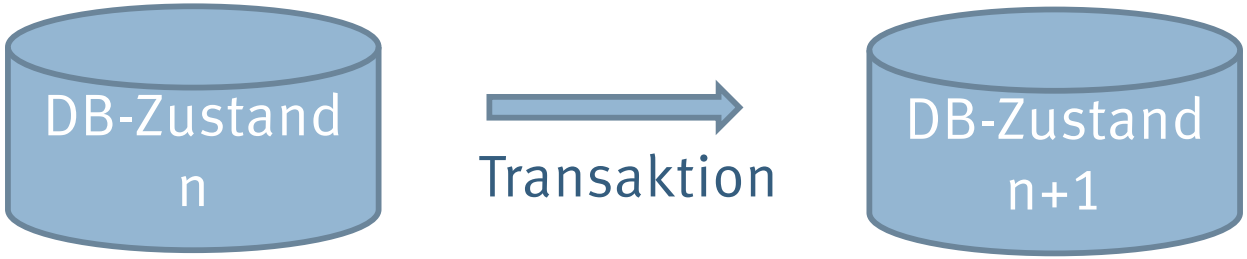


Transaktionen

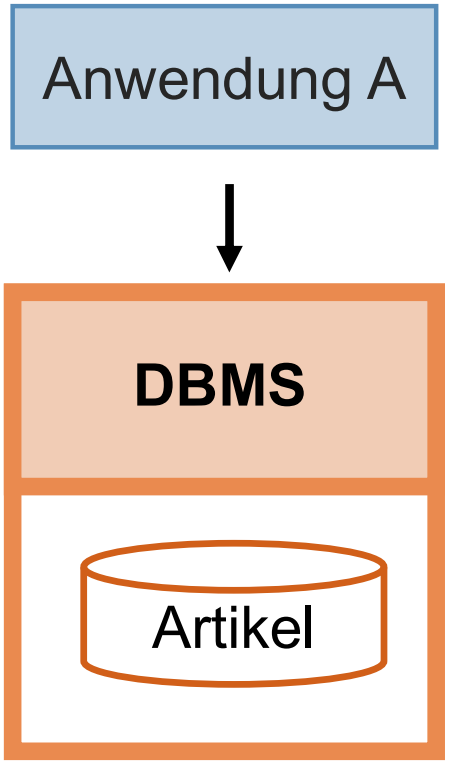
Was ist eine Transaktion?

Datenbankzustände

Die Datenbank soll zu jedem Zeitpunkt die in der Realität geltenden Zusammenhänge und Regeln wiederspiegeln.

Idee

Kontrolle des Datenzugriffs und damit der Datenintegrität durch das DBMS statt durch die Anwendungen



Vorteile

- ✓ Effektivere und sichere Kontrolle
- ✓ Einfachere Anwendungsprogrammierung
- ✓ Schlankere Clients

Implementierung

- → Statische Integritätsbedingungen
- → Dynamische Integritätsbedingungen

  Trigger, **Transaktionen**

Zusammenhängende Änderungen

Bestellung 1234 des Kunden 2310 wird versendet		
	Bestellung:	Bestellstatus „versendet“ setzen
	Lager:	Lagerbestand aktualisieren
	Kundenkonto:	mit Rechnungsbetrag belasten

Transaktion

Datenbankoperationen:

1. Bestellstatus auf „versendet“ setzen

```
UPDATE Bestellung SET Versandstatus = “versendet“
WHERE Bestellnummer = 1234
```

2. Lagerbestand aktualisieren

```
UPDATE Lager SET Lagerbestand = (SELECT Lagerbestand FROM Lager
                                 WHERE Anummer = 4812) -1
WHERE Anummer = 4812
```

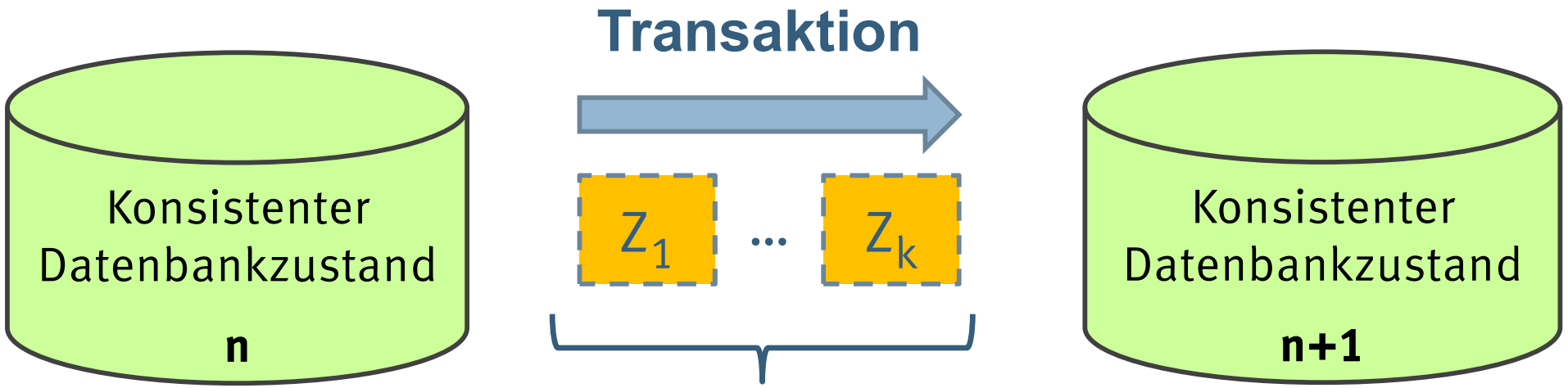
3. Kundenkonto mit Rechnungsbetrag belasten

```
UPDATE Konto SET Saldo  = (SELECT Saldo FROM Konto
                           WHERE Kundennummer=2310)
                         - (SELECT Betrag FROM Konto
                           WHERE Bestellnummer=1234)
WHERE Kundennummer = 2310
```

Transaktionsbegriff

Eine **Transaktion** ist eine inhaltlich **zusammenhängende Menge von Datenbankoperationen**, die ganz oder gar nicht ausgeführt werden.

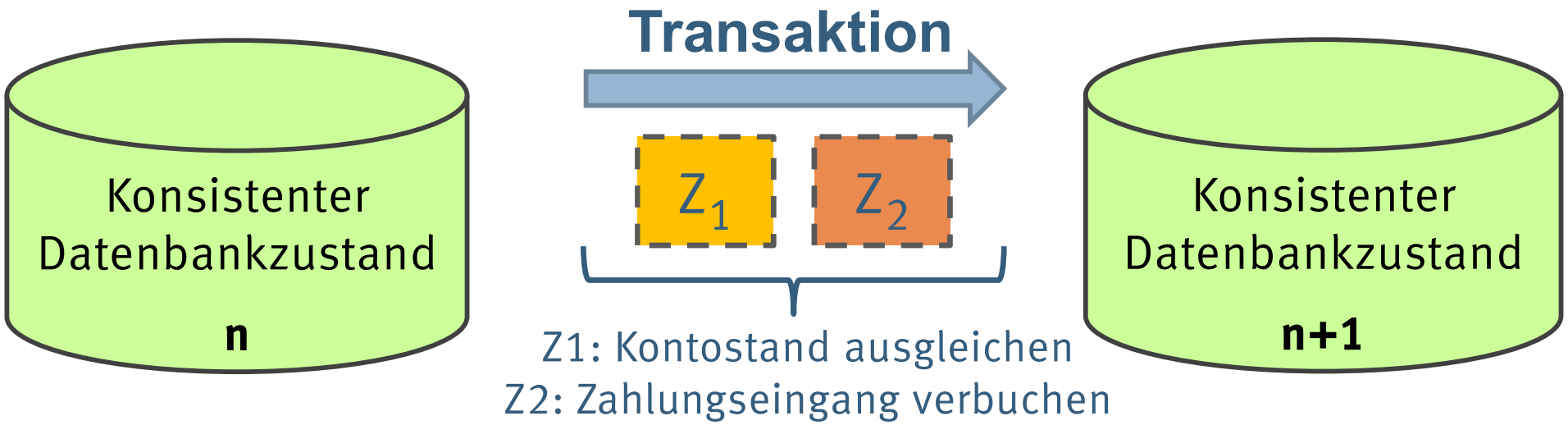
Eine **Transaktion** überführt einen konsistenten Datenbankzustand in einen wiederum konsistenten Datenbankzustand.

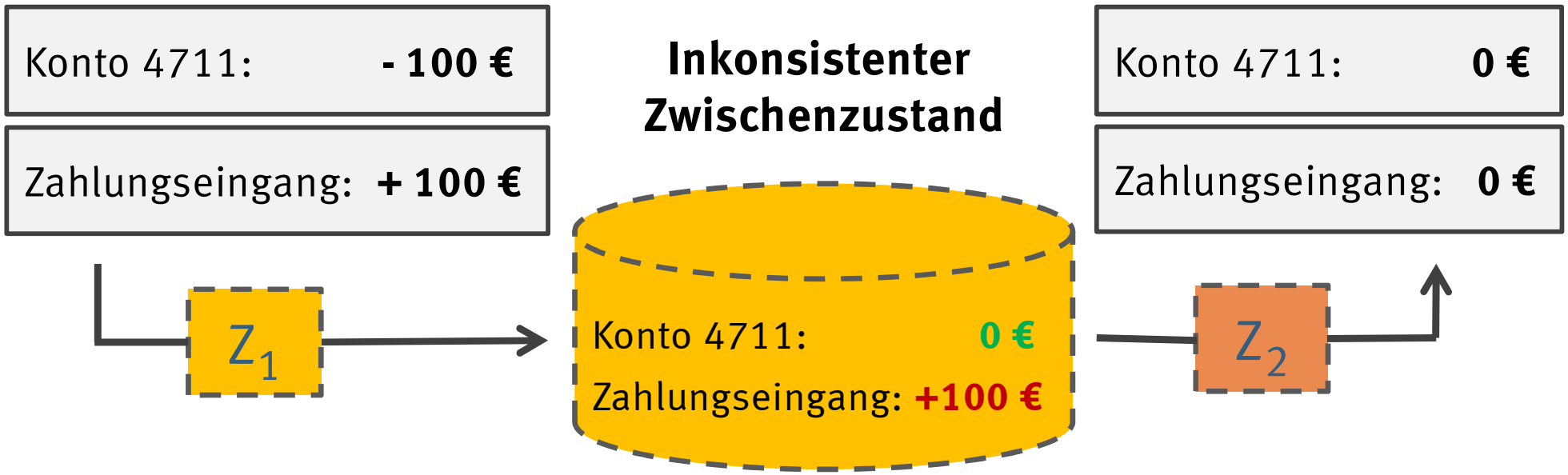


Der initiale Zustand (n=0) der Datenbank ist konsistent.

Zustandsänderung
durch DB-Operationen
(Insert, Update, Delete)

Beispiel: Zahlungseingang verbuchen

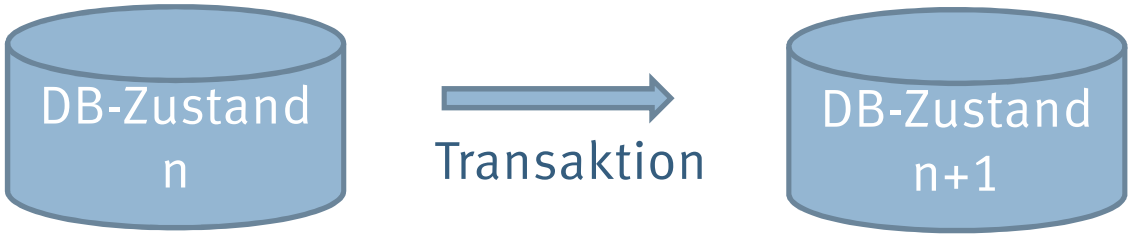




K1 Ist auf dieser Seite b:= 0 richtig? Sollte hier nicht Zahlungseingang = 100 stehen?
Kathrin Holl; 27.05.2018

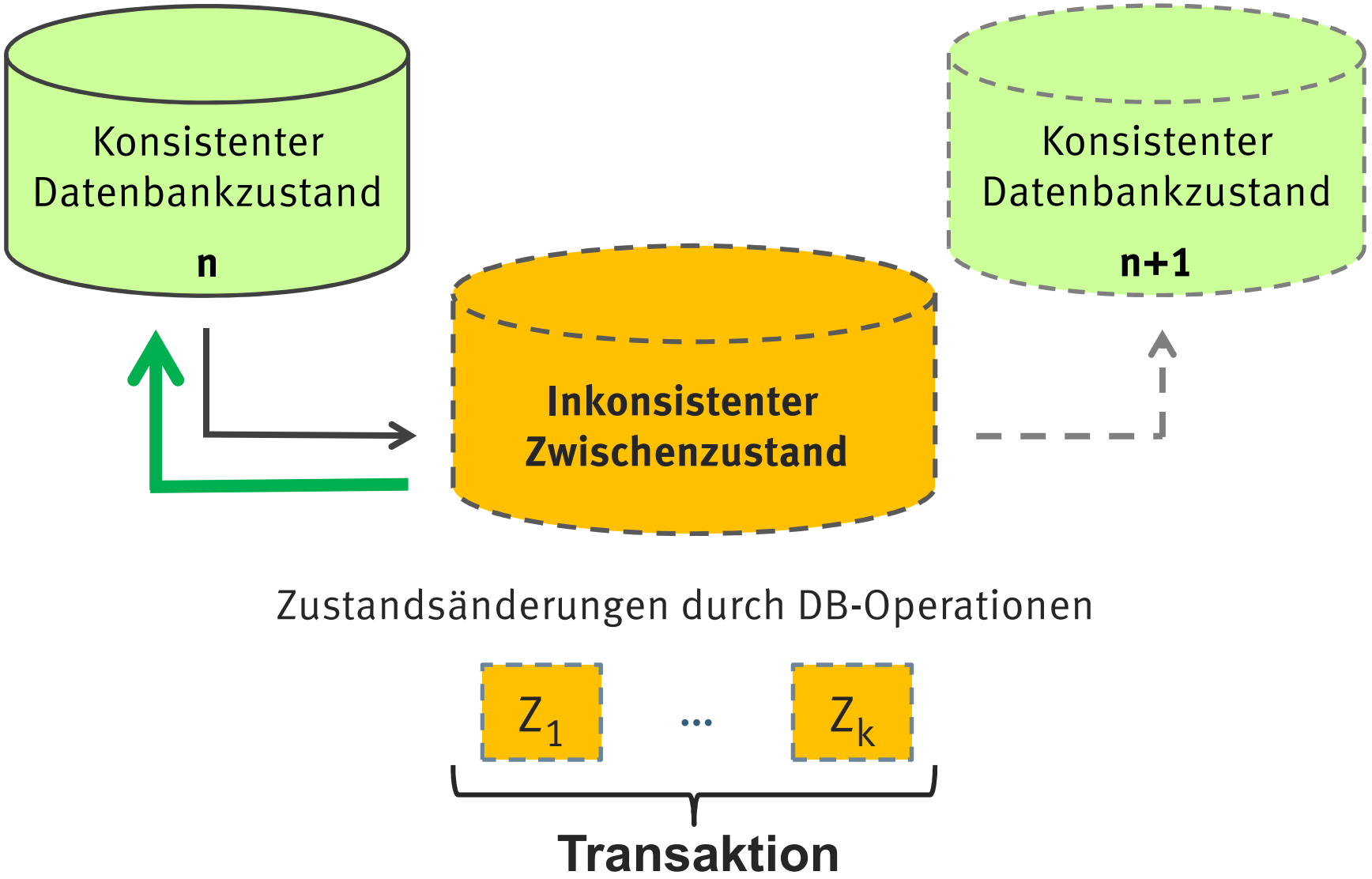
Transaktionen

- Die Datenbank soll zu jedem Zeitpunkt die in der Realität geltenden Zusammenhänge und Regeln exakt wiederspiegeln.
- Idee: Kontrolle des Datenzugriffs und damit der Datenintegrität durch das DBMS statt durch die Anwendungen
- Transaktionen überführen einen konsistenten Datenbankzustand in einen wiederum konsistenten Datenbankzustand.
- Eine Transaktion ist eine inhaltlich zusammenhängende Menge von Datenbankoperationen, die ganz oder gar nicht ausgeführt werden.
- Das heißt: Mehrere inhaltlich zusammengehörige Operationen (insert, update, delete) müssen zwingend *alle* erfolgreich ausgeführt oder *alle* abgebrochen und ggf. rückgängig gemacht werden. Diese Operationen ergeben in ihrer Gesamtheit eine Transaktion.



Transaktionen

Transaktionen sind atomar

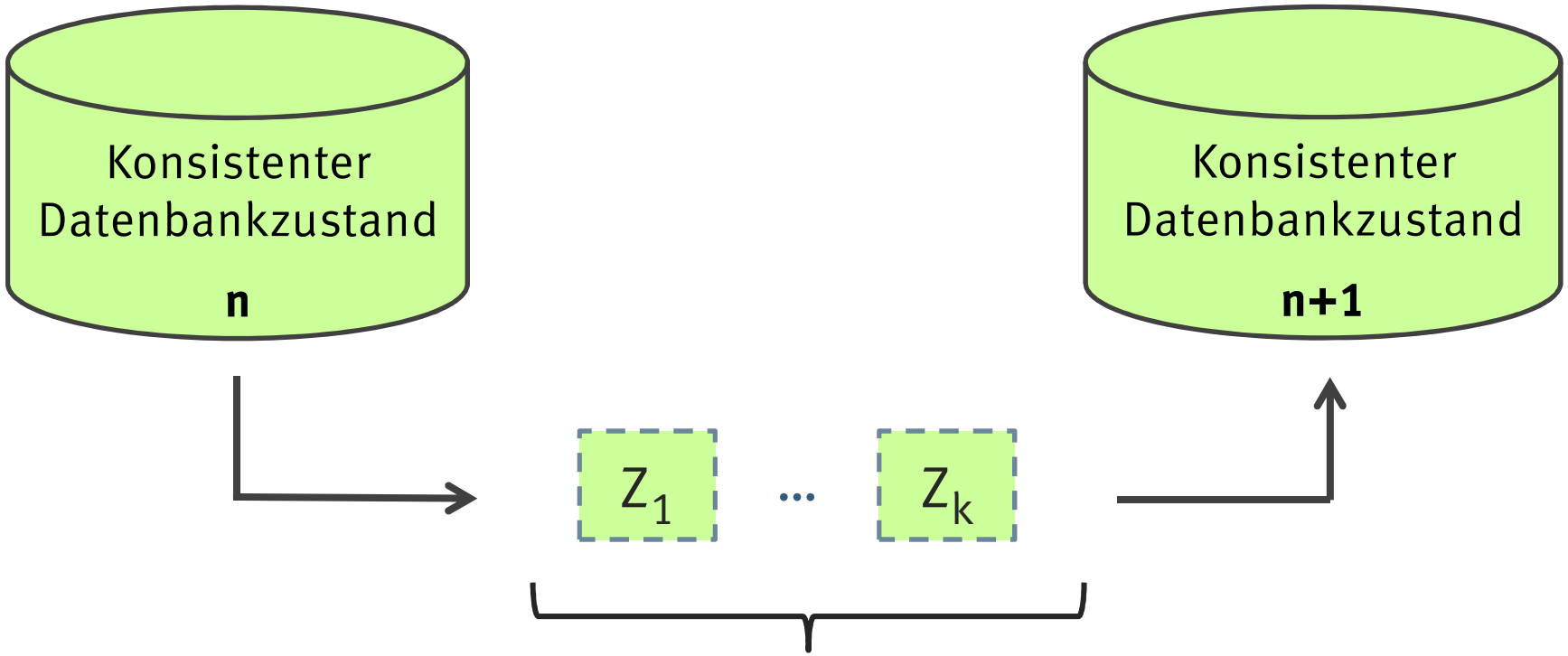

Konsistenter Datenbankzustand
n
Konsistenter Datenbankzustand
n+1
Inkonsistenter Zwischenzustand
Zustandsänderungen durch DB-Operationen
Z_1
...
Z_k
Transaktion

Folie 2

K1 Ist auf dieser Seite b:= 0 richtig? Sollte hier nicht Zahlungseingang = 100 stehen?
Kathrin Holl; 27.05.2018

Atomarität

Atomarität

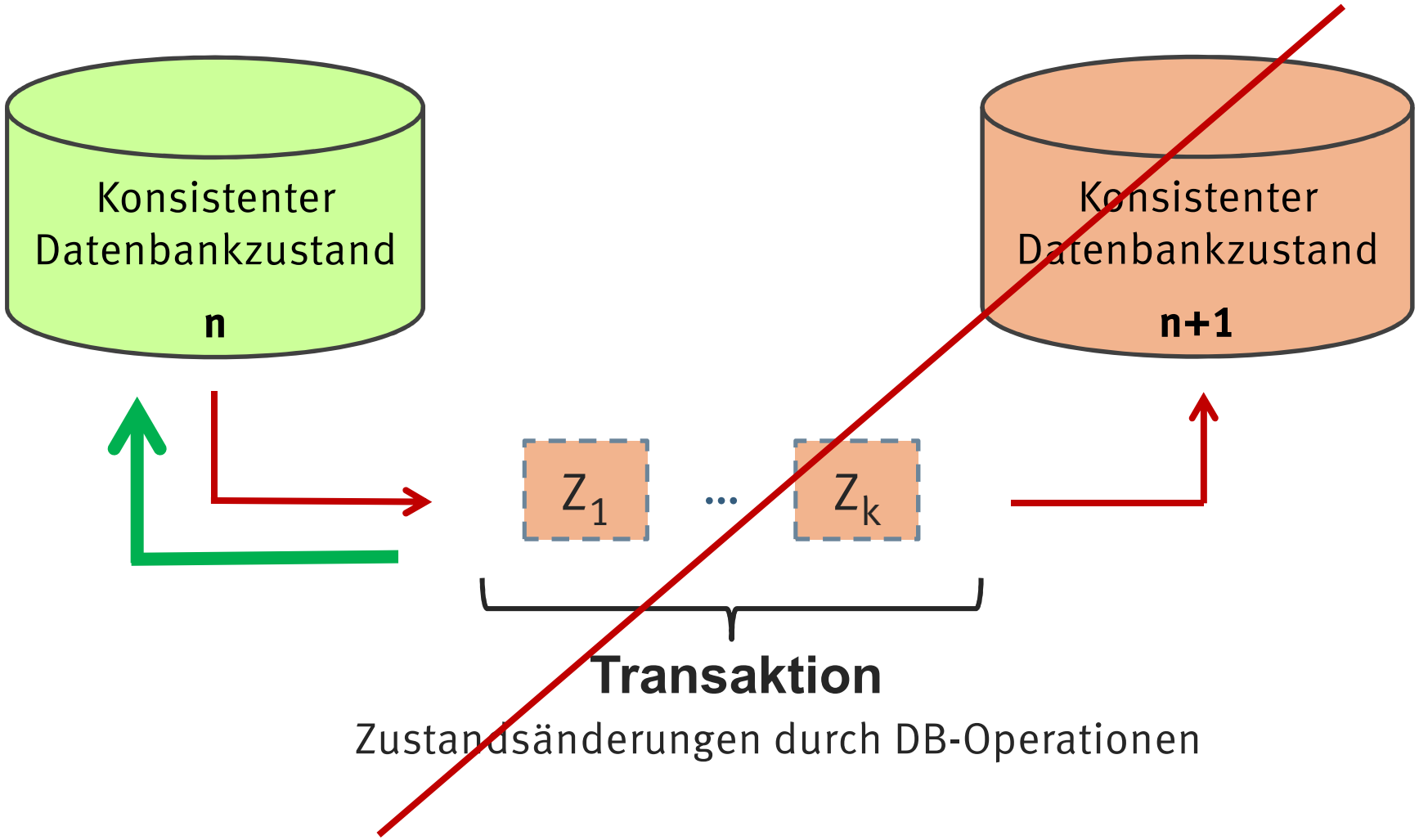


Zustandsänderungen durch DB-Operationen

Folie 4

K1 Ist auf dieser Seite b:= 0 richtig? Sollte hier nicht Zahlungseingang = 100 stehen?
Kathrin Holl; 27.05.2018

Atomarität


Konsistenter Datenbankzustand
n
Konsistenter Datenbankzustand
n+1
Z_1
...
Z_k
Transaktion
Zustandsänderungen durch DB-Operationen

