

# **Wirtschaftsinformatik 2**

## **LE 08 – Transaktionen**

Prof. Dr. Thomas Off

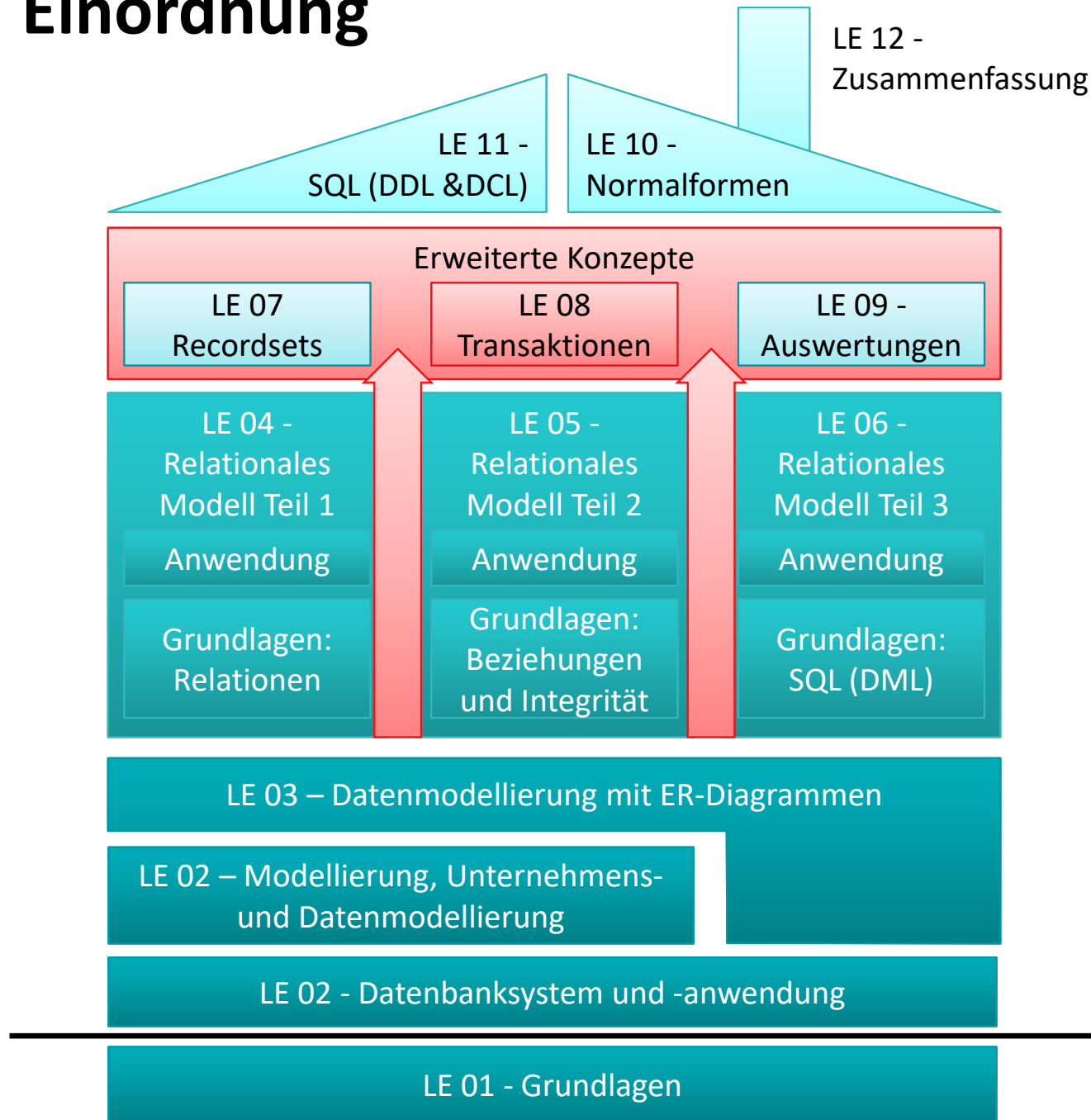
<http://www.ThomasOff.de/lehre/beuth/wi2>

# Ziel

## Ziel dieser Lehreinheit

- Konsistenzsicherung als Anforderung an Datenbanken wiederholen und vertiefen
- Konzepte der Transaktionen und deren Eigenschaften kennenlernen
- Anwendung von Transaktionen in SQL und mit MS Access
- Erläuterungen zum technischen Hintergrund der Transaktionsverarbeitung

# Einordnung



# Inhalt

## Ziel und Einordnung

## Rückblick

## Transaktionen

- Konsistenz und Integrität
  - Konsistenzsicherung als Ziel relationaler Datenbanken
  - Gefahren für die Konsistenz von Datenbanken
- Transaktionen und ACID
- Anwendung von Transaktionen
  - Anwendungsszenarien
  - Transaktionen in SQL
  - Transaktionen mit MS Access
- Technik der Transaktionsverarbeitung
  - Parallele Transaktionen und deren Probleme
  - Transaktionslevel
  - Umsetzung paralleler Transaktionen
  - Protokollierung
- Zusammenfassung

## Ausblick

# Rückblick



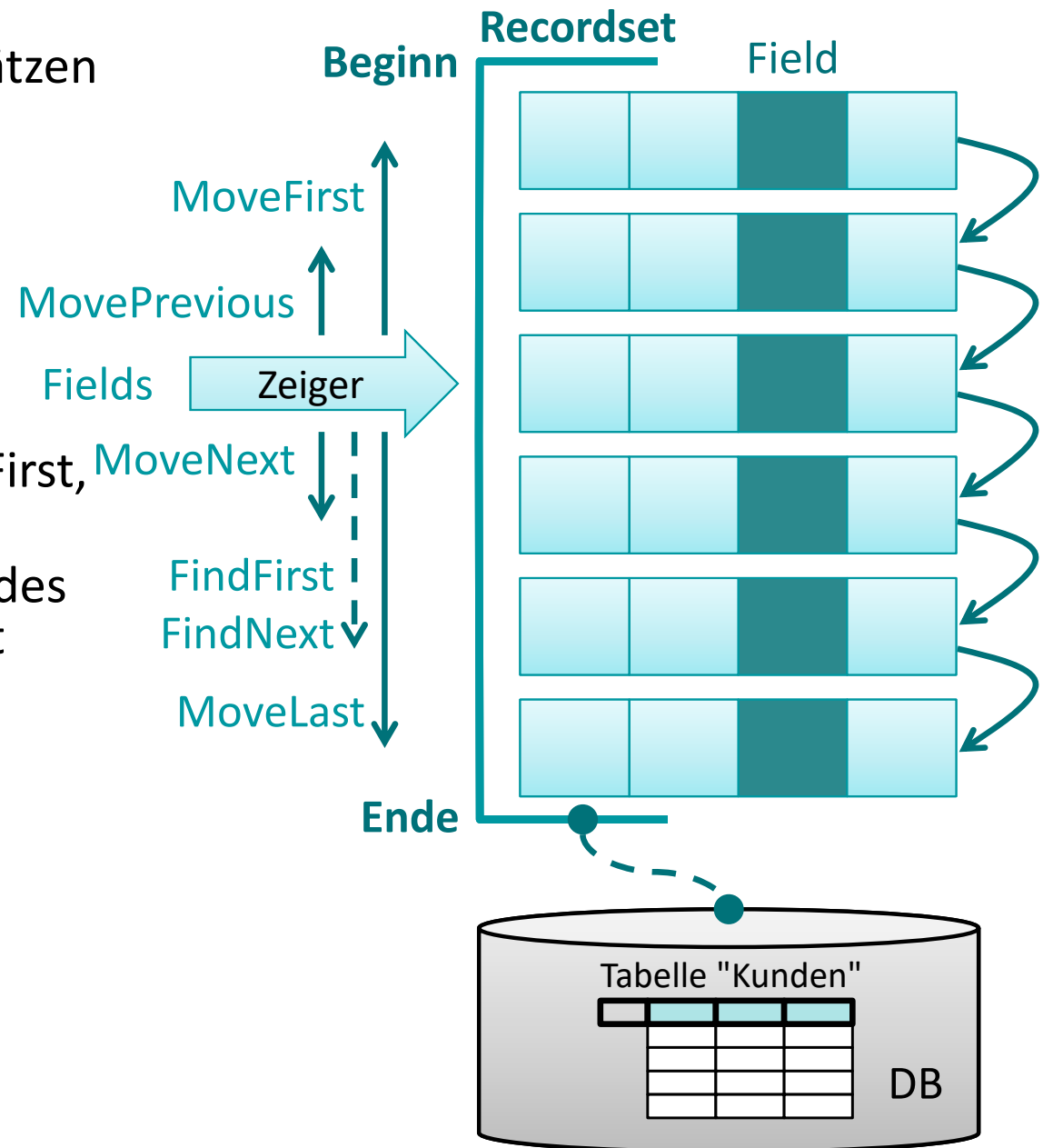
~~BHT~~

# Rückblick (LE 07)



## Recordset

- geordnete Menge von Datensätzen aus Tabelle(n) der Datenbank geladen
- Typen
  - Schnappschuss
  - Dynamische Verbindung
- Zeiger für Navigation über Datensätze (MoveNext, MoveFirst, ...)
- Zugriffsmöglichkeit auf Werte des Elementes, auf das Zeiger zeigt (Fields)
- Datensätze suchen (FindFirst, FindNext, ...)
- Modus für Ändern (Edit), Hinzufügen (AddNew) und anschließend Update oder alternativ Löschen (Delete)





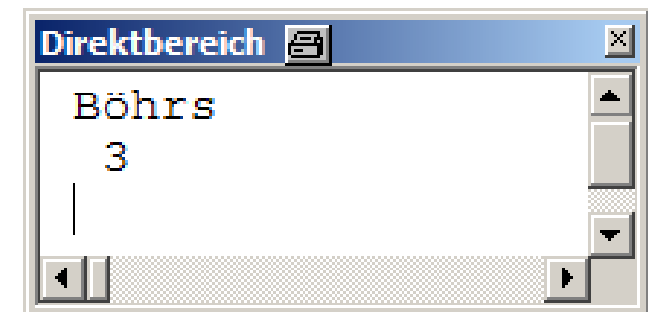
## Weitere VBA-Funktionen zum Datenzugriff

- Bisher: Zugriff auf mehrere Datensätze und deren Werte mittels Recordsets
- Jetzt: Ermittlung eines Ergebniswertes aus der Datenbank mittels Domänenfunktionen, z.B.
  - Ermitteln eines Wertes aus einem Datensatz,
  - Zählen aller Datensätze (anhand einer nicht leeren Spalte)

### ' Ausgabe

```
Debug.Print DLookup("kndName", "tblKunden", "kndIdPk=2")  
Debug.Print DCount("kndIdPk", "tblKunden")
```

	kndIdPk	kndName	kndVorname
+	1	Albers	Willi
+	2	Böhrs	Thomas
+	3	Dinkel	Ulrike
*	(Neu)		



**LE 08 - ~~Belegbuch~~ ~~Rechnungswesen~~ ~~Rechnungswesen~~**





# Rückblick (LE 02)



**Definition "Datenbanksystem": Zusammenfassung und Bereitstellung konsistenter, integrierter und untereinander in Beziehung stehender Daten und Informationen über die Organisation dieser Daten zur Nutzung in mehreren Anwendungen**



# Rückblick (LE 02)



Definition "Datenbanksystem": Zusammenfassung und Bereitstellung **konsistenter**, integrierter und untereinander in Beziehung stehender **Daten** und Informationen über die Organisation dieser Daten zur Nutzung in mehreren Anwendungen





## Anforderungen an Datenbanken in betrieblichen Anwendungen

– ...

– stellen sicher, dass die Daten korrekt sind

- physisch korrekt gespeichert
- logisch korrekt, so dass keine Widersprüche existieren
- semantisch korrekt, so dass keine unsinnigen Daten gespeichert sind

– ...

**➔ Integrität der gespeicherten Daten ist eine Voraussetzung für die Konsistenz der Datenbank**

# Inhalt

## Ziel und Einordnung

## Rückblick

## Transaktionen

- Konsistenz und Integrität
  - Konsistenzsicherung als Ziel relationaler Datenbanken
  - Gefahren für die Konsistenz von Datenbanken
- Transaktionen und ACID
- Anwendung von Transaktionen
  - Anwendungsszenarien
  - Transaktionen in SQL
  - Transaktionen mit MS Access
- Technik der Transaktionsverarbeitung
  - Parallele Transaktionen und deren Probleme
  - Transaktionslevel
  - Umsetzung paralleler Transaktionen
  - Protokollierung
- Zusammenfassung

## Ausblick

# Konsistenz und Integrität

## Ziel relationaler Datenbanken ist Sicherung der Konsistenz/Integrität der gespeicherten Daten

- Konsistenz bezeichnet Korrektheit der gespeicherten Daten
  - physische Konsistenz: Daten werden technisch korrekt gespeichert und korrekt gelesen
  - logische Konsistenz: die zu speichernden Daten und Zusammenhänge sind für sich genommen korrekt
  - semantische Konsistenz: es werden keine unsinnigen Daten gespeichert
- Konsistenz ist nur gewährleistet, wenn die gespeicherten Daten alle Integritätsbedingungen erfüllen

# Konsistenz und Integrität

## Ziel relationaler Datenbanken ist Sicherung der Konsistenz/Integrität der gespeicherten Daten

- wird erreicht für
  - Attribute/Spalten: durch Vorgabe und Prüfung des Wertebereichs
  - Entitäten/Relationen: durch Primärschlüssel und 1. Integritätsregel
    - Primärschlüssel muss eindeutig und darf niemals leer sein
  - Beziehungen/Fremdschlüssel: durch 2. Integritätsregel
    - Kein Fremdschlüssel (ungleich "leer"), dessen Wert im zugehörigen Primärschlüssel nicht existiert
  - Fachliche Zusammenhänge (z.B. Geburtsdatum von Kunden muss in der Vergangenheit liegen): durch Implementierung fachlicher Plausibilitätsregeln und Prüfung vor der Speicherung in der Datenbank

## Gefahren für die Konsistenz von Daten

- technische Fehler ausgelöst durch Rechnerabsturz, Stromausfall, Brand (Feuer, Löschwasser), ...
- logische und semantische Fehler durch konkurrierende Zugriffe mehrerer Benutzer



# Konsistenz und Integrität

## Beispiel 1: Überweisung zwischen Konten einer Bank

- Kunde Müller will Kunde Yilmaz 100 € überweisen

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500



# Konsistenz und Integrität

## Beispiel 1: Überweisung zwischen Konten einer Bank

- Kunde Müller will Kunde Yilmaz 100 € überweisen
- Konsistenz: Summe des Guthabens von Müller und Yilmaz ist vor und nach der Überweisung gleich

**Vorher:**

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

**Summe: 800 € + 500 € = 1300 €**

# Konsistenz und Integrität

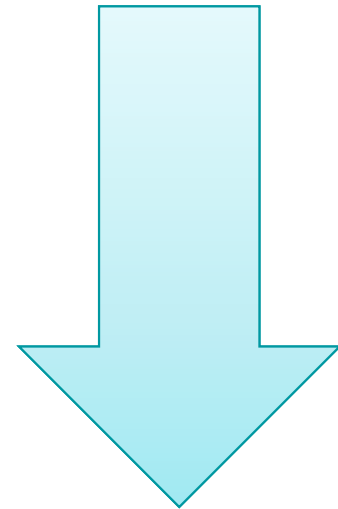
## Beispiel 1: Überweisung zwischen Konten einer Bank

- Kunde Müller will Kunde Yilmaz 100 € überweisen
- Konsistenz: Summe des Guthabens von Müller und Yilmaz ist vor und nach der Überweisung gleich

**Vorher:**

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

**Summe: 800 € + 500 € = 1300 €**



**Nacher:**

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	600

**Summe: 700 € + 600 € = 1300 €**

# Konsistenz und Integrität

## Beispiel 1: Überweisung zwischen Konten einer Bank

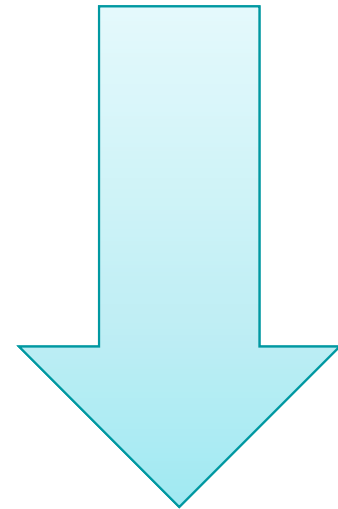
### – Ablauf

- Kontostand von Müller wird um 100 € reduziert
- Kontostand von Yilmaz wird um 100 € erhöht

**Vorher:**

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

**Summe: 800 € + 500 € = 1300 €**



**Nacher:**

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	600

**Summe: 700 € + 600 € = 1300 €**

# Konsistenz und Integrität

## Beispiel 1: Überweisung zwischen Konten einer Bank

- Ablauf
  - Kontostand von Müller wird um 100 € reduziert
  - Kontostand von Yilmaz wird um 100 € erhöht
- SQL

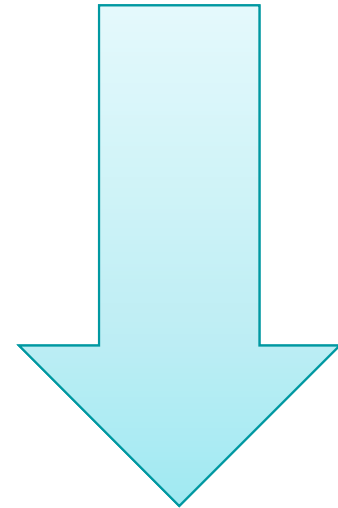
```
UPDATE Konten
  SET Saldo = Saldo - 100
  WHERE KtoNr = 2345;

UPDATE Konten
  SET Saldo = Saldo + 100
  WHERE KtoNr = 4567;
```

**Vorher:**

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

**Summe: 800 € + 500 € = 1300 €**



**Nacher:**

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	600

**Summe: 700 € + 600 € = 1300 €**

# Konsistenz und Integrität

## Beispiel 1: Überweisung zwischen Konten einer Bank

- Ablauf
  - Kontostand von Müller wird um 100 € reduziert
  - Kontostand von Yilmaz wird um 100 € erhöht
- SQL

```
UPDATE Konten
  SET Saldo = Saldo - 100
  WHERE KtoNr = 2345;

UPDATE Konten
  SET Saldo = Saldo + 100
  WHERE KtoNr = 4567;
```

**Vorher:**

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

**Summe: 800 € + 500 € = 1300 €**



...



