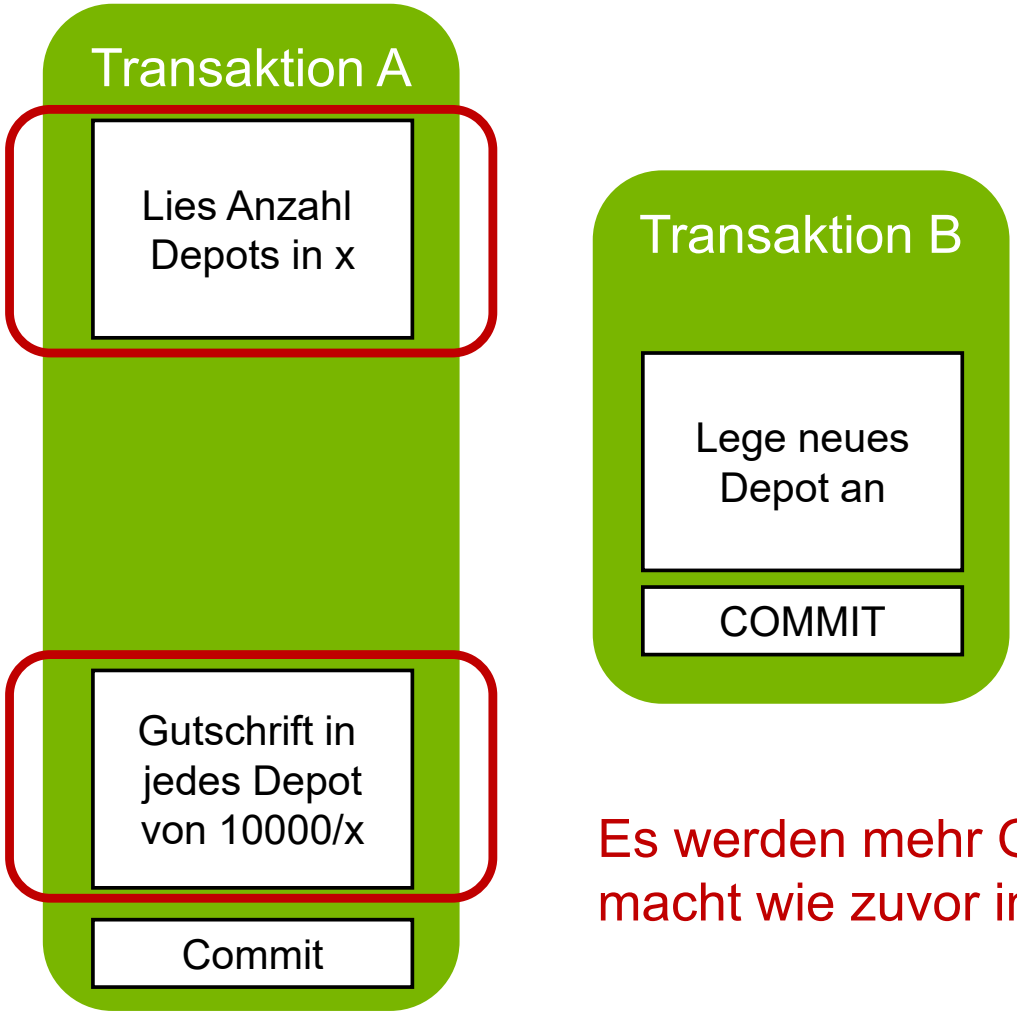
Update geht verloren

Zwei Transaktionen modifizieren parallel denselben Datensatz und nach Ablauf dieser beiden Transaktionen wird nur die Änderung von einer von ihnen übernommen.



Depotbestand 500 statt 300

6 Phantom Read

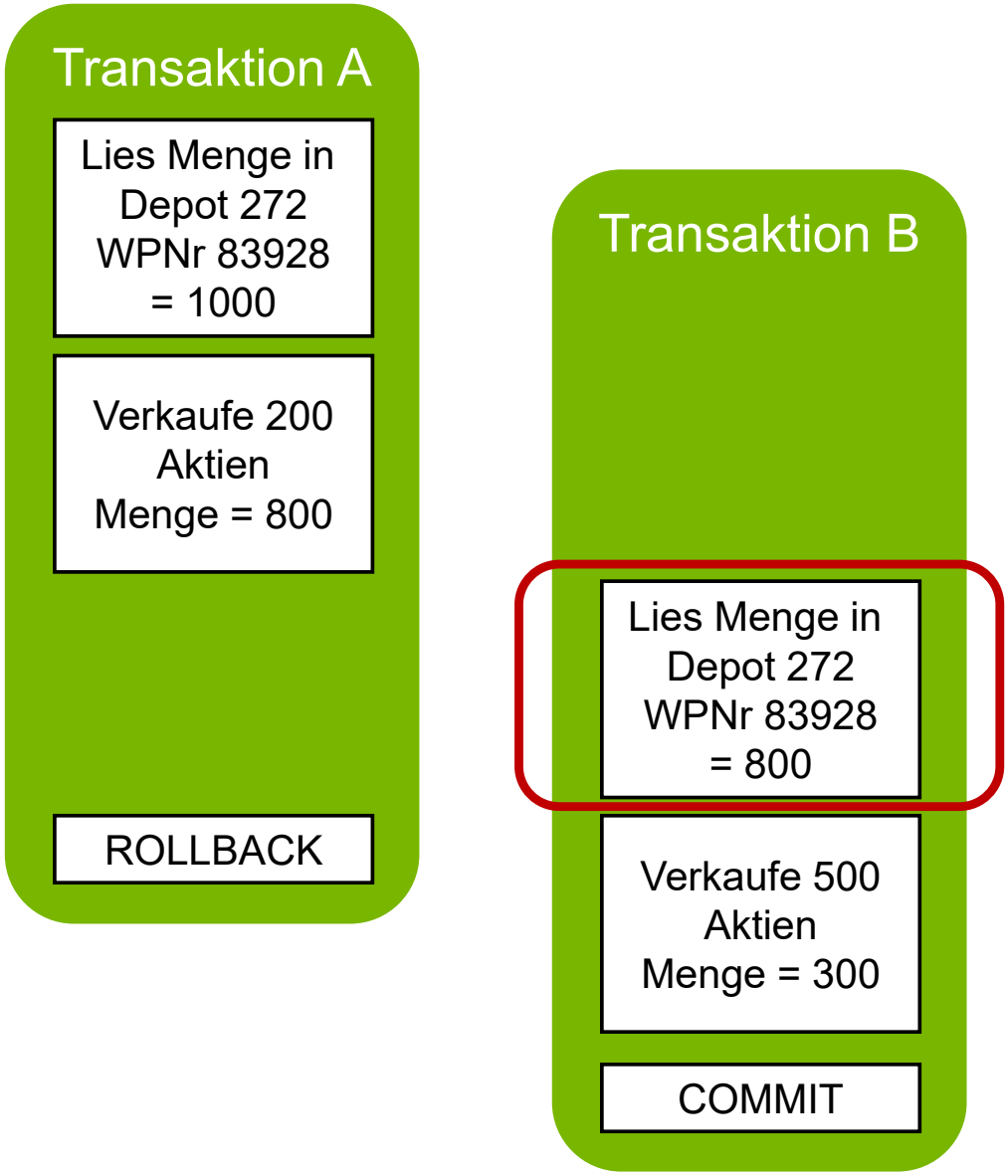


Anzahlen, Updates, etc. beziehen sich während einer Transaktion auf unterschiedliche Datensätze, weil eine andere Transaktion Datensätze hinzugefügt, entfernt oder verändert hat.

Werbeaktion kostet nicht 10000 sondern 10000/x*(x+1)

Es werden mehr Gutschriften (Updates) gemacht wie zuvor in Anzahl x gespeichert wurden.

6 Dirty Read

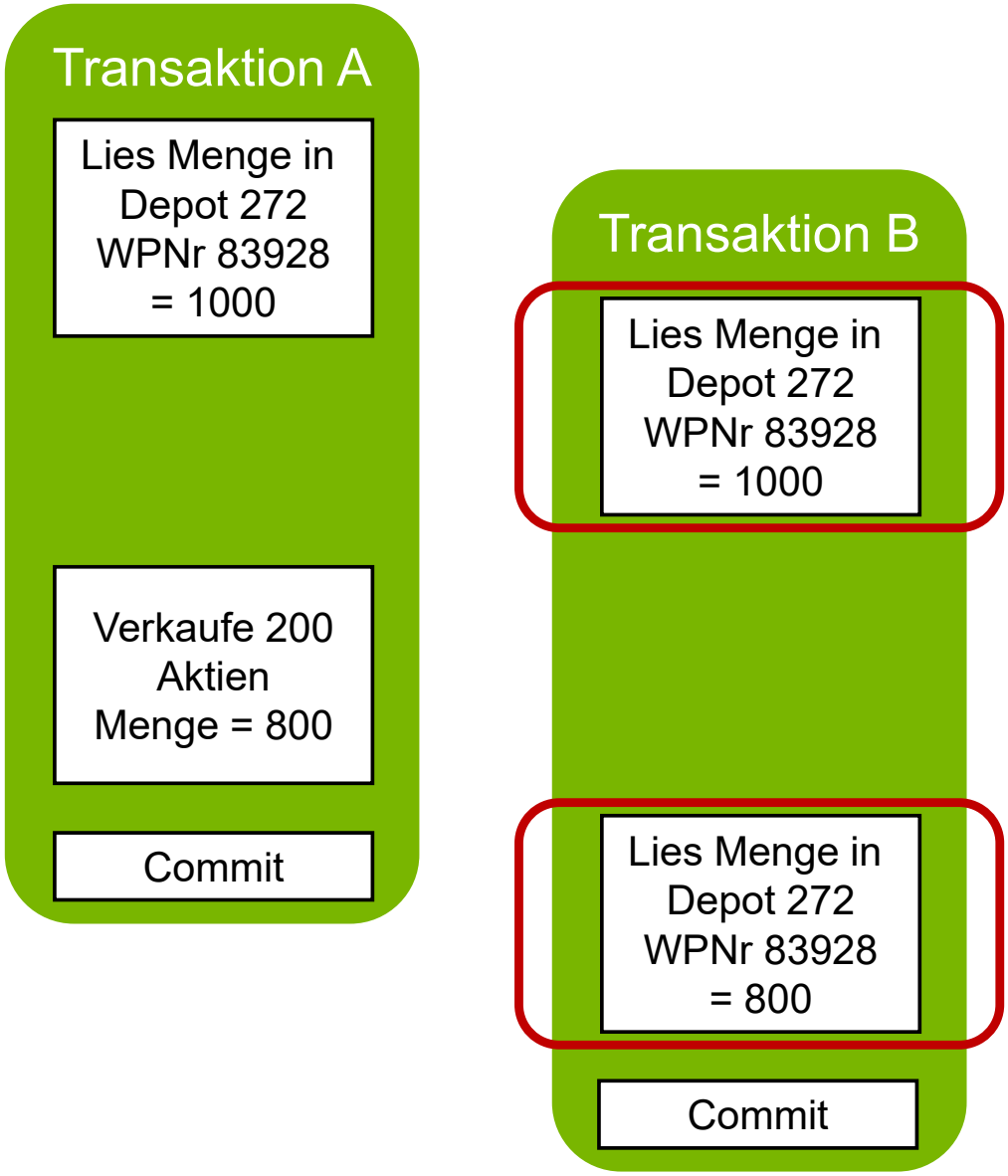


Daten einer noch nicht abgeschlossenen Transaktion werden von einer anderen Transaktion gelesen.