Atomarität

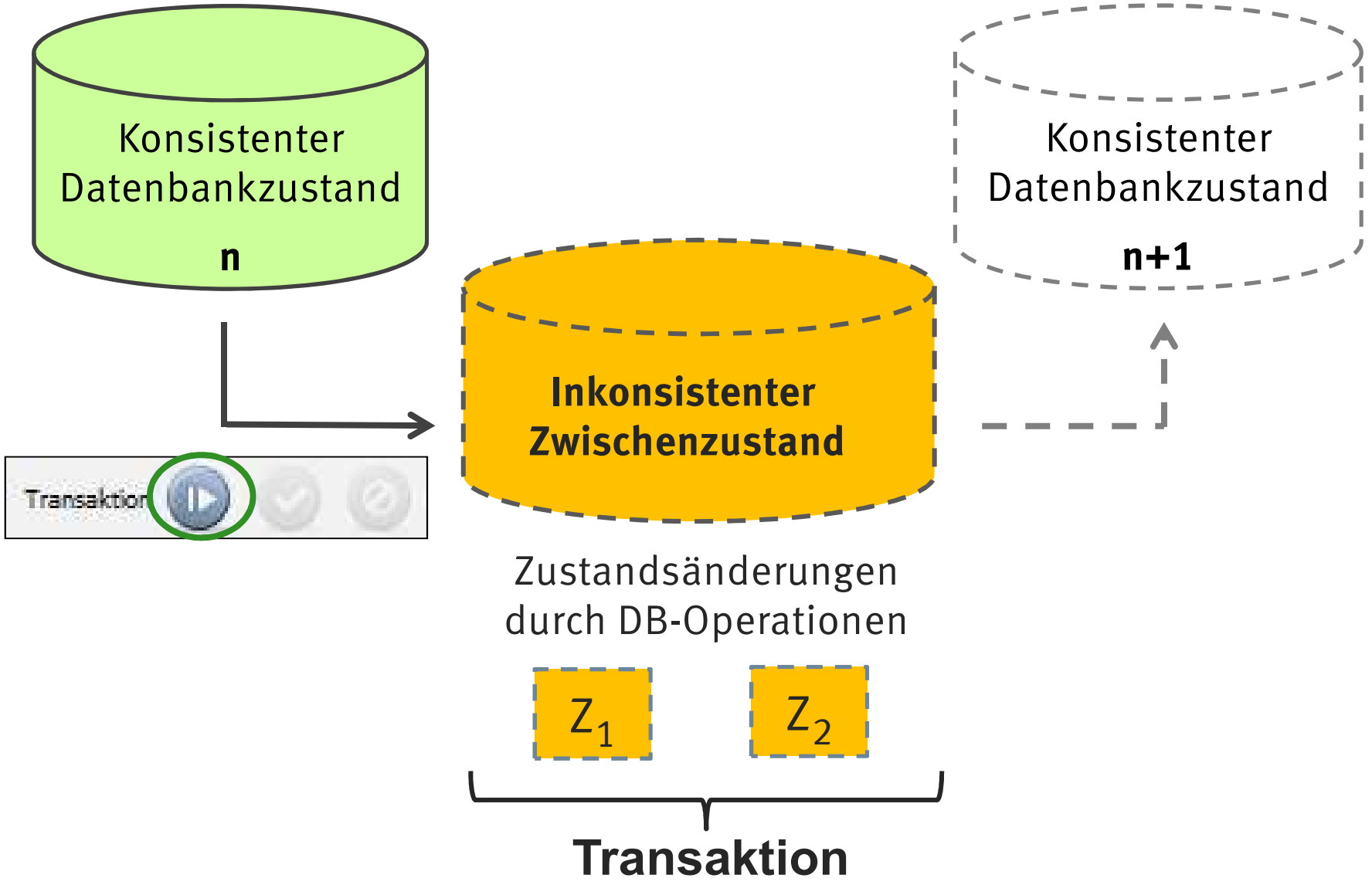
Eine Transaktion ist eine inhaltlich zusammenhängende Menge von Datenbankoperationen, die ganz oder gar nicht ausgeführt werden.

→ **Zusammengehörende Datenbankoperationen werden zu einer Transaktion zusammengefasst.**

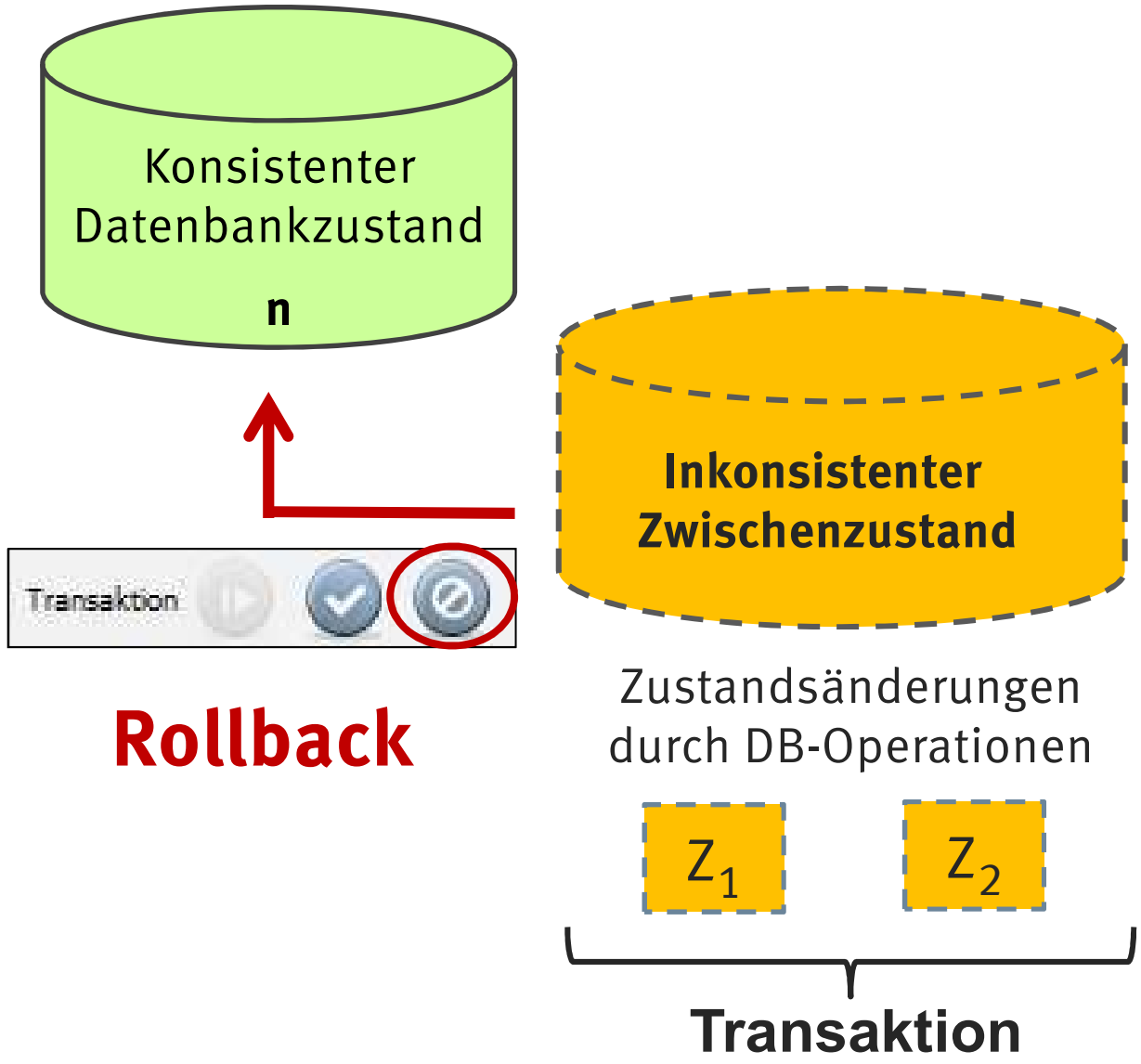
- **Begin of Transaction (BOT)**
  Beginn der Menge zusammenhängender Datenbankoperationen

- **End of Transaction (EOT)**
  Ende der Menge zusammenhängender Datenbankoperationen

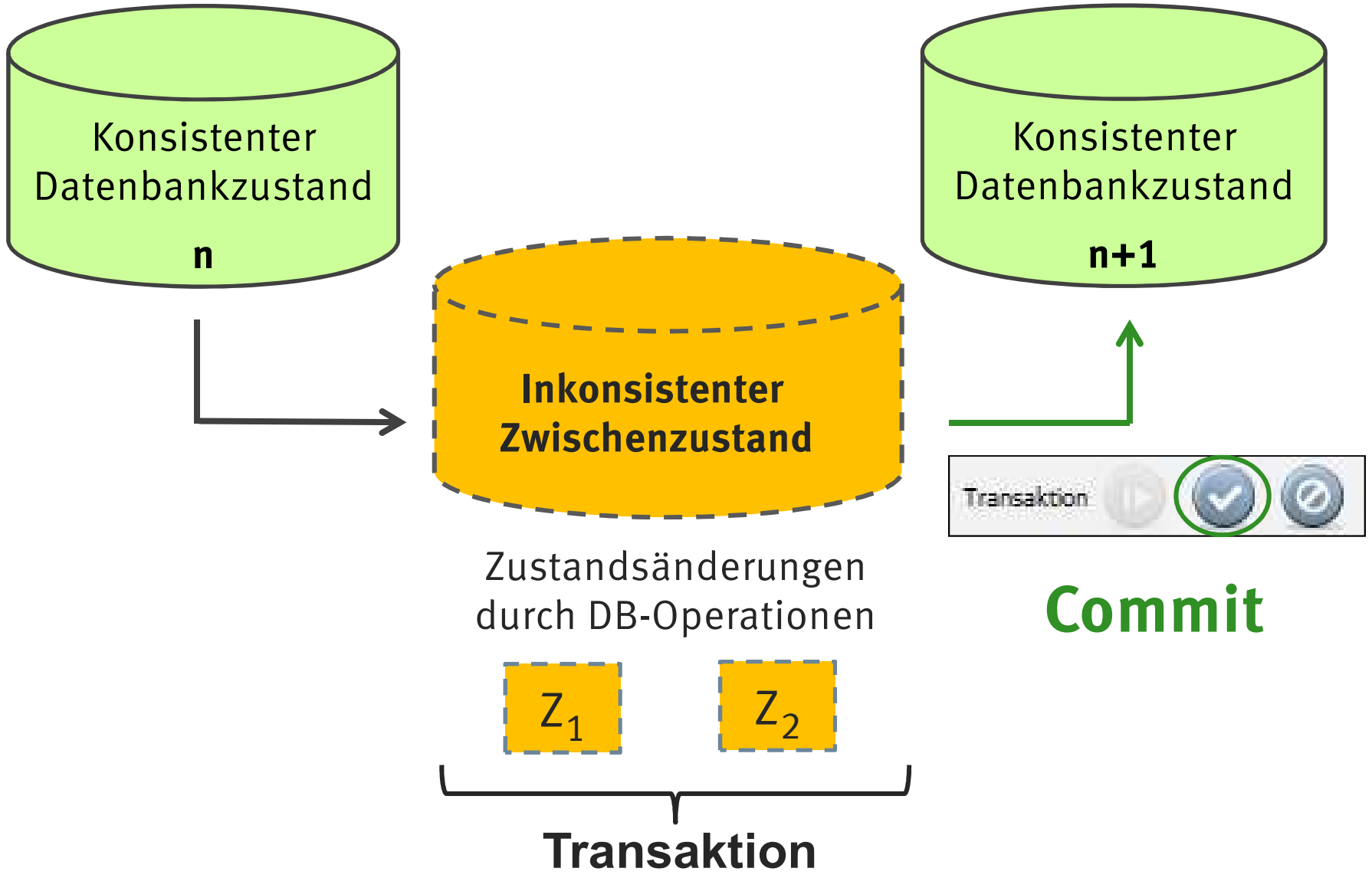
Atomarität


Konsistenter Datenbankzustand
n
Konsistenter Datenbankzustand
n+1
Inkonsistenter Zwischenzustand
Transaktion
Zustandsänderungen durch DB-Operationen
Z_1
Z_2
Transaktion

Atomarität


Konsistenter
Datenbankzustand
n
Transaktion
Rollback
Inkonsistenter
Zwischenzustand
Zustandsänderungen
durch DB-Operationen
Z_1
Z_2
Transaktion

Atomarität


Konsistenter Datenbankzustand
n
Konsistenter Datenbankzustand
n+1
Inkonsistenter Zwischenzustand
Transaktion
Zustandsänderungen durch DB-Operationen
Commit
Z_1
Z_2
Transaktion

Commit und Rollback

Commit

- Sicherung der Änderungen der Transaktion
- Ausgeführt beim Erreichen eines korrekten DB-Zustandes (Integritätsbedingungen erfüllt)

Rollback

- Rücksetzung der DB in einen vorherigen, konsistenten Zustand
- erfolgt bei einer fehlerhaften Transaktion, z.B. nach einem Transaktionsabbruch oder bei Verletzung einer Integritätsbedingung

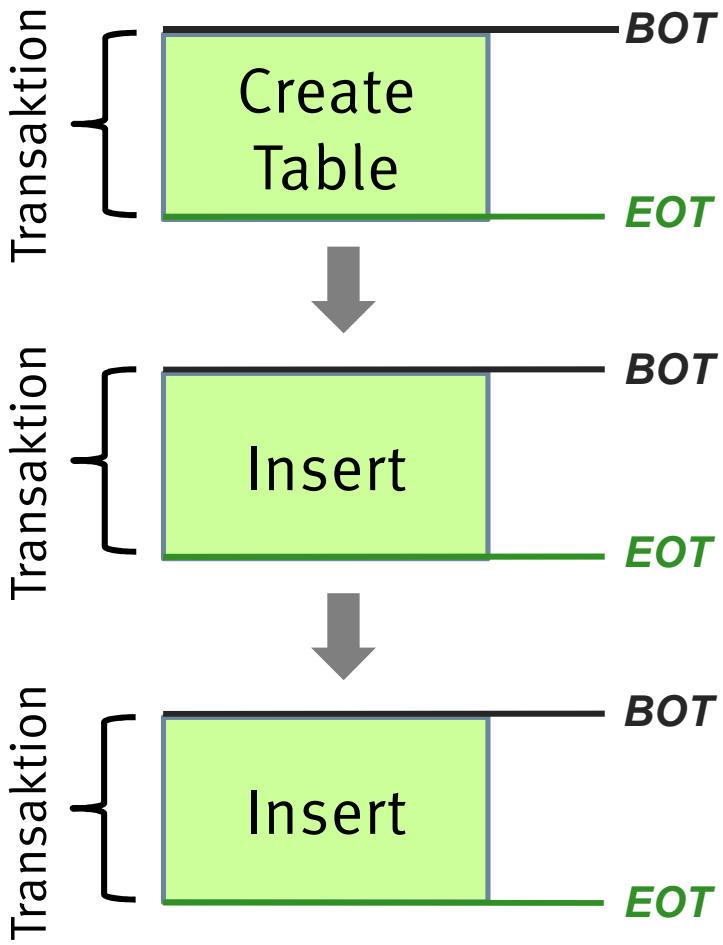
Autocommit

Transaktionssteuerung

Standard:
set autocommit on

Java:
con.setAutocommit(true)

Jedes Statement wird in einer eigenen Transaktion ausgeführt



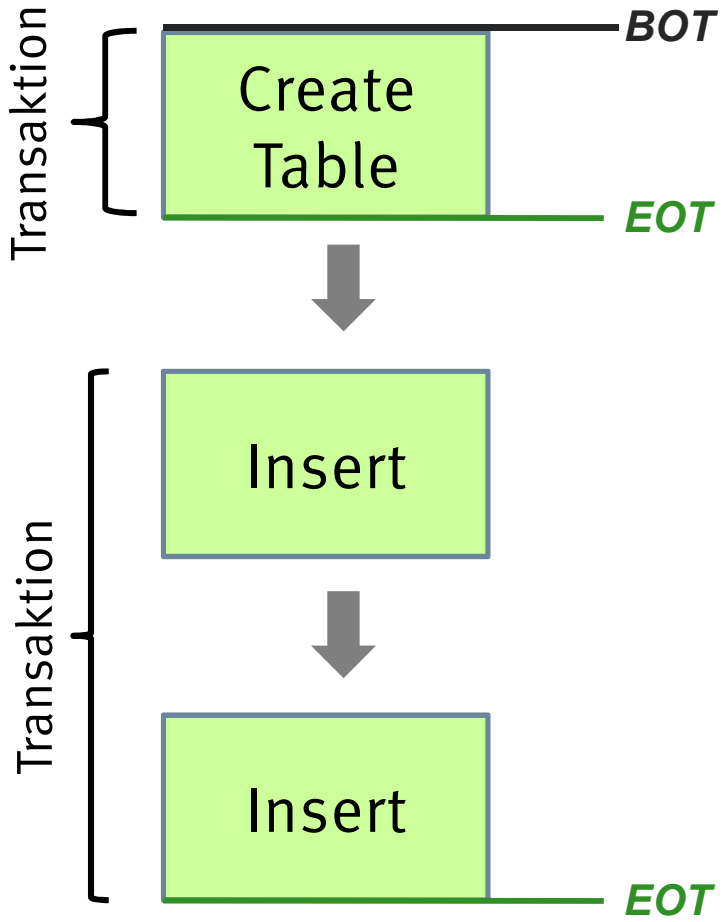
BOT: Begin of Transaction
EOT: End of Transaction

Transaktionssteuerung

set Autocommit off

Java:
con.setAutocommit(false)

Statements werden in einer Transaktion zusammen-gefasst bis zum nächsten Commit bzw. Rollback. Nach DDL-Befehlen erfolgt dies automatisch.



Java:
- con.commit()
- con.rollback()

Zusammenfassung

ACID-Eigenschaften von Transaktionen

1. Atomarität (*Atomicity*)
2. Konsistenz (*Consistency*)
3. Isoliertheit (*Isolation*)
4. Dauerhaftigkeit (*Durability*)

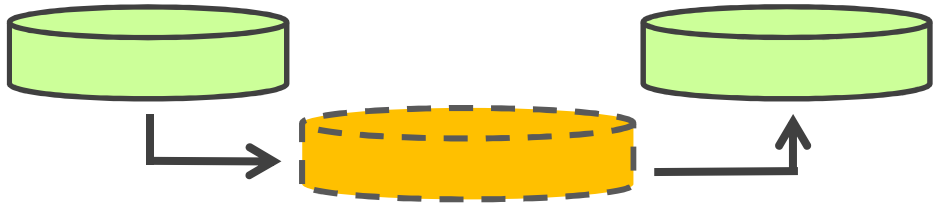
Zusammenfassung

(Wiederholung:)

» Eine **Transaktion** ist eine inhaltlich zusammenhängende Folge von Datenbank-Operationen.

» Ein Übergang zwischen zwei konsistenten Datenbank-Zuständen durch eine Transaktion erfolgt entweder ganz (vollständig), oder gar nicht. Diese Eigenschaft von Transaktionen nennen wir „**Atomarität**“ (oder auch „Abgeschlossenheit“).

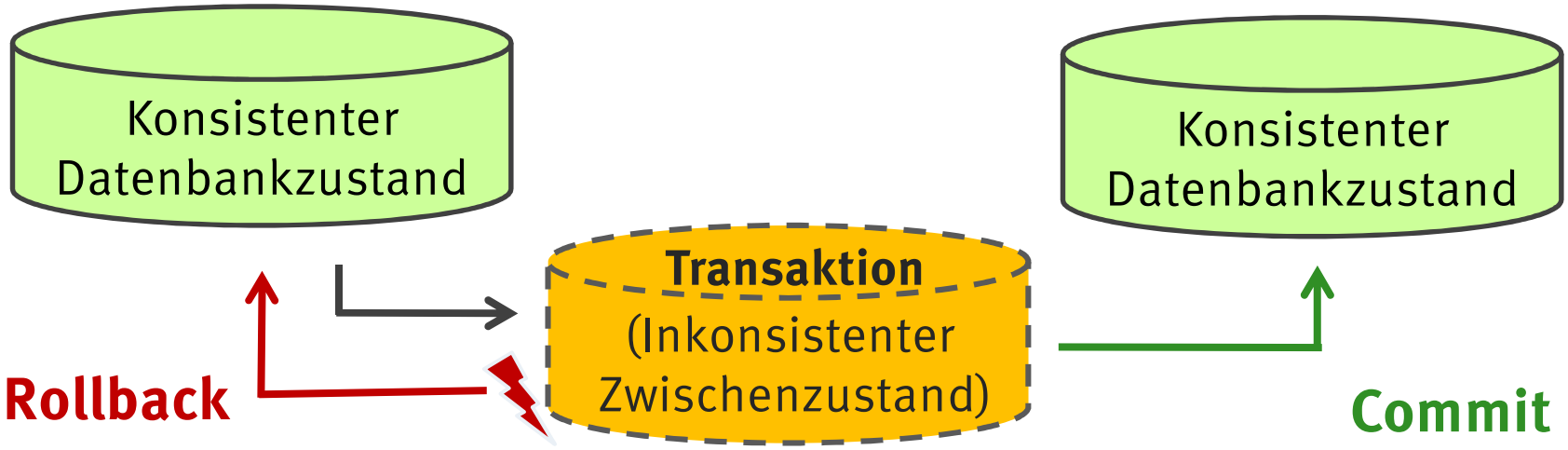
» Während der Ausführung einer Transaktion befindet sich die Datenbank solange in einem inkonsistenten Zustand, bis **alle** zusammengehörigen Operationen *erfolgreich* und *vollständig* ausgeführt worden sind.



Zusammenfassung

» Stellt sich während einer Transaktion heraus, dass diese *nicht* vollständig abgeschlossen werden kann, wird ein „**Rollback**“ durchgeführt. Alle bis dahin bereits ausgeführten DB-Operationen werden dadurch rückgängig gemacht und die Datenbank kehrt zu ihrem Ausgangszustand zurück.

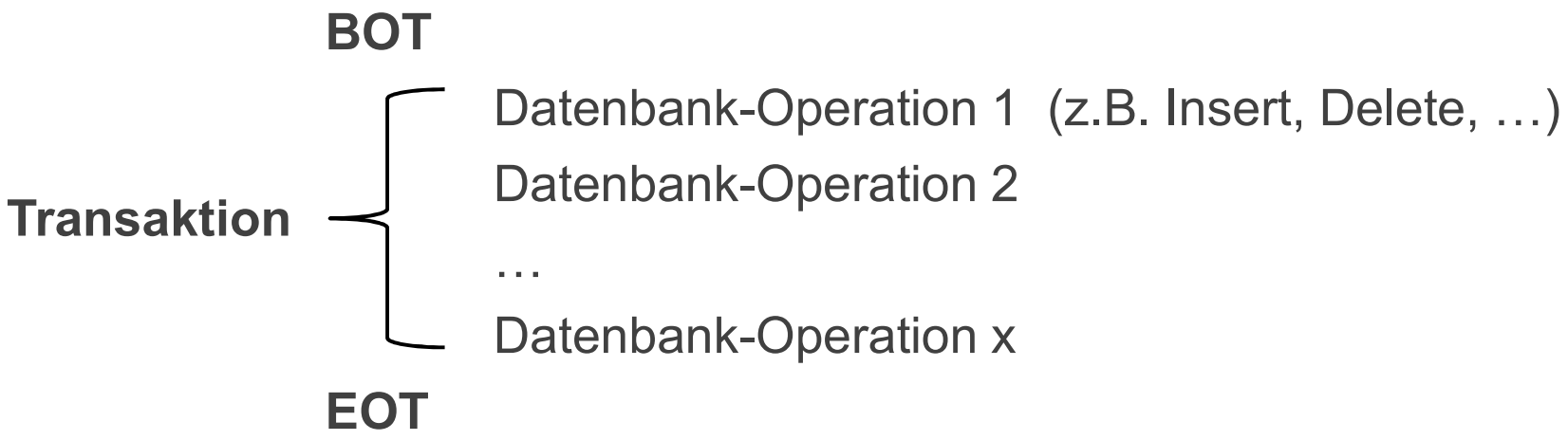
» Im Positiv-Falle erfolgt das sogenannte „**Commit**“:
Die Transaktion wird vollständig wirksam und gültig. Die Datenbank geht in einen neuen, konsistenten Zustand über.