**Nacher:**

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	600

**Summe: 700 € + 600 € = 1300 €**

# Konsistenz und Integrität

## Beispiel 1: Überweisung zwischen Konten einer Bank

– Ablauf

- **Kontostand von Müller wird um 100 € reduziert**
- Kontostand von Yilmaz wird um 100 € erhöht

– SQL

```
UPDATE Konten
  SET Saldo = Saldo - 100
  WHERE KtoNr = 2345;

UPDATE Konten
  SET Saldo = Saldo + 100
  WHERE KtoNr = 4567;
```

**Vorher:**

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

**Summe: 800 € + 500 € = 1300 €**



# Konsistenz und Integrität

## Beispiel 1: Überweisung zwischen Konten einer Bank

– Ablauf

- **Kontostand von Müller wird um 100 € reduziert**
- Kontostand von Yilmaz wird um 100 € erhöht

– SQL

```
UPDATE Konten
  SET Saldo = Saldo - 100
  WHERE KtoNr = 2345;

UPDATE Konten
  SET Saldo = Saldo + 100
  WHERE KtoNr = 4567;
```

**Vorher:**

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

**Summe: 800 € + 500 € = 1300 €**



**Zwischenzeitlich:**

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	<b>700</b>
	4567	Yilmaz	500

**Summe: 700 € + 500 € = 1200 €**

# Konsistenz und Integrität

## Beispiel 1: Überweisung zwischen Konten einer Bank

### – Ablauf

- Kontostand von Müller wird um 100 € reduziert
- **Kontostand von Yilmaz wird um 100 € erhöht**

### – SQL

```
UPDATE Konten
  SET Saldo = Saldo - 100
  WHERE KtoNr = 2345;

UPDATE Konten
  SET Saldo = Saldo + 100
  WHERE KtoNr = 4567;
```

### Vorher:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

**Summe: 800 € + 500 € = 1300 €**



### Zwischenzeitlich:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	500

**Summe: 700 € + 500 € = 1200 €**



### Nacher:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	<b>600</b>

**Summe: 700 € + 600 € = 1300 €**



# Konsistenz und Integrität

## Beispiel 1: Überweisung zwischen Konten einer Bank

- Angenommen technischer Fehler (Rechnerabsturz, Stromausfall, Feuer, Löschwasser, ...) tritt zwischenzeitlich ein



**Vorher:**

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

**Summe: 800 € + 500 € = 1300 €**



**Zwischenzeitlich:**

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	500

**Summe: 700 € + 500 € = 1200 €**



**Nacher:**

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	600

**Summe: 700 € + 600 € = 1300 €**

# Konsistenz und Integrität

## Beispiel 1: Überweisung zwischen Konten einer Bank

- Angenommen technischer Fehler (Rechnerabsturz, Stromausfall, Feuer, Löschwasser, ...) tritt zwischenzeitlich ein
- ... dann ist die Datenbank nicht mehr konsistent (es fehlen 100 EUR)

**Vorher:**

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

**Summe: 800 € + 500 € = 1300 €**



**Nachher?**

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	500

**Summe: 700 € + 500 € = 1200 €**

# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC



Quelle: <http://maps.google.com/>

# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

- Konsistenz bedeutet
  - Vorher: es ist kein Flug gebucht
  - Nachher: es ist der gesamte Flug, bestehen aus einem Flug pro Teilstrecke gebucht
    - TXL → MUC
    - MUC → TLS



Quelle: <http://maps.google.com/>

# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

– Ablauf?



# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

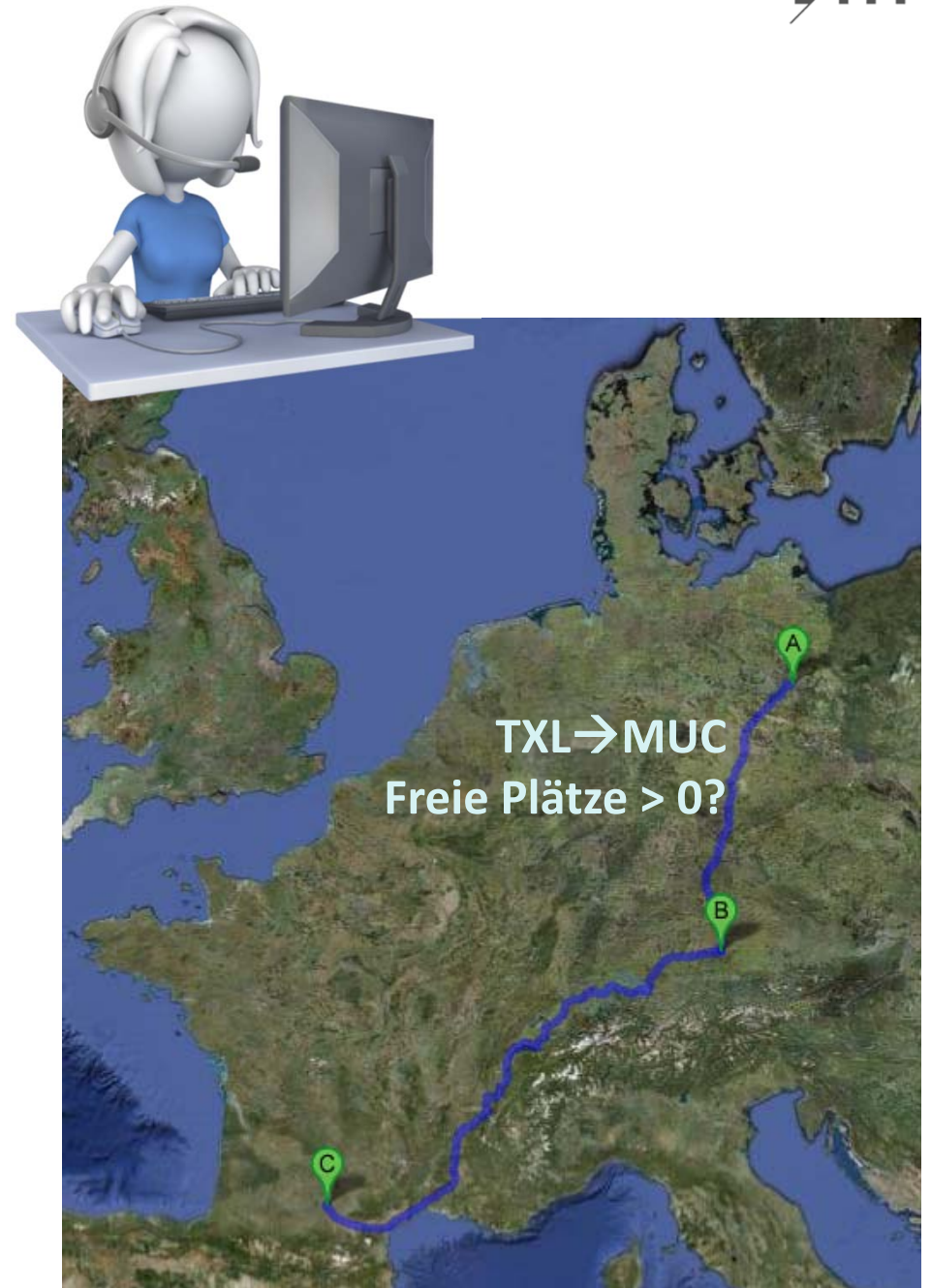
- Ablauf
  - Lesen, ob freie Plätze
  - Buchen, wenn auf beiden Teilstrecken ein Platz frei ist



# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

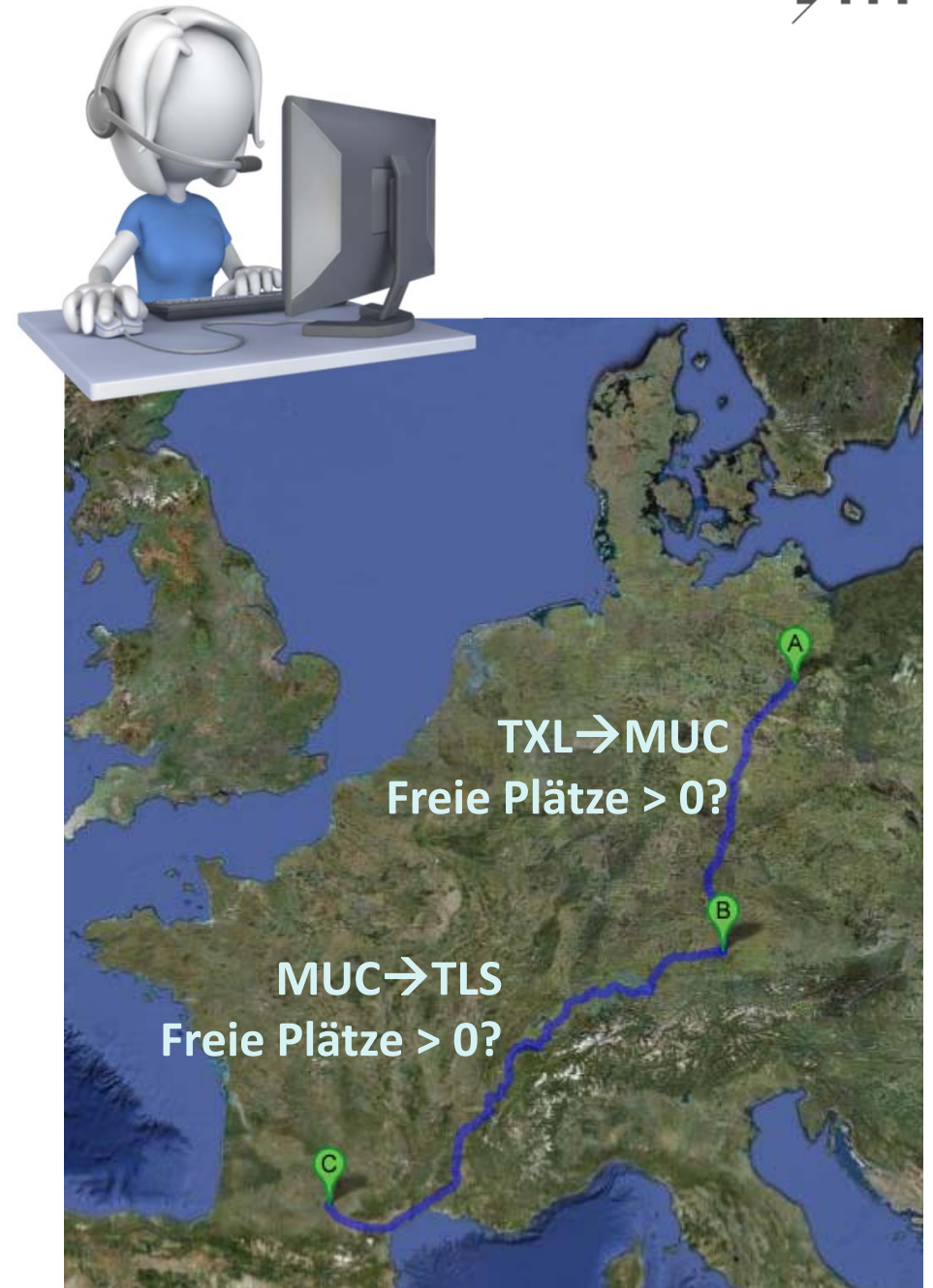
- Ablauf
  - Lesen, ob freie Plätze
    - auf Teilstrecke TXL nach MUC verfügbar ist und
  - Buchen, wenn auf beiden Teilstrecken ein Platz frei ist



# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

- Ablauf
  - Lesen, ob freie Plätze
    - auf Teilstrecke TXL nach MUC verfügbar ist und
    - auf Teilstrecke MUC nach TLS verfügbar ist
  - Buchen, wenn auf beiden Teilstrecken ein Platz frei ist