Depotbestand 300 statt 500

Nicht commitete Daten

6 Non-Repeatable Read



Wiederholte Lesevorgänge liefern unterschiedliche Ergebnisse.

Inkonsistente Mengen in Transaktion B

Erst Rückgabewert 1000, dann Wert 800.

6 Serialisierbarkeit

Problem:

- Mehrbenutzerbetrieb unterstützen
- ACID-Prinzip garantieren

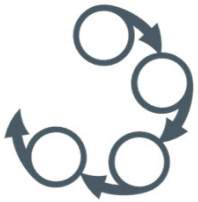
Lösung: Parallele Transaktionen erlauben solange diese korrekt synchronisieren.

Prinzip der Serialisierbarkeit
Ein System paralleler Transaktionen heißt korrekt synchronisiert, wenn es eine serielle Ausführung gibt, die denselben Datenbankzustand erzeugt.

Wann sind Transaktionen korrekt synchronisiert?

- Präzendenzgraph (nächste Folie) besitzt keine Zyklen

6 Präzendenzgraph



Der Präzedenzgraph zu einem Ablauf S ist ein gerichteter Graph mit

(1) den Knoten $T_1, T_2, \ldots T_s$ für jede Transaktion T_i in S

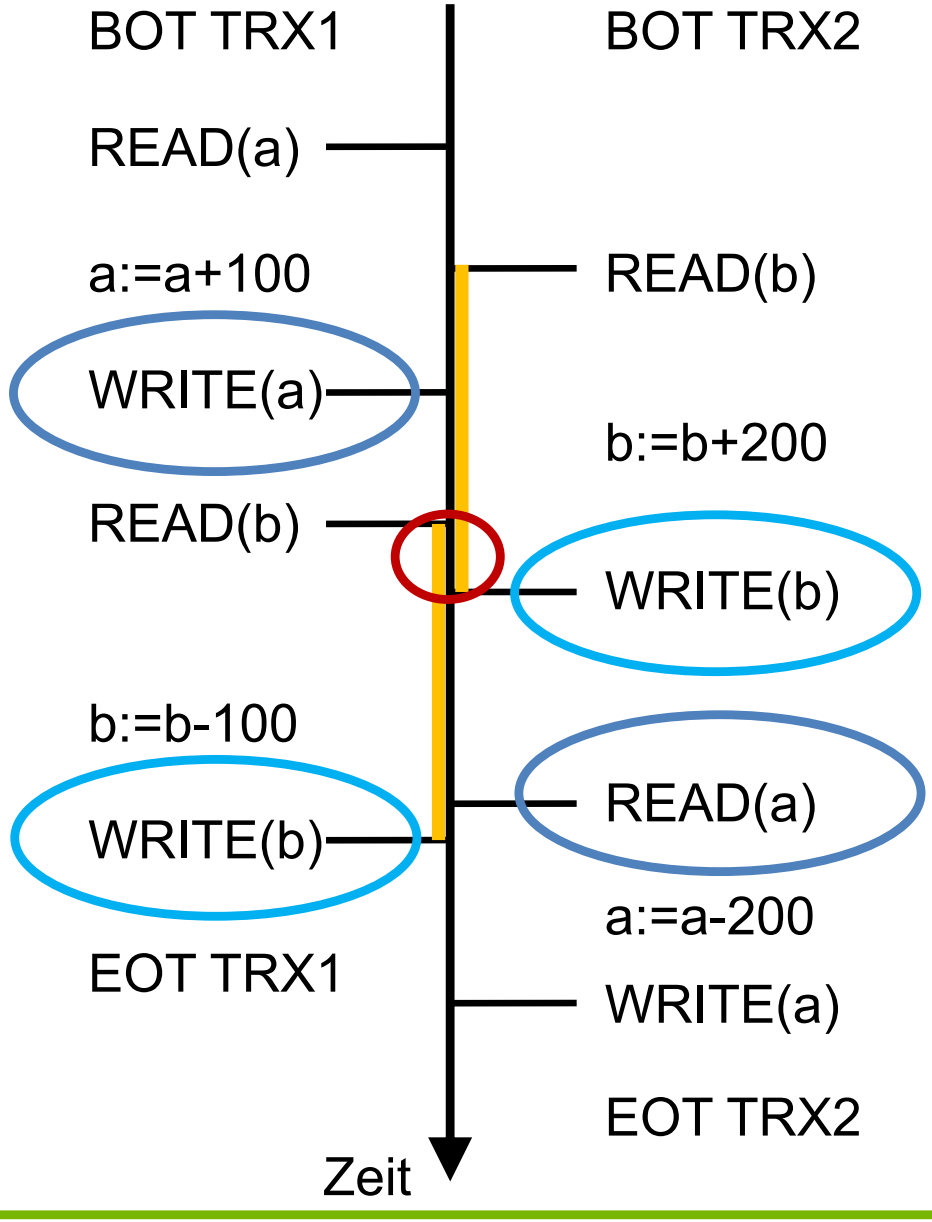
(2) der Kante $T_i \rightarrow T_j$, falls es ein T_i.Read/Write(x) vor T_j.Read/Write(x) gibt und mind. eine Operation ein write ist. (In Konflikt stehende Aktionen)

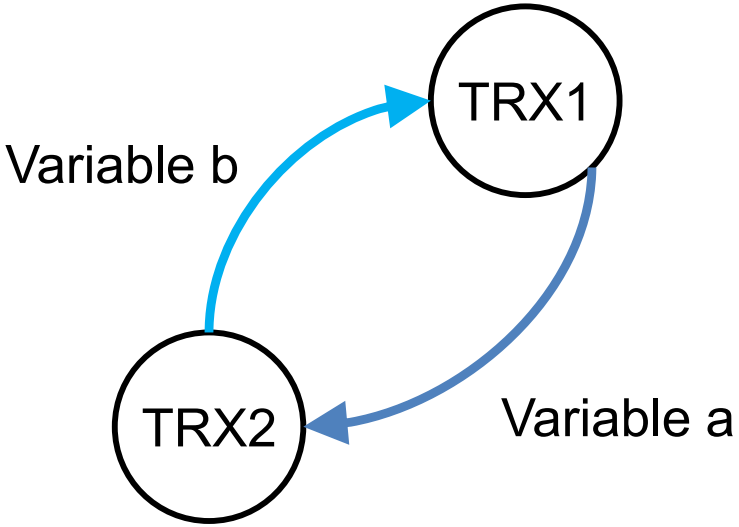
* In Konflikt stehende Aktionen bzgl. Datenobjekt x:

- Betrachte Aktionen, die auf gleiches Datenobjekt x zugreifen
- Mindestens eine Aktion ist ein **write**
- Aktionen gehören zu unterschiedlichen Transaktionen T_i und T_j

Der Bereich, wo gemeinsam auf Variable x zugegriffen wird in T_i bzw. T_j heißt Konfliktbereich.

6 Zugriffssequenz & Präzendenzgraph – Beispiel





Rot: Konfliktbereich bzgl. Variable b

6 Notwendigkeit der Synchronisation

- Zufälligkeiten in den Ergebnissen verhindern
  (späteres Update gewinnt –> zufälliges Update)
- Gegenseitiges Überschreiben von Datenwerten verhindern (siehe Lost Update)
- Vermeidung von Fehlern und Inkonsistenzen in der Datenbasis (siehe weitere Probleme zuvor)

6 Wie erreichen wir eine Synchronisation?

- **Optimistisches Verfahren**
  - Keine Zentral-Instanz
  - Annahme, dass Konflikte zwischen konkurrierenden Transaktionen selten vorkommen
  - Arbeiten auf „Kopien“
  - Validierung der Transaktion (Test, ob Zyklus im Präzedenzgraph)
  - ➢ Bei konfliktfreiem Abschluss: Übertragung der Kopien

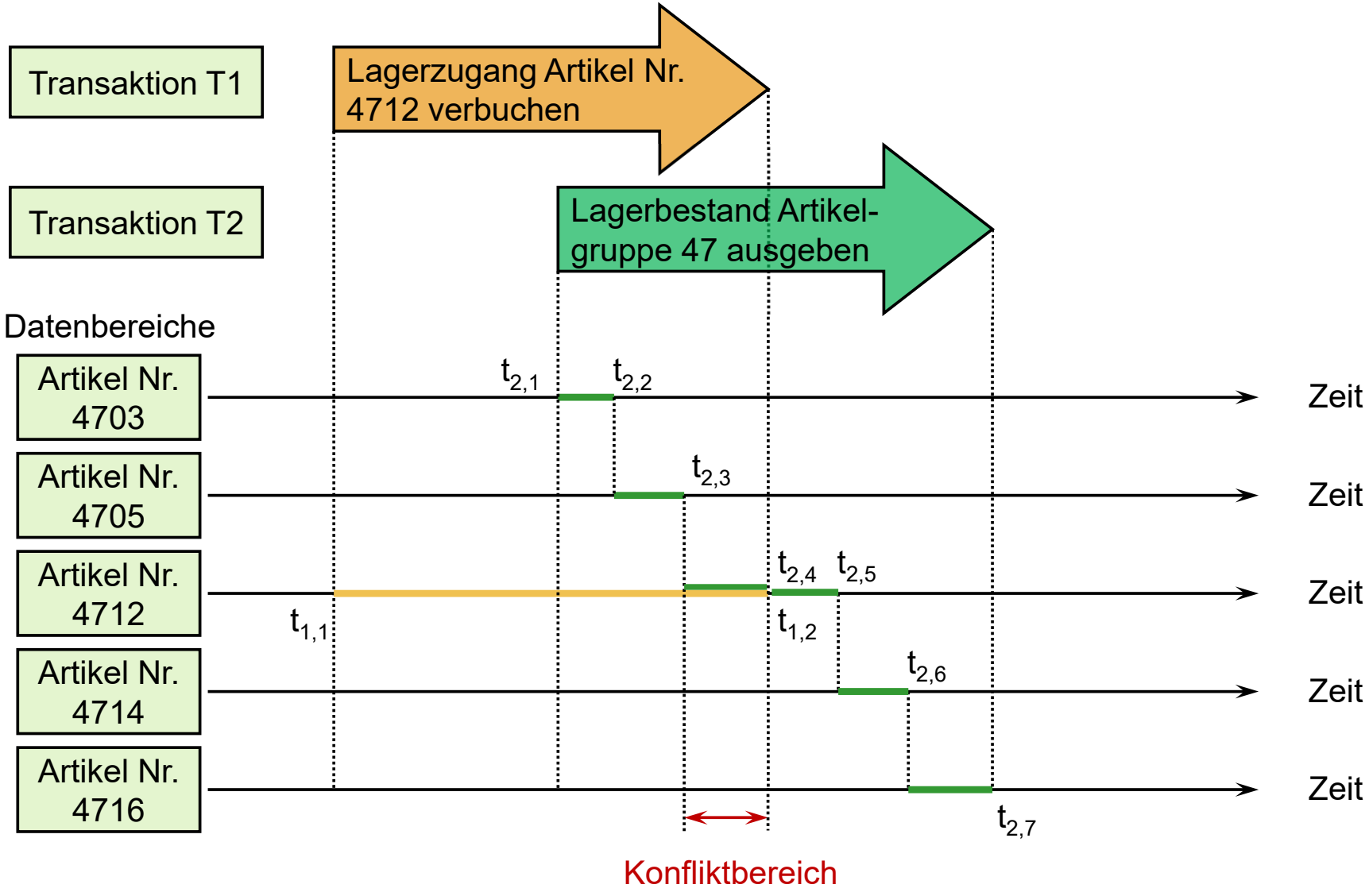
- **Pessimistische Verfahren / Sperrverfahren**
  - Transaktionen mit Zugriffssperren versehen
    => Kann nur eine Transaktion nach der anderen auf eine Tabelle zugreifen, dann gibt es keine Zyklen im Präzendenzgraphen.
  - Exklusivität für Datenbereiche => Sperrprotokoll nötig
  - Mehrere Verfahrensvarianten

6 Transaktion – Bestandteile

- Start der Transaktion
  - **Sperren auf DB-Tabellen setzen**
  - SQL-Befehlssequenz
  - **Sperren aufheben**
  - COMMIT zur Übernahme der Änderungen in die DB
  - ROLLBACK zum Abbruch der Transaktion
- Ende der Transaktion

Wann immer eine Transaktion T1 eine Sperre auf eine Tabelle x setzen will, wird überprüft, ob schon eine andere Transaktion T2 eine Sperre auf x gesetzt hat. Falls ja, dann wartet T1, bis Freigabe von T2 auf x erfolgt ist.