Zusammenfassung

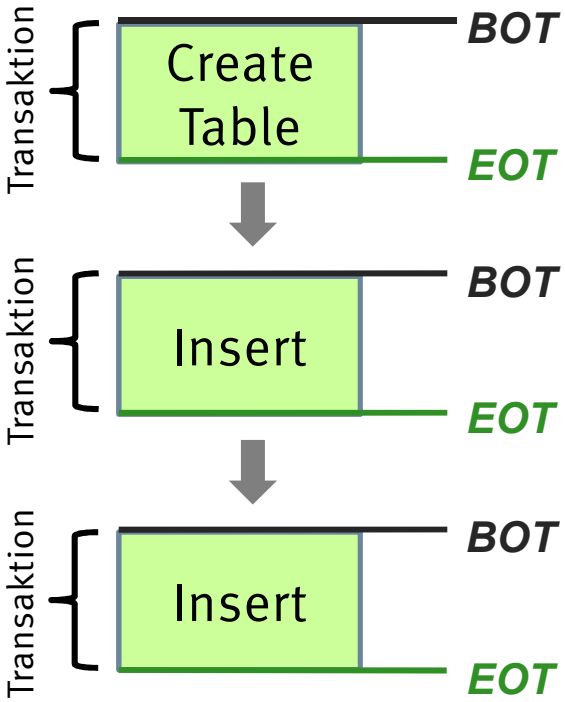
» Beginn einer Transaktion: **BOT** (Begin of Transaction)

» Ende einer Transaktion: **EOT** (End of Transaction)



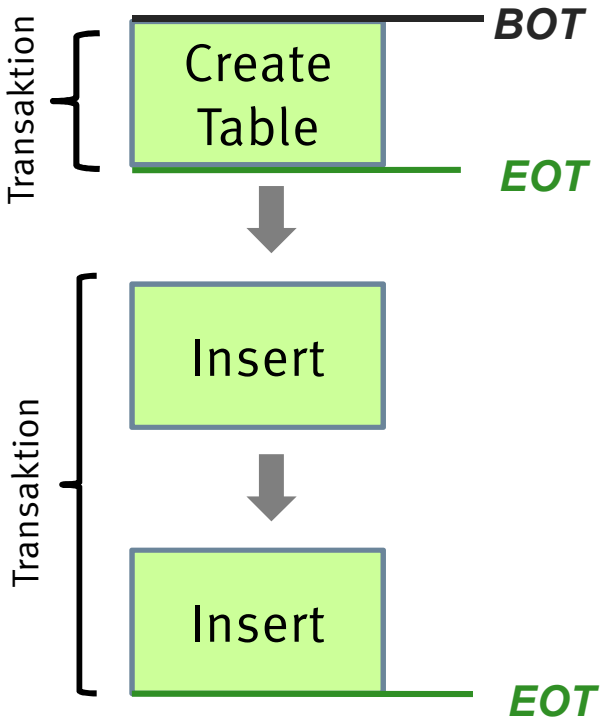
Zusammenfassung

Autocommit on

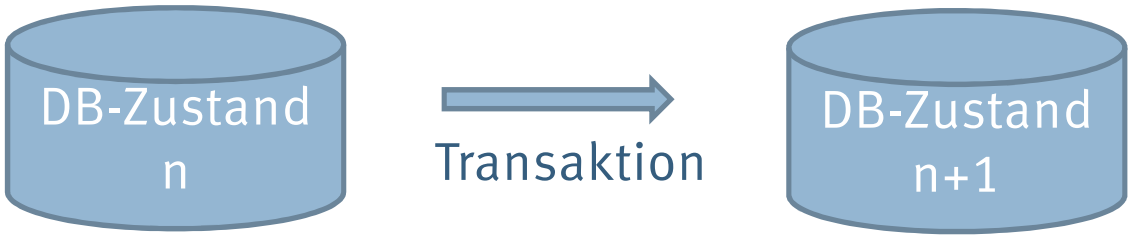


Jedes Statement wird in einer eigenen Transaktion ausgeführt.

Autocommit off



Eine Transaktion endet durch ein explizites Commit oder Rollback. Nach DDL-Befehlen erfolgt ein implizites Commit.



Transaktionen

Transaktionen und Datenbankprogramme

Beispiel

Warenkorb Meitner

Artikel-nummer	Anzahl
~~4810~~	~~1~~
~~4820~~	~~2~~

Bestellung

Bestell-nummer	Kunden-nummer	Datum	Bestell-status
10002	2310	10.12.2010	Offen

Bestellposition

Bestell-nummer	Artikel-nummer	Anzahl
10002	4810	1
10002	4820	2

Wie können diese Änderungsoperationen implementiert werden?

1. Lösungsmöglichkeit : Eine gespeicherte Prozedur

1. Einfügen der Bestellung in die Tabelle „Bestellung“

```
INSERT INTO Bestellung (bestellnummer, kundennummer,
                        datum, bestellstatus)
VALUES (bestellnr, kundennr, current_date, ‘offen’);
```

2. Ermittlung der automatisch generierten Bestellnummer

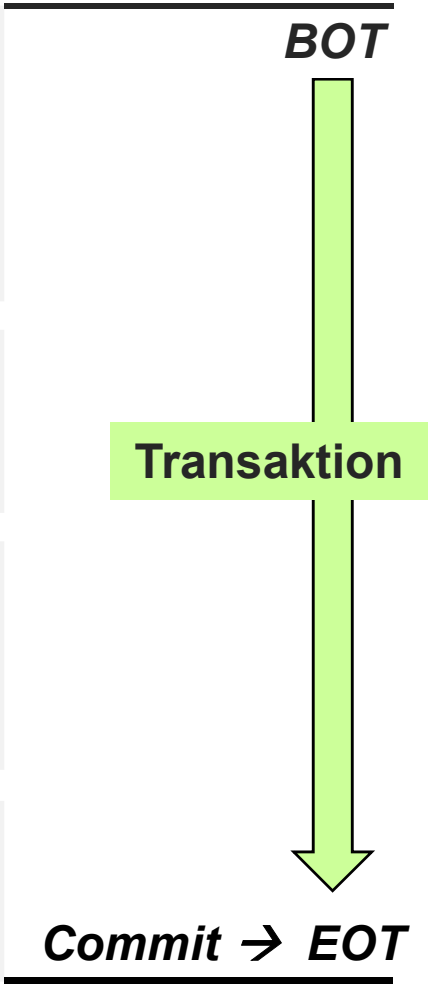
```
SELECT …
```

3. Übertragung der Einträge aus dem Warenkorb in die Tabelle „Bestellposition“

```
INSERT INTO Bestellposition …
```

4. **Löschen der Einträge im Warenkorb**

```
DELETE FROM Warenkorb …
```



Implementierung

Implementierung

Oracle
(mit Transaktions-Kontext)

```
CREATE  PROCEDURE bestellung(IN kundennr INT)

AS  keineArtikel      EXCEPTION; …

BEGIN

     START Transaction;

        SELECT count(*) INTO AnzahlArtikel …
        IF AnzahlArtikel=0  THEN RAISE keineArtikel;
        INSERT INTO bestellung  …
        INSERT INTO bestellposition …
        DELETE FROM Warenkorb where Kundennummer=kundennr;

     Commit;

     EXCEPTION
            WHEN keineArtikel
                  THEN
                    rollback;
                    raise_application_error(-20500,'Fehler: keine Artikel ');
END;
```

2. Lösungsmöglichkeit : Eine gespeicherte Prozedur und ein AFTER-Trigger

Gespeicherte Prozedur:

1. Einfügen der Bestellung in die Tabelle „Bestellung“

```
INSERT INTO Bestellung (bestellnummer, kundennummer,
                        datum, bestellstatus)
VALUES (bestellnr, kundennr, current_date, ‘offen’);
```

BOT

(Auto-) Commit → EOT

Trigger After InsertON Bestellung:

2. Ermittlung der automatisch generierten Bestellnummer

(new.Bestellnummer)

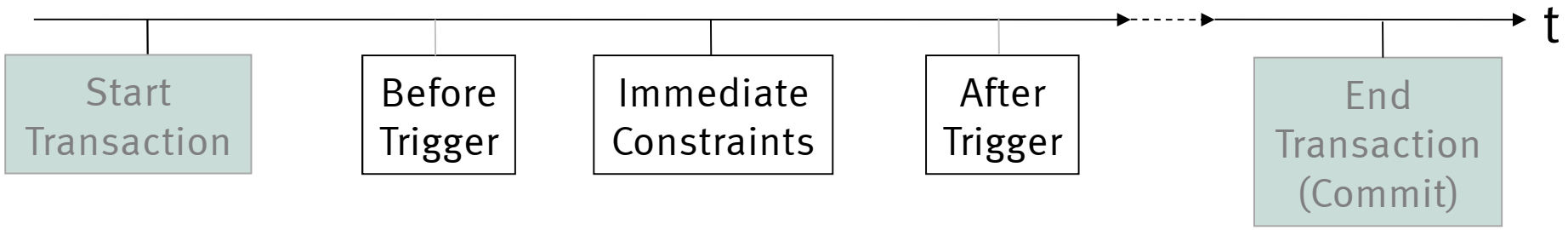
3. Übertragung der Einträge aus dem Warenkorb in die Tabelle „Bestellposition“

INSERT INTO Bestellposition …

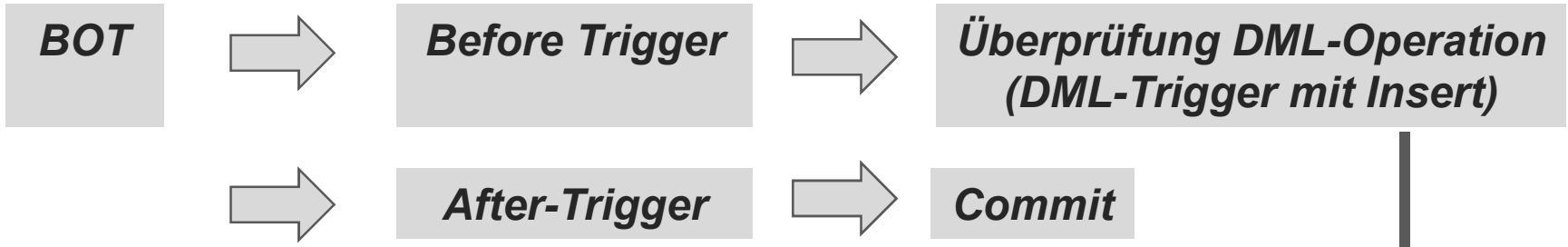
4. **Löschen der Einträge im Warenkorb**

DELETE FROM Warenkorb …

Aufrufreihenfolge



2. Lösungsmöglichkeit : Eine gespeicherte Prozedur und ein AFTER-Trigger



... Trigger After InsertON Bestellung:

2. Ermittlung der automatisch generierten Bestellnummer

(new.Bestellnummer)

3. Übertragung der Einträge aus dem Warenkorb in die Tabelle „Bestellposition“

INSERT INTO Bestellposition …

4. **Löschen der Einträge im Warenkorb**

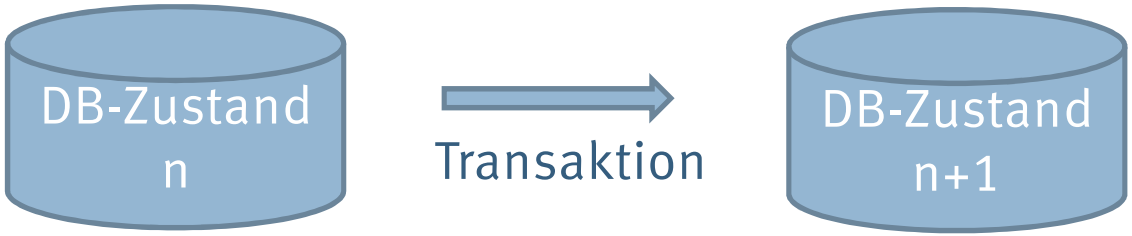
DELETE FROM Warenkorb …

Prozeduren und Trigger

Es wurden beispielhaft zwei Umsetzungsvarianten für einen Bestellvorgang vorgestellt:

- Umsetzung aller Abwicklungsschritte als gespeicherte Prozedur aus mehreren Schritten, umgesetzt in einer einzelnen Transaktion

- Umsetzung als gespeicherte Prozedur mit darauf folgendem After Insert Trigger



Transaktionen

Isolierte Ausführung

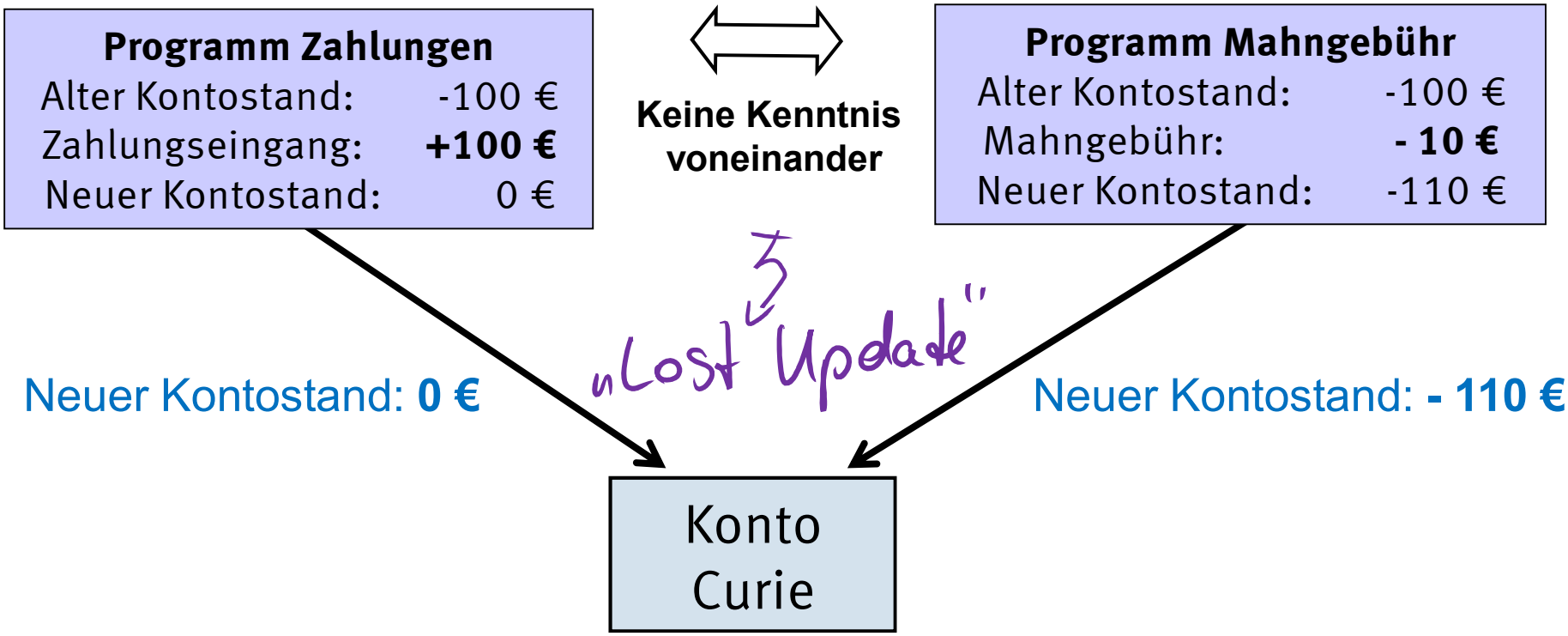
ACID-Eigenschaften von Transaktionen

1. Atomarität (***A**tomicity)*
2. Konsistenz *(**C**onsistency)*
3. Isoliertheit (***I**solation)*
4. Dauerhaftigkeit (**D***urability)*

Problemstellung Mehrbenutzerbetrieb

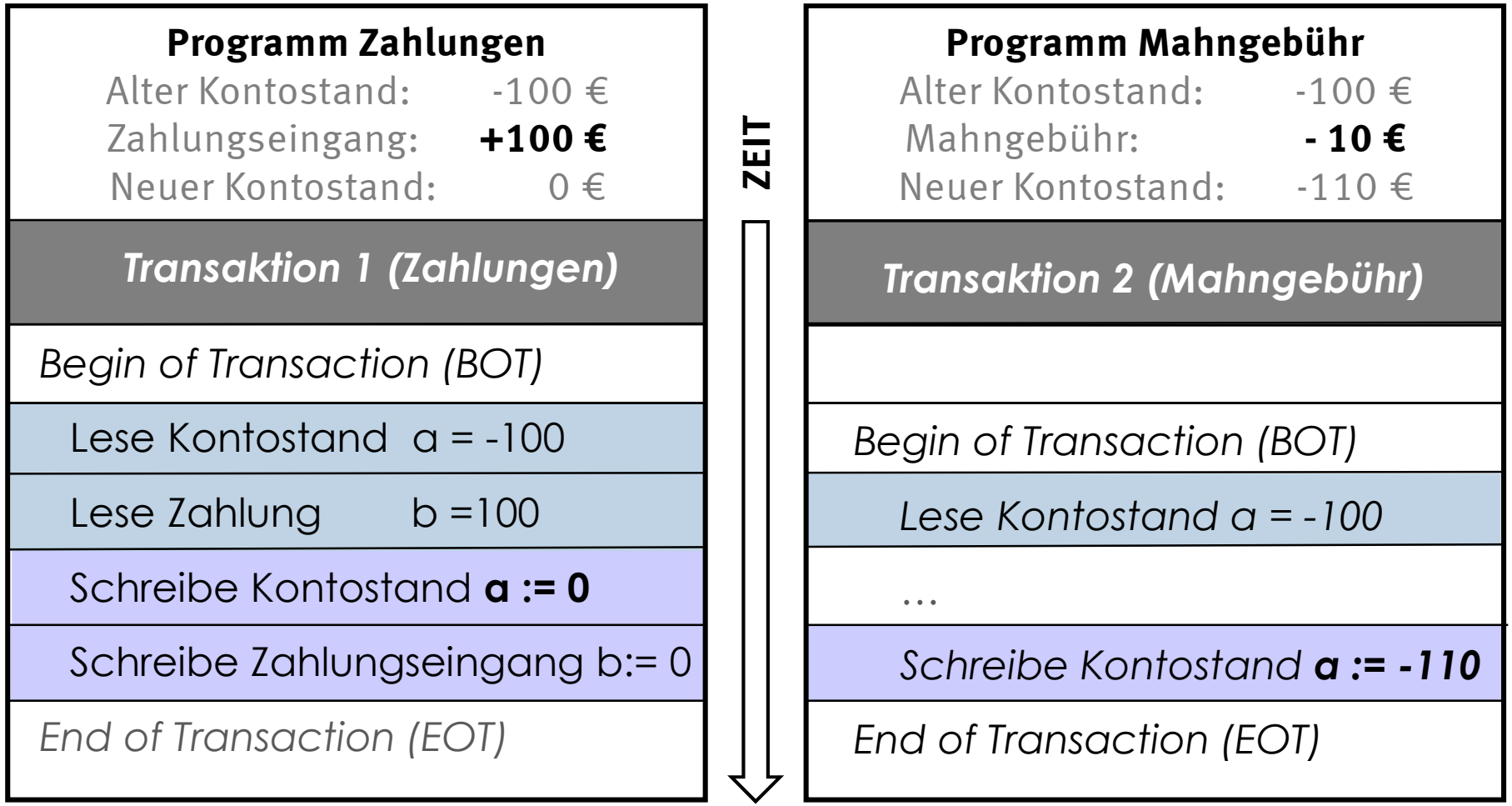
Problem: Gleichzeitiger, ändernder Zugriff mehrerer Benutzer auf eine Dateneinheit

Zielsetzung: Ablaufintegrität, Wahrung der Korrektheit von Daten



Lost Update

Lost Update: Änderung der Transaktion 1 wird durch Transaktion 2 überschrieben, da beide auf gemeinsame Ressourcen zugreifen.

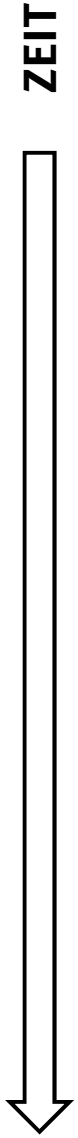


Lösungsansatz:

Zugriffs-Sperren und DBMS-Synchronisation

Synchronisation von Transaktionen

Transaktion 1 (Zahlungen)
Begin of Transaction (BOT)
Lese Kontostand a = -100
Lese Zahlungseingang b =100
Schreibe Kontostand a:= 0
Schreibe Zahlungseingang b:= 0
End of Transaction (EOT)
…



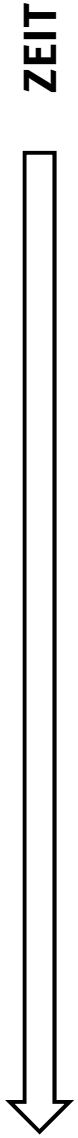
Synchronisation von Transaktionen durch Sperren:

Sperrmechanismen simulieren einen Ein-Benutzer-Betrieb auf der Datenbank.

Das DBMS sorgt so durch adäquate Synchronisation der Datenzugriffe für einen korrekten Ablauf.