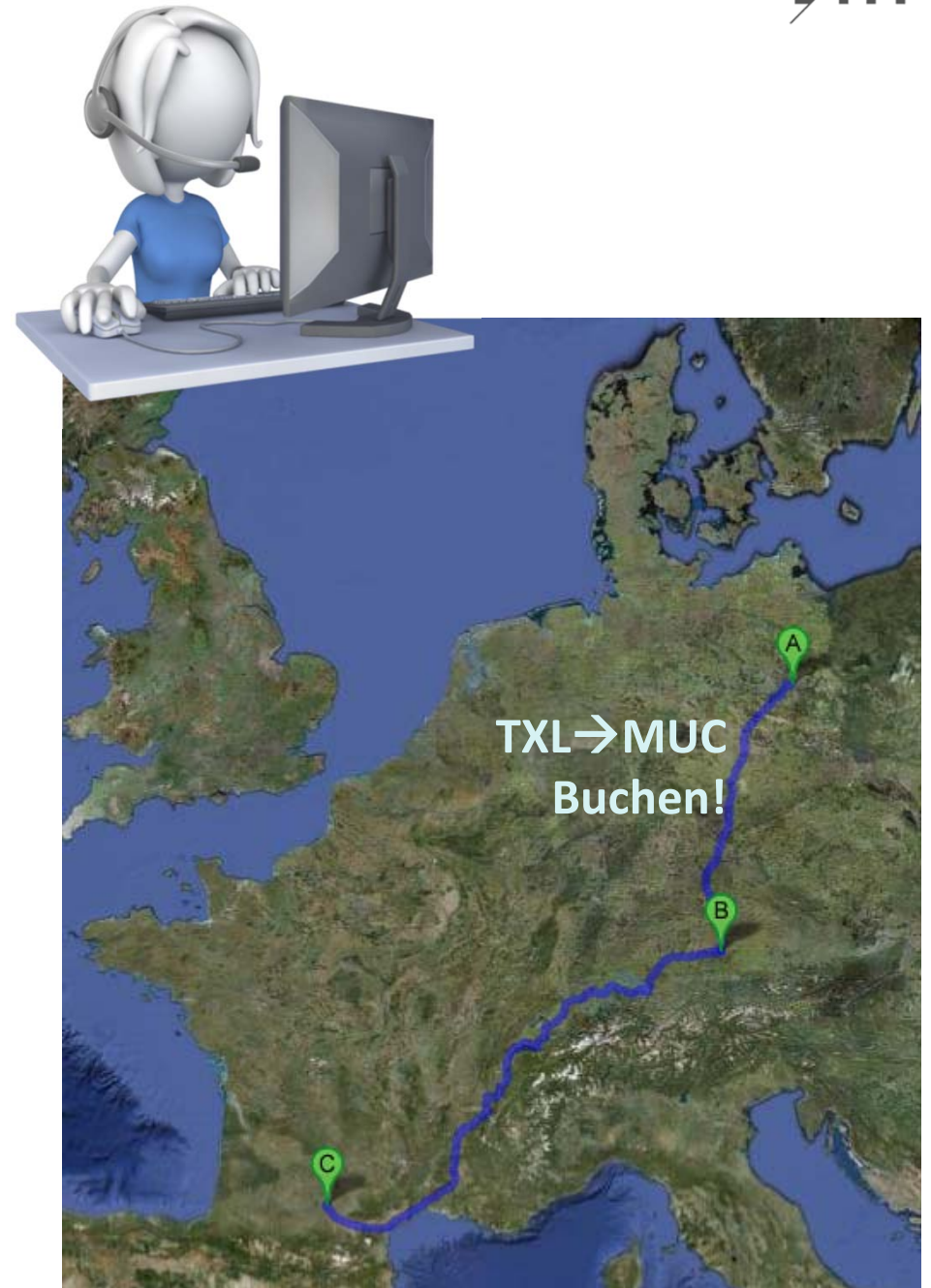




# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

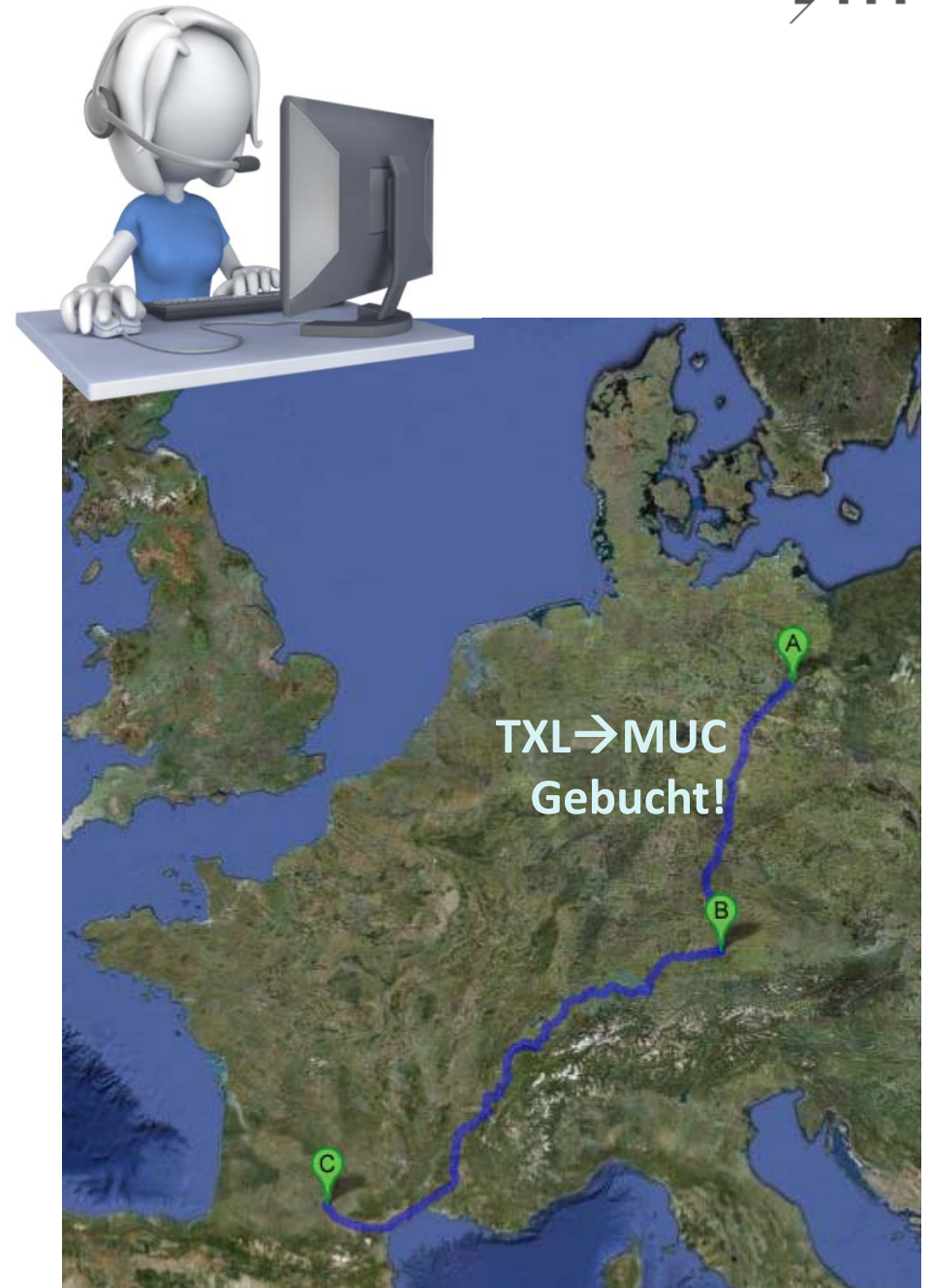
- Ablauf
  - Lesen, ob freie Plätze
    - auf Teilstrecke TXL nach MUC verfügbar ist und
    - auf Teilstrecke MUC nach TLS verfügbar ist
  - Buchen, wenn auf beiden Teilstrecken ein Platz frei ist
    - Teilstrecke TXL-MUC buchen



# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

- Ablauf
  - Lesen, ob freie Plätze
    - auf Teilstrecke TXL nach MUC verfügbar ist und
    - auf Teilstrecke MUC nach TLS verfügbar ist
  - Buchen, wenn auf beiden Teilstrecken ein Platz frei ist
    - Teilstrecke TXL-MUC buchen

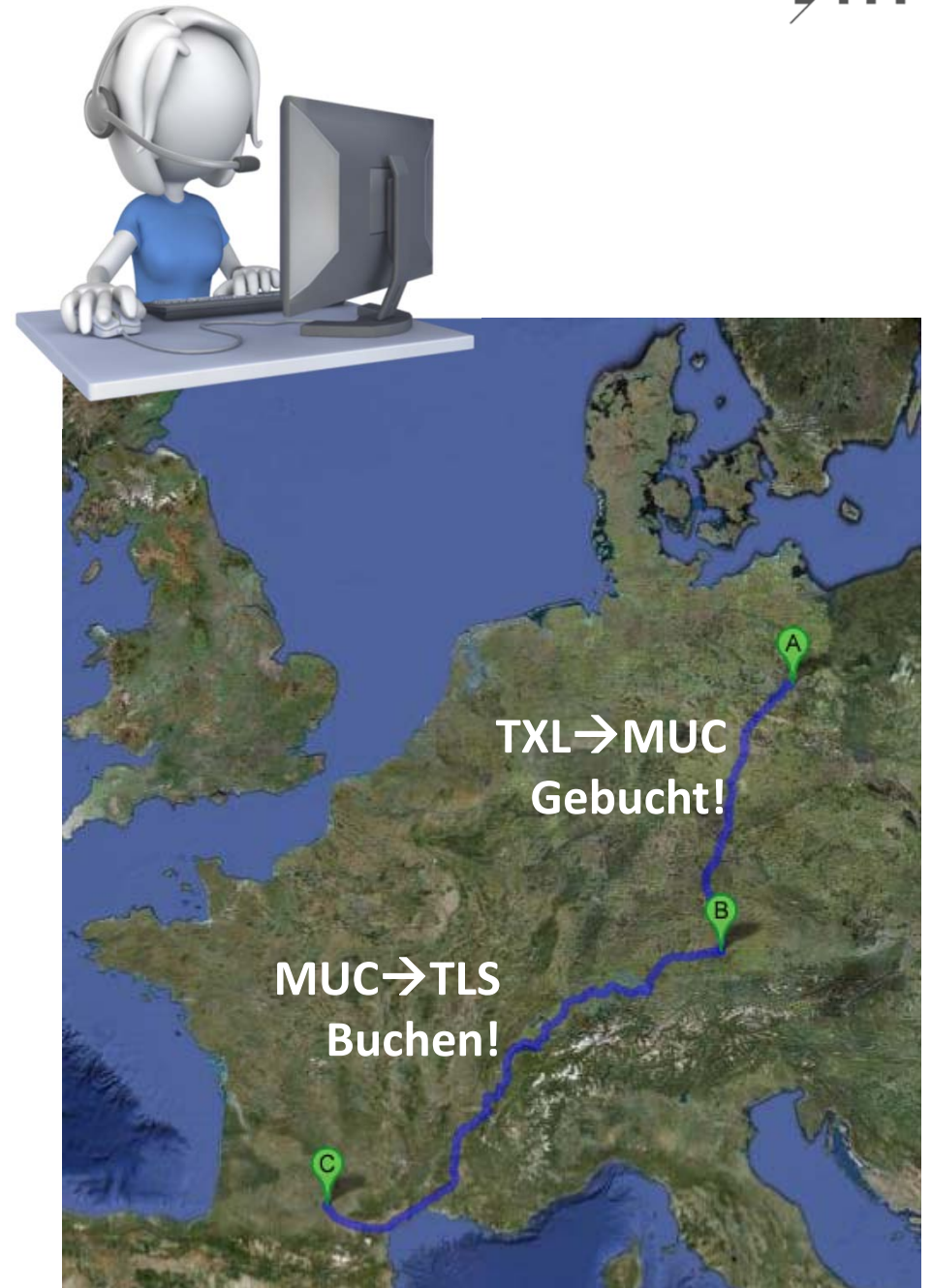


Quelle: <http://maps.google.com/>

# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

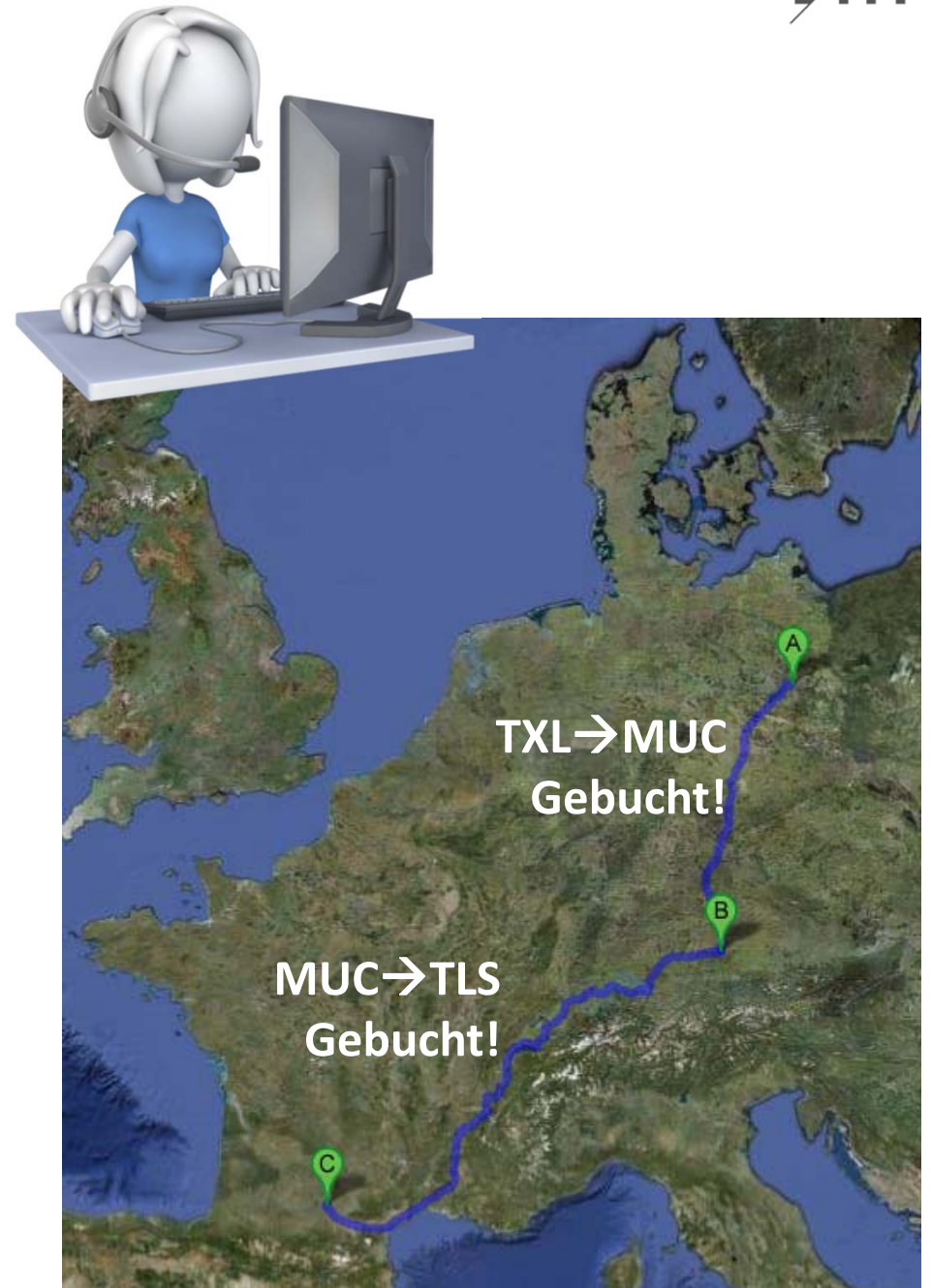
- Ablauf
  - Lesen, ob freie Plätze
    - auf Teilstrecke TXL nach MUC verfügbar ist und
    - auf Teilstrecke MUC nach TLS verfügbar ist
  - Buchen, wenn auf beiden Teilstrecken ein Platz frei ist
    - Teilstrecke TXL-MUC buchen
    - Teilstrecke MUC-TLS buchen



# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

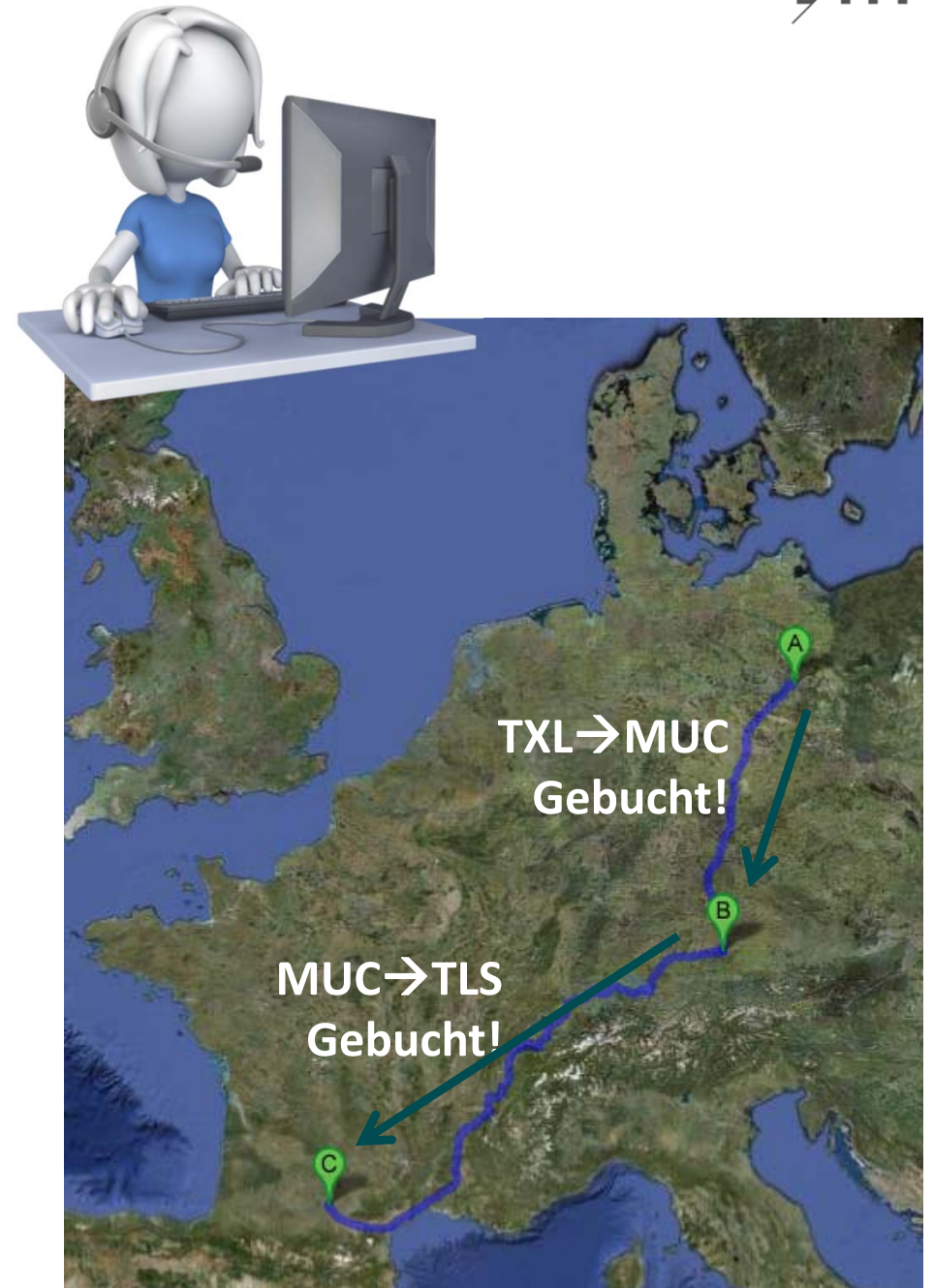
- Ablauf
  - Lesen, ob freie Plätze
    - auf Teilstrecke TXL nach MUC verfügbar ist und
    - auf Teilstrecke MUC nach TLS verfügbar ist
  - Buchen, wenn auf beiden Teilstrecken ein Platz frei ist
    - Teilstrecke TXL-MUC buchen
    - Teilstrecke MUC-TLS buchen



# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

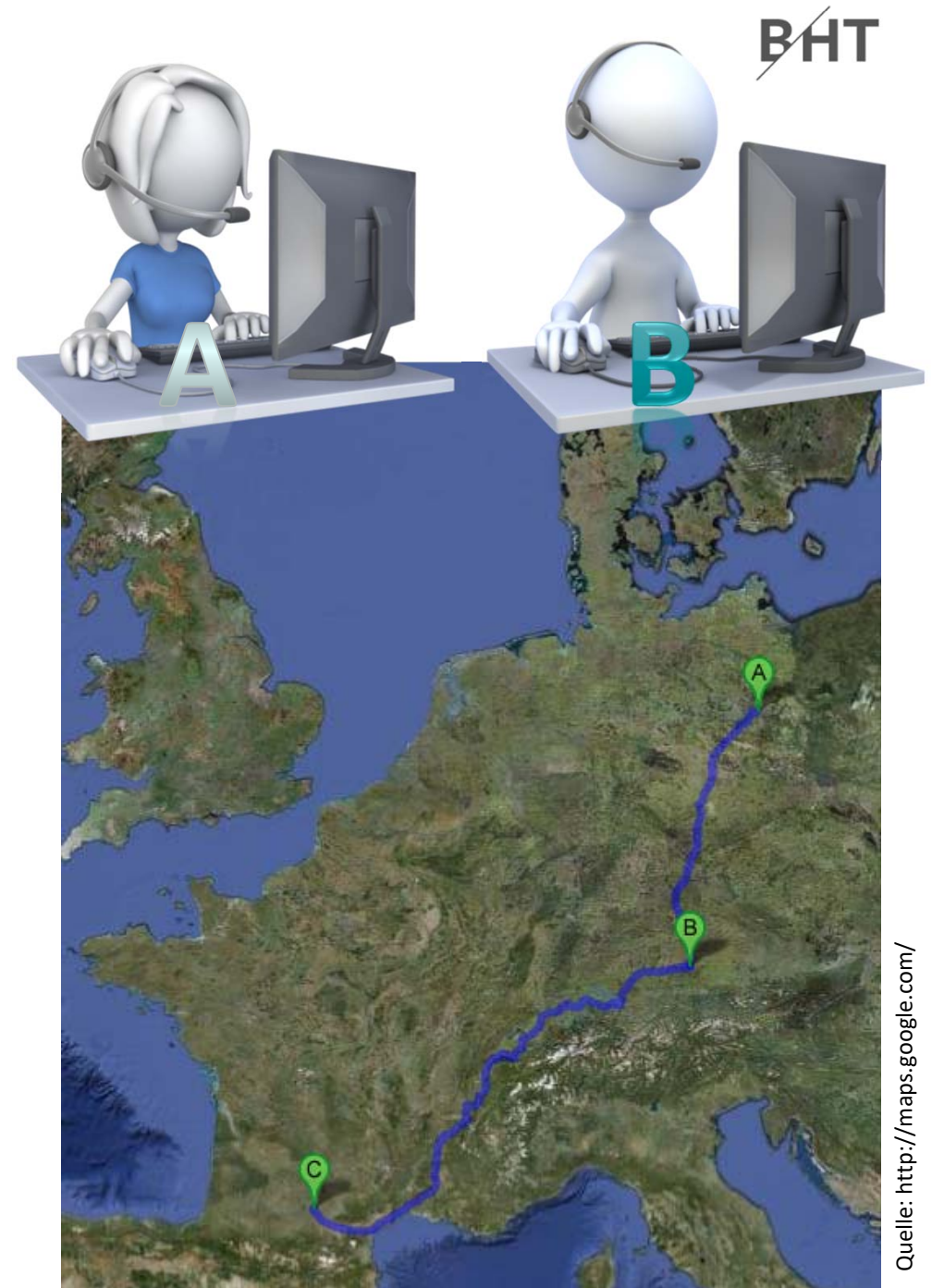
- Konsistenter Zustand: es ist der gesamte Flug, bestehend aus einem Flug pro Teilstrecke gebucht
  - TXL → MUC
  - MUC → TLS



# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

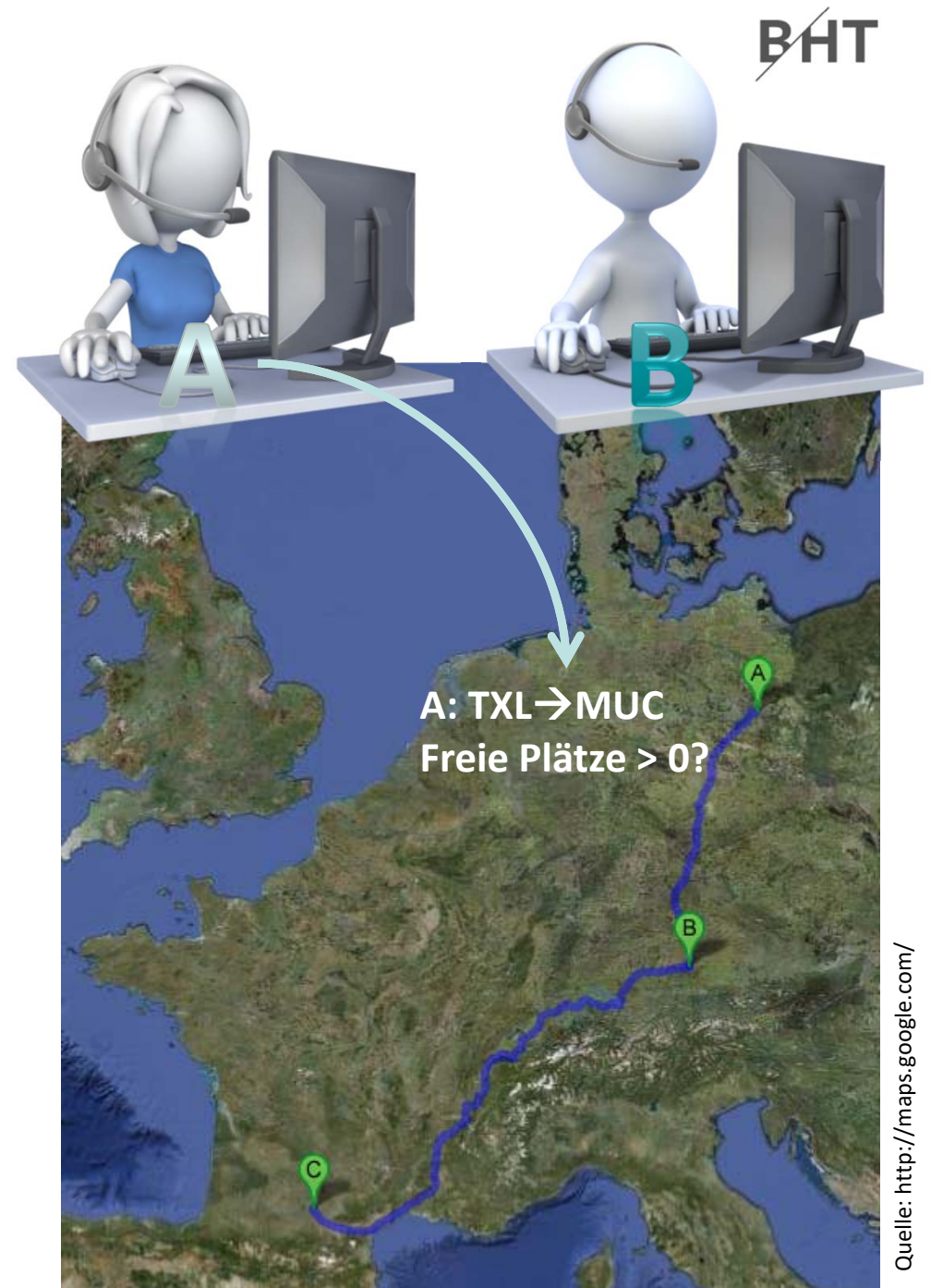
- Ablauf mit mehreren Benutzern



# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

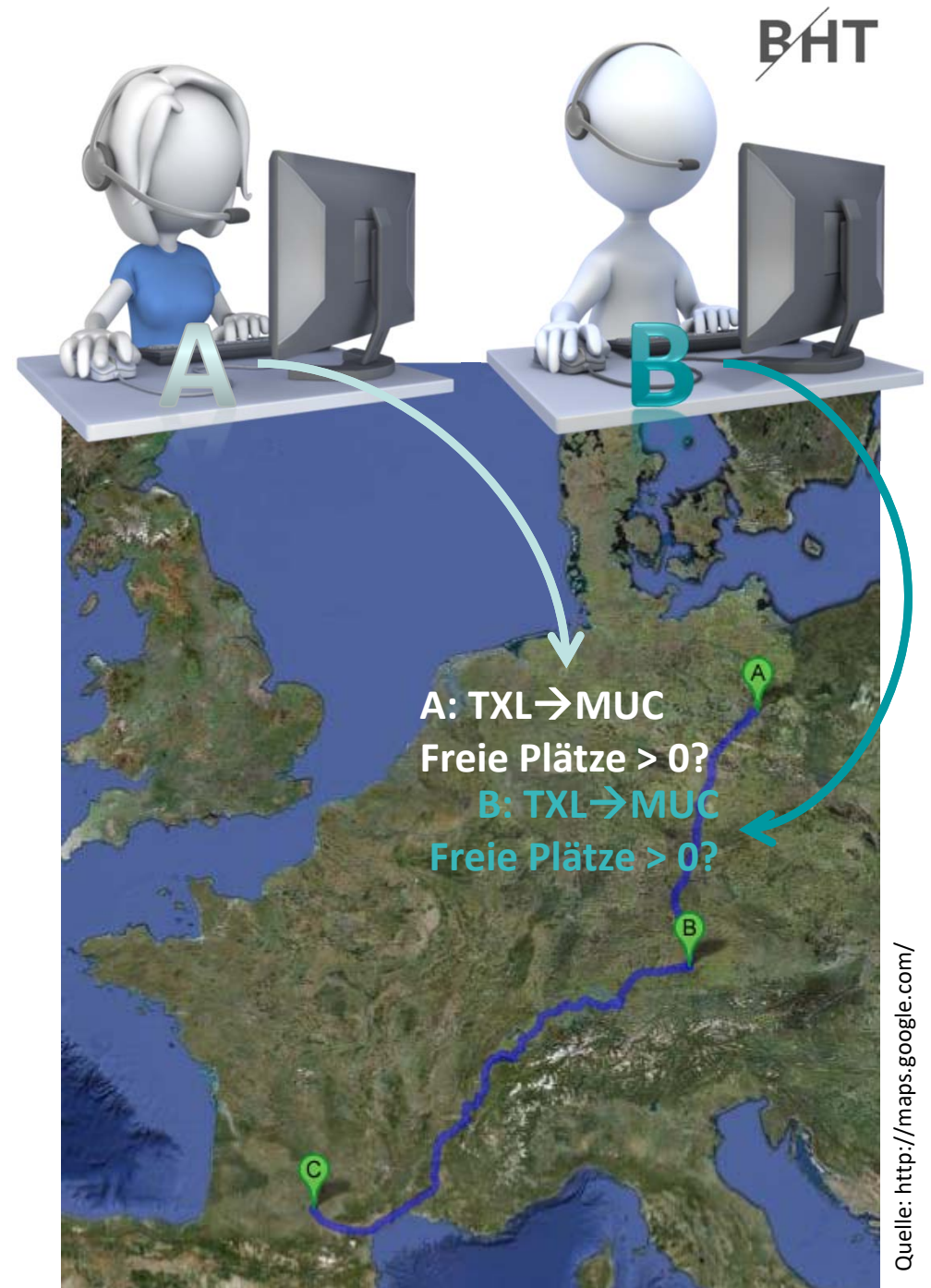
- Ablauf mit mehreren Benutzern
  - A: Lesen freier Plätze



# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

- Ablauf mit mehreren Benutzern
  - A: Lesen freier Plätze
  - B: Lesen freier Plätze

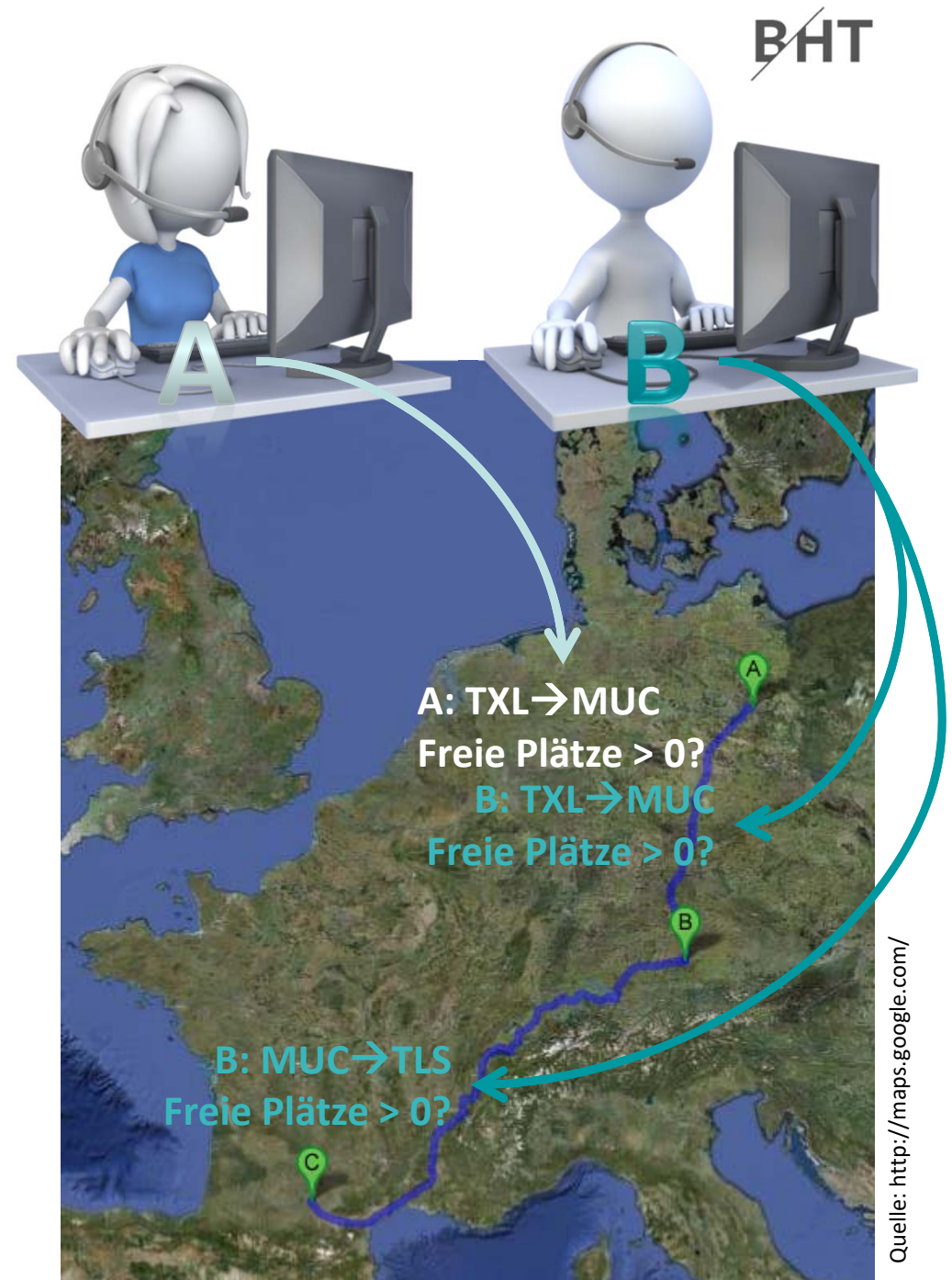




# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

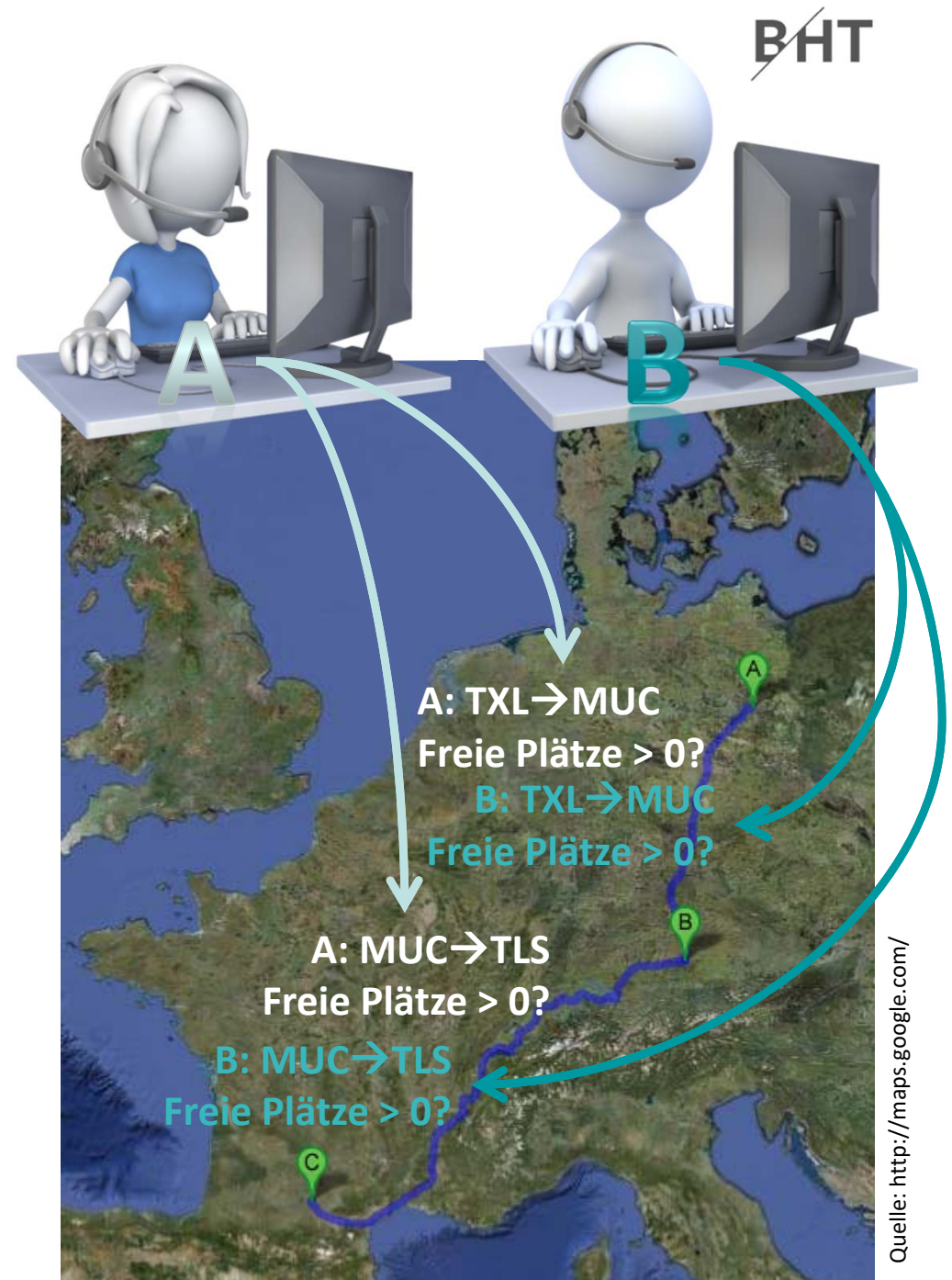
- Ablauf mit mehreren Benutzern
  - A: Lesen freier Plätze
  - B: Lesen freier Plätze



# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

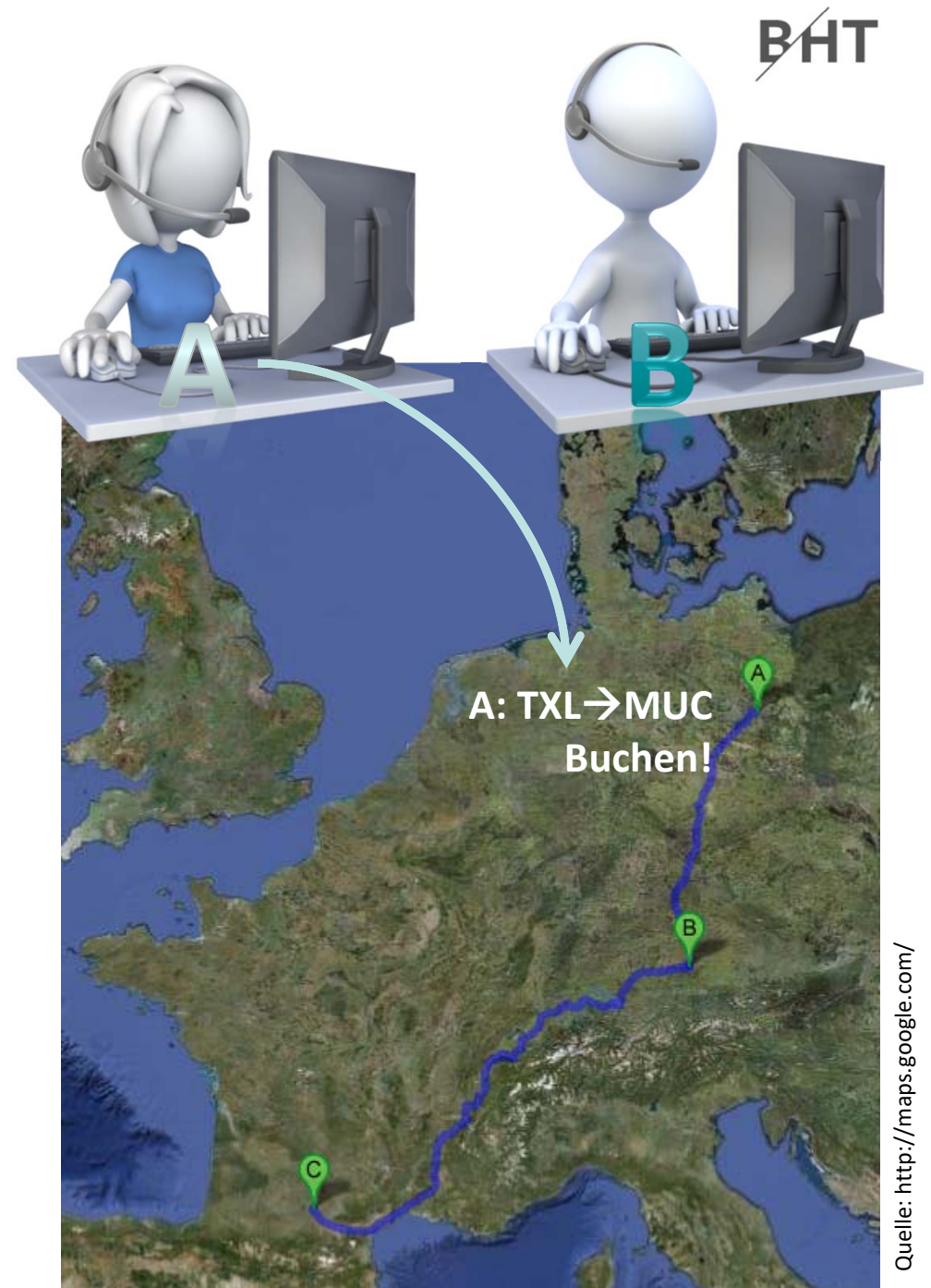
- Ablauf mit mehreren Benutzern
  - A: Lesen freier Plätze
  - B: Lesen freier Plätze



# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

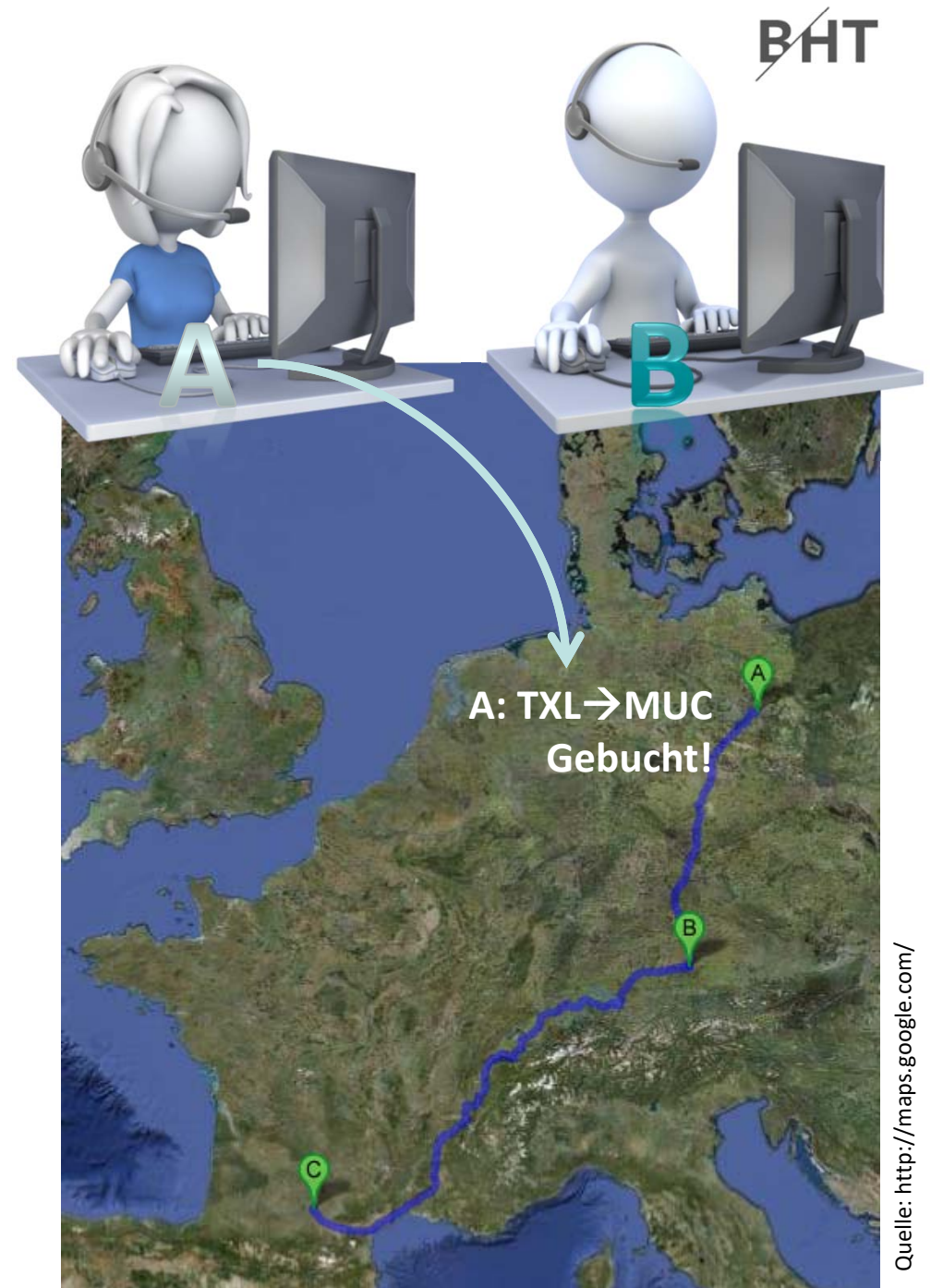
- Ablauf mit mehreren Benutzern
  - A: Lesen freier Plätze
  - B: Lesen freier Plätze
  - A: Buchen Teilstrecke 1



# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

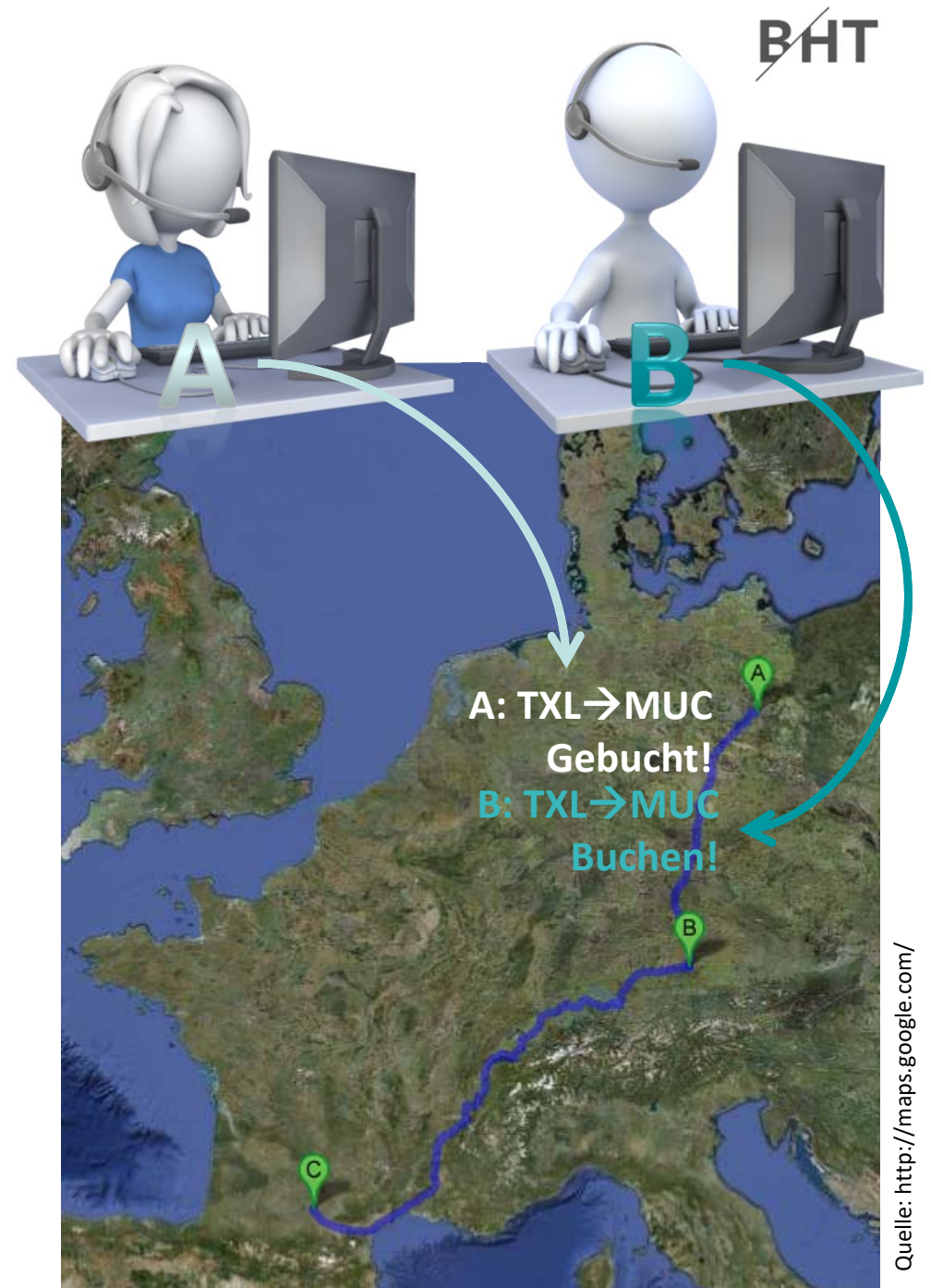
- Ablauf mit mehreren Benutzern
  - A: Lesen freier Plätze
  - B: Lesen freier Plätze
  - A: Buchen Teilstrecke 1



# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

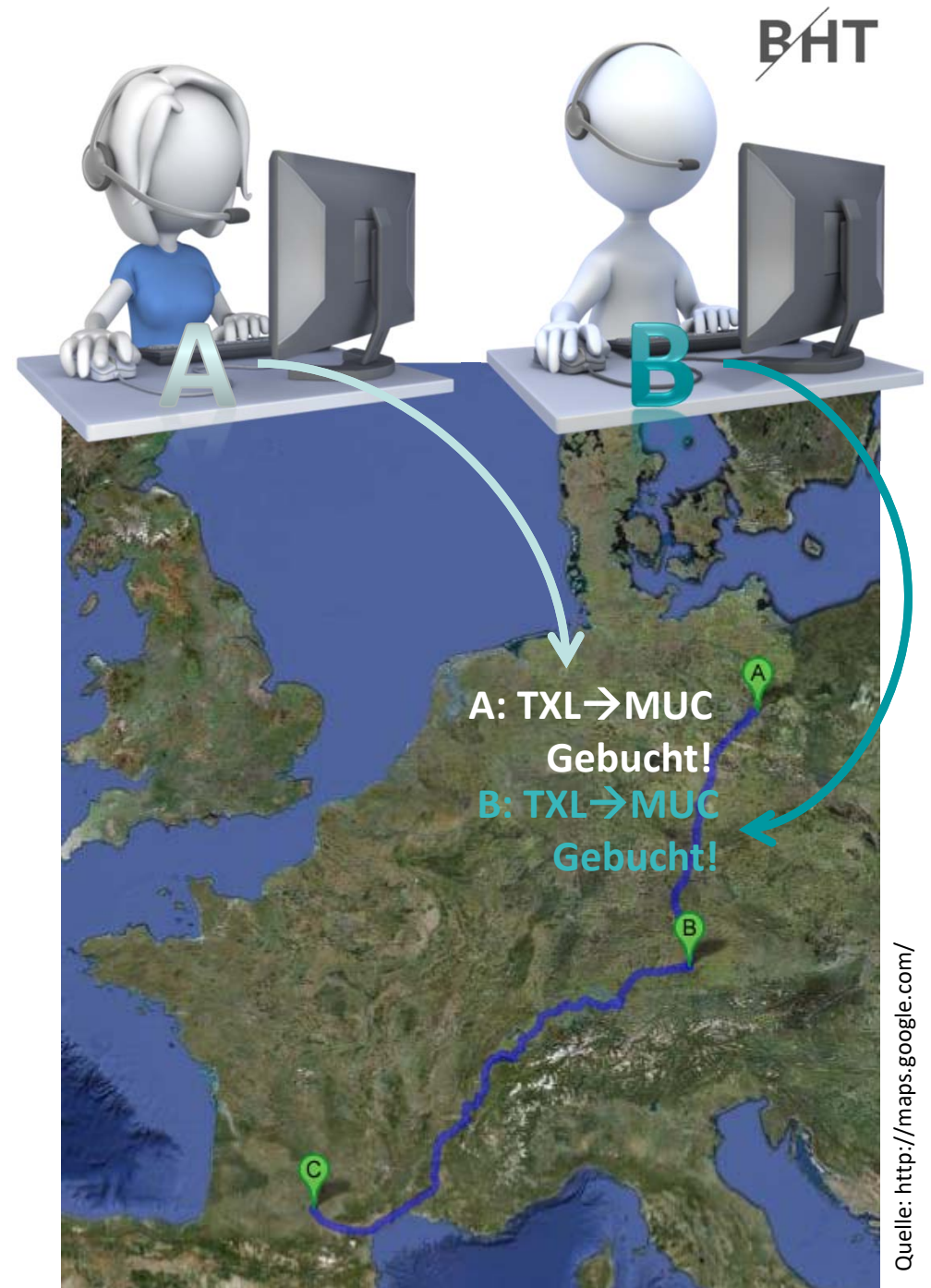
- Ablauf mit mehreren Benutzern
  - A: Lesen freier Plätze
  - B: Lesen freier Plätze
  - A: Buchen Teilstrecke 1
  - B: Buchen Teilstrecke 1



# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

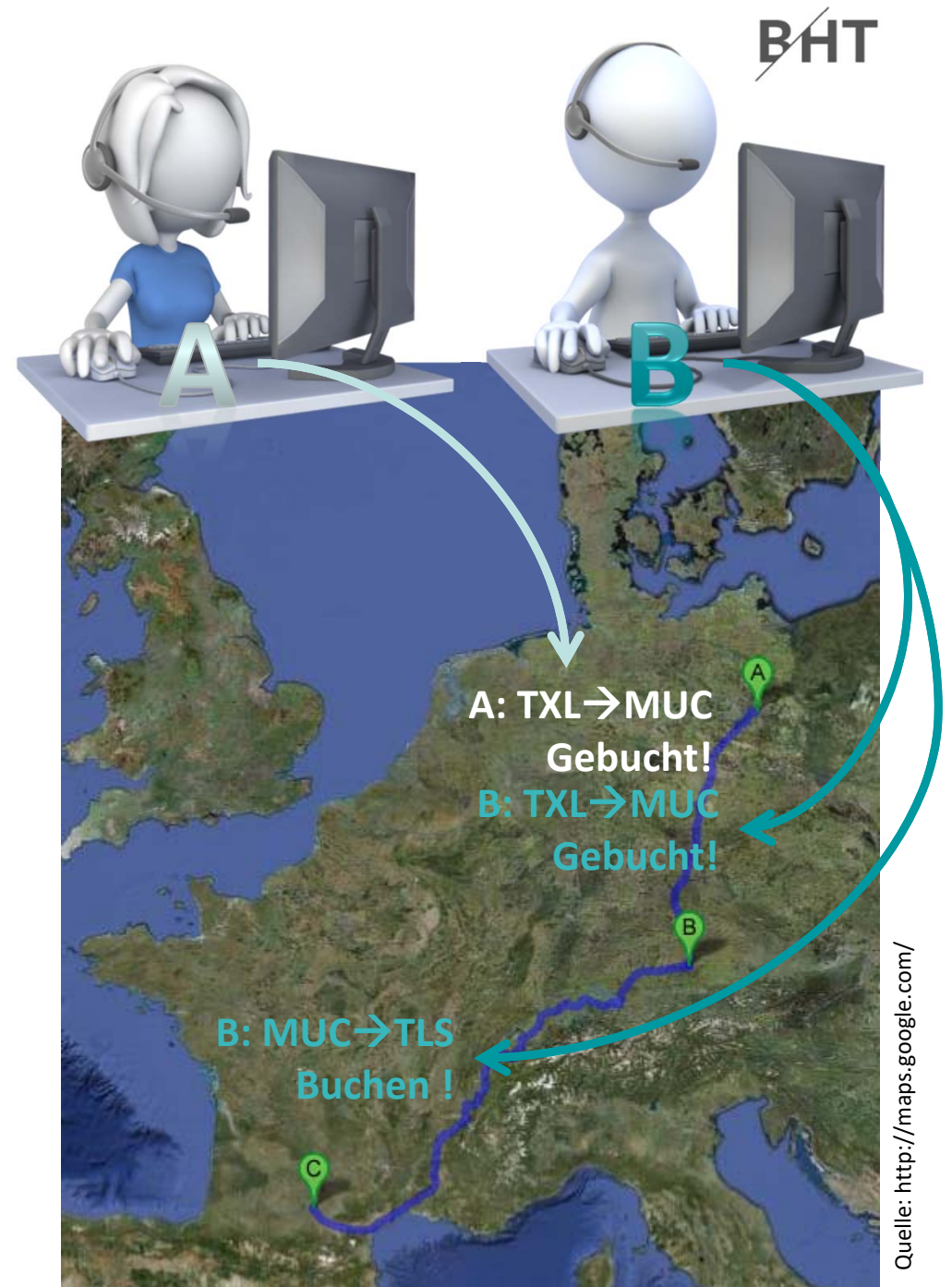
- Ablauf mit mehreren Benutzern
  - A: Lesen freier Plätze
  - B: Lesen freier Plätze
  - A: Buchen Teilstrecke 1
  - B: Buchen Teilstrecke 1



# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

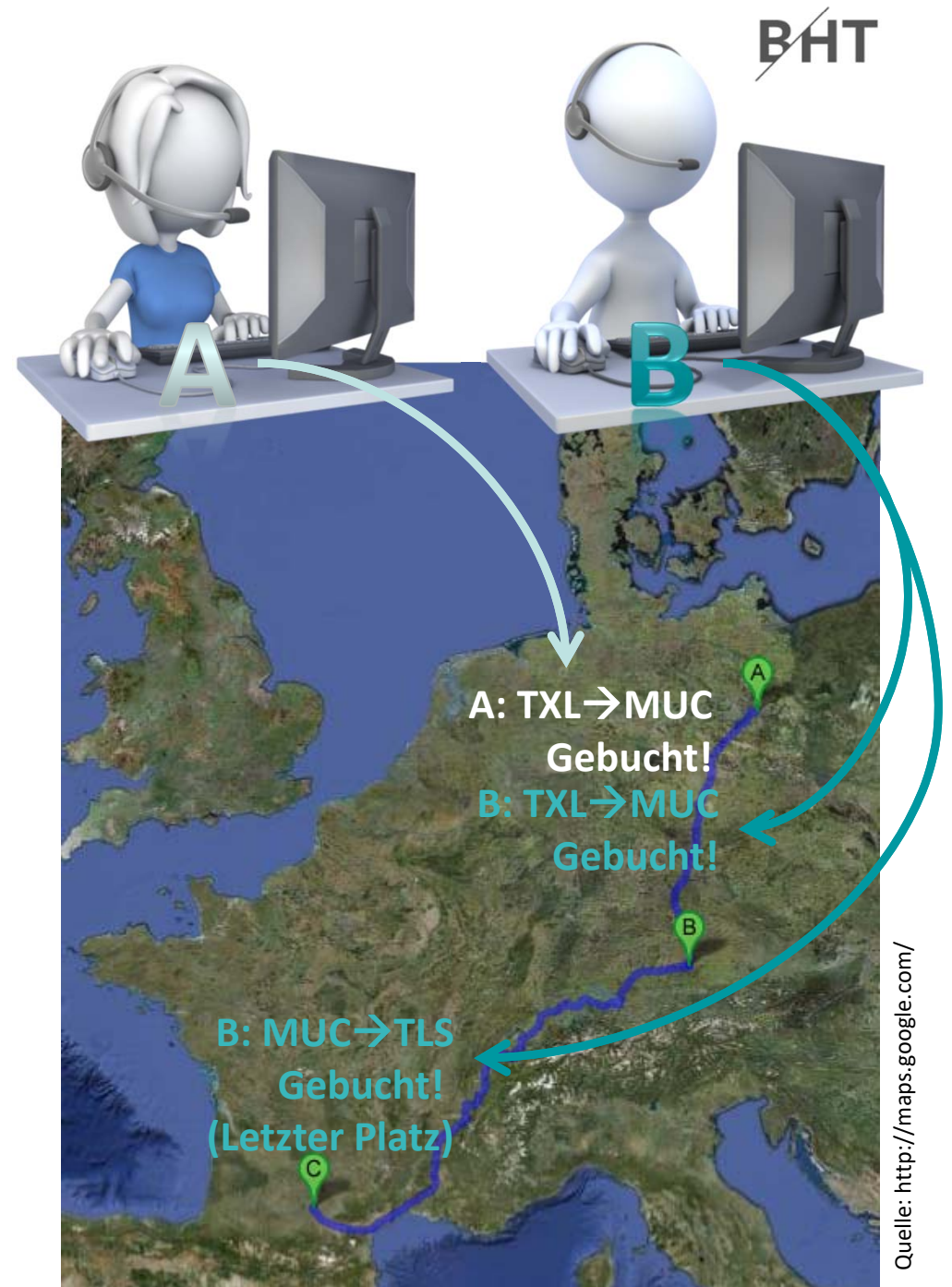
- Ablauf mit mehreren Benutzern
  - A: Lesen freier Plätze
  - B: Lesen freier Plätze
  - A: Buchen Teilstrecke 1
  - B: Buchen Teilstrecke 1
  - B: Buchen Teilstrecke 2



# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

- Ablauf mit mehreren Benutzern
  - A: Lesen freier Plätze
  - B: Lesen freier Plätze
  - A: Buchen Teilstrecke 1
  - B: Buchen Teilstrecke 1
  - B: Buchen Teilstrecke 2 → letzter Platz, puh... das war knapp

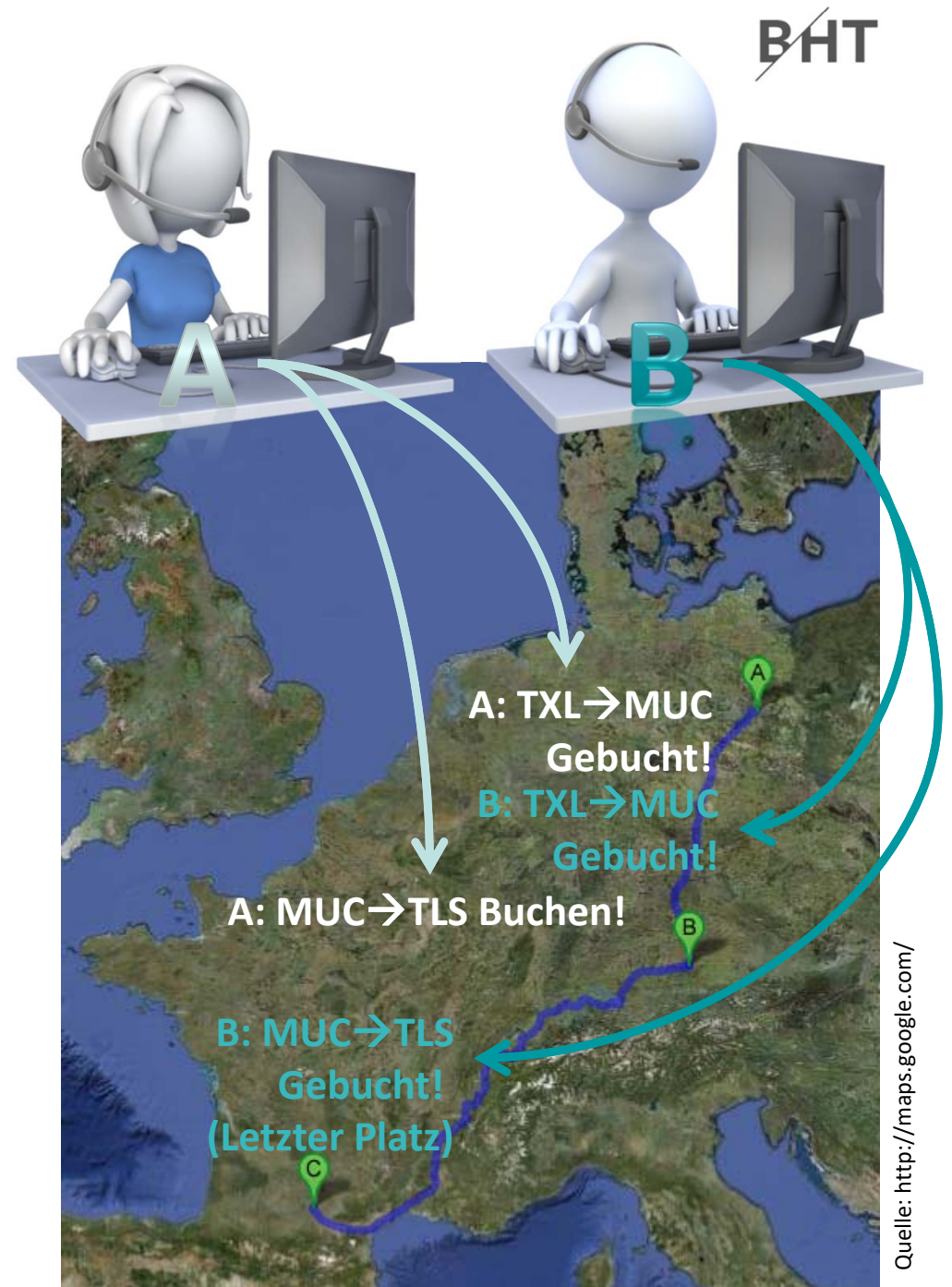




# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

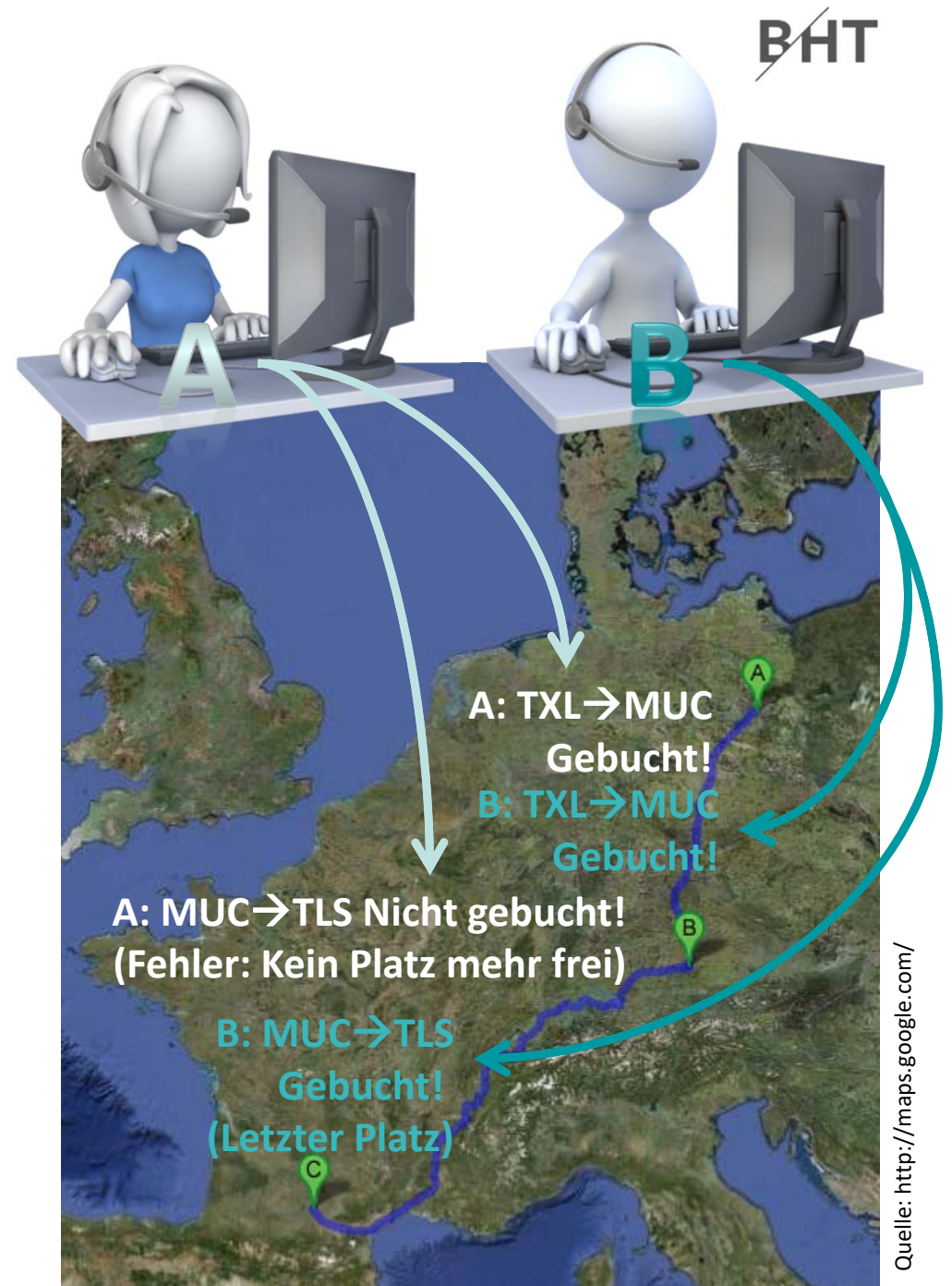
- Ablauf mit mehreren Benutzern
  - A: Lesen freier Plätze
  - B: Lesen freier Plätze
  - A: Buchen Teilstrecke 1
  - B: Buchen Teilstrecke 1
  - B: Buchen Teilstrecke 2 → letzter Platz, puh... das war knapp
  - A: Buchen Teilstrecke 2



# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

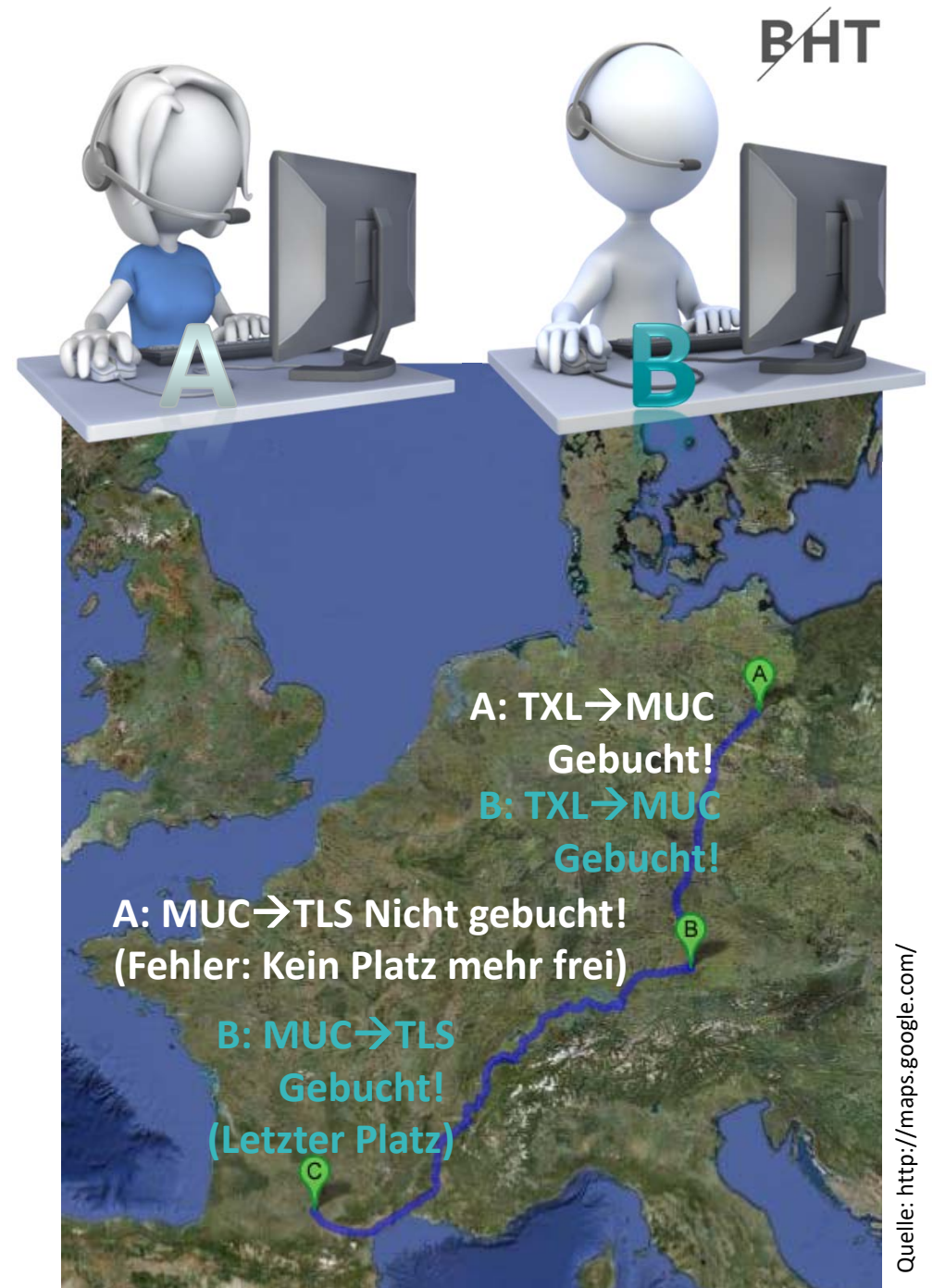
- Ablauf mit mehreren Benutzern
  - A: Lesen freier Plätze
  - B: Lesen freier Plätze
  - A: Buchen Teilstrecke 1
  - B: Buchen Teilstrecke 1
  - B: Buchen Teilstrecke 2 → letzter Platz, puh... das war knapp
  - A: Buchen Teilstrecke 2 → Fehler, kein Platz mehr frei



# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

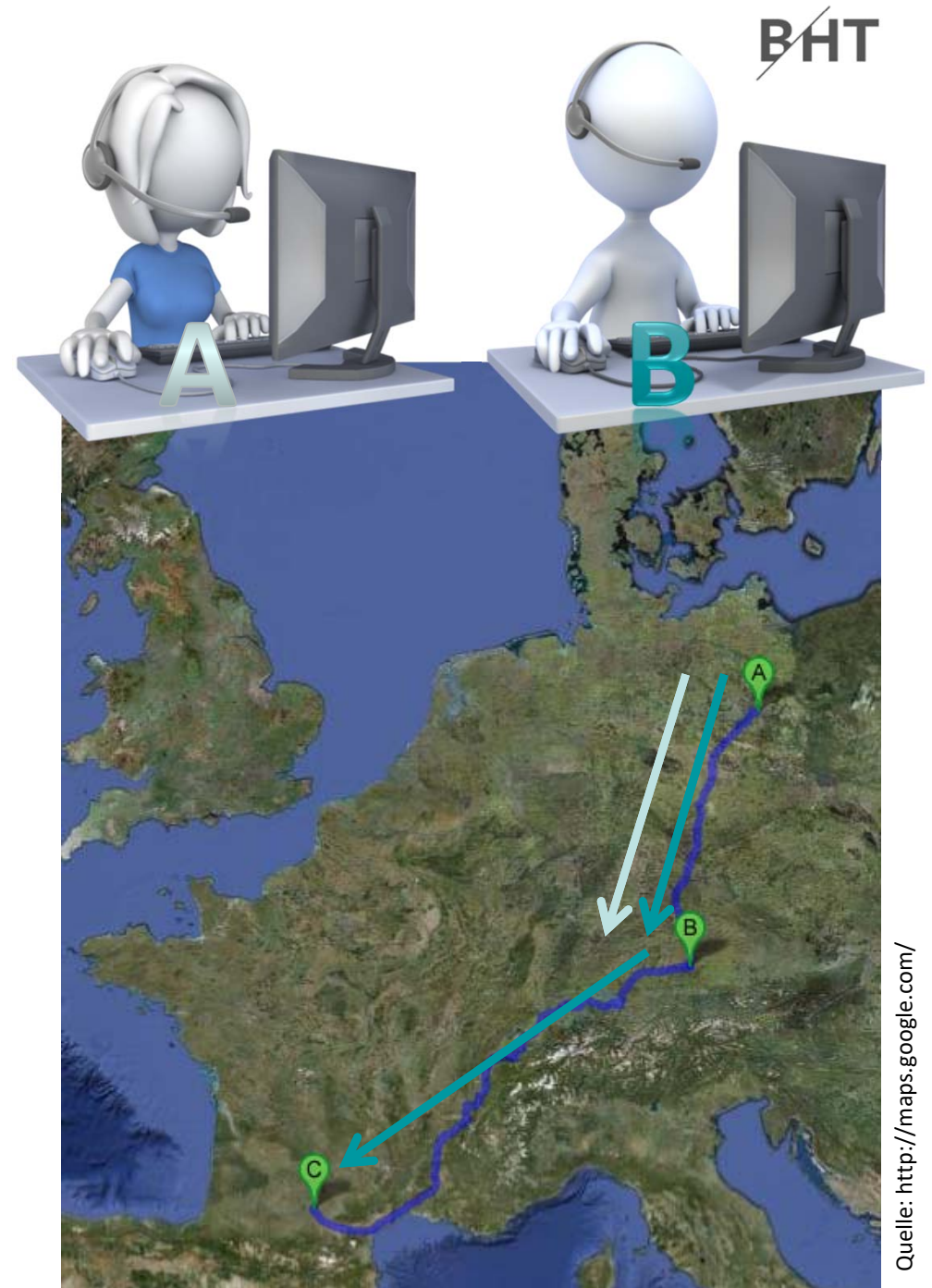
- Ablauf mit mehreren Benutzern
- Kann bei konkurrierenden Zugriffen dazu führen, dass Operationen nicht für alle Benutzer erfolgreich abgeschlossen werden



# Konsistenz und Integrität

## Beispiel 2: Flugbuchung TXL nach TLS via MUC

- Ablauf mit mehreren Benutzern
- Kann bei konkurrierenden Zugriffen dazu führen, dass Operationen nicht für alle Benutzer erfolgreich abgeschlossen werden
- Konsistenzbedingung
  - nicht erfüllt für A
  - erfüllt für B



## Zwischenstand

- es gibt elementare Operationen
  - SELECT
  - INSERT
  - UPDATE
  - DELETE
- elementare Operationen werden immer vollständig oder gar nicht ausgeführt, d.h.
  - führen die Daten von einem technisch konsistenten Zustand in einen anderen technisch konsistenten Zustand
  - können logische Konsistenz nicht gewährleisten (z.B. Verschwinden von 100 € oder Buchung einer Teilstrecke anstelle der gesamten Reise)

# Inhalt

## Ziel und Einordnung

## Rückblick

## Transaktionen

- Konsistenz und Integrität
  - Konsistenzsicherung als Ziel relationaler Datenbanken
  - Gefahren für die Konsistenz von Datenbanken
- Transaktionen und ACID
- Anwendung von Transaktionen
  - Anwendungsszenarien
  - Transaktionen in SQL
  - Transaktionen mit MS Access
- Technik der Transaktionsverarbeitung
  - Parallele Transaktionen und deren Probleme
  - Transaktionslevel
  - Umsetzung paralleler Transaktionen
  - Protokollierung
- Zusammenfassung

## Ausblick

# Transaktionen

## Definition: Folge von Datenbankoperationen,

- die hinsichtlich der Konsistenz/Integritätsbedingungen als atomare Einheit angesehen wird.<sup>1</sup>
- die ausgehend von einem konsistenten Zustand der Datenbank immer in einen konsistenten Zustand führt.<sup>2</sup>
- die mit besonderen Kommandos
  - begonnen,
  - erfolgreich abgeschlossen oder
  - nicht erfolgreich beendet wird



1) vgl. A. Fink, G. Schneiderreit, S. Voß: Grundlagen der Wirtschaftsinformatik, Physika-Verlag (Springer), 2001; S. 139

2) vgl. E. Schicker: Datenbanken und SQL, Teubner Verlag, 1996; S. 59f.

# Transaktionen

## ACID-Eigenschaften

- Atomarität
- Konsistenz (Consistency)
- Isolation
- Dauerhaftigkeit





# Transaktionen

## ACID-Eigenschaften

- Atomarität
  - Transaktion wird entweder vollständig oder gar nicht ausgeführt
  - tritt bei einer Operation der Transaktion ein Fehler auf, werden diese Operation und alle bereits (erfolgreich) ausgeführten Operationen zurückgesetzt
- Konsistenz (Consistency)
- Isolation
- Dauerhaftigkeit



# Transaktionen

## ACID-Eigenschaften

- Atomarität
- Konsistenz (Consistency)
  - Transaktion führt die Datenbank stets von einem konsistenten Zustand in den nächsten konsistenten Zustand
  - Vor und nach der Ausführung der Transaktion sind stets alle Integritätsbedingungen erfüllt
- Isolation
- Dauerhaftigkeit



# Transaktionen

## ACID-Eigenschaften

- Atomarität
- Konsistenz (Consistency)
- Isolation
  - Transaktionen laufen isoliert voneinander ab, d.h. aus Sicht des Benutzers verhält sich Datenbank so, als sei er der einzige Benutzer
  - parallele Transaktionen werden so ausgeführt, als würden sie nacheinander ablaufen, aber tatsächlich laufen sie parallel ab
  - DBMS stellt Isolation durch verschiedene Mechanismen sicher (z.B. aus Performance-Gründen nicht immer nacheinander sinnvoll)
- Dauerhaftigkeit



# Transaktionen

## ACID-Eigenschaften

- Atomarität
- Konsistenz (Consistency)
- Isolation
- Dauerhaftigkeit
  - abgeschlossene Transaktionen müssen auch nach einem unmittelbar anschließenden Fehlerzustand gespeichert sein
  - insbesondere, auch wenn
    - Stromausfall zum Löschen des Cache-Speichers im RAM führt
    - Festplattendefekt die Datenbank-Datei zerstört