6 Synchronisationsnotwendigkeit



6 Sperrverfahren

- Sperrobjekte
- Sperrmodi
  - Lesesperren (Shared Locks = kein Schreiben von anderen Transaktionen erlaubt)
  - Schreibsperren (Exclusive Locks = kein Zugriff von anderen Transaktionen erlaubt)
- Sperrprotokoll
  - Zweiphasen-Sperrprotokoll
  - Varianten des Zweiphasen-Sperrprotokolls

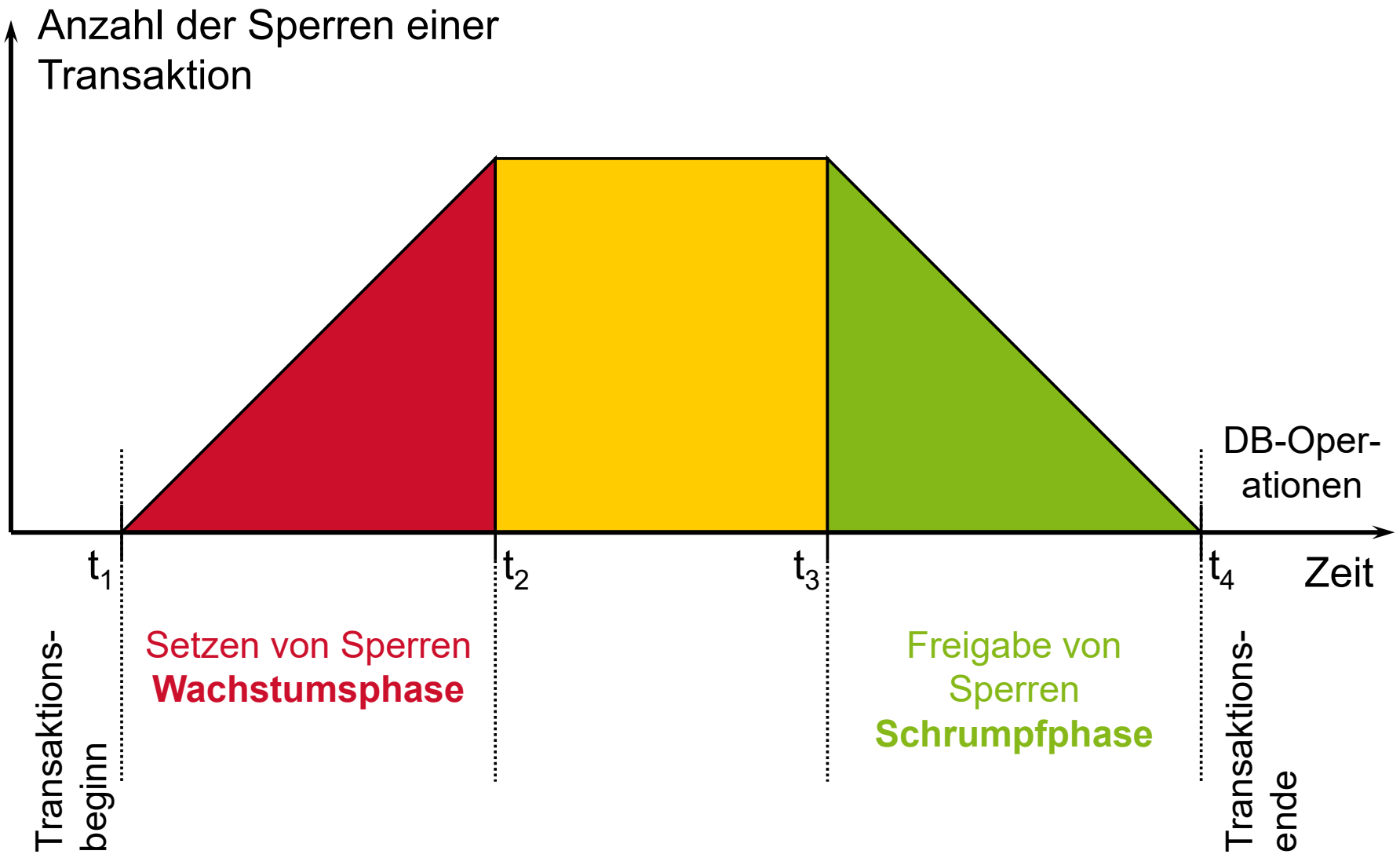
6 Zweiphasen-Sperrprotokoll

1. Jeder Bereich, auf den eine Transaktion zugreift, muss gesperrt werden.

2. Es ist der zulässige, am wenigsten einschränkende Sperrmodi zu wählen.

3. Die Sperre für einen Bereich darf von der Transaktion erst dann freigegeben werden, wenn der Bereich nicht mehr bearbeitet werden muss und keine weiteren Sperren mehr gesetzt werden müssen (auch für andere Bereiche)
   - $\Rightarrow$ Keine Vermischung von Sperren und Freigeben.
   - $\Rightarrow$ Jeder Bereich darf von einer Transaktion nur einmal gesperrt werden.

6 Zweiphasen-Sperrprotokoll



6 Technische Realisierung des Protokolls

- Alle DB-Änderungen werden in einem Logfile protokolliert
  - Transaction ID (TID)
  - Begin of Transaction (BOT)
  - Before-Images
    TID OID (Object Id) Objektwert vor Änderung
  - After-Images
    TID OID (Object Id) Objektwert nach Änderung
  - End of Transaction (EOT) / COMMIT
  - ROLLBACK
- Erst wird Logfile beschrieben, dann die DB-Änderung durchgeführt
- Mit Hilfe des Logfiles kann jede Transaktion
  - rückgängig gemacht (Rollback, Undo)
  - wiederholt (Rollforward, Redo)

  werden

Verarbeitung von Transaktionen

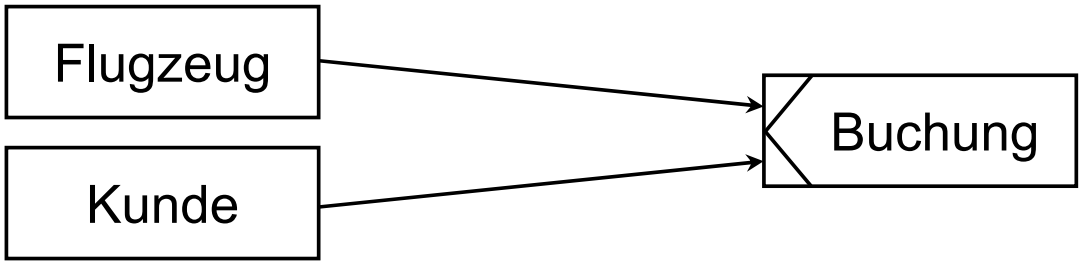
- Direktes Schreiben in DB langsam

⇒ Arbeit auf Zwischenspeicher

⇒ Gewährleistung der Übernahme in DB über Checkpoints

  - Übernahme aller abgeschlossenen, aber nicht physisch übernommenen Transaktionen vom Cache in DB
  - Vermerk des Checkpoints im Logfile
  - Ausführung des Checkpoints, wenn
    - Logfile ausreichend gefüllt
    - DB idle
    - vordefinierte Zeit verstrichen

6 Beispiel: Flugzeug und Buchung (1)



FLUGZEUG				
FLZNr	FlugNr	AbflZeit	Sitzplätze	FreieSitzplätze

BUCHUNG					
BuNr	FLZNr	FlugNr	AbflZeit	KNR	Plätze

Kunde					
KNr	Name	Vorname	Strasse	PLZ	Ort

6 Beispiel: Flugzeug und Buchung (2)

Aktueller DB-Stand:

FLUGZEUG				
FLZNr	FlugNr	AbflZeit	Sitzplätze	FreieSitzplätze
375	288	12:50	440	2

Gewünschte Buchungen:

BUCHUNG					
BuNr	FLZNr	FlugNr	AbflZeit	KNR	Plätze
4967	375	288	12:50	1645	1
4968	375	288	12:50	9542	2

Beispiel: Flugzeug und Buchung (3)

Transaktion zum Buchen von Flügen (Pseudocode)