Synchronisation von Transaktionen

Transaktion 1 (Zahlungen)
Begin of Transaction (BOT)
Schreibsperre a setzen
Lese Kontostand a = -100
Schreibsperre b setzen
Lese Zahlungseingang b =100
Schreibe Kontostand a:= 0
Schreibe Zahlungseingang b:= 0
Die Sperren auf a und b freigeben
End of Transaction (EOT)
…

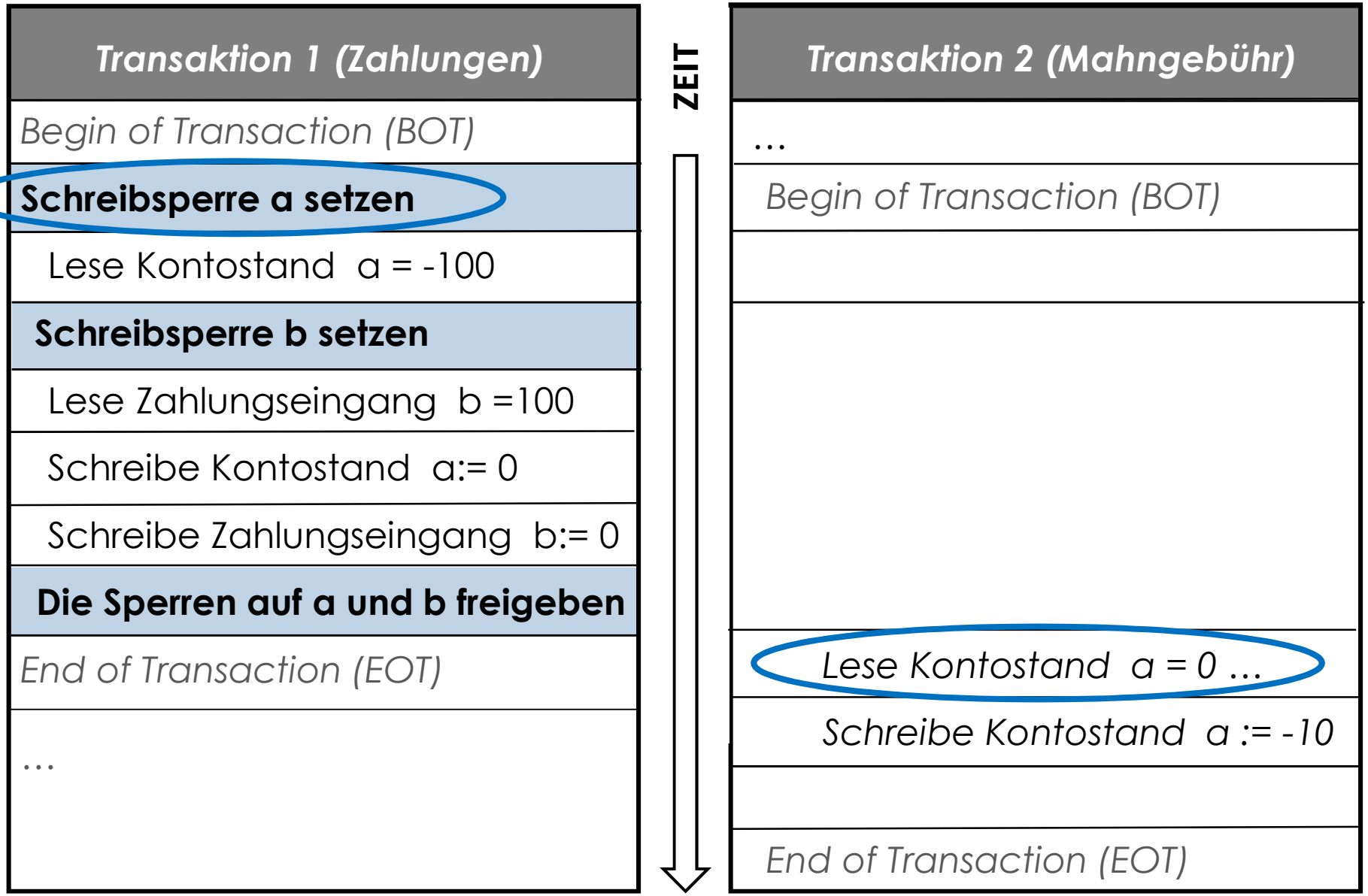


Synchronisation von Transaktionen durch Sperren:

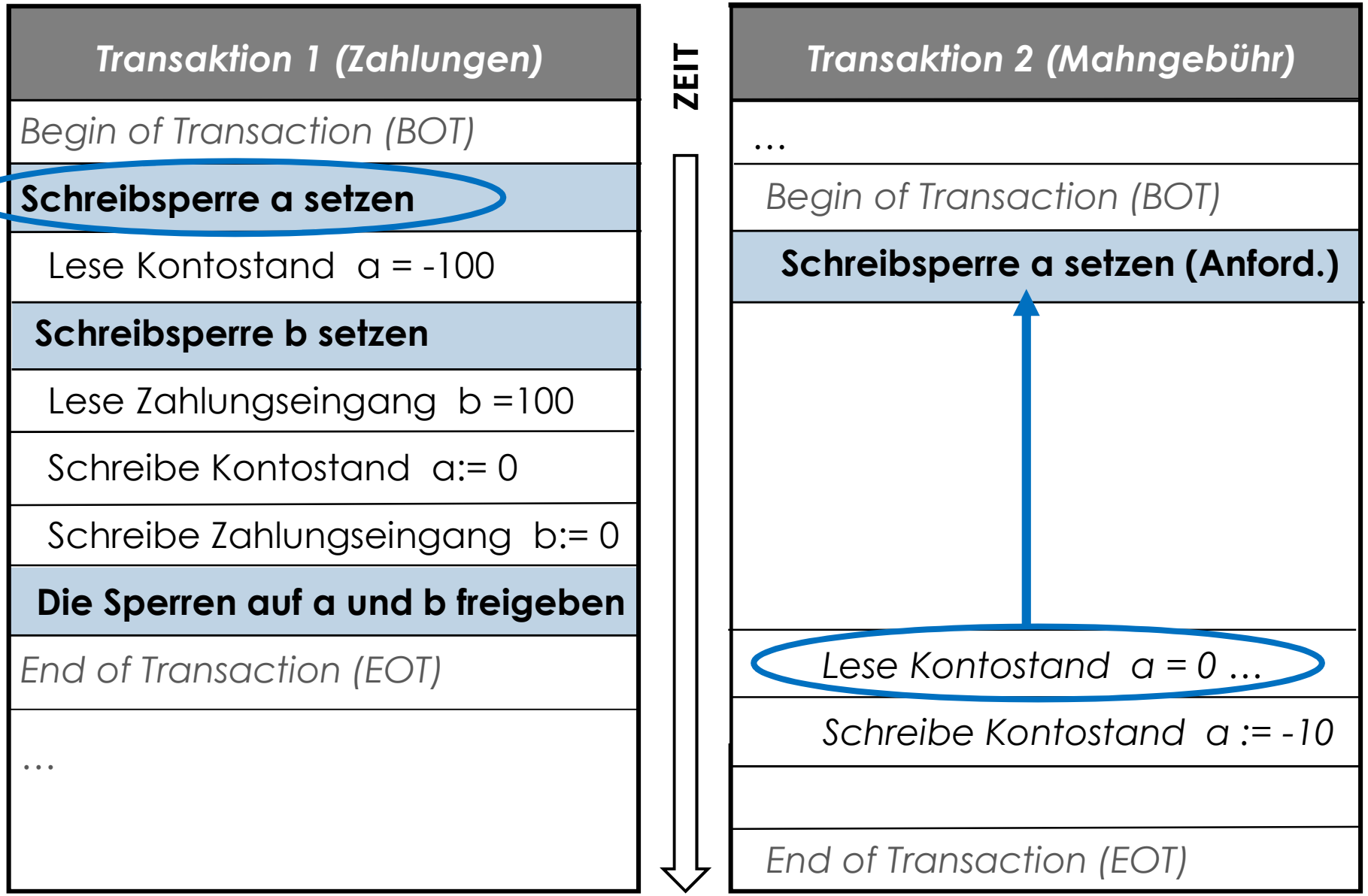
Sperrmechanismen simulieren einen Ein-Benutzer-Betrieb auf der Datenbank.

Das DBMS sorgt so durch adäquate Synchronisation der Datenzugriffe für einen korrekten Ablauf.

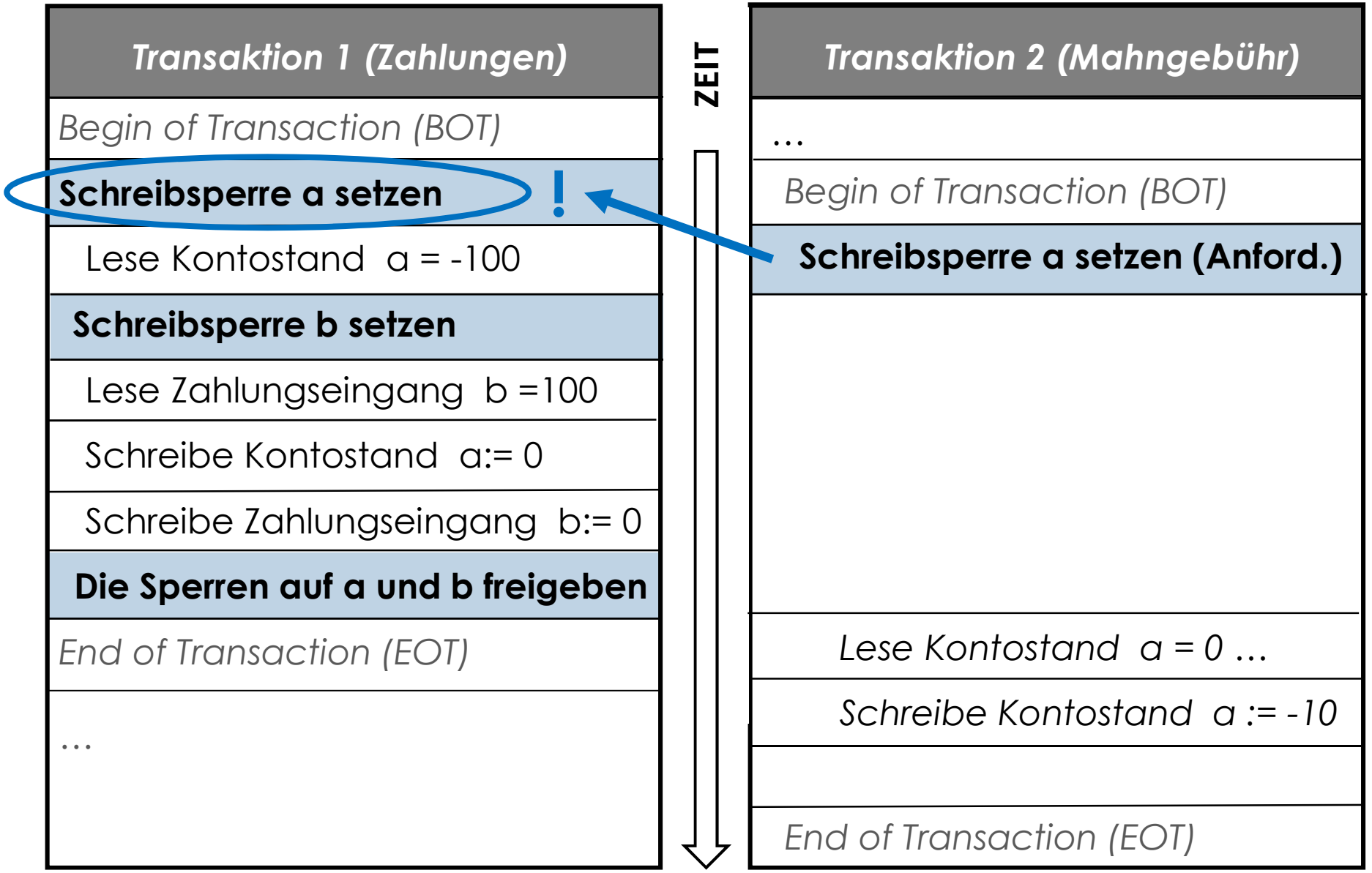
Synchronisation von Transaktionen



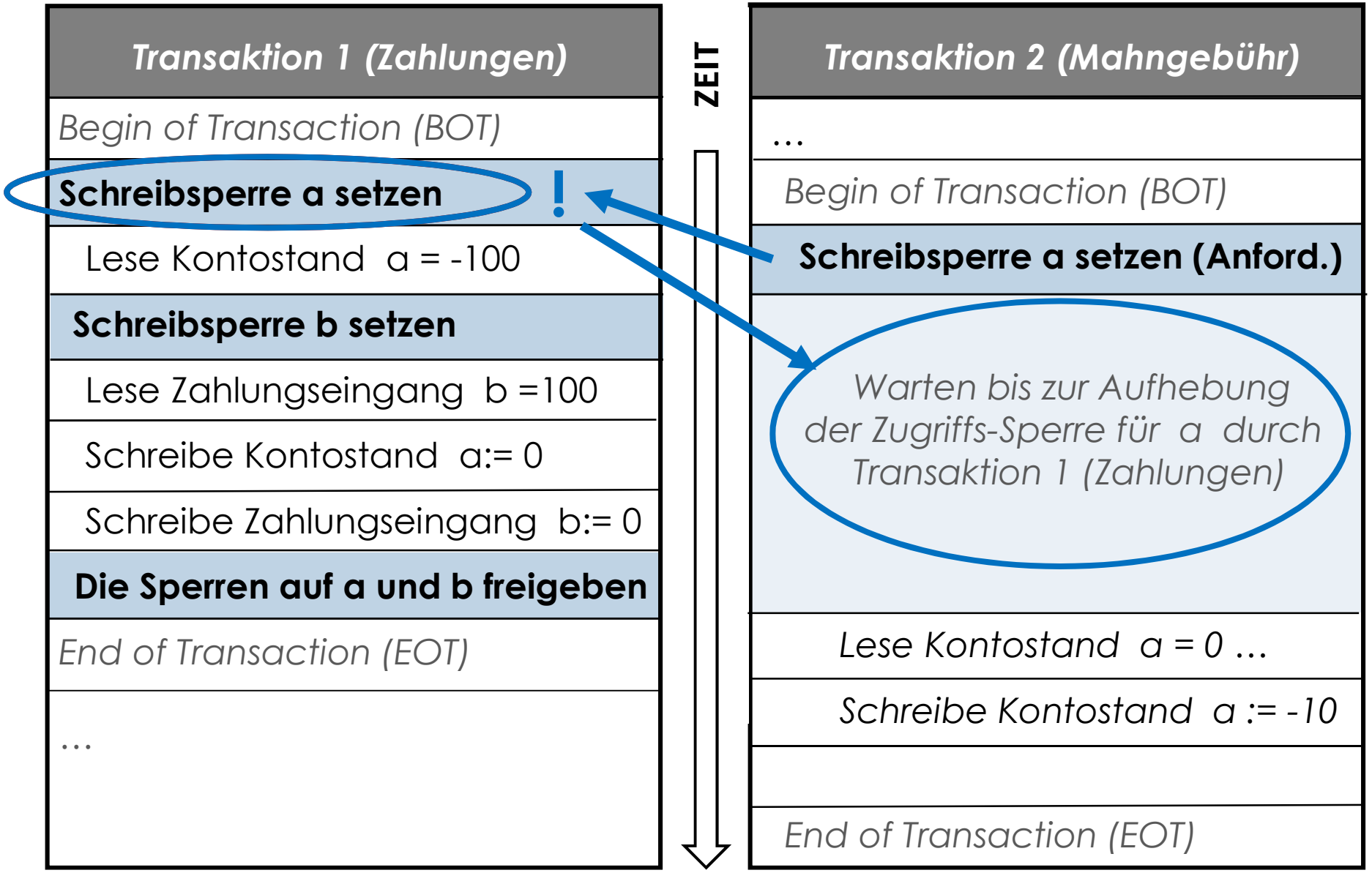
Synchronisation von Transaktionen



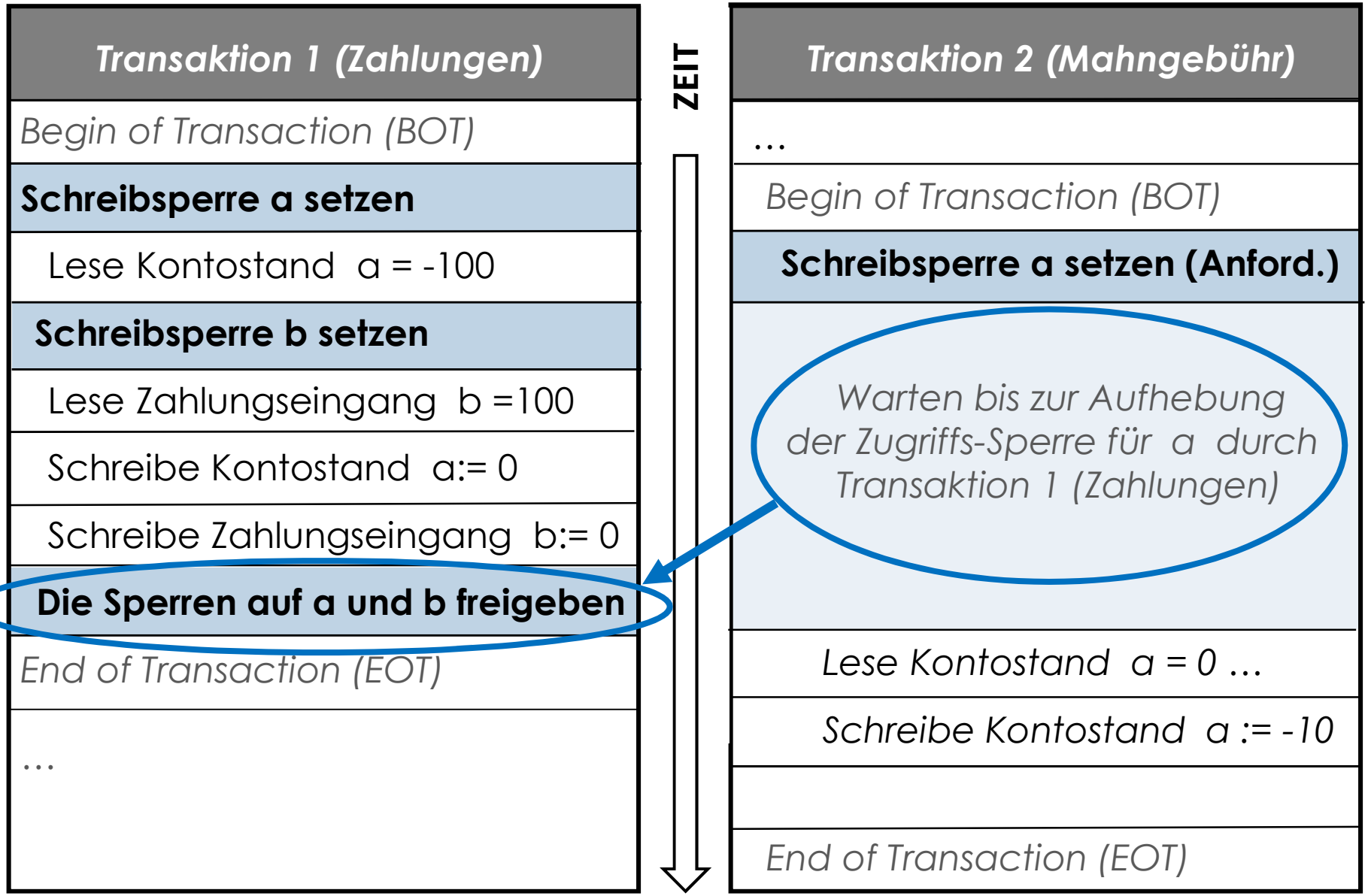
Synchronisation von Transaktionen



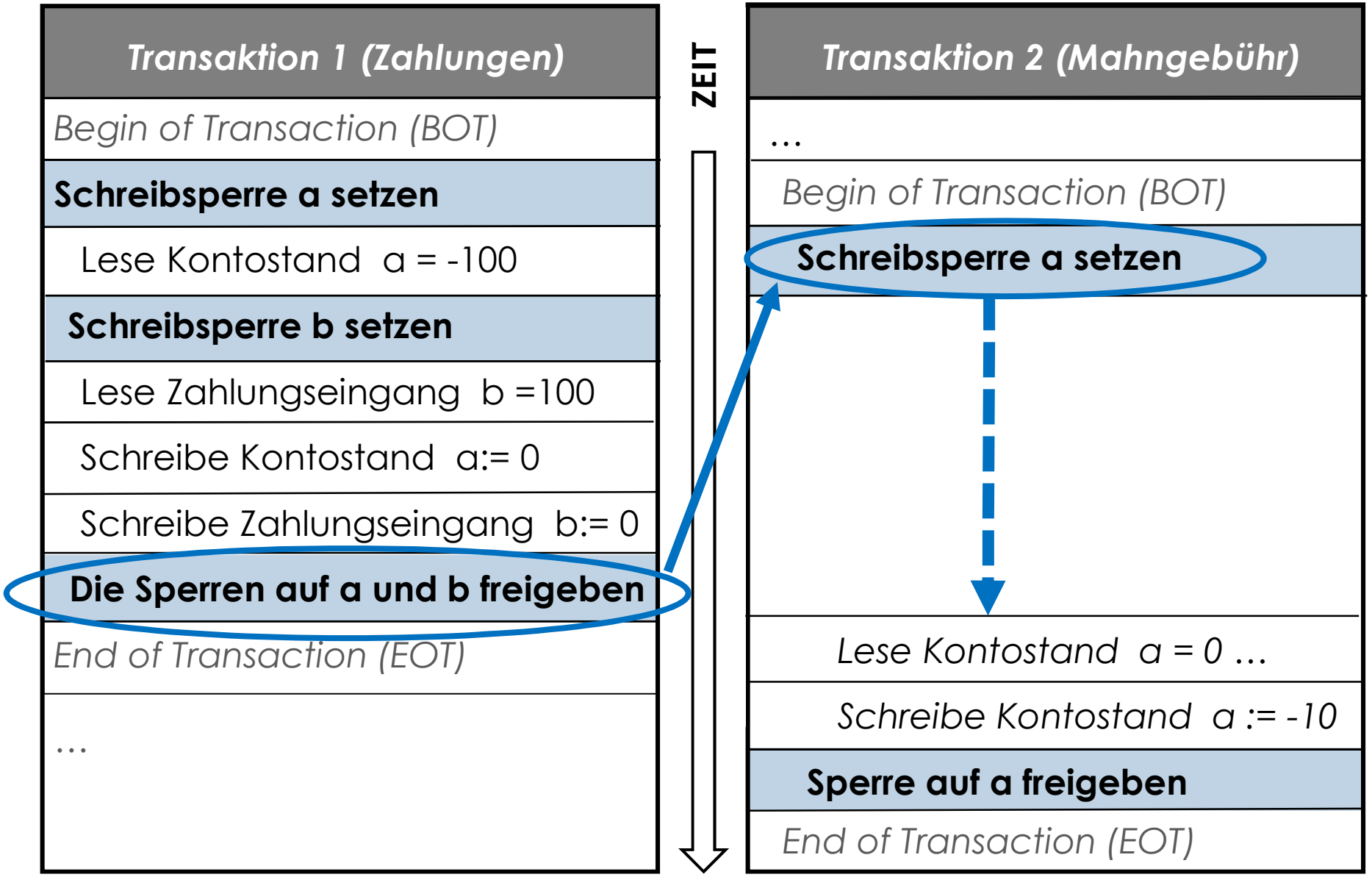
Synchronisation von Transaktionen



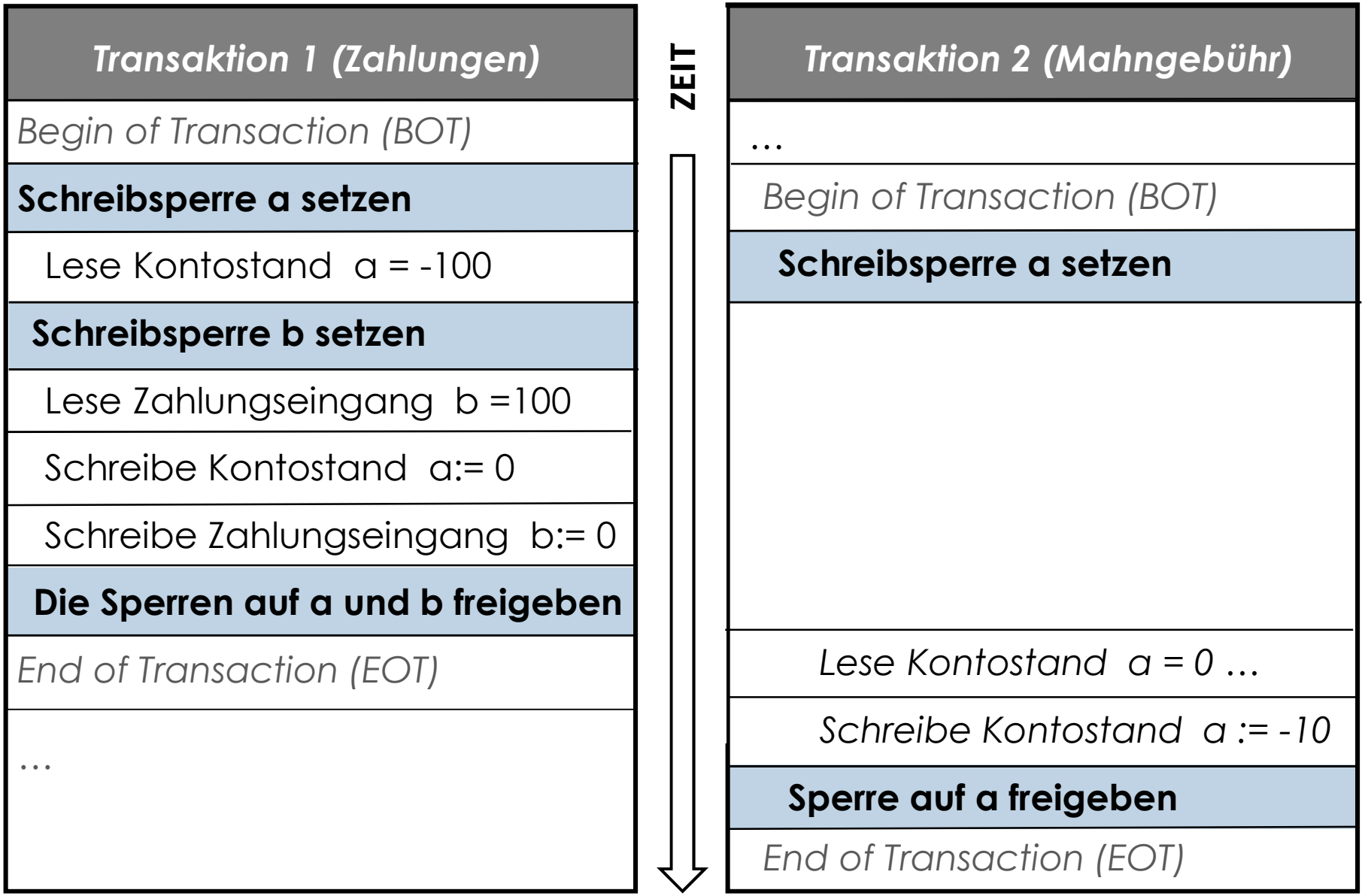
Synchronisation von Transaktionen



Synchronisation von Transaktionen



Synchronisation von Transaktionen



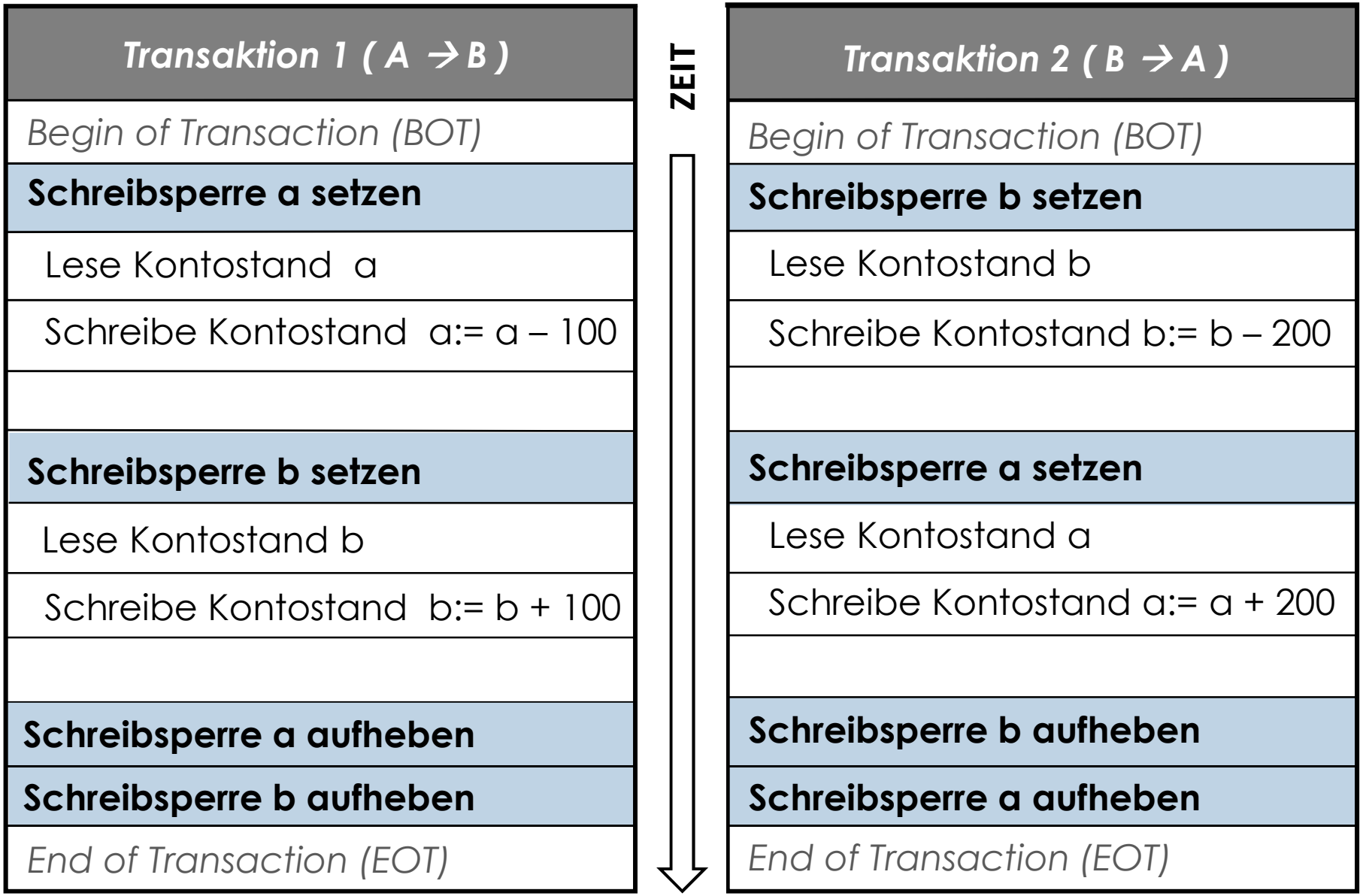
Problemstellung Dead-Locks:

Wie Zugriffs-Sperren den Programmablauf stören können

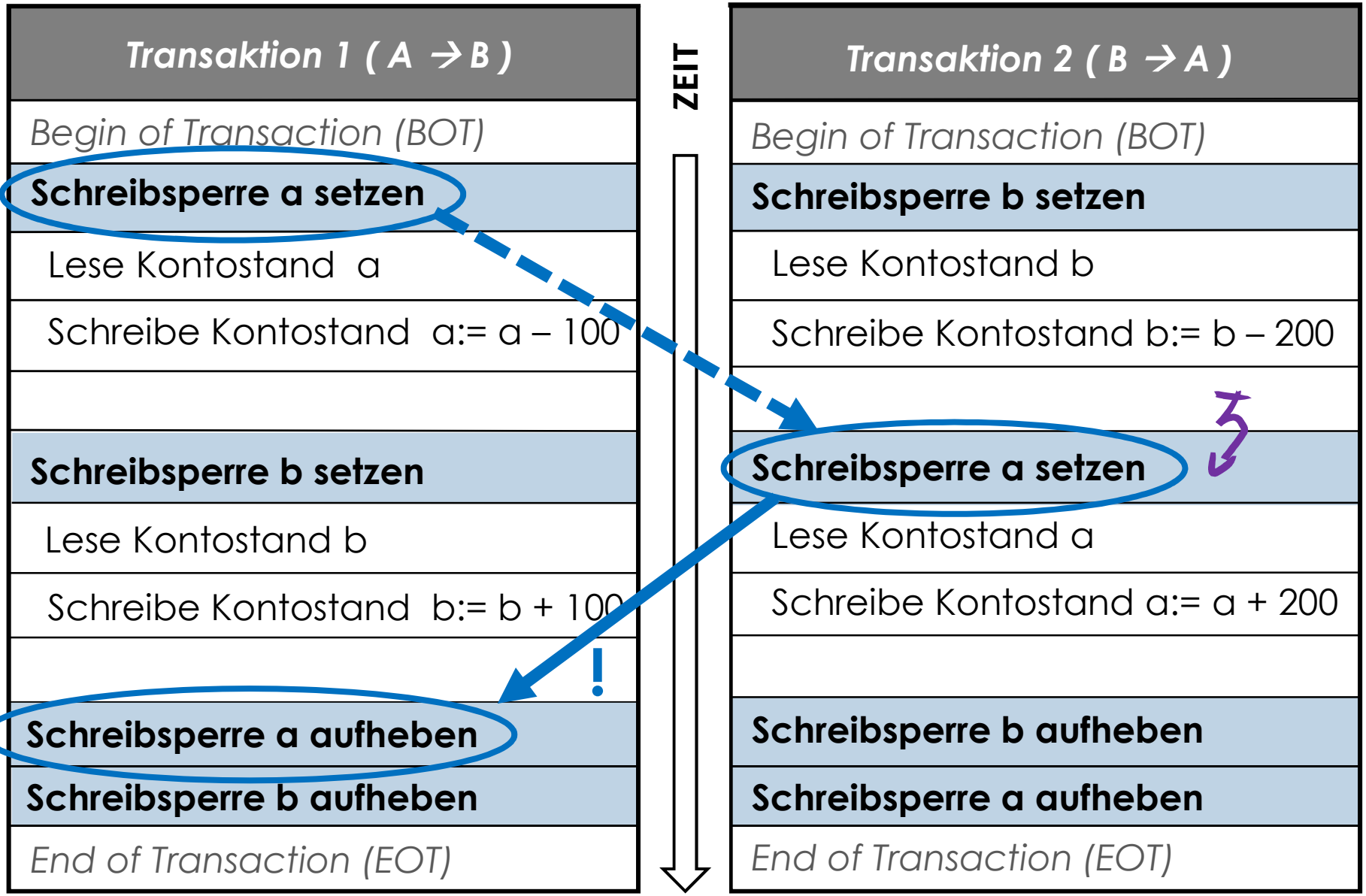
Dead-Lock

Transaktion 1 (A → B)	ZEIT	*Transaktion 2 (B → A)*
Begin of Transaction (BOT)		*Begin of Transaction (BOT)*
Lese Kontostand a		Lese Kontostand b
Schreibe Kontostand a:= a – 100		Schreibe Kontostand b:= b – 200
Lese Kontostand b		Lese Kontostand a
Schreibe Kontostand b:= b + 100		Schreibe Kontostand a:= a + 200
End of Transaction (EOT)		*End of Transaction (EOT)*

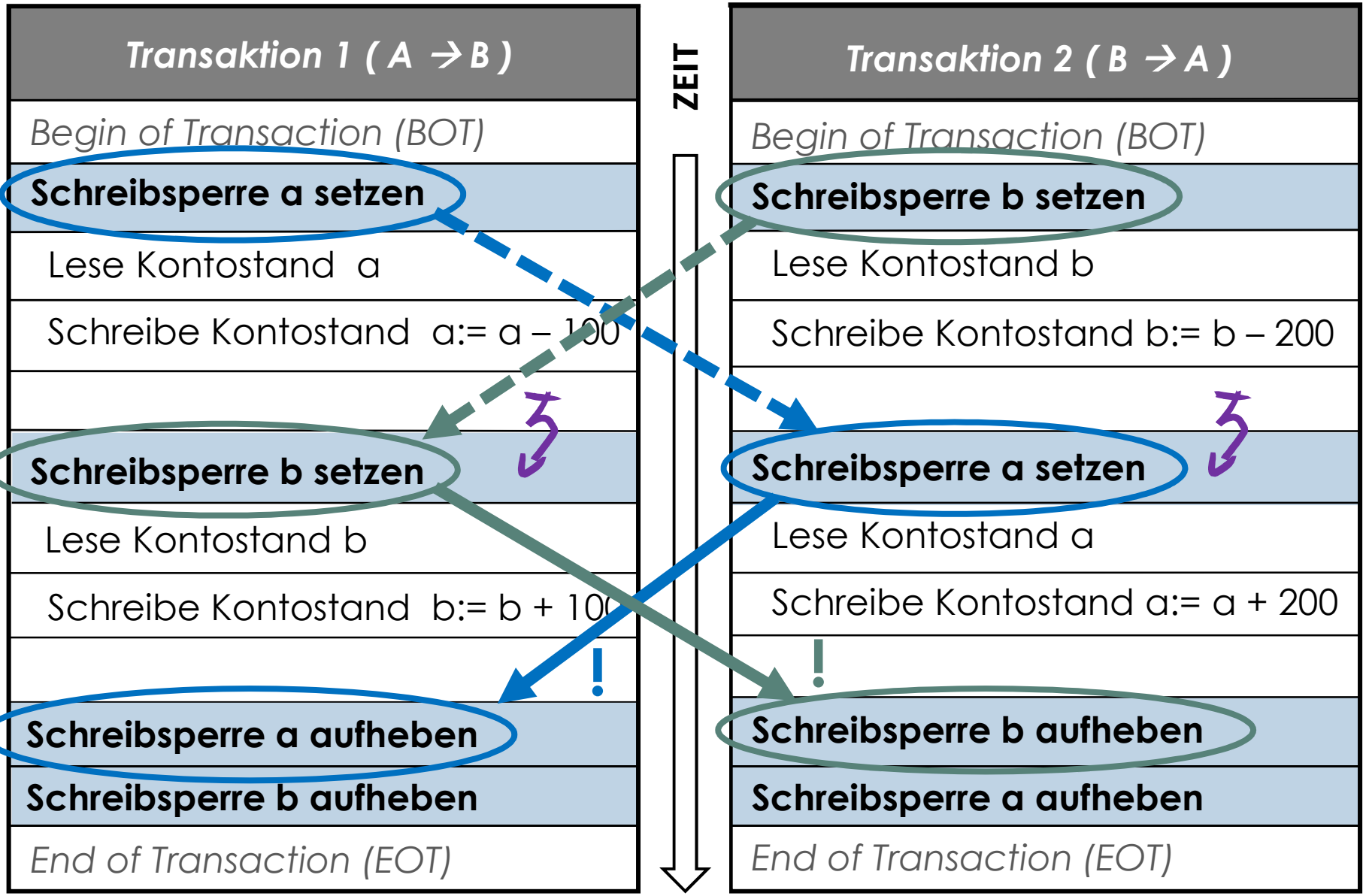
Dead-Lock


Transaktion 1 (A → B)
Begin of Transaction (BOT)
Schreibsperre a setzen
Lese Kontostand a
Schreibe Kontostand a:= a – 100
Schreibsperre b setzen
Lese Kontostand b
Schreibe Kontostand b:= b + 100
Schreibsperre a aufheben
Schreibsperre b aufheben
End of Transaction (EOT)
ZEIT
Transaktion 2 (B → A)
Begin of Transaction (BOT)
Schreibsperre b setzen
Lese Kontostand b
Schreibe Kontostand b:= b – 200
Schreibsperre a setzen
Lese Kontostand a
Schreibe Kontostand a:= a + 200
Schreibsperre b aufheben
Schreibsperre a aufheben
End of Transaction (EOT)

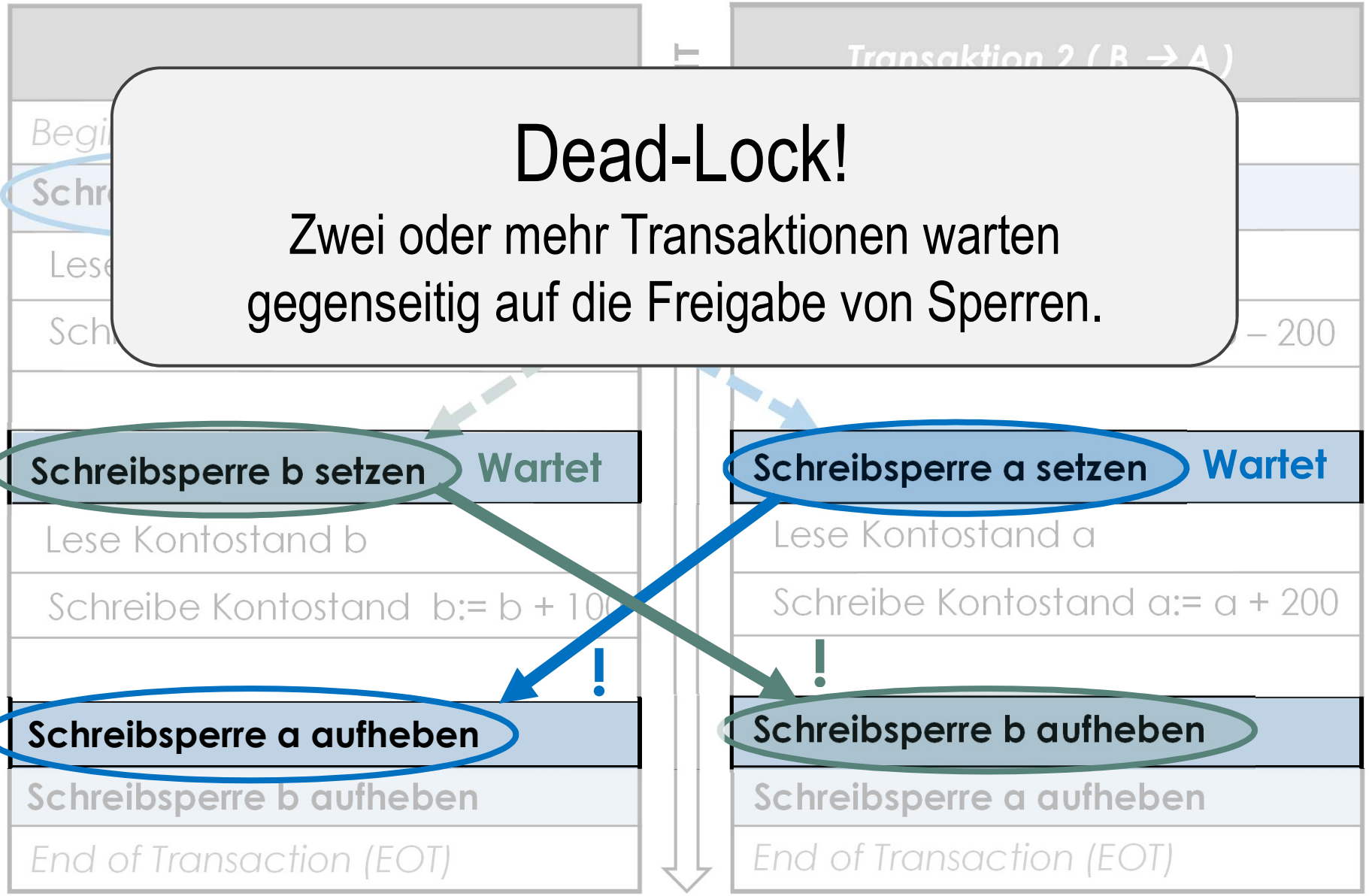
Dead-Lock


Transaktion 1 (A → B)
Begin of Transaction (BOT)
Schreibsperre a setzen
Lese Kontostand a
Schreibe Kontostand a:= a – 100
Schreibsperre b setzen
Lese Kontostand b
Schreibe Kontostand b:= b + 100
!
Schreibsperre a aufheben
Schreibsperre b aufheben
End of Transaction (EOT)
ZEIT
Transaktion 2 (B → A)
Begin of Transaction (BOT)
Schreibsperre b setzen
Lese Kontostand b
Schreibe Kontostand b:= b – 200
Schreibsperre a setzen
Lese Kontostand a
Schreibe Kontostand a:= a + 200
Schreibsperre b aufheben
Schreibsperre a aufheben
End of Transaction (EOT)

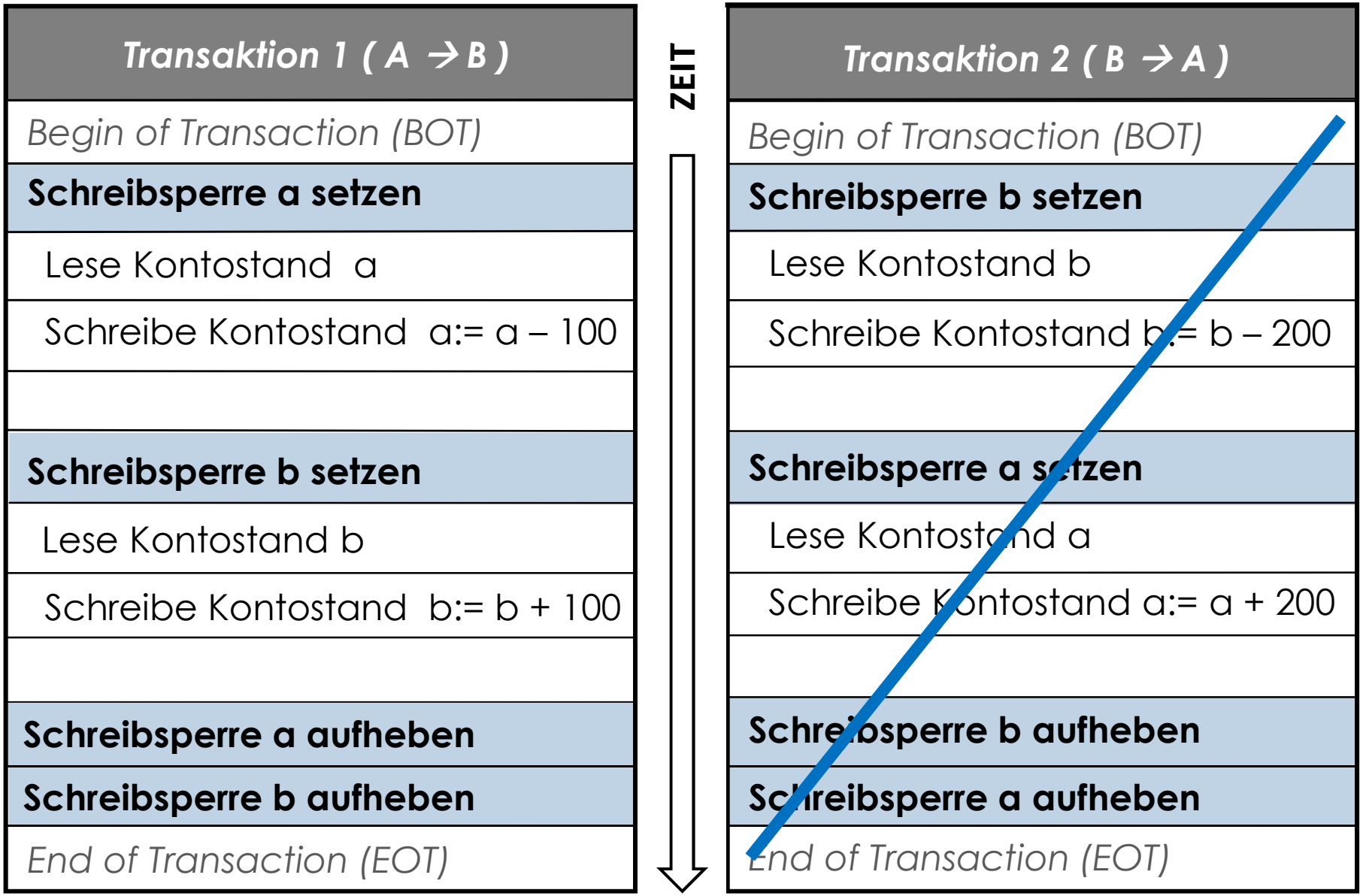
Dead-Lock


Transaktion 1 (A → B)
Begin of Transaction (BOT)
Schreibsperre a setzen
Lese Kontostand a
Schreibe Kontostand a:= a – 100
Schreibsperre b setzen
Lese Kontostand b
Schreibe Kontostand b:= b + 100
Schreibsperre a aufheben
Schreibsperre b aufheben
End of Transaction (EOT)
ZEIT
Transaktion 2 (B → A)
Begin of Transaction (BOT)
Schreibsperre b setzen
Lese Kontostand b
Schreibe Kontostand b:= b – 200
Schreibsperre a setzen
Lese Kontostand a
Schreibe Kontostand a:= a + 200
Schreibsperre b aufheben
Schreibsperre a aufheben
End of Transaction (EOT)

Dead-Lock


Dead-Lock!
Zwei oder mehr Transaktionen warten gegenseitig auf die Freigabe von Sperren.
Schreibsperre b setzen
Wartet
Lese Kontostand b
Schreibe Kontostand b:= b + 100
Schreibsperre a aufheben
Schreibsperre b aufheben
End of Transaction (EOT)
Schreibsperre a setzen
Wartet
Lese Kontostand a
Schreibe Kontostand a:= a + 200
Schreibsperre b aufheben
Schreibsperre a aufheben
End of Transaction (EOT)

Dead-Lock


Transaktion 1 (A → B)
Begin of Transaction (BOT)
Schreibsperre a setzen
Lese Kontostand a
Schreibe Kontostand a:= a – 100
Schreibsperre b setzen
Lese Kontostand b
Schreibe Kontostand b:= b + 100
Schreibsperre a aufheben
Schreibsperre b aufheben
End of Transaction (EOT)
ZEIT
Transaktion 2 (B → A)
Begin of Transaction (BOT)
Schreibsperre b setzen
Lese Kontostand b
Schreibe Kontostand b:= b – 200
Schreibsperre a setzen
Lese Kontostand a
Schreibe Kontostand a:= a + 200
Schreibsperre b aufheben
Schreibsperre a aufheben
End of Transaction (EOT)

Zusammenfassung und Ergänzung

ACID-Eigenschaften von Transaktionen

1	Atomarität	(*Atomicity*)
2	Konsistenz	(*Consistency*)
3	Isoliertheit	(*Isolation*)
4	Dauerhaftigkeit	(*Durability*)

Das Prinzip der **Isolation** ist wie die anderen ACID-Prinzipien eine wichtige Anforderung an ein DBMS, um parallele Transaktionen und den Mehrbenutzerbetrieb auf einer Datenbank zu ermöglichen.

Das ACID-Prinzip „Isolation“:

Trennung von Transaktionen, so dass eine Transaktion nicht von einer parallel ablaufenden Transaktion in einen undefinierten Zustand gebracht werden kann, bspw. durch (unkoordinierte) Änderung gemeinsam genutzter oder Lesen unvollständig bearbeiteter Daten

Ziel der Datenbanküberwachung:

Ablaufintegrität im Mehrbenutzerbetrieb - Wahrung der Korrektheit von Daten zu jedem Zeitpunkt bei gleichzeitigen, ändernden Zugriffsversuchen auf eine Dateneinheit von verschiedenen Stellen aus

Probleme bei mangelnder Transaktions-Isolation

- **Lost Updates (vorgestellt)**:
  Zwei Transaktionen versuchen parallel dieselben Daten zu ändern. Die spätere Änderung setzt sich durch und überschreibt die frühere.
- **Dirty Read**: Veränderliche Daten einer noch nicht vollständig abgeschlossenen Transaktion werden von einer anderen Transaktion vor Abschluss gelesen.
- **Non-Repeatable Read**: Wiederholte Lesevorgänge liefern unterschiedliche Ergebnisse.

... und weitere.