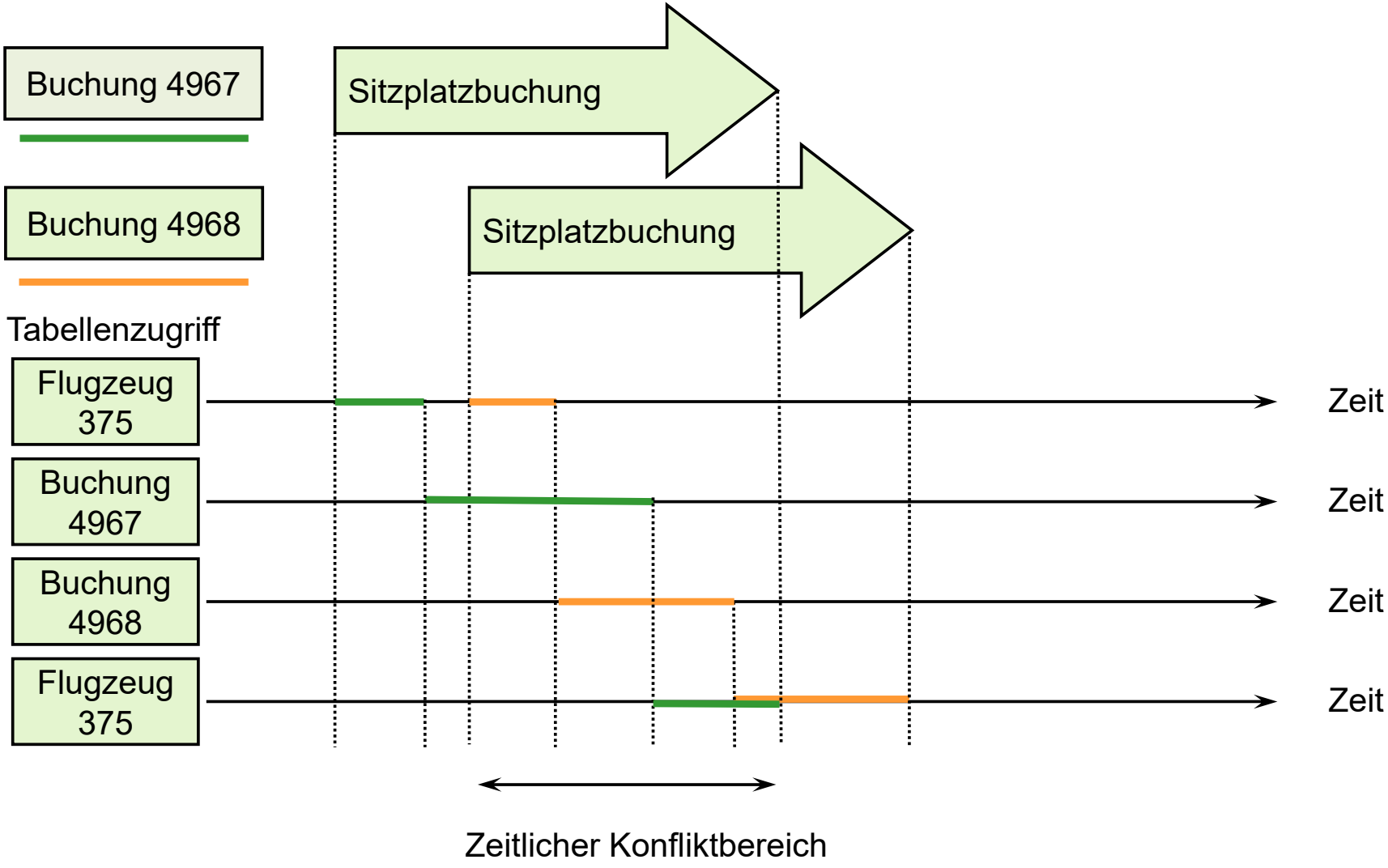
```
TRANSACTION Flugbuchung (FLZNr, FlugNr, AbflZeit, KNR, Plätze)
BEGIN
    EXCLUSIVELOCK(Flugzeug);
    EXCLUSIVELOCK(Buchung);
    IF Flugzeug(FLZNr, FlugNr, AbflZeit).FreieSitzplätze >= Plätze
    THEN
            INSERT INTO Buchung VALUES (BuNr, FLZNr, FlugNr, AbflZeit, KNR, Plätze);
            UPDATE Flugzeug
            SET FreieSitzplätze = FreieSitzplätze – Plätze
            WHERE (Flugzeug.FLZNr, Flugzeug.FlugNr, Flugzeug.AbflZeit) = (FLZNr, FlugNr, AbflZeit);
     ELSE
            OUTPUT ('Nur noch Flugzeug (FLZNr, FlugNr, AbflZeit).FreieSitzplätze vorhanden');
    UNLOCK(Buchung);
    UNLOCK(Flugzeug);
ENDTRANSACTION;
```

FLUGZEUG				
FLZNr	FlugNr	AbflZeit	Sitz-plätze	Freie Sitzplätze

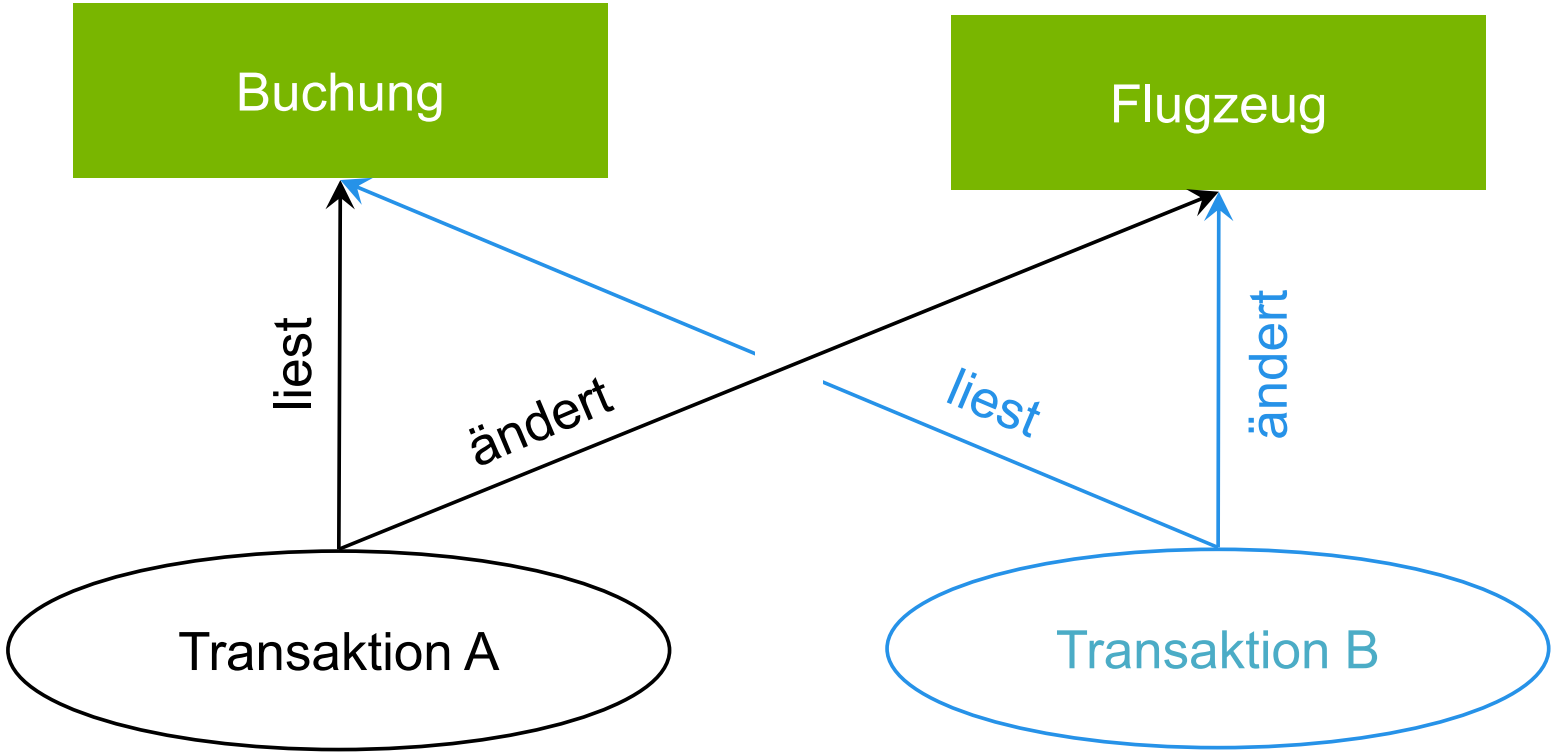
BUCHUNG					
BuNr	FLZNr	FlugNr	Abfl Zeit	KNR	Plätze

KUNDE					
KNr	Name	Vor-name	Strasse	PLZ	Ort

6 Beispiel: Flugzeug und Buchung (4)



6 Zweites Beispiel paralleler Transaktionen



Verringerung der freien Sitzplätze um die gebuchte Anzahl aus Buchung 4967

Ausgabe augenblicklich freier Plätze und Buchungsmenge

6 Pseudocode zum zweiten Beispiel (1)

Ablaufplan 1

```
A   TRANSACTION A (375, 288, 12:50, 1645, 1);
A   BEGIN
A      EXCLUSIVELOCK(Flugzeug);
A      EXCLUSIVELOCK(Buchung);
A      IF Flugzeug(375, 288, 12:50).FreieSitzplätze >= 1
A      THEN
A            INSERT INTO Buchung VALUES (4967, 375, 288, 12:50, 1645, 1);
A            UPDATE Flugzeug
A            SET FreieSitzplätze = FreieSitzplätze – 1
A            WHERE (Flugzeug.FLZNr, Flugzeug.FlugNr, Flugzeug.AbflZeit) = (375, 288, 12:50);
A      ELSE
A            OUTPUT ('Nur noch Flugzeug (375, 288, 12:50).FreieSitzplätze vorhanden');
A      UNLOCK(Buchung);
A      UNLOCK(Flugzeug);
A   ENDTRANSACTION A;
B   TRANSACTION B
B   BEGIN
B      SHAREDLOCK(Buchung);
B      READ(Plätze);
B      SHAREDLOCK(Flugzeug);
B      READ(FreieSitzplätze);
B      OUTPUT(Plätze, FreieSitzplätze);
B      UNLOCK(Buchung);
B      UNLOCK(Flugzeug);
B   ENDTRANSACTION B;
```

Keine Zyklen im Präzendenzgraph
➔ Korrekt synchronisiert.

6 Pseudocode zum zweiten Beispiel (2)

Ablaufplan 2

```
B   TRANSACTION B
B   BEGIN
B      SHAREDLOCK(Buchung);
B      READ(Plätze);
B      SHAREDLOCK(Flugzeug);
B      READ(FreieSitzplätze);
B      OUTPUT(Plätze, FreieSitzplätze);
B      UNLOCK(Buchung);
B      UNLOCK(Flugzeug);
B   ENDTRANSACTION B;
A   TRANSACTION A (375, 288, 12:50, 1645, 1);
A   BEGIN
A      EXCLUSIVELOCK(Flugzeug);
A      EXCLUSIVELOCK(Buchung);
A      IF Flugzeug(375, 288, 12:50).FreieSitzplätze >= 1
A      THEN
A             INSERT INTO Buchung VALUES (4967, 375, 288, 12:50, 1645, 1);
A             UPDATE Flugzeug
A             SET FreieSitzplätze = FreieSitzplätze – 1
A             WHERE (Flugzeug.FLZNr, Flugzeug.FlugNr, Flugzeug.AbflZeit) = (375, 288, 12:50);
A      ELSE
A             OUTPUT ('Nur noch Flugzeug (375, 288, 12:50).FreieSitzplätze vorhanden');
A      UNLOCK(Buchung);
A      UNLOCK(Flugzeug);
A   ENDTRANSACTION A;
```

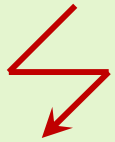
Keine Zyklen im Präzendenzgraph
➔ Korrekt synchronisiert.

6 Pseudocode zum zweiten Beispiel (3)

Ablaufplan 3

```
A   TRANSACTION A (375, 288, 12:50, 1645, 1);
A   BEGIN
B   TRANSACTION B
B   BEGIN
B       SHAREDLOCK(Buchung);
A       EXCLUSIVELOCK(Flugzeug);
A       EXCLUSIVELOCK(Buchung);          A wartet
B       READ(Plätze);
B       SHAREDLOCK(Flugzeug);            B wartet
B       READ(FreieSitzplätze);
B       OUTPUT(Plätze, FreieSitzplätze);
B       UNLOCK(Buchung);
B       UNLOCK(Flugzeug);
B   ENDTRANSACTION B;
A       IF Flugzeug(375, 288, 12:50).FreieSitzplätze >= 1
A       THEN
A               INSERT INTO Buchung VALUES (4967, 375, 288, 12:50, 1645, 1);
A               UPDATE Flugzeug
A               SET FreieSitzplätze = FreieSitzplätze – 1
A               WHERE (Flugzeug.FLZNr, Flugzeug.FlugNr, Flugzeug.AbflZeit) = (375, 288, 12:50);
A       ELSE
A               OUTPUT ('Nur noch Flugzeug (375, 288, 12:50).FreieSitzplätze vorhanden');
A       UNLOCK(Buchung);
A       UNLOCK(Flugzeug);
A   ENDTRANSACTION A;
```

Keine Zyklen im Präzendenzgraph
➔ Korrekt synchronisiert.