Lösungsansatz:

Synchronisation von Transaktionen durch Sperren

Sperrmechanismen simulieren einen Ein-Benutzer-Betrieb auf der Datenbank und isolieren zusammengehörige Transaktionsabläufe von anderen Zugriffen. Das DBMS sorgt somit durch adäquate Synchronisation der Datenzugriffe für einen korrekten Ablauf (Ablaufintegrität) auch im Mehrbenutzer-Betrieb.

Dead-Lock

Durch Sperrmechanismen kann ein neues Problem entstehen:
Wenn mehrere Transaktionen gleichzeitig auf die Freigabe von Sperren durch die jeweils anderen warten, kann es passieren, dass sie sich gegenseitig in der Weiterverarbeitung blockieren.

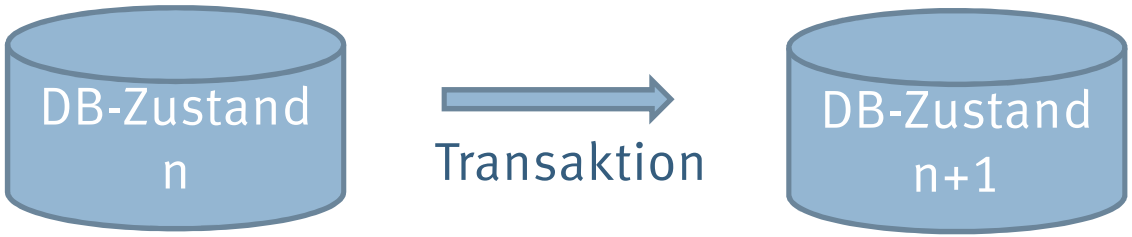
Lösung:
Ein auf diese Weise „festgefahrener“ Zustand kann durch das DBMS aufgelöst werden, indem es bei Bedarf einzelne Transaktionen zurück setzt (Rollback). Sperren werden so freigegeben.
Durch Einstufung von Transaktionen nach Priorität können hierbei bestimmte Transaktionen bevorzugt abgearbeitet werden.

Hinweise für Java-Programme zum Thema Isolation

- Serialisierung von Transaktionen erzwingen: Setzen des Transaktions-Isolationslevels

  con.setTransactionIsolation(con.*TRANSACTION_SERIALIZABLE*);

- Bei Timeout aufgrund einer nicht erhaltenen Sperre:
  - Exception wird geworfen
  - Wiederausführen der Transaktion ist erforderlich (→ im Catch-Block anstoßen)



Transaktionen

Dauerhaftigkeit

ACID-Eigenschaften von Transaktionen

1. Atomarität (*Atomicity*)
2. Konsistenz (*Consistency*)
3. Isoliertheit (*Isolation*)
4. Dauerhaftigkeit (*Durability*)

Problemstellung:

Fehler während des Datenbankbetriebs

Fehlerarten

Transaktionsfehler

- Rollback durch die Anwendung
- Verletzung von Integritätsbedingungen
- Verletzung von Zugriffsrechten
- Abbruch zur Auflösung einer Blockade (Dead-Locks)

Systemfehler

- Stromausfall
- Betriebssystemausfall
- Sonstiger Hardware- / Speicherfehler

Plattenfehler

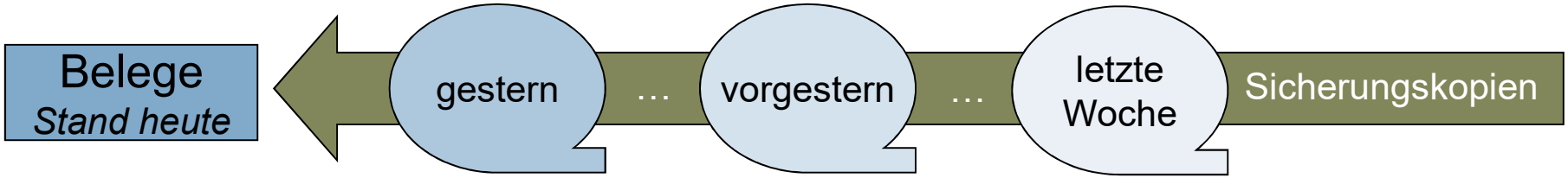
- Physischer Defekt der Festplatten-Oberfläche
- Fehlfunktion des Controllers

Arten der Datensicherung

Backup, Restore, Recovery

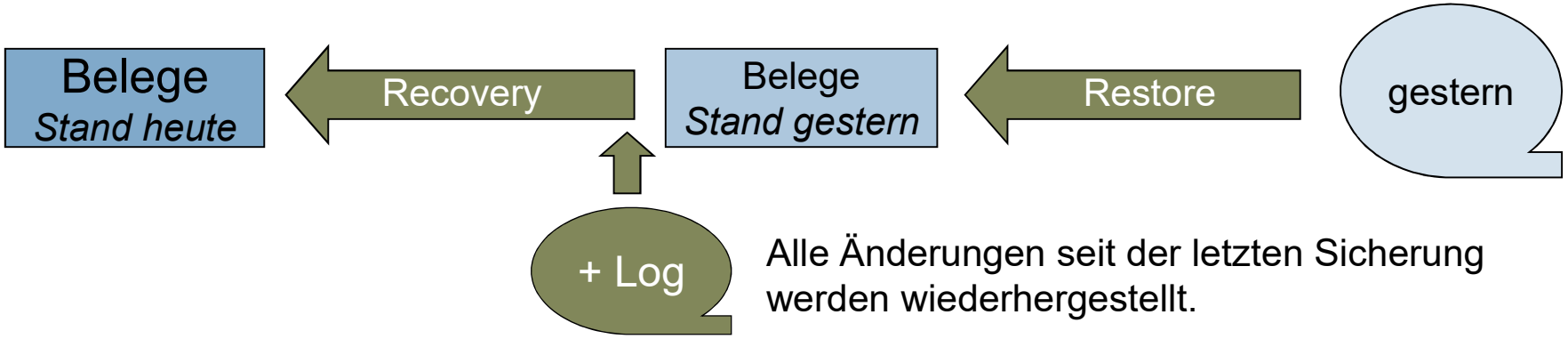
Datensicherungsarten

Dateisystem



Problem: Zwischenzeitliche Änderungen gehen bei Wiederherstellung verloren.

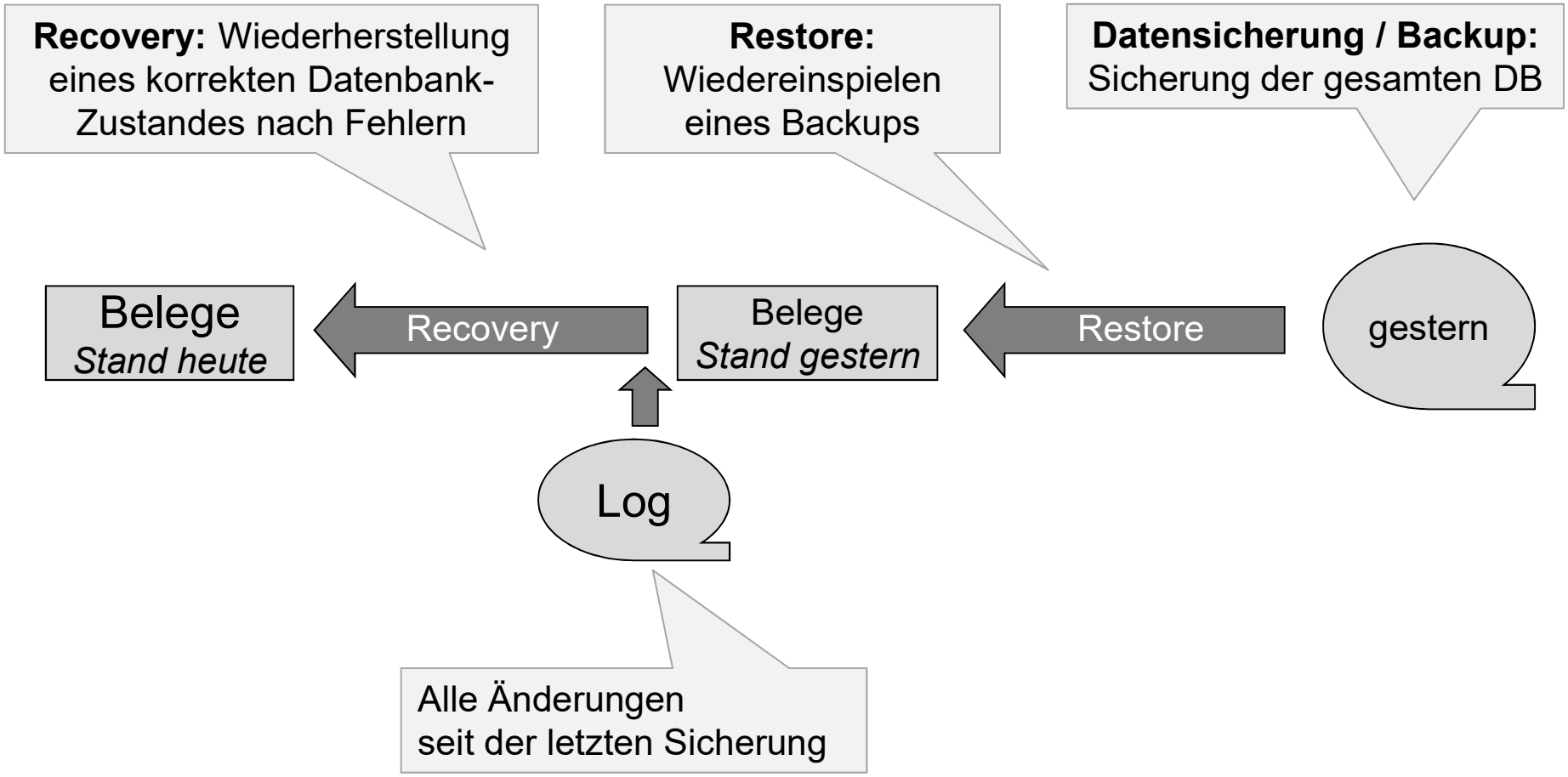
Datenbank



Alle Änderungen seit der letzten Sicherung werden wiederhergestellt.

Lösung: Nach dem Wiederherstellen der Sicherungsdatei (**Restore**) werden zwischenzeitliche Änderungen mit Hilfe des Logs wiederhergestellt (**Recovery**).

Begriffe

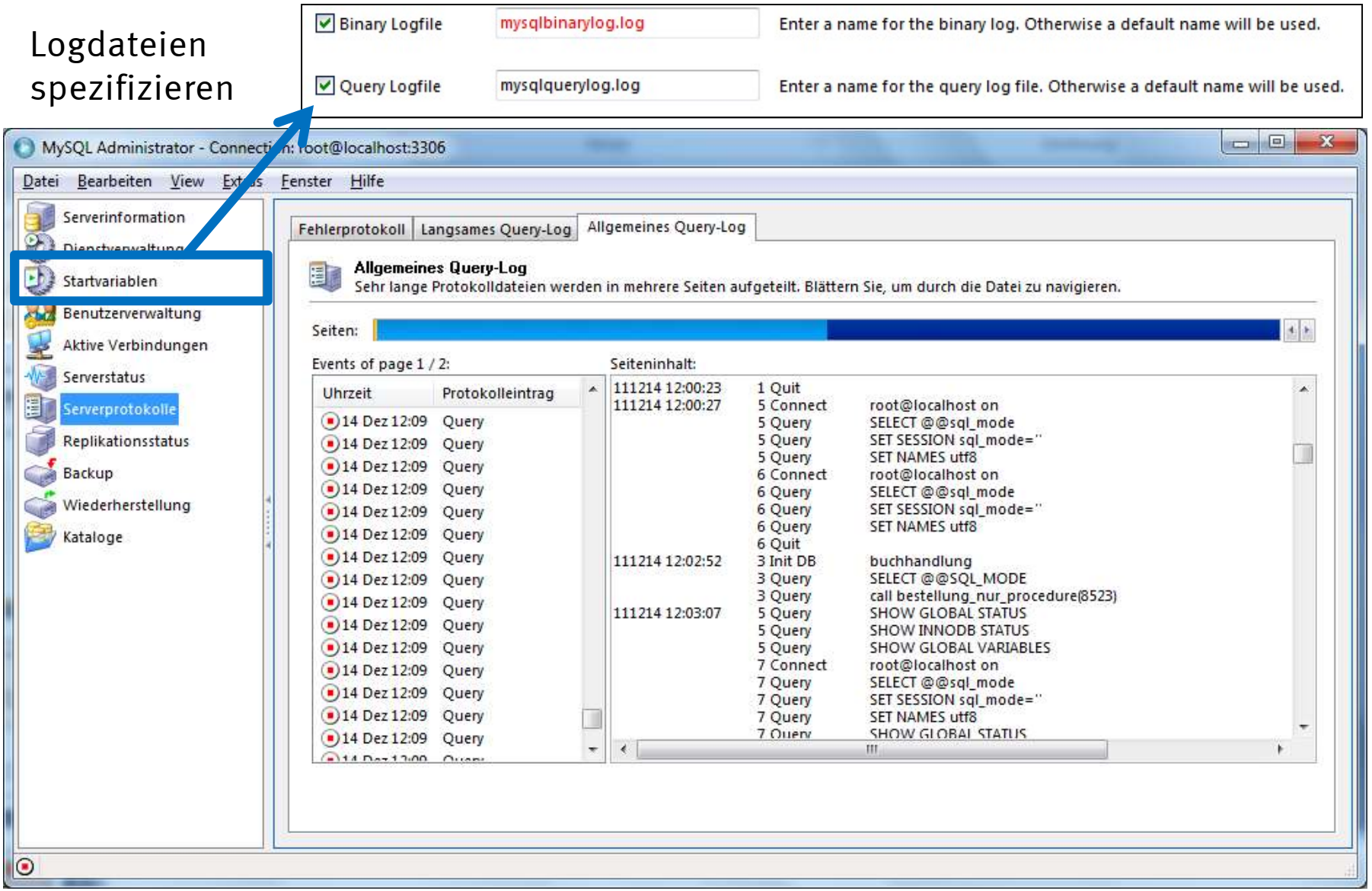

Recovery: Wiederherstellung eines korrekten Datenbank-Zustandes nach Fehlern
Restore: Wiedereinspielen eines Backups
Datensicherung / Backup: Sicherung der gesamten DB
Belege *Stand heute*
Recovery
Belege *Stand gestern*
Restore
gestern
Log
Alle Änderungen seit der letzten Sicherung

Beispiel

Query-Logdateien in MySQL

Query-Logdateien in MySQL

Logdateien spezifizieren



Anm.: Zugriff auf Serverlogs nur auf localhost, MySQL muss als Service gestartet sein.

Beispiel einer Änderungs-Logdatei (MySQL)

Das DBMS protokolliert alle Änderungen der Datenbasis im DataLog (Logbuch)

Beispiel

```
1  Begin of Transaction (BOT)
     Einfügen Kundennummer 2310, Saldo=0
     Schreibe Saldo = -100
     Commit
   End of Transaction (EOT)
```

Binäre Logdatei von MySQL:

TransaktionsID

```
# at 1966
#111214 14:09:26 server id 1  end_log_pos 2034  Query   thread_id=10    exec_tim
e=0     error_code=0
1 SET TIMESTAMP=1323868166/*!*/;
BEGIN
/*!*/;
# at 2034
#111214 14:09:26 server id 1  end_log_pos 2133  Query   thread_id=10    exec_tim
e=0     error_code=0
SET TIMESTAMP=1323868166/*!*/;
INSERT INTO konten VALUES (2310,0.0)
/*!*/;
# at 2133
#111214 14:09:26 server id 1  end_log_pos 2250  Query   thread_id=10    exec_tim
e=0     error_code=0
SET TIMESTAMP=1323868166/*!*/;
Update konten SET Saldo=-100.0 WHERE Kundennummer=2310
/*!*/;
# at 2250
#111214 14:09:26 server id 1  end_log_pos 2277  Xid = 341
COMMIT/*!*/;
DELIMITER ;
```

Anm.: Die binäre Logdatei von MySQL unterstützt nur serialisierte Transaktionen. Ausgelesen mit dem MySQL-Tool mysqlbinlog

Beispiel einer Änderungs-Logdatei (MySQL)

Das DBMS protokolliert alle Änderungen der Datenbasis im DataLog (Logbuch)

Beispiel

```
Begin of Transaction (BOT)
2  Einfügen Kundennummer 2310, Saldo=0
   Schreibe Saldo = -100
   Commit
End of Transaction (EOT)
```

Binäre Logdatei von MySQL:

```
# at 1966
#111214 14:09:26 server id 1  end_log_pos 2034  Query   thread_id=10    exec_tim
e=0     error_code=0
SET TIMESTAMP=1323868166/*!*/;
BEGIN
/*!*/;
# at 2034
#111214 14:09:26 server id 1  end_log_pos 2133  Query   thread_id=10    exec_tim
e=0     error_code=0
SET TIMESTAMP=1323868166/*!*/;
2 INSERT INTO konten VALUES (2310,0.0)
/*!*/;
# at 2133
#111214 14:09:26 server id 1  end_log_pos 2250  Query   thread_id=10    exec_tim
e=0     error_code=0
SET TIMESTAMP=1323868166/*!*/;
Update konten SET Saldo=-100.0 WHERE Kundennummer=2310
/*!*/;
# at 2250
#111214 14:09:26 server id 1  end_log_pos 2277  Xid = 341
COMMIT/*!*/;
DELIMITER ;
```

Anm.: Die binäre Logdatei von MySQL unterstützt nur serialisierte Transaktionen. Ausgelesen mit dem MySQL-Tool mysqlbinlog

Beispiel einer Änderungs-Logdatei (MySQL)

Das DBMS protokolliert alle Änderungen der Datenbasis im DataLog (Logbuch)

Beispiel

```
Begin of Transaction (BOT)
 Einfügen Kundennummer 2310, Saldo=0
3 Schreibe Saldo = -100
 Commit
End of Transaction (EOT)
```

Binäre Logdatei von MySQL:

```
# at 1966
#111214 14:09:26 server id 1  end_log_pos 2034  Query   thread_id=10    exec_tim
e=0     error_code=0
SET TIMESTAMP=1323868166/*!*/;
BEGIN
/*!*/;
# at 2034
#111214 14:09:26 server id 1  end_log_pos 2133  Query   thread_id=10    exec_tim
e=0     error_code=0
SET TIMESTAMP=1323868166/*!*/;
INSERT INTO konten VALUES (2310,0.0)
/*!*/;
# at 2133
#111214 14:09:26 server id 1  end_log_pos 2250  Query   thread_id=10    exec_tim
e=0     error_code=0
SET TIMESTAMP=1323868166/*!*/;
3 Update konten SET Saldo=-100.0 WHERE Kundennummer=2310
/*!*/;
# at 2250
#111214 14:09:26 server id 1  end_log_pos 2277  Xid = 341
COMMIT/*!*/;
DELIMITER ;
```

Anm.: Die binäre Logdatei von MySQL unterstützt nur serialisierte Transaktionen. Ausgelesen mit dem MySQL-Tool mysqlbinlog

Beispiel einer Änderungs-Logdatei (MySQL)

Das DBMS protokolliert alle Änderungen der Datenbasis im DataLog (Logbuch)

Beispiel

```
Begin of Transaction (BOT)
 Einfügen Kundennummer 2310, Saldo=0
 Schreibe Saldo = -100
4 Commit
End of Transaction (EOT)
```

Binäre Logdatei von MySQL:

```
# at 1966
#111214 14:09:26 server id 1  end_log_pos 2034  Query   thread_id=10    exec_tim
e=0     error_code=0
SET TIMESTAMP=1323868166/*!*/;
BEGIN
/*!*/;
# at 2034
#111214 14:09:26 server id 1  end_log_pos 2133  Query   thread_id=10    exec_tim
e=0     error_code=0
SET TIMESTAMP=1323868166/*!*/;
INSERT INTO konten VALUES (2310,0.0)
/*!*/;
# at 2133
#111214 14:09:26 server id 1  end_log_pos 2250  Query   thread_id=10    exec_tim
e=0     error_code=0
SET TIMESTAMP=1323868166/*!*/;
Update konten SET Saldo=-100.0 WHERE Kundennummer=2310
/*!*/;
# at 2250
#111214 14:09:26 server id 1  end_log_pos 2277  Xid = 341
4 COMMIT/*!*/;
DELIMITER ;
```

Anm.: Die binäre Logdatei von MySQL unterstützt nur serialisierte Transaktionen. Ausgelesen mit dem MySQL-Tool mysqlbinlog

Beispiel einer Änderungs-Logdatei (MySQL)

Das DBMS protokolliert alle Änderungen der Datenbasis im DataLog (Logbuch)

Beispiel

1. Begin of Transaction (BOT)
2. **Einfügen** Kundennummer 2310, Saldo=0
3. **Schreibe** Saldo = -100
4. **Commit**

   End of Transaction (EOT)

Binäre Logdatei von MySQL:

```
# at 1966
#111214 14:09:26 server id 1  end_log_pos 2034  Query   thread_id=10    exec_tim
e=0     error_code=0
1 SET TIMESTAMP=1323868166/*!*/;
BEGIN
/*!*/;
# at 2034
#111214 14:09:26 server id 1  end_log_pos 2133  Query   thread_id=10    exec_tim
e=0     error_code=0
2 SET TIMESTAMP=1323868166/*!*/;
INSERT INTO konten VALUES (2310,0.0)
/*!*/;
# at 2133
#111214 14:09:26 server id 1  end_log_pos 2250  Query   thread_id=10    exec_tim
e=0     error_code=0
3 SET TIMESTAMP=1323868166/*!*/;
Update konten SET Saldo=-100.0 WHERE Kundennummer=2310
/*!*/;
# at 2250
#111214 14:09:26 server id 1  end_log_pos 2277  Xid = 341
4 COMMIT/*!*/;
DELIMITER ;
```

Anm.: Die binäre Logdatei von MySQL unterstützt nur serialisierte Transaktionen. Ausgelesen mit dem MySQL-Tool mysqlbinlog

Recovery Situationen

Situation beim Recovery

1 Abgeschlossene Transaktionen

- Die Transaktion ist abgeschlossen **und**
- die erfolgten Änderungen sind vollständig physikalisch gespeichert

→ **Recovery**: Keine Fehlerbehandlung erforderlich

2 Redo-Transaktionen

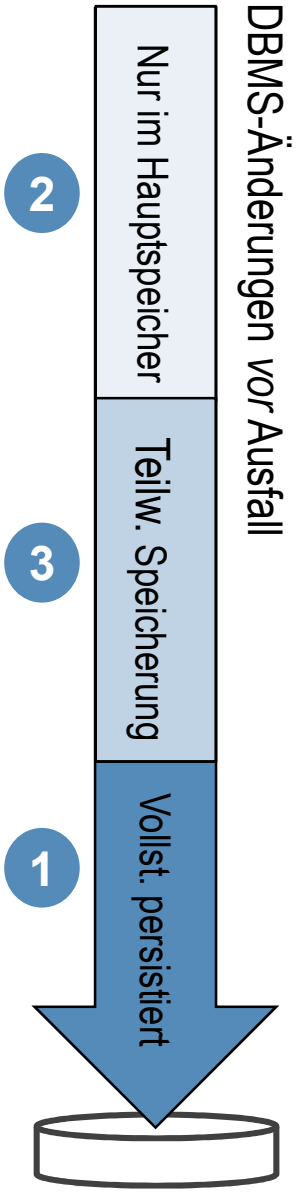
- Die Änderungen sind vor dem Fehler **nur** im (flüchtigen) Hauptspeicher durchgeführt worden, selbst wenn die Transaktion **vollständig** durchlaufen wurde
- Daher sind die Änderungen noch **nicht** physikalisch gespeichert worden

→ **Recovery:** Wiederholen (**Redo**) und Speichern der Änderungen

3 Undo-Transaktionen

- Die Transaktion ist **nicht** abgeschlossen worden.
- Teil-Änderungen können **dennoch** bereits im (flüchtigen) Hauptspeicher und/oder physikalisch zwischengespeichert worden sein.

→ **Recovery**: Rückgängigmachen (**Undo**) der Änderungen in umgekehrter Reihenfolge → **Rollback**



Recovery von System-/Plattenfehlern

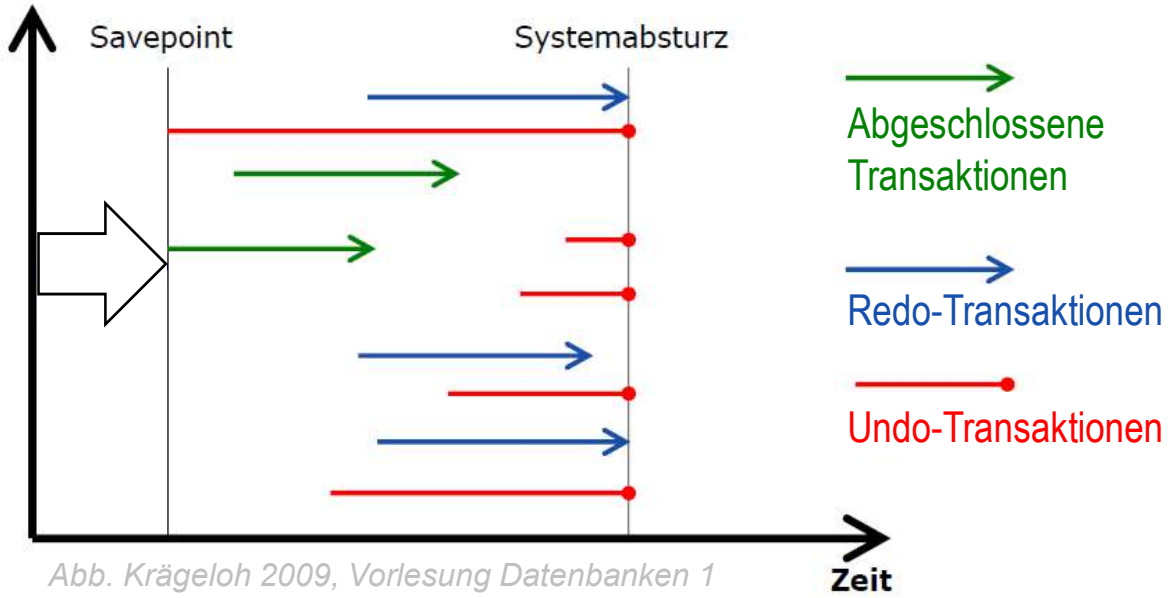


Abb. Krägeloh 2009, Vorlesung Datenbanken 1

Transaktionsart	Recovery-Maßnahme
Abgeschlossen	Nicht erforderlich
Redo	Erneutes Einspielen des Änderungen (Redo)
Undo	Zurücksetzung der Transaktionen (Undo) in umgekehrter Reihenfolge

Zusammenfassung

Dauerhaftigkeit

ACID-Eigenschaften von Transaktionen

1	Atomarität	(*Atomicity*)
2	Konsistenz	(*Consistency*)
3	Isoliertheit	(*Isolation*)
4	Dauerhaftigkeit	(*Durability*)

Das Prinzip der **Dauerhaftigkeit** stellt sicher, dass Ergebnisse von Transaktionen dauerhaft gespeichert und gesichert werden.

Wiederholung ACID

Regeln für Transaktionen		Transaktionsausführung
1 Atomarität	(*Atomicity*)	„ganz oder gar nicht“
2 Konsistenz	(*Consistency*)	Konsistenzerhaltend (Korrekter Inhalt der DB bei EOT)
3 Isoliertheit	(*Isolation*)	Isoliert (jede Transaktion für sich, Simulation Ein-Benutzer-Betrieb)
4 Dauerhaftigkeit	(*Durability*)	Dauerhafte Speicherung der Ergebnisse

DB-Fehlerarten (Beispiele)

- Transaktionsfehler
- Systemfehler
- Plattenfehler

DataLog
Logbuch / Logdatei(en)

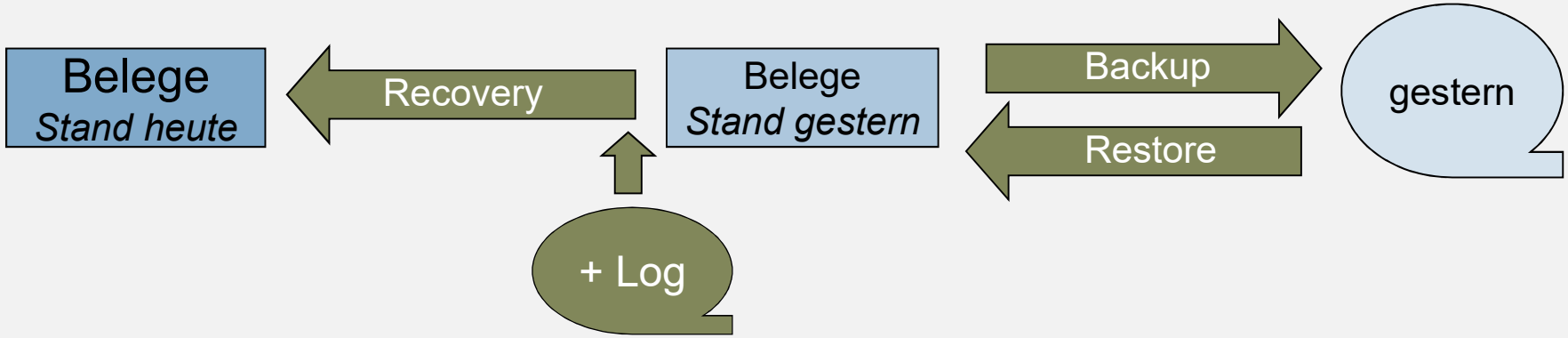
Das DBMS protokolliert alle Änderungen der Datenbasis im DataLog. Dieses wird im Falle eines notwendigen Recovery nach DB-Fehlern genutzt, um eine korrekte Datenbasis wiederherzustellen.

Wichtige Begriffe

Datensicherung / Backup: *Sicherung der gesamten DB*

Restore: *Wiedereinspielung des Backups*

Recovery: *Wiederherstellung eines korrekten DB-Zustandes nach Fehlern (mit Hilfe des DB-Logs)*



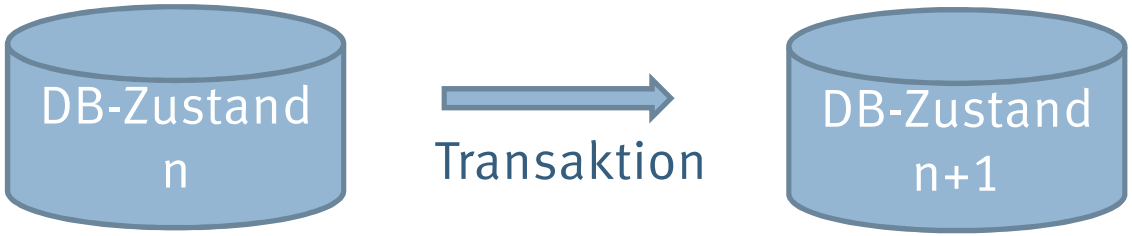
Nach dem Wiederherstellen der Sicherungsdatei (**Restore**) werden zwischenzeitliche Änderungen mit Hilfe des Logs wiederhergestellt (**Recovery**).

Situationen beim Recovery

- **Abgeschlossene Transaktionen**
  Transaktion abgeschlossen, Änderungen vollständg physikalisch gespeichert.
  → *Keine Fehlerbehandlung erforderlich.*

- **Redo-Transaktionen**
  Transaktion abgeschlossen, aber Änderungen waren noch nicht dauerhaft physikalisch gespeichert (nur im Hauptspeicher).
  → ***Recovery****: Wiederholung (Redo) der Änderungen.*

- **Undo-Transaktionen**
  Transaktion nicht abgeschlossen, Teil-Änderungen können jedoch bereits physikalisch (zwischen-)gespeichert vorliegen.
  → ***Recovery****: Rollback der Änderungen*



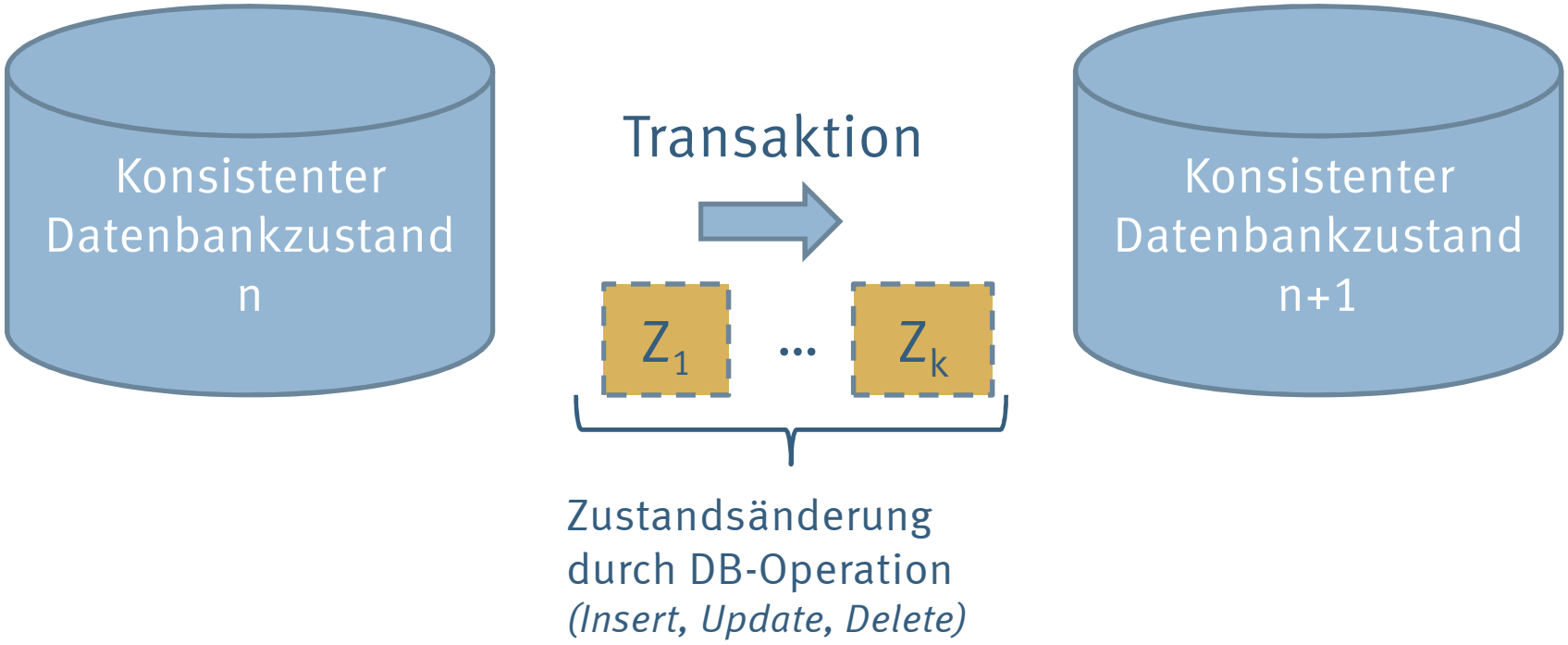
Transaktionen

Konsistenz

Transaktionsbegriff

Eine **Transaktion** überführt einen konsistenten Datenbankzustand in einen wiederum konsistenten Datenbankzustand.

Der initiale Zustand (Zustand n=0) der Datenbank ist konsistent.



Konsistenzüberwachung - Datenintegrität

Umsetzung mittels DBMS:

z.B. Primär/Fremdschlüssel

- Formulierung von (statischen) Integritätsregeln

  (siehe nächste Folie)

z.B. Trigger

- Zentrale Prüfprogramme überwachen die Regeleinhaltung

  (diese prüfen auf Plausibilität und formale Korrektheit)

- Zurückweisung von DB-Änderungen bei Integritäts-

  verletzung

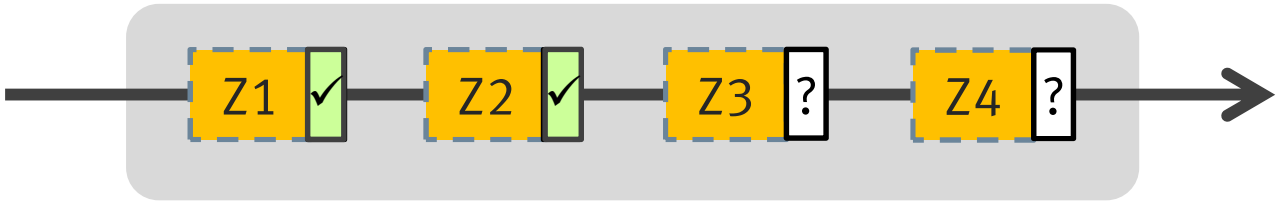
Rollback

Ziel: Semantische Datenintegrität

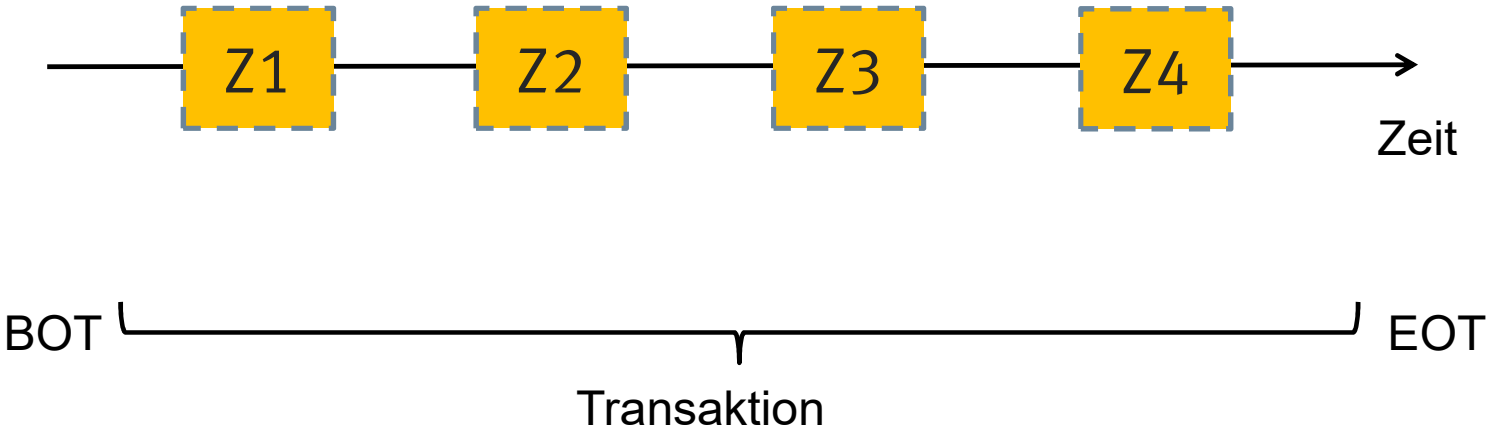
Umsetzung mittels DBMS:

- **Statische Integritätsbedingungen**
  - **Views** — Einschränkung von Änderungsoperationen, Berücksichtigung von Benutzerrechten, deklarativ bei Definition
  - **Constraints** — mit Tabellendeklaration definiert
  - **Assertions** — nur SQL 1999, deklariert unabhängig von Tabellendeklaration
- **Dynamische Integritätsbedingungen**
  - **Trigger** — prozedural, unabhängig von der Tabellendeklaration
  - **Transaktionen** — Zusammengehörigkeit von Änderungsoperationen, unabhängig von der Tabellendeklaration

Überprüfung der Integrität von Datenbankoperationen



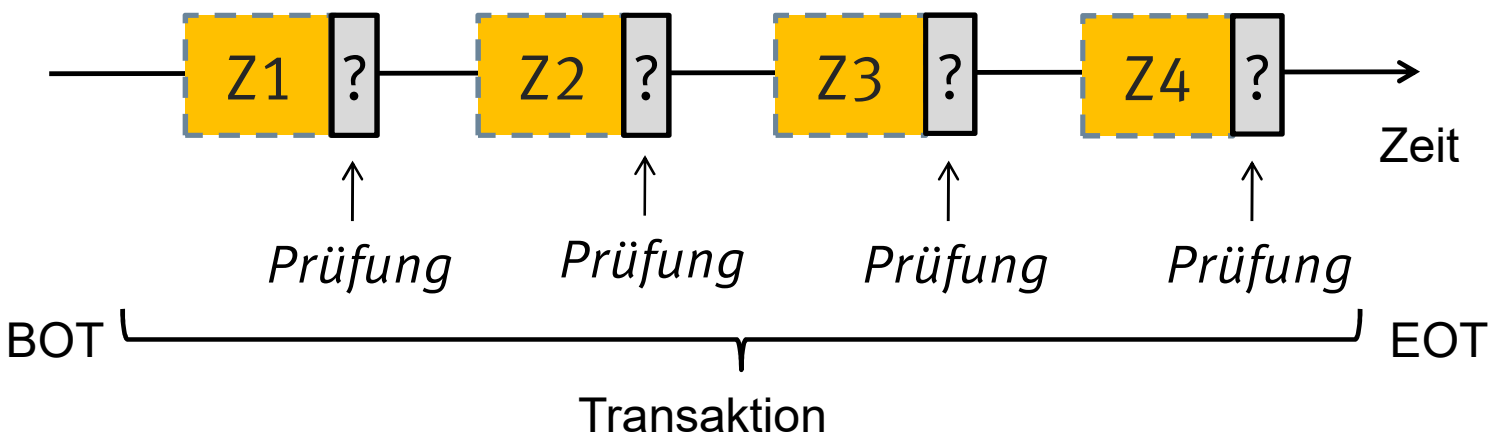
Zeitpunkt der Überprüfung



- **Standard:**
  Jede DML-Anweisung wird sofort nach der Anweisung geprüft auf **Ausführbarkeit** und **Übereinstimmung mit den Integritäts-bedingungen**

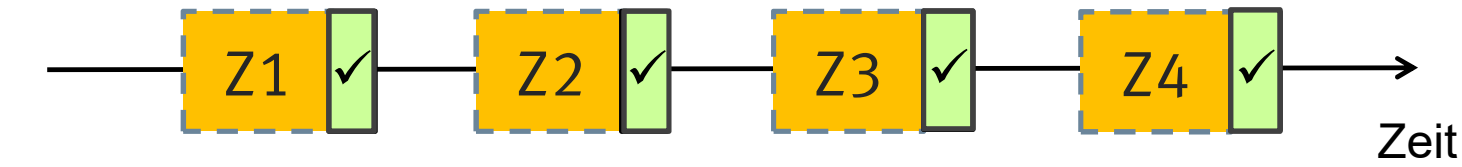
DML: Data Manipulation Language (DML)
SQL Kommandos zur Modifikation der Daten (INSERT, UPDATE and DELETE)

Zeitpunkt der Überprüfung



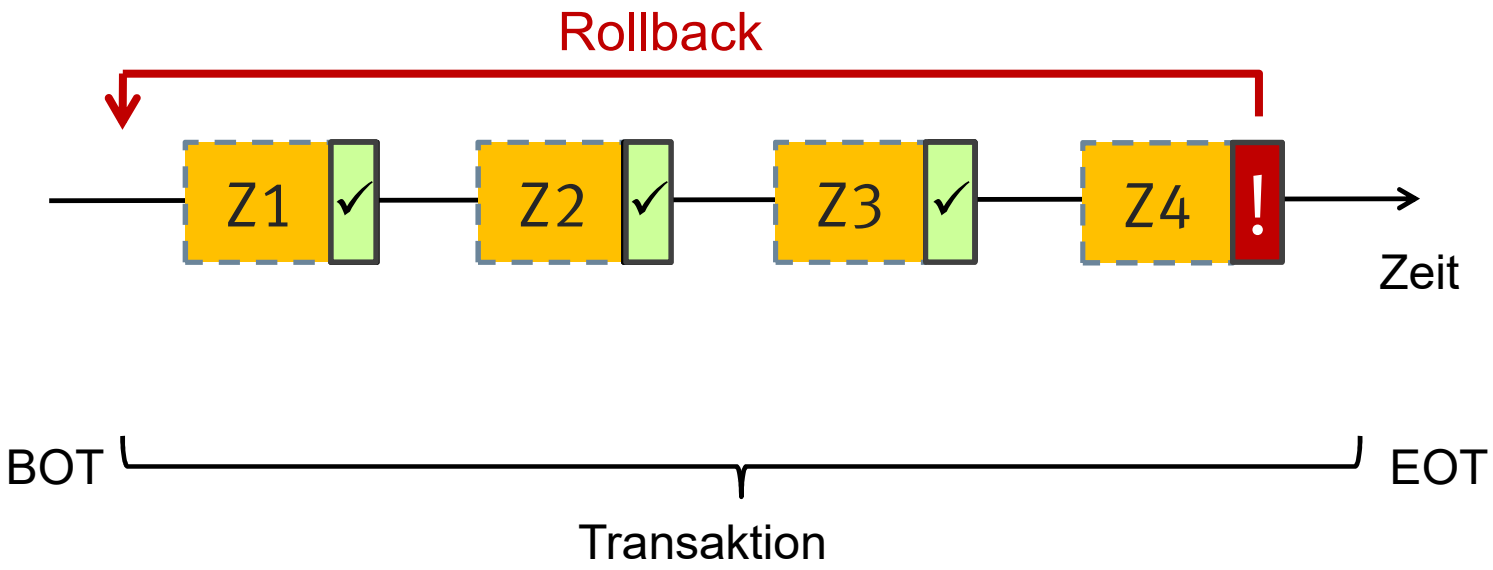
- **Standard:**
  Jede DML-Anweisung wird sofort nach der Anweisung geprüft auf **Ausführbarkeit** und **Übereinstimmung mit den Integritäts-bedingungen**

Zeitpunkt der Überprüfung



- **Standard:**
  Jede DML-Anweisung wird sofort nach der Anweisung geprüft auf **Ausführbarkeit** und **Übereinstimmung mit den Integritäts-bedingungen**

Zeitpunkt der Überprüfung

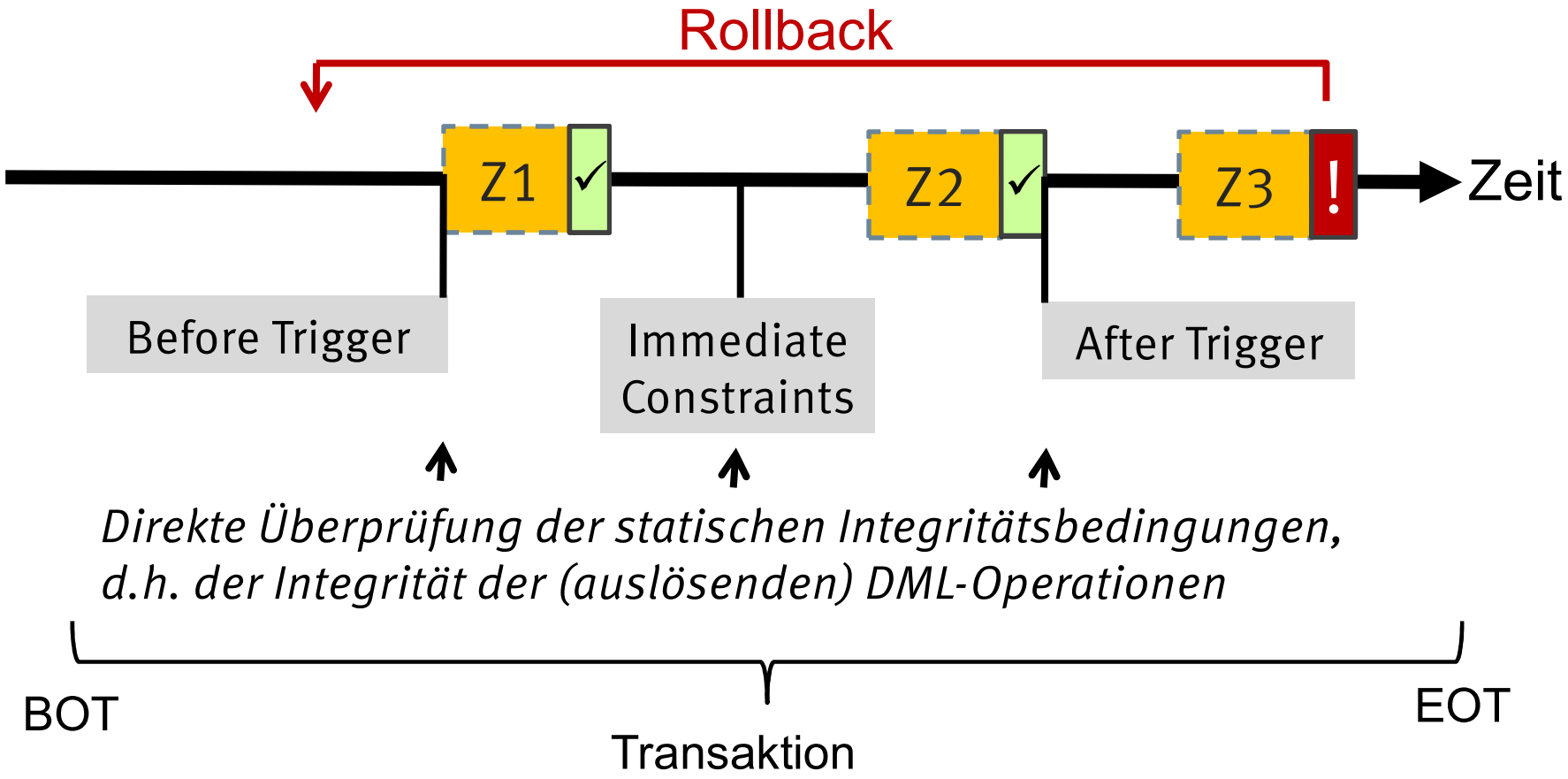


- **Standard:**
  Jede DML-Anweisung wird sofort nach der Anweisung geprüft auf **Ausführbarkeit** und **Übereinstimmung mit den Integritäts-bedingungen**