Aber Deadlock!!

6 Pseudocode zum zweiten Beispiel (4)

Ablaufplan 4

```
A   TRANSACTION A (375, 288, 12:50, 1645, 1);
A   BEGIN
B   TRANSACTION B
B   BEGIN
B      SHAREDLOCK(Buchung);
B      READ(Plätze);
B      SHAREDLOCK(Flugzeug);
B      READ(FreieSitzplätze);
B      OUTPUT(Plätze, FreieSitzplätze);
B      UNLOCK(Buchung);
B      UNLOCK(Flugzeug);
A      EXCLUSIVELOCK(Flugzeug);
A      EXCLUSIVELOCK(Buchung);
B   ENDTRANSACTION B;
A      IF Flugzeug(375, 288, 12:50).FreieSitzplätze >= 1
A      THEN
A        UPDATE Flugzeug
A              SET FreieSitzplätze = FreieSitzplätze – 1
A              WHERE (Flugzeug.FLZNr, Flugzeug.FlugNr, Flugzeug.AbflZeit) = (375, 288, 12:50);
A              INSERT INTO Buchung VALUES (4967, 375, 288, 12:50, 1645, 1);
A      ELSE
A              OUTPUT ('Nur noch Flugzeug (375, 288, 12:50).FreieSitzplätze vorhanden');
A      UNLOCK(Buchung);
A      UNLOCK(Flugzeug);
A   ENDTRANSACTION A;
```

Transaction A muss Lock nach hinten verschieben.

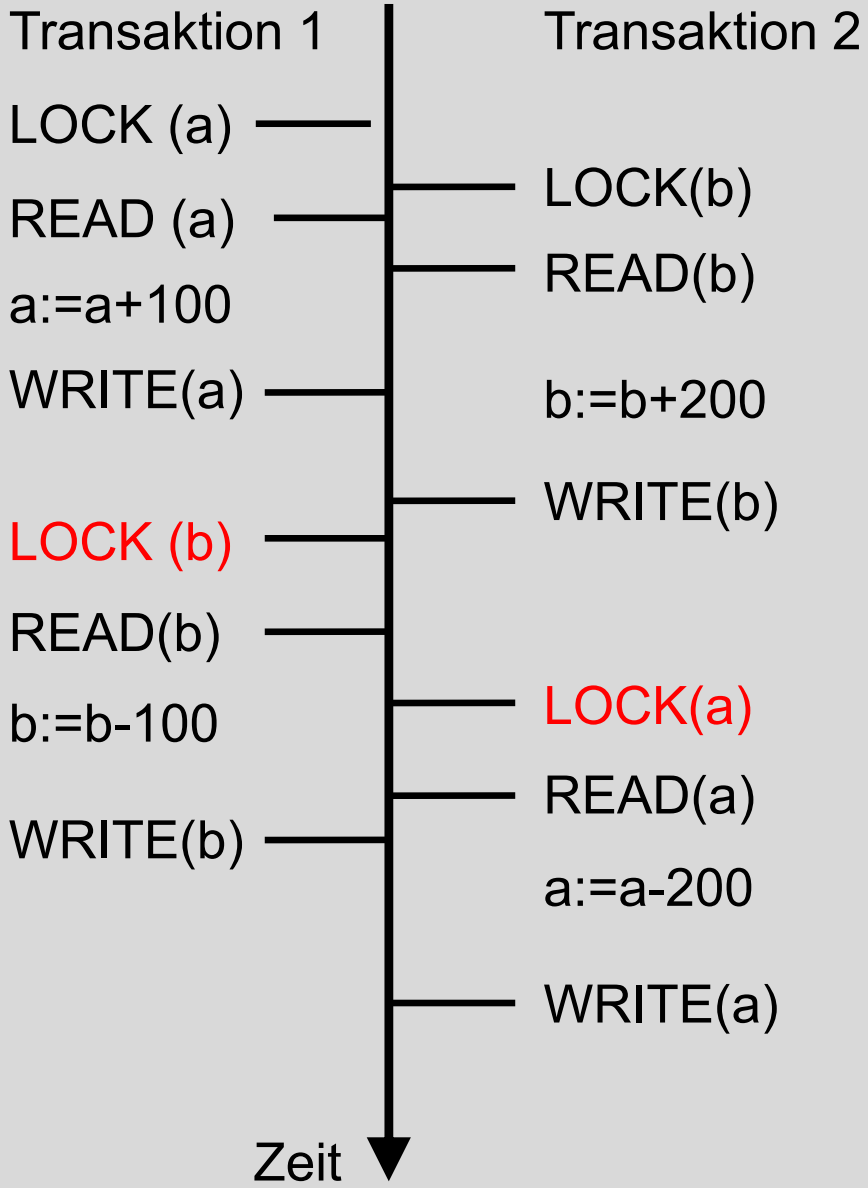
Allgemeine Regel?

6 Deadlock

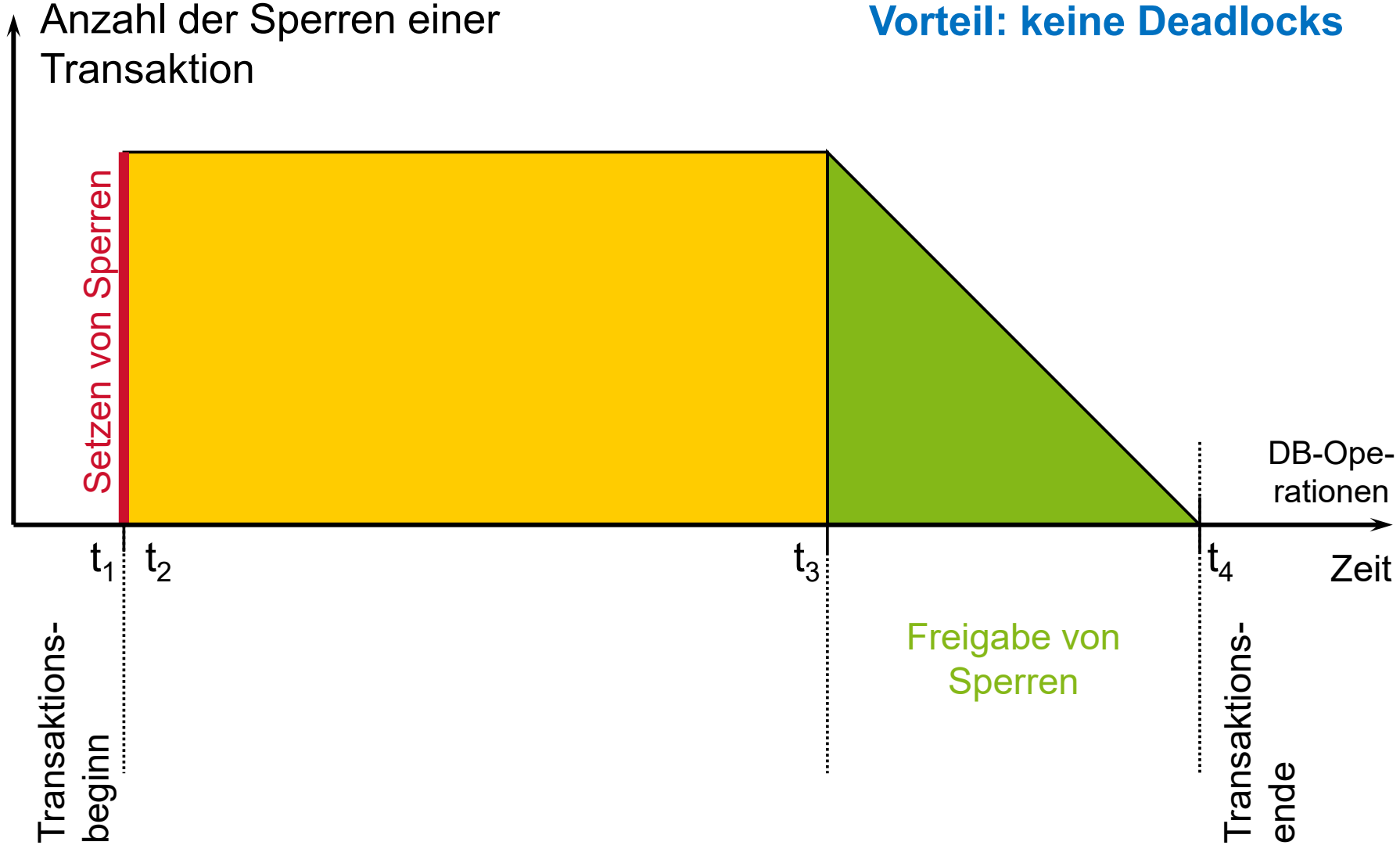
Deadlock = Gegenseitiges Warten
Situation, bei der parallel ausgeführte Transaktionen wechselseitig auf das Aufheben gesetzter Sperren warten und von sich aus gemäß des Sperrprotokolls nicht in der Lage sind, den Wartezustand aufzulösen

- Lösungsstrategien
  - Entdeckung und Behebung => Präzendenzgraph nutzen
  - Vermeidung