- **Wirkung im Fehlerfall:**
  - Rücksetzen der verletzten Anweisung
  - Rollback der gesamten Transaktion

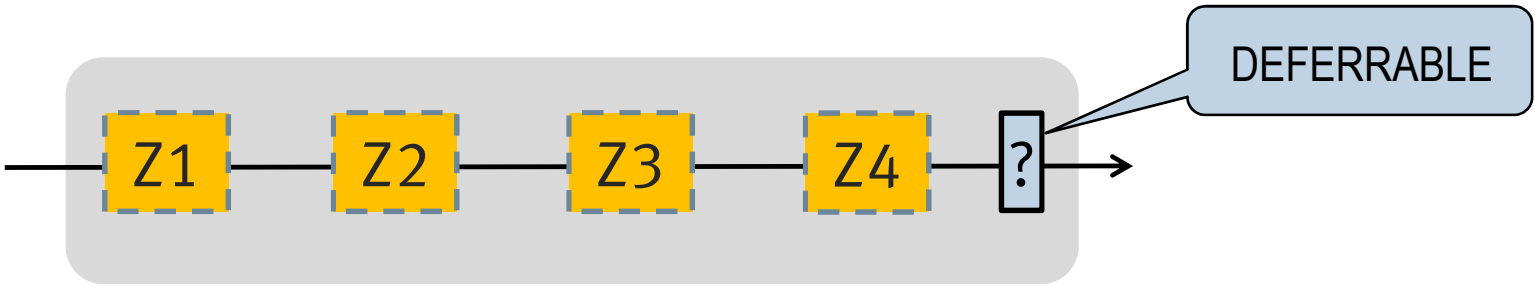
Zeitpunkt der Überprüfung

Die Überprüfung der DML-Operationen bezieht Trigger-Operationen mit ein:



Verschieben von Integritätsprüfungen das Transaktionsende

(DEFERRABLES)



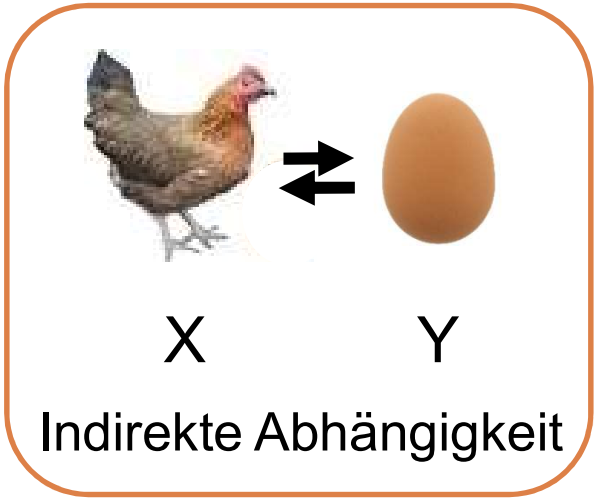
Indirekte Abhängigkeiten – Problemstellung

Constraint:

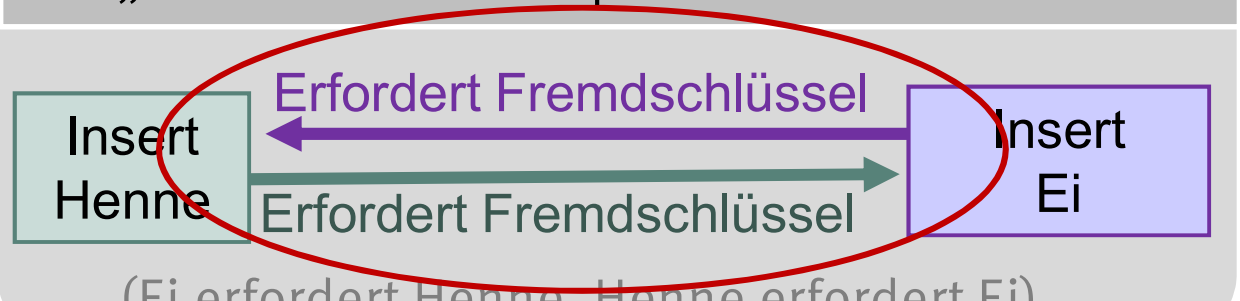
„Ein neuer Kunde kann nur eingefügt werden, wenn dieser bereits Artikel im Warenkorb hat“

(Kunde erfordert Artikel, Artikel erfordert Kunde)

Strukturell gleichwertig:



1. „Zu jedem Ei Y gehört eine Henne X“
2. „Eine Henne X schlüpft aus einem Ei Y“



(Ei erfordert Henne, Henne erfordert Ei)

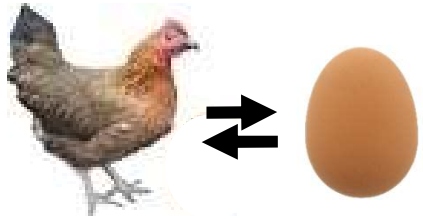
Indirekte Abhängigkeiten – Problemstellung

Constraint:

„Ein neuer Kunde kann nur eingefügt werden, wenn dieser bereits Artikel im Warenkorb hat“

(Kunde erfordert Artikel, Artikel erfordert Kunde)

Strukturell gleichwertig:



X Y

Indirekte Abhängigkeit

1. „Zu jedem Ei Y gehört eine Henne X“
2. „Eine Henne X schlüpft aus einem Ei Y“

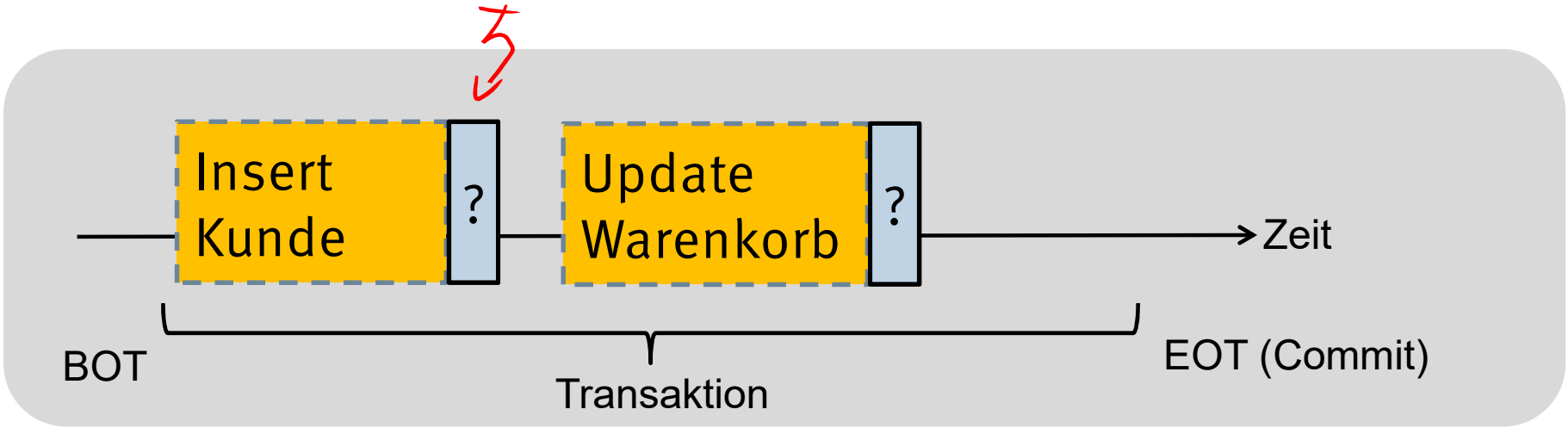
(Ei erfordert Henne, Henne erfordert Ei)

✓ Oracle

✗ MySQL

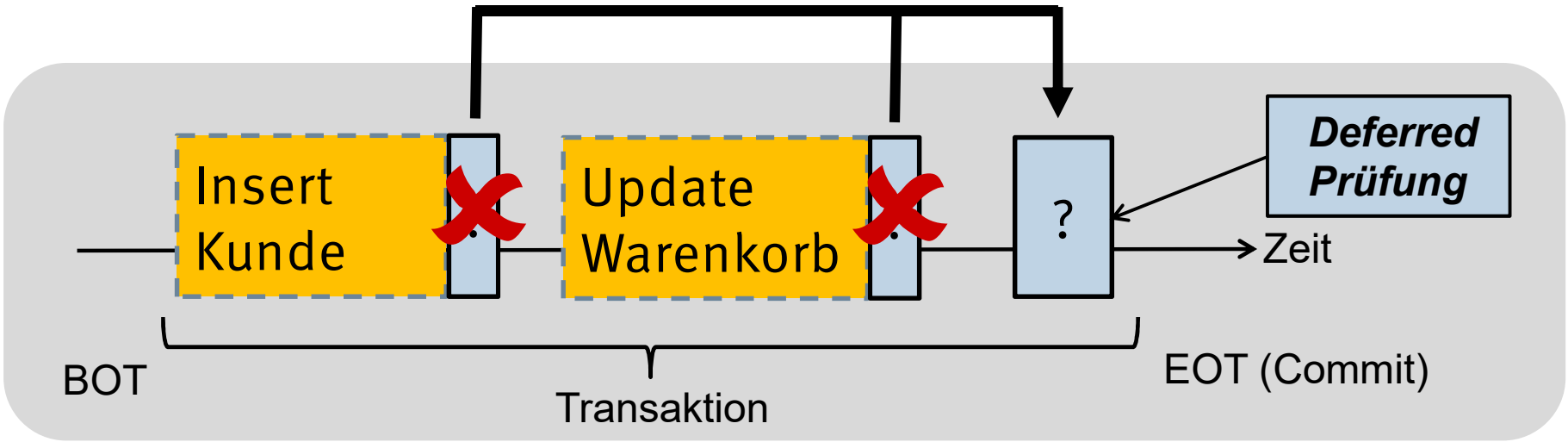
Verschiebung der Integritätsprüfung (Oracle)

Die Überprüfung der statischen Integritätsbedingungen erfolgt nicht sofort, sondern wird an das **Ende der Transaktion** verschoben (deferred).



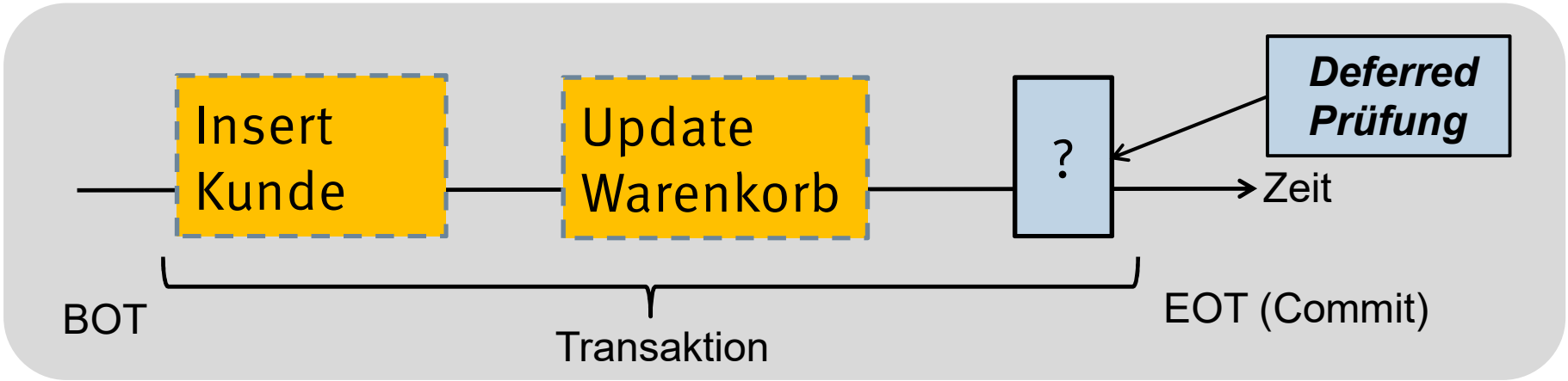
Verschiebung der Integritätsprüfung (Oracle)

Die Überprüfung der statischen Integritätsbedingungen erfolgt nicht sofort, sondern wird an das **Ende der Transaktion** verschoben (deferred).



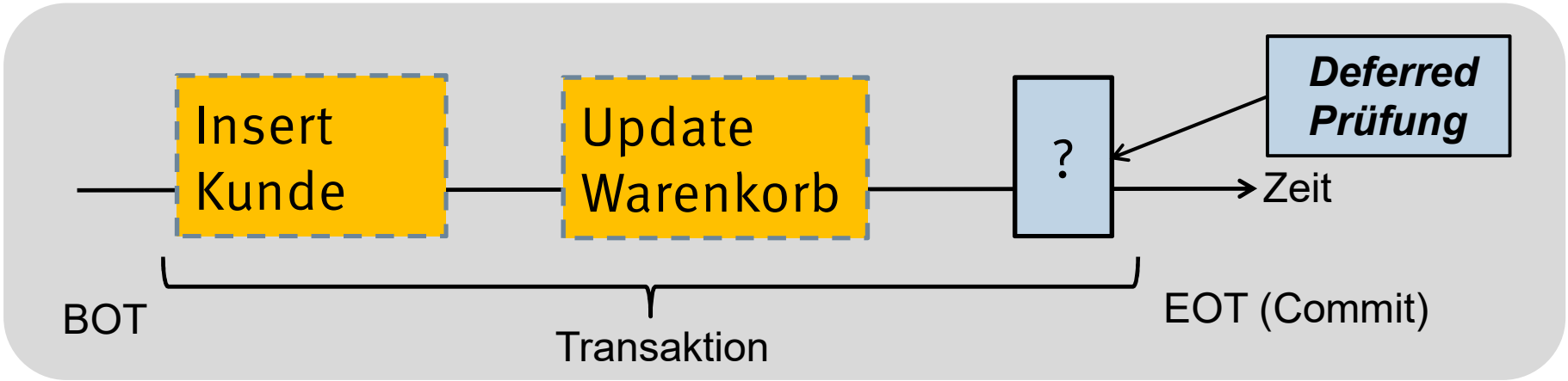
Im Fehlerfall wird die gesamte Transaktion zurückgesetzt. (Rollback)

Verschiebung der Integritätsprüfung




```
CREATE TABLE Warenkorb(
...
    FOREIGN KEY(Kundennummer)
    REFERENCES Kunden(Kundennummer)
    ON DELETE CASCADE DEFERRABLE
);
```

Oracle: Verschiebung der Überprüfung möglich



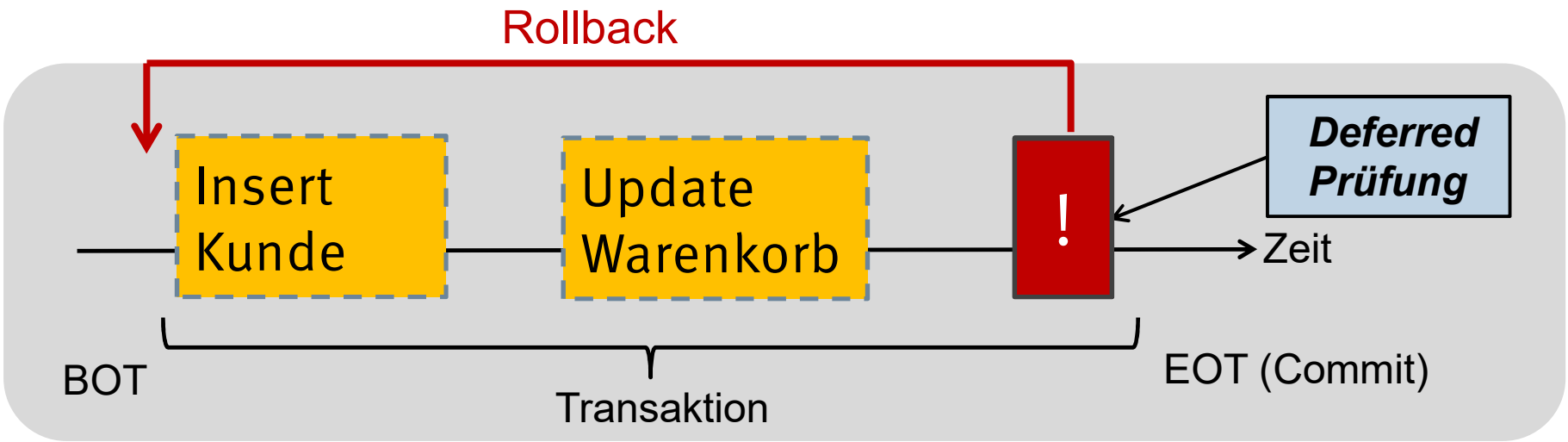

```
CREATE TABLE Warenkorb(
…
    FOREIGN KEY(Kundennummer)
    REFERENCES Kunden(Kundennummer)
    ON DELETE CASCADE DEFERRABLE
);
```

Nur für Oracle:

DEFERRABLE
= prüfen verschieben

IMMIDIATE (default)
= sofort prüfen

Verschiebung der Integritätsprüfung



Im Fehlerfall wird die gesamte Transaktion zurückgesetzt. (Rollback)

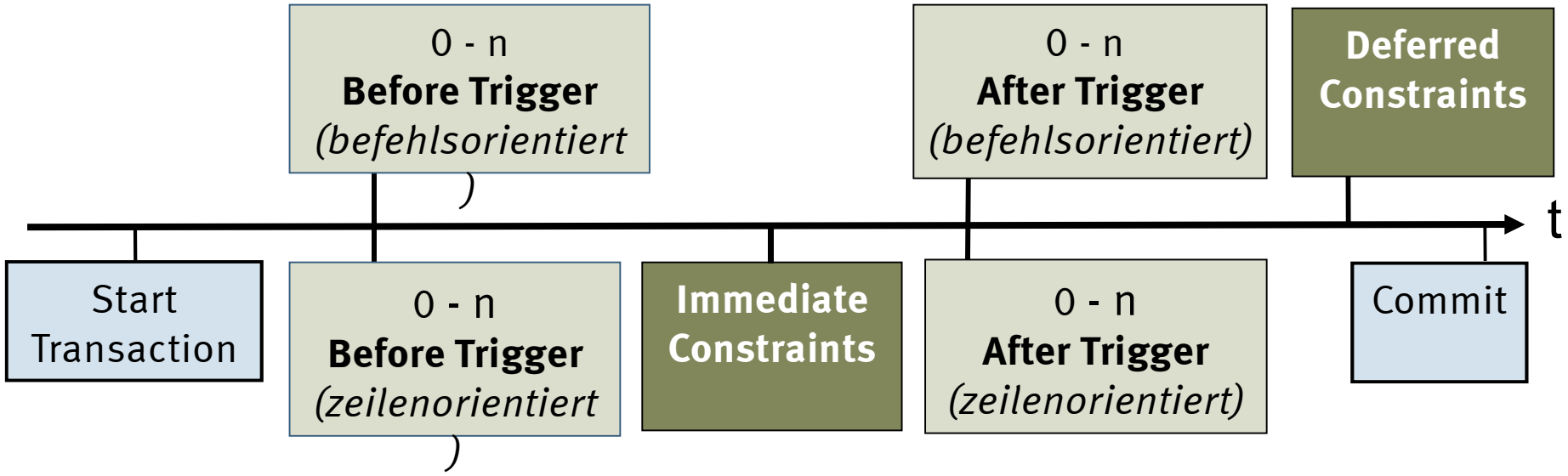
```
CREATE TABLE Warenkorb(
…
    FOREIGN KEY(Kundennummer)
    REFERENCES Kunden(Kundennummer)
    ON DELETE CASCADE DEFERRABLE
);
```

Nur für Oracle:

DEFERRABLE
= prüfen verschieben

IMMIDIATE (default)
= sofort prüfen

Ausführungsreihenfolge der Integritätsprüfung



Oracle:

- zeilen- und befehlsorientierte Trigger sind möglich
- keine Einschränkung bzgl. der Anzahl von Triggern pro Tabelle

Zusammenfassung

Ziel der Datenbanküberwachung:

Konsistenz, d.h. semantische Datenintegrität,
die Korrektheit der Daten zu jedem Zeitpunkt

ACID-Eigenschaften von Transaktionen

1	Atomarität	(*Atomicity*)
2	Konsistenz	(*Consistency*)
3	Isoliertheit	(*Isolation*)
4	Dauerhaftigkeit	(*Durability*)

DBMS – Maßnahmen:

- Statische Integritätsregeln

  (z.B. Primär-/Fremdschlüssel, Views, Constraints)

- Dynamische Transaktionsbedingungen, zentrale

  Prüfprogramme für Plausibilität und formale Korrektheit

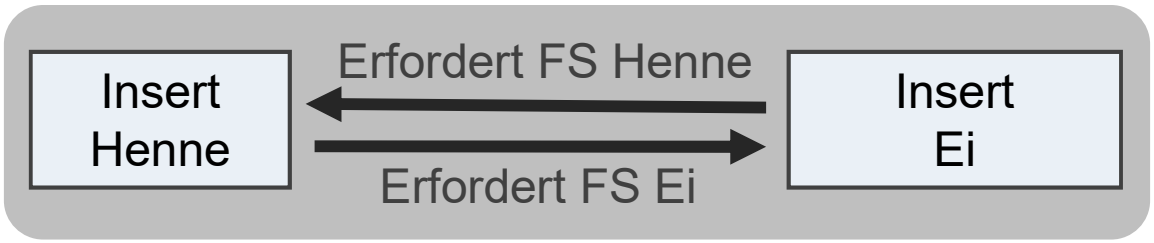
  (z.B. Trigger, Transaktionen)

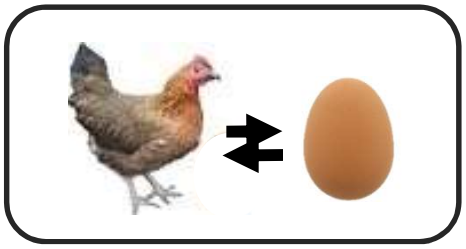
- Zurückweisung von Änderungen bei Integritätsverletzung

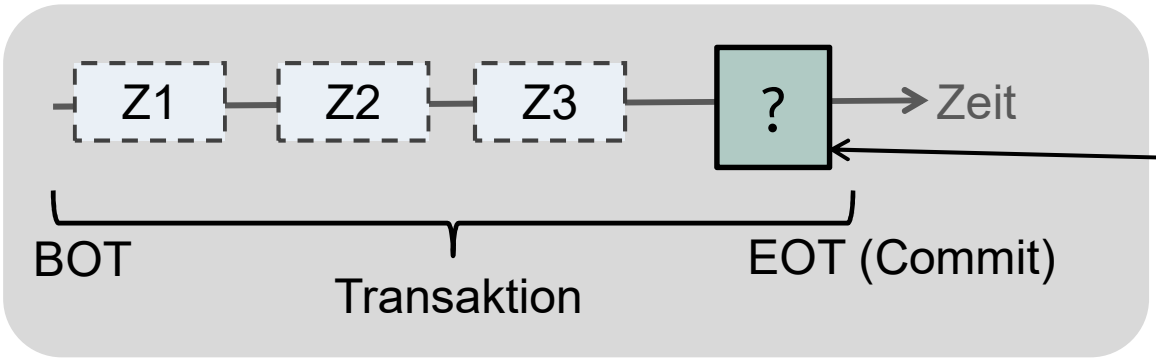
  (Rollback)

Oracle

Das Verschieben der Überprüfung von Transaktions-internen Anweisungen an das Ende der Transaktion ermöglicht die Ausführung von Anweisungen auch bei Vorhandensein indirekter Abhängigkeiten (Deferrables)







Deferred Prüfung

Nicht mgl. in MySQL