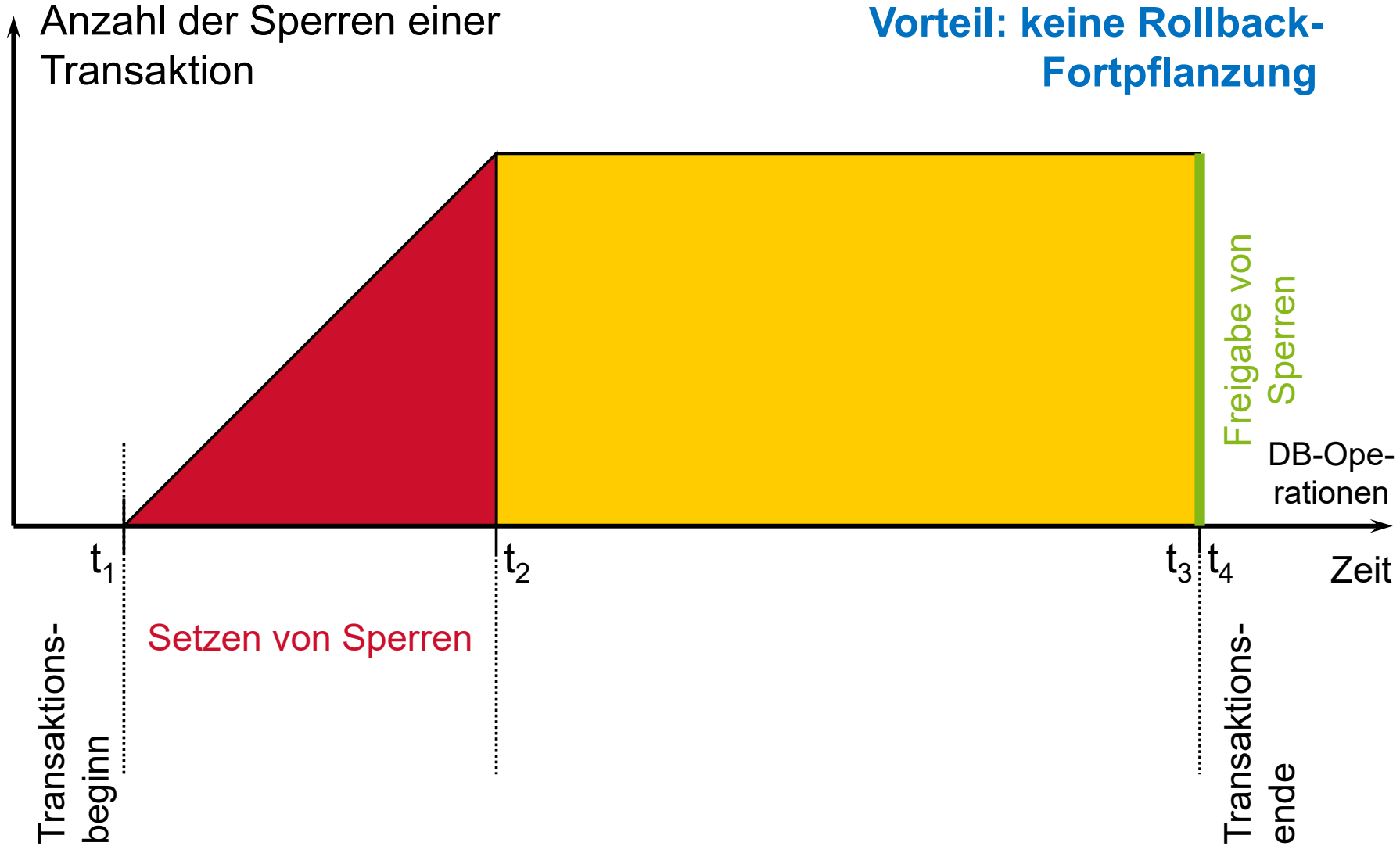
6 Deadlock – Vereinfachtes Beispiel



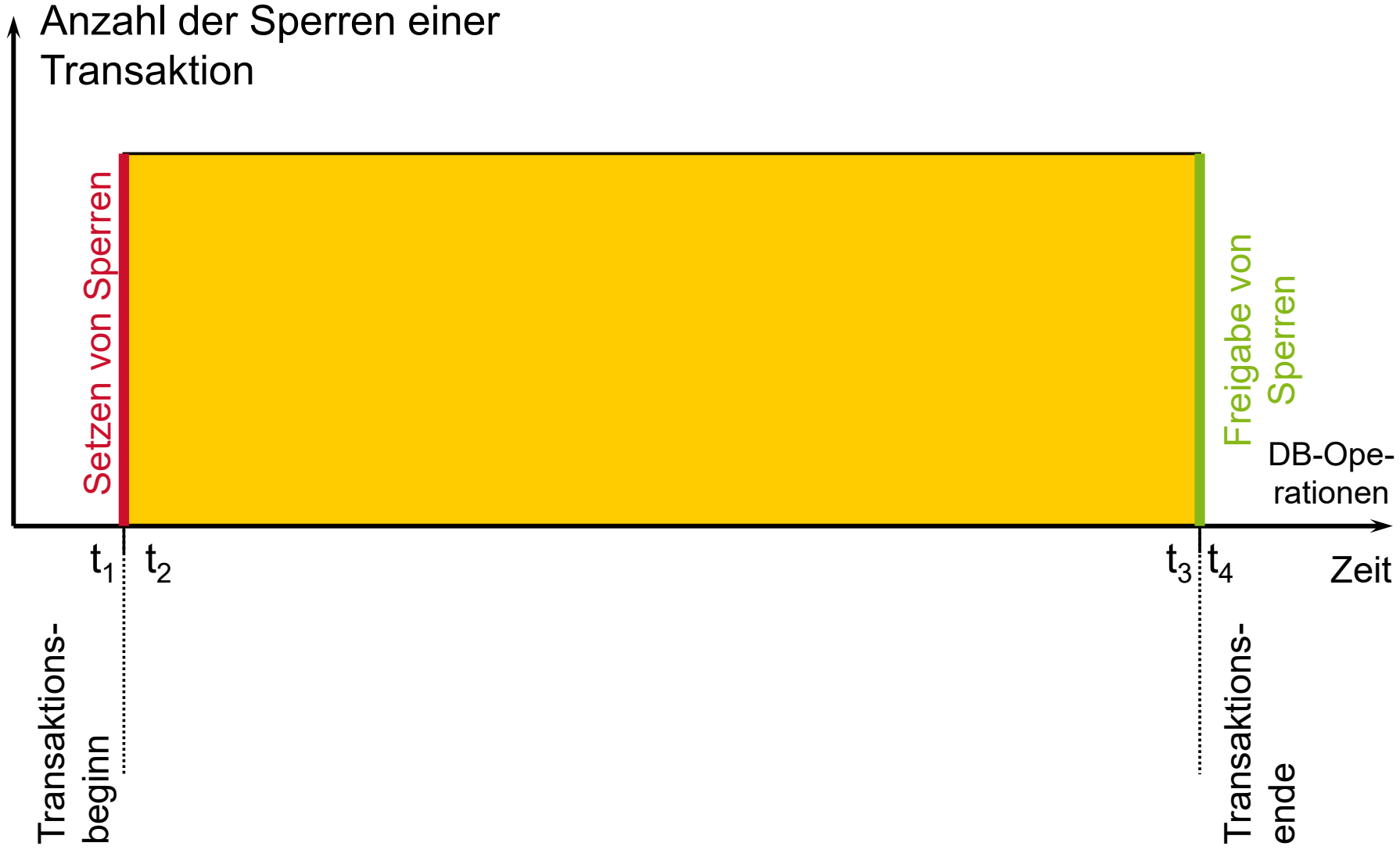
6 Preclaiming



6 Sperren bis zum Transaktionsende



6 Striktes Zweiphasen-Sperrprotokoll



6 Datenrekonstruktion

- „Reparatur“ undefinierter Datenbankzustände
- Wiederherstellen eines korrekten Datenbankzustandes nach Fehlerauftreten

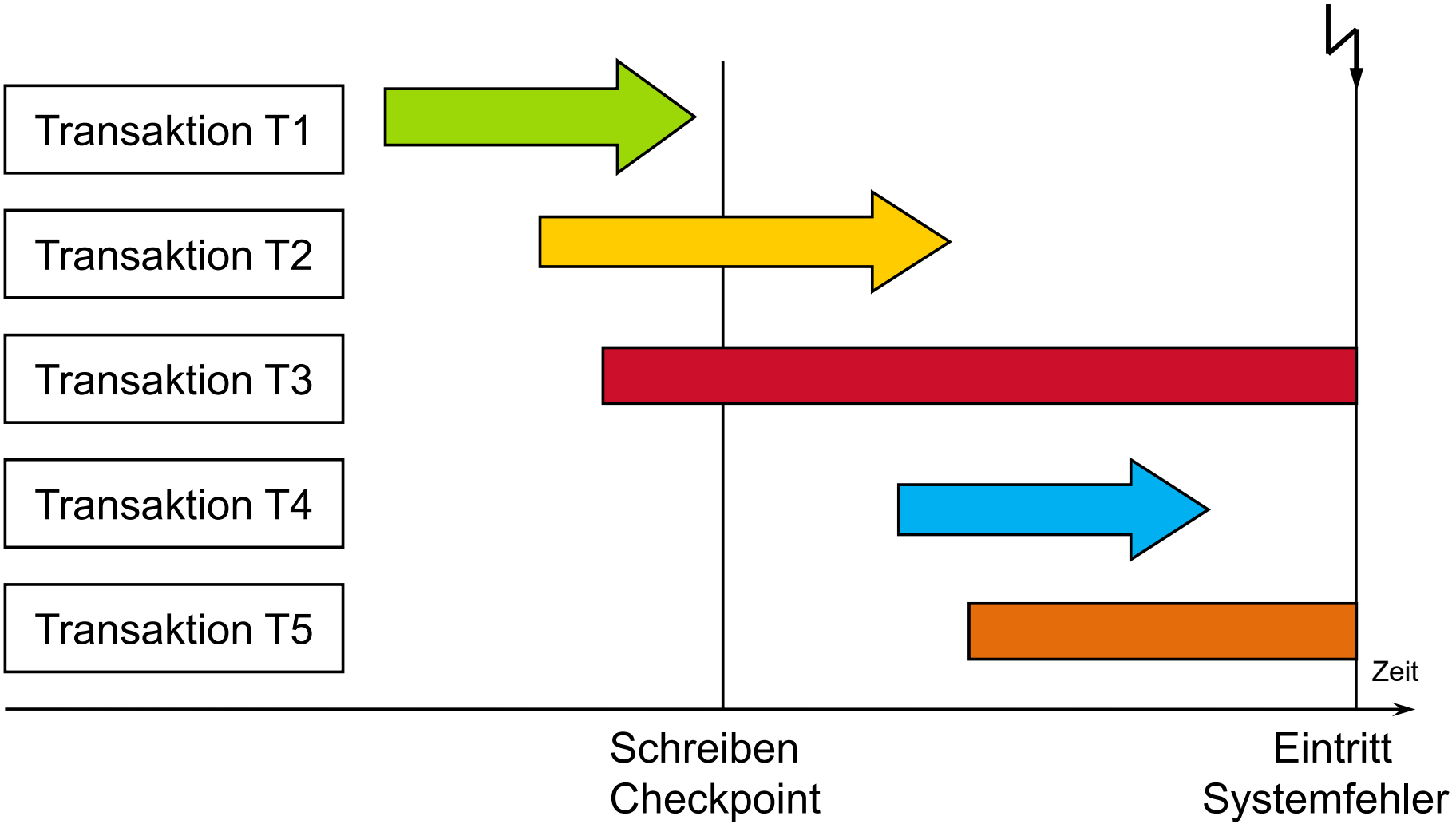
- Behebung von Transaktionsfehlern
  - Basis: Update-Kopien
  - Basis: Logdatei
- Restart: Behebung von BS- oder DBMS-Fehlern
  - Basis: Logfile mit Checkpoints
- Rekonstruktion zur Behebung von Speicherfehlern
  - Basis: Dump und Logfile

6 Rollback / Rollforward

- Zur Wahrung der Datenkonsistenz müssen nicht komplett durchgeführte Transaktionen:
  - zurückgerollt, also rückgängig gemacht werden (Rollback)
  - erneut ausgeführt werden, falls alle Infos vorliegen (Rollforward)

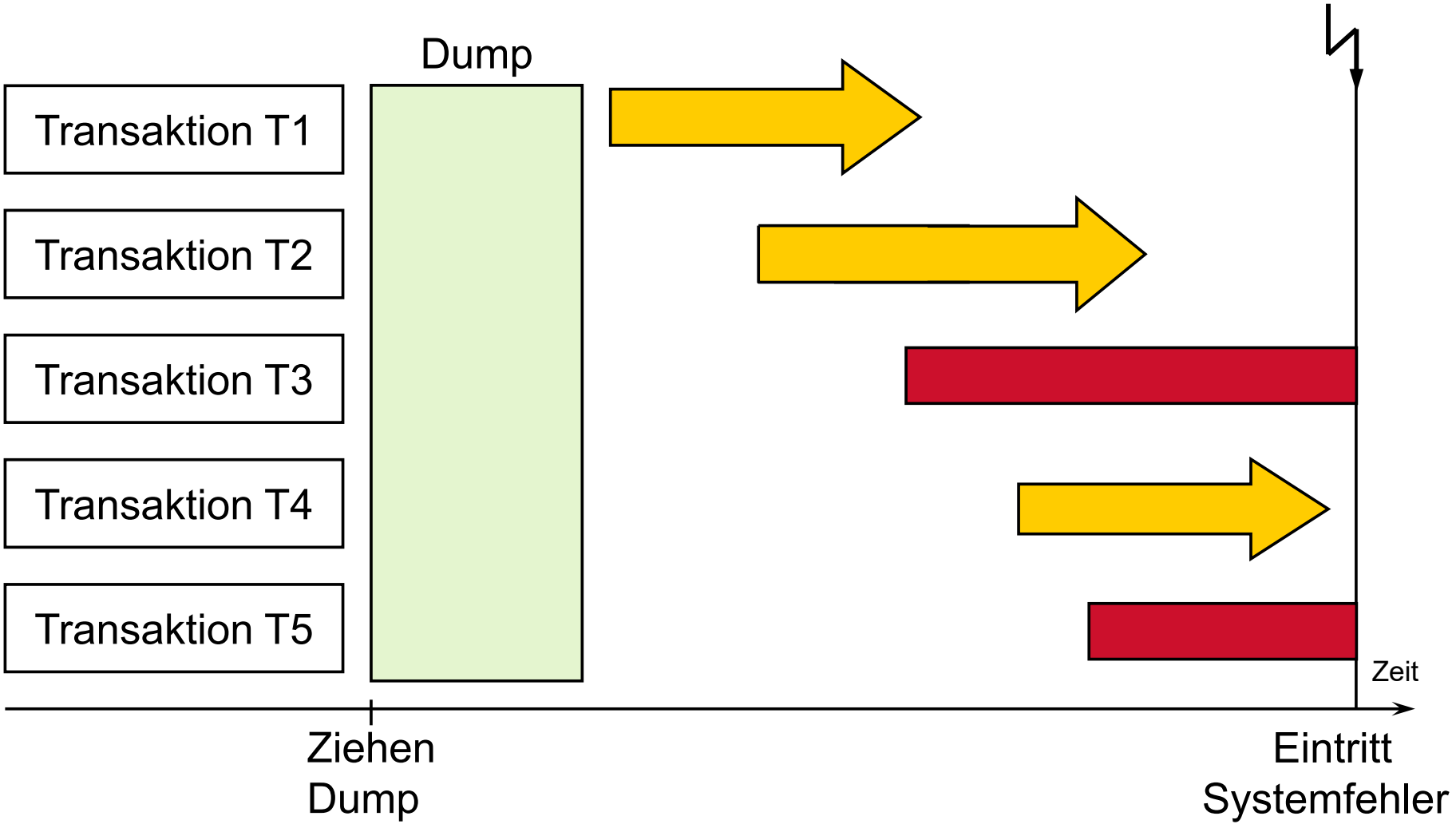
- Lösungsalternativen:
  1. Arbeiten auf Updatekopien
  2. Logfile zur Protokollierung aller Aktivitäten und Before/After-Images

6 Beispiel: Restart

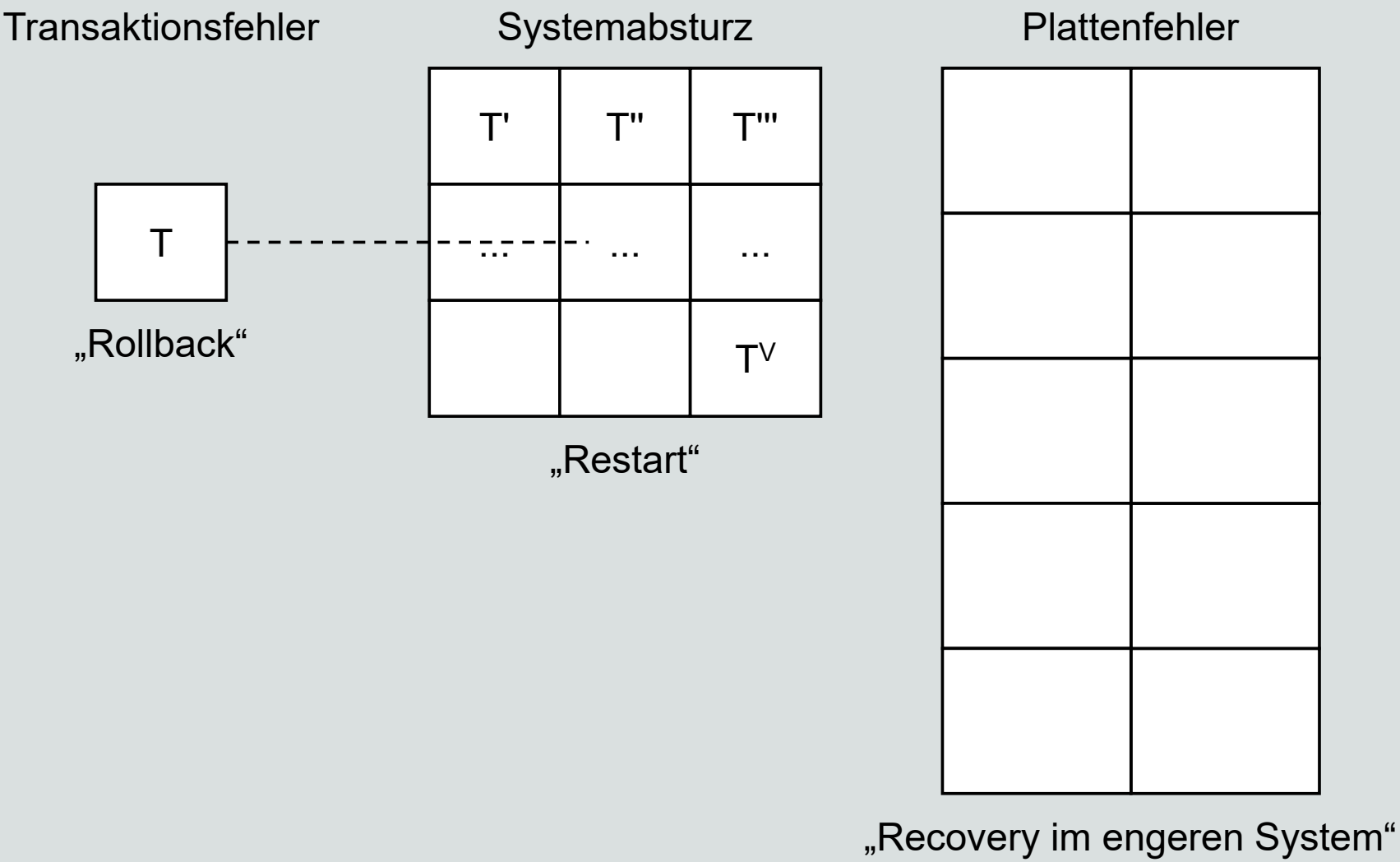


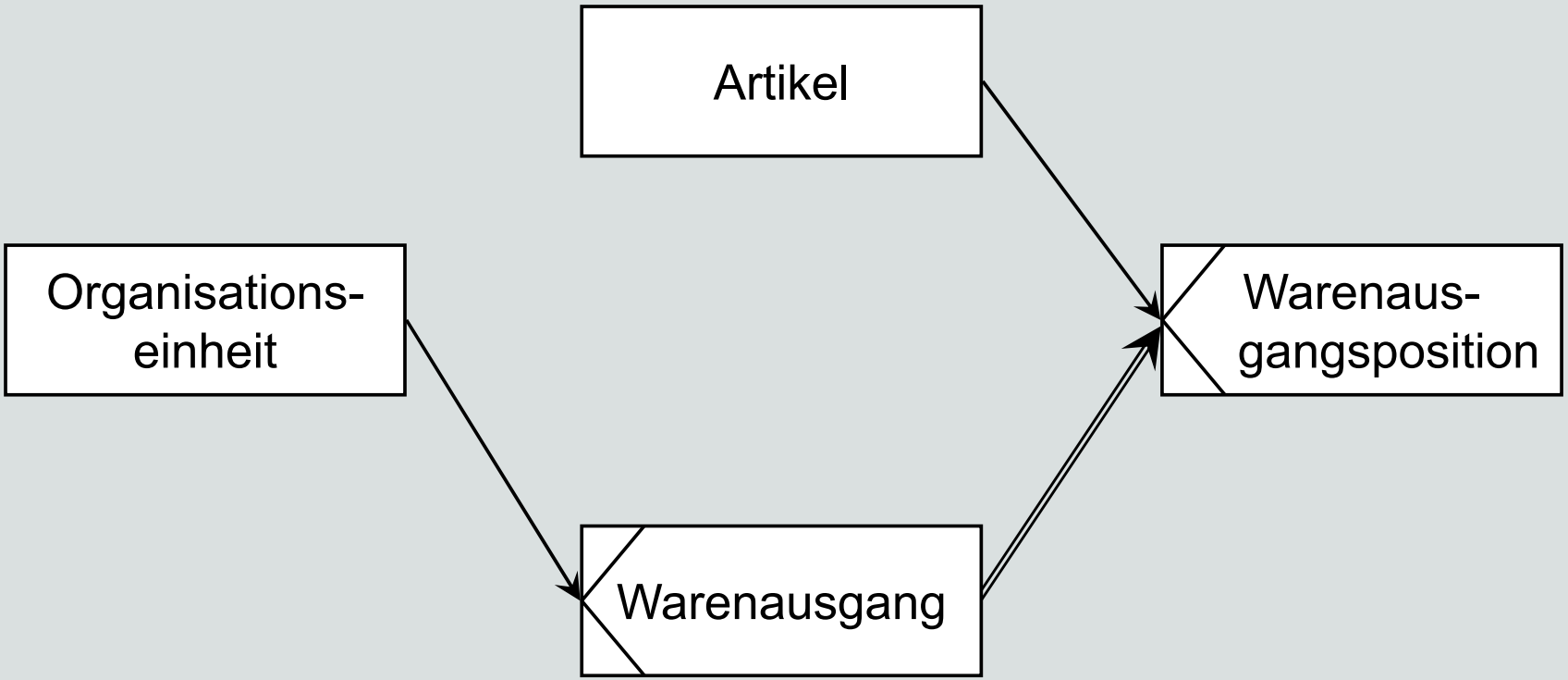
Wie soll man mit den Transaktionen umgehen?

6 Beispiel: Rekonstruktion



6 Transaktionsfehler, Systemabsturz und Plattenfehler




Artikel
Organisations-
einheit
Warenaus-
gangsposition
Warenausgang

Datenrekonstruktion

Ausgangstabellen

Artikel			
Artikelnr	Bezeichnung	Bestand	Preis
A0001	Aspirin Plus C	100	15.00
A0002	Paracetamol	90	23.50
A0003	Doppel Spalt N	30	17.50

Warenausgang		
Warenausgangsnr	Organisationsnr	Datum

Warenausgangsposition			
Warenausgangsnr	Warenausgangspos	Artikelnr	Menge

Rollback – Transaktion und Logfile

```
BOT Transaktion 1
    INSERT INTO Warenausgang (19960003, 4711, ´01022013´);
    INSERT INTO Warenausgangsposition (19960003, 1, A0001, 10);
    INSERT INTO Warenausgangsposition (19960003, 2, A0002, 5);
    UPDATE Artikel SET Bestand = Bestand - 10 WHERE Artikelnr = A0001;
    UPDATE Artikel SET Bestand = Bestand - 5 WHERE Artikelnr = A0002;
EOT Transaktion 1
```

Inhalt des Logfiles

...	
CHECKPOINT 1234 (...)	
...	
BOT Transaktion 1	
Warenausgang ()	Warenausgang **(19960003, 4711, ´01022013´)**;
Warenausgangsposition ()	Warenausgangsposition **(19960003, 1, A0001, 10)**;
Warenausgangsposition ()	Warenausgangsposition **(19960003, 2, A0002, 5)**;
Artikel (A0001, Aspirin Plus C, 100, 15.00)	Artikel (A0001, Aspirin Plus C, **90**, 15.00)
Artikel (A0002, Paracetamol, 90, 23.50)	Artikel (A0002, Paracetamol, **85**, 23.50)
EOT Transaktion 1	
...	

6 Rollback – Rücksetzen der Transaktion

...	
CHECKPOINT 1234 (...)	
...	
BOT Transaktion 1	
Warenausgang ()	Warenausgang **(19960003, 4711, ´01022013´)**;
Warenausgangsposition ()	Warenausgangsposition **(19960003, 1, A0001, 10)**;
Warenausgangsposition ()	Warenausgangsposition **(19960003, 2, A0002, 5)**;
Artikel (A0001, Aspirin Plus C, 100, 15.00)	Artikel (A0001, Aspirin Plus C, **90**, 15.00)
Artikel (A0002, Paracetamol, 90, 23.50)	Artikel (A0002, Paracetamol, **85**, 23.50)
EOT Transaktion 1	
...	

```
BOT Rollback Transaktion 1
    UPDATE Artikel SET Bestand = 90 WHERE Artikelnr = A0002;
    UPDATE Artikel SET Bestand = 100 WHERE Artikelnr = A0001;
    DELETE FROM Warenausgangsposition WHERE Warenausgangsnr = 19960003
    AND Warenausgangspos=1;
    DELETE FROM Warenausgangsposition WHERE Warenausgangsnr = 19960003
    AND Warenausgangspos=2;
    DELETE FROM Warenausgang WHERE Warenausgangsnr = 19960003;
EOT Rollback Transaktion 1
```

System-Neustart – Transaktionen

```
BOT Transaktion 1
    INSERT INTO Warenausgang (19960003, 4711, ´01022013´);
    INSERT INTO Warenausgangsposition (19960003, 1, A0001, 10);
    INSERT INTO Warenausgangsposition (19960003, 2, A0002, 5);
    UPDATE Artikel SET Bestand = Bestand - 10 WHERE Artikelnr = A0001;
    UPDATE Artikel SET Bestand = Bestand - 5 WHERE Artikelnr = A0002;
EOT Transaktion 1
BOT Transaktion 2
    INSERT INTO Warenausgang(19960004, 1211, ´01022013´);
    INSERT INTO Warenausgangsposition (19960004, 1, A0003, 1);

SYSTEMZUSAMMENBRUCH !
```

System-Neustart – Logfile

...	
CHECKPOINT 1234 (...)	
...	
BOT Transaktion 1	
Warenausgang ()	Warenausgang **(19960003, 4711, ´01022013´)**;
Warenausgangsposition ()	Warenausgangsposition **(19960003, 1, A0001, 10)**;
Warenausgangsposition ()	Warenausgangsposition **(19960003, 2, A0002, 5)**;
Artikel (A0001, Aspirin Plus C, 100, 15.00)	Artikel (A0001, Aspirin Plus C, **90**, 15.00)
Checkpoint 1235 (..., Transaktion1)	
Artikel (A0002, Paracetamol, 90, 23.50)	Artikel (A0002, Paracetamol, **85**, 23.50)
EOT Transaktion 1	
...	
BOT Transaktion 2	
Warenausgang ()	Warenausgang **(19960004, 1211,´01022013´)**;
Warenausgangsposition ()	Warenausgangsposition **(19960004, 1, A0003, 1)**;
SYSTEMZUSAMMENBRUCH	

6 System-Neustart – Restart

Vorgehen

- Ausführen nicht vollendeter Transaktionen
- Rücksetzen abgebrochener Transaktionen

- Mögliche Varianten
  a) Transaktionen werden bei EOT spätestens physisch hinausgeschrieben.
     ⇒ Nur Rollback abgebrochener Transaktionen
  b) Spätestens beim Checkpoint wird alles physisch hinausgeschrieben.
     ⇒ Rollback abgebrochener Transaktionen
     ⇒ Redo nicht vollendeter Transaktionen

6 System-Neustart – Restart

...	
CHECKPOINT 1234 (...)	
...	
BOT Transaktion 1	
Warenausgang ()	Warenausgang **(19960003, 4711, ´01022013´)**;
Warenausgangsposition ()	Warenausgangsposition **(19960003, 1, A0001, 10)**;
Warenausgangsposition ()	Warenausgangsposition **(19960003, 2, A0002, 5)**;
Artikel (A0001, Aspirin Plus C, 100, 15.00)	Artikel (A0001, Aspirin Plus C, **90**, 15.00)
Checkpoint 1235 (..., Transaktion1)	
Artikel (A0002, Paracetamol, 90, 23.50)	Artikel (A0002, Paracetamol, **85**, 23.50)
EOT Transaktion 1	
...	
BOT Transaktion 2	
Warenausgang ()	Warenausgang **(19960004, 1211,´01022013´)**;
Warenausgangsposition ()	Warenausgangsposition **(19960004, 1, A0003, 1)**;
SYSTEMZUSAMMENBRUCH	

```
BOT Transaktion Systemneustart;
    DELETE FROM Warenausgangsposition WHERE Warenausgangsnr = 19960004;
    DELETE FROM Warenausgang WHERE Warenausgangsnr = 19960004;
    UPDATE Artikel SET Bestand = 85 WHERE Artikelnr = A0002;
EOT Transaktion Systemneustart;
```

6 Rekonstruktion – Transaktionen

```
BOT Transaktion 1
    INSERT INTO Warenausgang (19960003, 4711, ´01022013´);
    INSERT INTO Warenausgangsposition (19960003, 1, A0001, 10);
    INSERT INTO Warenausgangsposition (19960003, 2, A0002, 5);
    UPDATE Artikel SET Bestand = Bestand - 10 WHERE Artikelnr = A0001;
    UPDATE Artikel SET Bestand = Bestand - 5 WHERE Artikelnr = A0002;
EOT Transaktion 1
BOT Transaktion 2
    INSERT INTO Warenausgang(19960004, 1211, ´01022013´);
    INSERT INTO Warenausgangsposition (19960004, 1, A0003, 1);

SYSTEMZUSAMMENBRUCH !
```

Rekonstruktion – Logfile

...	
CHECKPOINT 1236[Datenbanksicherung]	
...	
BOT Transaktion 1	
Warenausgang ()	Warenausgang **(19960003, 4711, ´01022013´)**;
Warenausgangsposition ()	Warenausgangsposition **(19960003, 1, A0001, 10)**;
Warenausgangsposition ()	Warenausgangsposition **(19960003, 2, A0002, 5)**;
Artikel (A0001, Aspirin Plus C, 100, 15.00)	Artikel (A0001, Aspirin Plus C, **90**, 15.00)
Artikel (A0002, Paracetamol, 90, 23.50)	Artikel (A0002, Paracetamol, **85**, 23.50)
EOT Transaktion 1	
...	
BOT Transaktion 2	
Warenausgang ()	Warenausgang **(19960004, 1211,´01022013´)**;
Warenausgangsposition ()	Warenausgangsposition **(19960004, 1, A0003, 1)**;
SPEICHERFEHLER	

Rekonstruktion

Vorgehen

1. Aufspielen der Datenbankkopie
2. Ausführen aller Transaktionen mit EOT-Marke im LOGFILE

```
BOT Transaktion 1
    INSERT INTO Warenausgang (19960003, 4711, ´01022013´);
    INSERT INTO Warenausgangsposition (19960003, 1, A0001, 10);
    INSERT INTO Warenausgangsposition (19960003, 2, A0002, 5);
    UPDATE Artikel SET Bestand = 90 WHERE Artikelnr = A0001;
    UPDATE Artikel SET Bestand = 85 WHERE Artikelnr = A0002;
EOT Transaktion 1
```

6 Aufgaben

In einer Bank erfolgen folgende zwei Transaktionen:

- T1(Umbuchung) transferiert 400,-- EUR von Konto a nach Konto b, wobei zunächst Konto a belastet wird und danach die Gutschrift auf Konto b erfolgt.
- T2(Zinsgutschriften) läuft gleichzeitig zu T1 ab und schreibt den Konten a und b die 0,5 % Zinseinkünfte gut.

Die beiden Tabellen geben zwei Alternativen für einen verzahnten Ablauf an, ohne, dass eine Mehrbenutzersynchronisation existiert.

n	T1 (Umbuchung)	T2 (Zinsgutschrift)
1	Read(a)	
2	a = a - 400	
3	Write(a)	
4		Read(a)
5		a = a * 1,005
6	Read(b)	
7	b = b + 400	
8	**Rollback**	
9		Write(a)
10		Read(b)
11		b = b * 1,005
12		Write(b)

Alternative 1

n	T1 (Umbuchung)	T2 (Zinsgutschrift)
1	Read(a)	
2	a = a - 400	
3		Read(a)
4		a = a * 1,005
5	Write(a)	
6		Write(a)
7	Read(b)	
8	b = b + 400	
9	Write(b)	
10		Read(b)
11		b = b * 1,005
12		Write(b)

Alternative 2

6 Aufgaben

a) Erläutern Sie für beide Alternativen den jeweiligen Effekt der Ausführung. Wie bezeichnet man das jeweilige konkrete Problem aufgrund des unkontrollierten parallelen Zugriffs?

b) Erstellen Sie, gemäß dem verzahnten Ablauf der beiden Transaktionen in Alternative 2, den Präzedenzgraphen. Interpretieren Sie das Ergebnis in Hinblick auf die Serialisierbarkeit.

c) Geben Sie tabellarisch einen möglichen verzahnten Ablauf beider Transaktionen für Alternative 2 an, so dass
   - wir keine Sperren und keine Konflikte haben
   - es den Anforderungen des Zwei-Phasen-Sperrprotokolls (2PL) genügt.

6 Kontrollfragen

- Vor welchen Herausforderungen steht die Transaktionsverwaltung?
- Erläutern Sie die Konflikte „Lost Update“, „Phantom Read“, „Dirty Read“ und „Non-repeatable Read“.
- Was besagt das ACID-Prinzip der Transaktionsverwaltung? Erläutern Sie die Bedeutung dieses Akronyms.
- Wie ist beim Durchführen eines Recovery nach einem Systemabsturz vorzugehen?
- Geben Sie ein Beispiel einer Verklemmung („Deadlock“) an.
- Wie unterscheiden sich pessimistische und optimistische Sperrverfahren?
- Was versteht man unter exklusiven Sperren?
- Erläutern Sie das Prinzip des „Zwei-Phasen-Sperrprotokolls“.