# Transaktionen

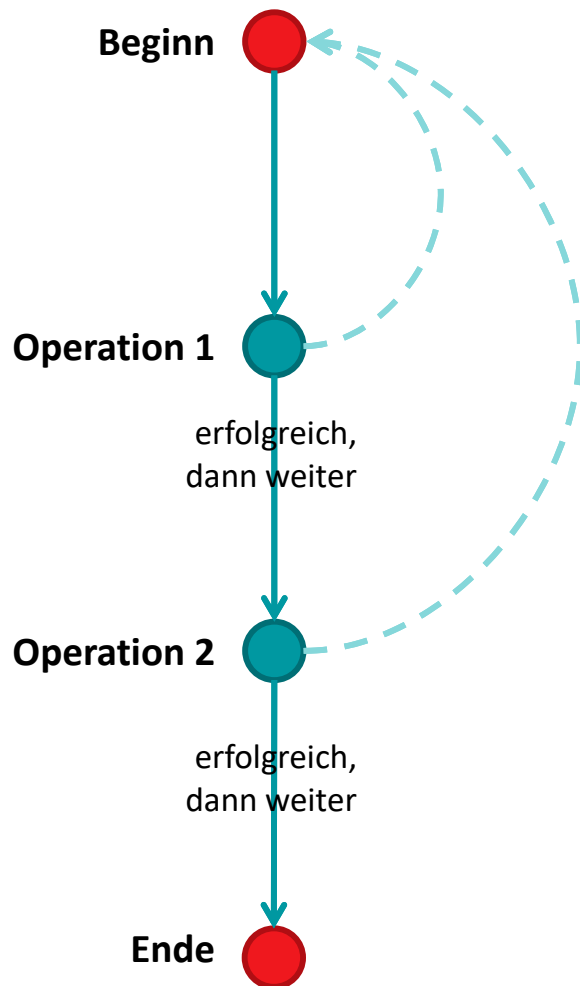
## ACID-Eigenschaften

- Atomarität
- Konsistenz (Consistency)
- Isolation
- Dauerhaftigkeit



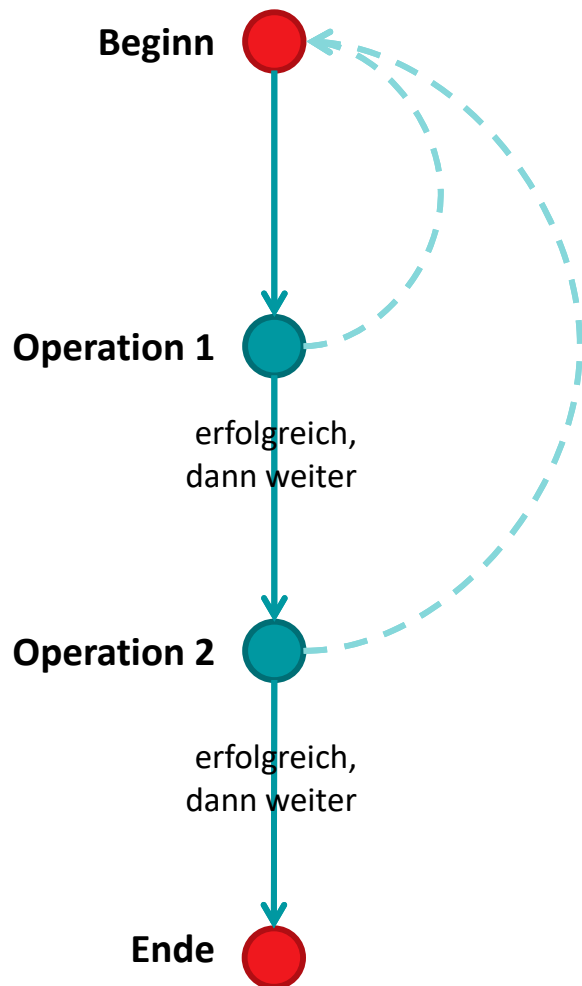
# Transaktionen

Erfolgreich  
abgeschlossene  
Transaktion

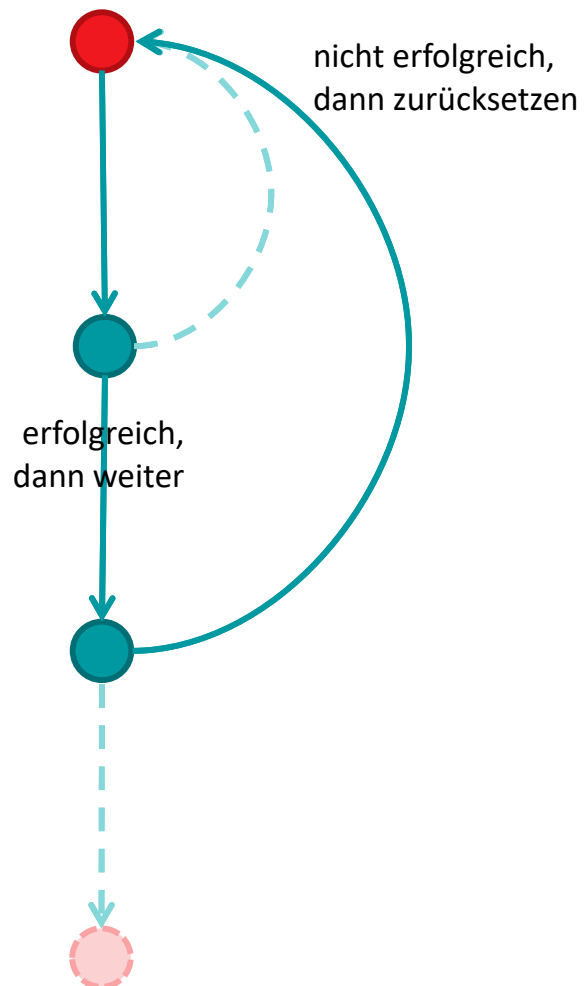


# Transaktionen

**Erfolgreich  
abgeschlossene  
Transaktion**

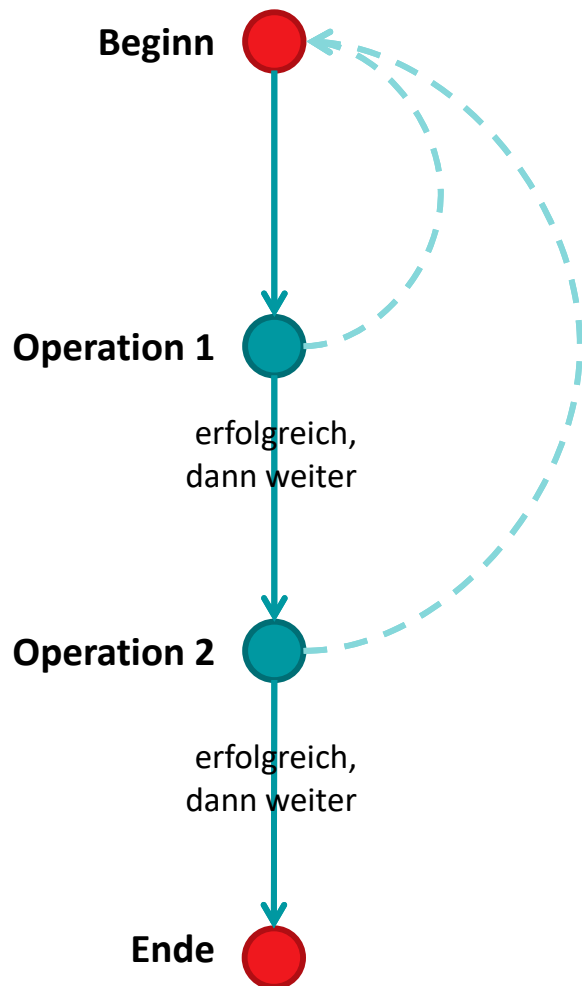


**Nicht erfolgreich  
beendete  
Transaktion**

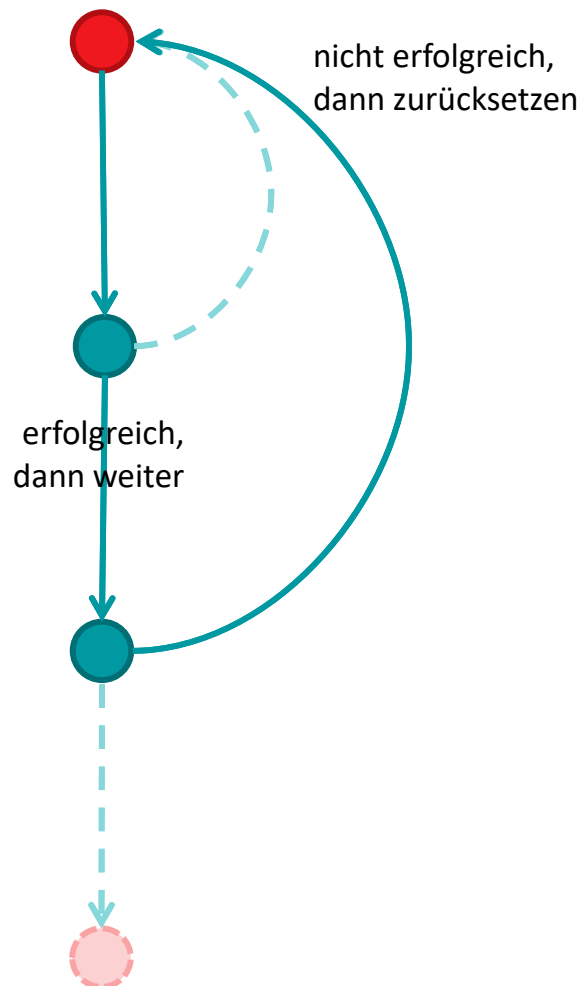


# Transaktionen

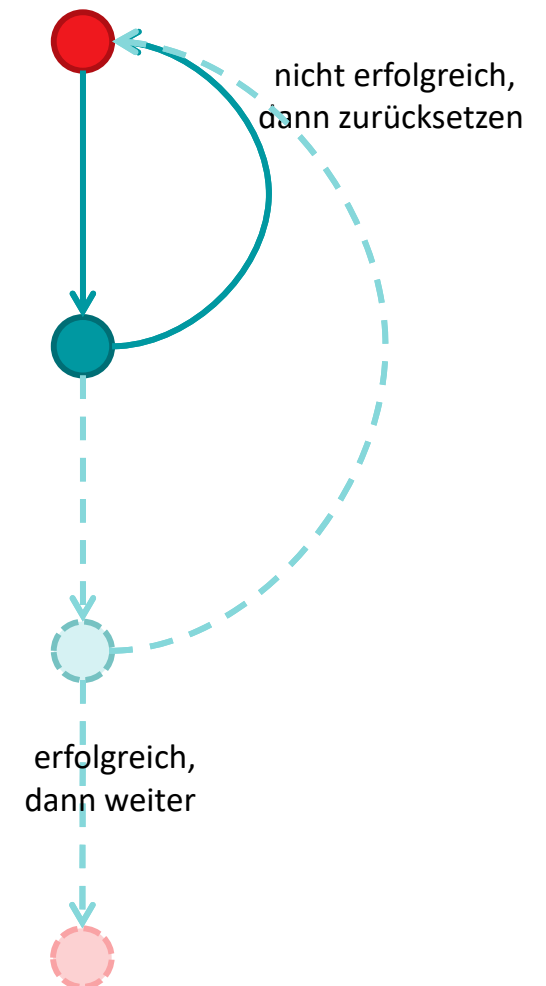
**Erfolgreich  
abgeschlossene  
Transaktion**



**Nicht erfolgreich  
beendete  
Transaktion**



**Nicht erfolgreich  
beendete  
Transaktion**





# Inhalt

## Ziel und Einordnung

## Rückblick

## Transaktionen

- Konsistenz und Integrität
  - Konsistenzsicherung als Ziel relationaler Datenbanken
  - Gefahren für die Konsistenz von Datenbanken
- Transaktionen und ACID
- Anwendung von Transaktionen
  - Anwendungsszenarien
  - Transaktionen in SQL
  - Transaktionen mit MS Access
- Technik der Transaktionsverarbeitung
  - Parallele Transaktionen und deren Probleme
  - Transaktionslevel
  - Umsetzung paralleler Transaktionen
  - Protokollierung
- Zusammenfassung

## Ausblick

# Anwendung von Transaktionen

## Anwendung der Transaktionen auf

- Beispiel 1: Überweisung zwischen Konten
- Beispiel 2: Flugbuchung von TXL nach TLS via MUC

# Anwendung von Transaktionen

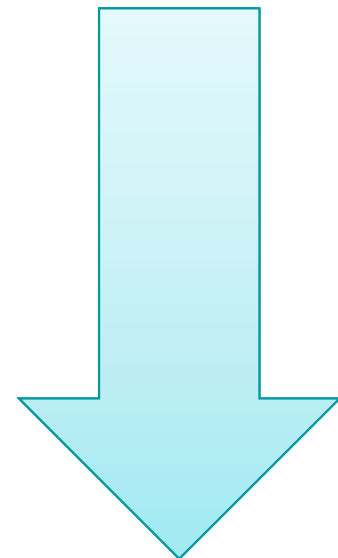
## Mit Transaktion:



## Vorher:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

**Summe: 800 € + 500 € = 1300 €**



## Nacher:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	<b>600</b>

**Summe: 700 € + 600 € = 1300 €**

# Anwendung von Transaktionen

Mit Transaktion:



**Beginn Transaktion**

```
UPDATE Konten
SET Saldo =
    Saldo - 100
WHERE KtoNr = 2345;
```

```
UPDATE Konten
SET Saldo =
    Saldo + 100
WHERE KtoNr = 4567;
```

**Abschluss Transaktion**



Vorher:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

**Summe: 800 € + 500 € = 1300 €**

# Anwendung von Transaktionen

Mit Transaktion:

Beginn Transaktion

```
UPDATE Konten
SET Saldo =
    Saldo - 100
WHERE KtoNr = 2345;
```

```
UPDATE Konten
SET Saldo =
    Saldo + 100
WHERE KtoNr = 4567;
```

Abschluss Transaktion



Vorher:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

Summe: 800 € + 500 € = 1300 €

# Anwendung von Transaktionen

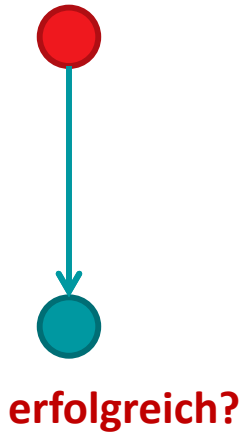
## Mit Transaktion:

### Beginn Transaktion

```
UPDATE Konten
SET Saldo =
    Saldo - 100
WHERE KtoNr = 2345;
```

```
UPDATE Konten
SET Saldo =
    Saldo + 100
WHERE KtoNr = 4567;
```

### Abschluss Transaktion



## Vorher:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

Summe: 800 € + 500 € = 1300 €

## Zwischenzeitlich:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	500

Summe: 700 € + 500 € = 1200 €

# Anwendung von Transaktionen

## Mit Transaktion:

### Beginn Transaktion

```
UPDATE Konten  
SET Saldo =  
    Saldo - 100  
WHERE KtoNr = 2345;
```

```
UPDATE Konten  
SET Saldo =  
    Saldo + 100  
WHERE KtoNr = 4567;
```

### Abschluss Transaktion




## Vorher:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

Summe: 800 € + 500 € = 1300 €

## Zwischenzeitlich:



Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	500

Summe: 700 € + 500 € = 1200 €

# Anwendung von Transaktionen

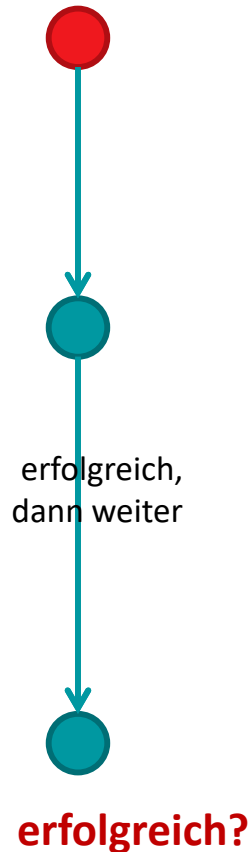
## Mit Transaktion:

### Beginn Transaktion

```
UPDATE Konten
SET Saldo =
    Saldo - 100
WHERE KtoNr = 2345;
```

```
UPDATE Konten
SET Saldo =
    Saldo + 100
WHERE KtoNr = 4567;
```

### Abschluss Transaktion



## Vorher:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

Summe: 800 € + 500 € = 1300 €

## Zwischenzeitlich:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	500

Summe: 700 € + 500 € = 1200 €

## Nacher:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	600

Summe: 700 € + 600 € = 1300 €



# Anwendung von Transaktionen

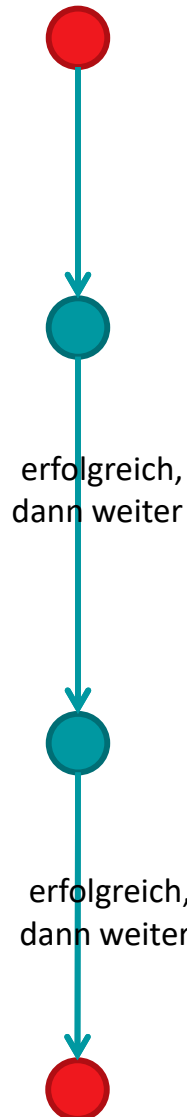
## Mit Transaktion:

### Beginn Transaktion

```
UPDATE Konten  
SET Saldo =  
    Saldo - 100  
WHERE KtoNr = 2345;
```

```
UPDATE Konten  
SET Saldo =  
    Saldo + 100  
WHERE KtoNr = 4567;
```

### Abschluss Transaktion




## Vorher:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

Summe: 800 € + 500 € = 1300 €


## Zwischenzeitlich:



Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	500

Summe: 700 € + 500 € = 1200 €

## Nacher:



Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	600

Summe: 700 € + 600 € = 1300 €

# Anwendung von Transaktionen

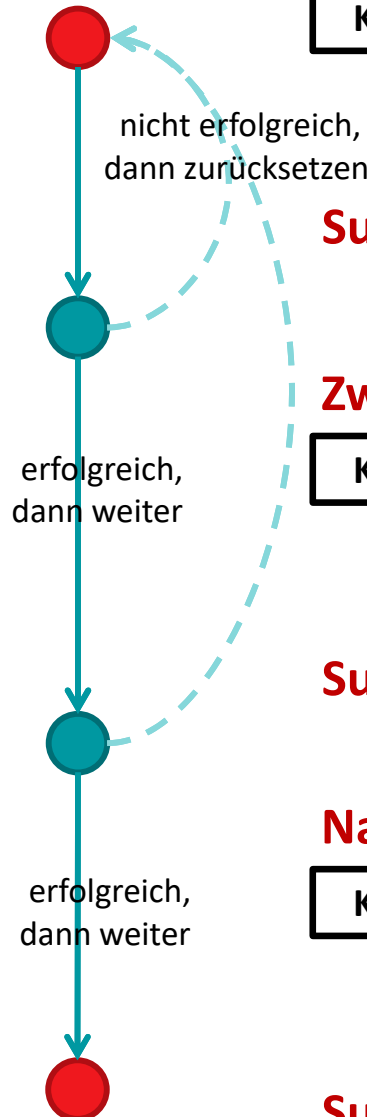
## Mit Transaktion:

### Beginn Transaktion

```
UPDATE Konten  
SET Saldo =  
    Saldo - 100  
WHERE KtoNr = 2345;
```

```
UPDATE Konten  
SET Saldo =  
    Saldo + 100  
WHERE KtoNr = 4567;
```

### Abschluss Transaktion




## Vorher:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

**Summe: 800 € + 500 € = 1300 €**


## Zwischenzeitlich:



Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	500

**Summe: 700 € + 500 € = 1200 €**

## Nacher:



Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	<b>600</b>

**Summe: 700 € + 600 € = 1300 €**

# Anwendung von Transaktionen

Mit Transaktion:



**Beginn Transaktion**

```
UPDATE Konten
SET Saldo =
    Saldo - 100
WHERE KtoNr = 2345;
```

```
UPDATE Konten
SET Saldo =
    Saldo + 100
WHERE KtoNr = 4567;
```

**Abschluss Transaktion**



Vorher:


Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

**Summe: 800 € + 500 € = 1300 €**

# Anwendung von Transaktionen

Mit Transaktion:

Beginn Transaktion



```
UPDATE Konten
SET Saldo =
    Saldo - 100
WHERE KtoNr = 2345;
```

```
UPDATE Konten
SET Saldo =
    Saldo + 100
WHERE KtoNr = 4567;
```

Abschluss Transaktion



Vorher:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

Summe: 800 € + 500 € = 1300 €

# Anwendung von Transaktionen

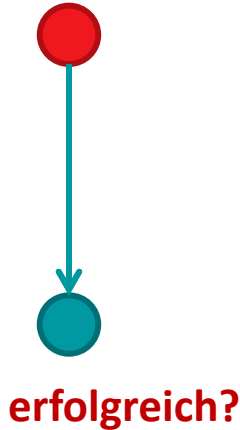
## Mit Transaktion:

### Beginn Transaktion

```
UPDATE Konten  
SET Saldo =  
    Saldo - 100  
WHERE KtoNr = 2345;
```

```
UPDATE Konten  
SET Saldo =  
    Saldo + 100  
WHERE KtoNr = 4567;
```

### Abschluss Transaktion




## Vorher:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

Summe: 800 € + 500 € = 1300 €

## Zwischenzeitlich:



Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	500

Summe: 700 € + 500 € = 1200 €

# Anwendung von Transaktionen

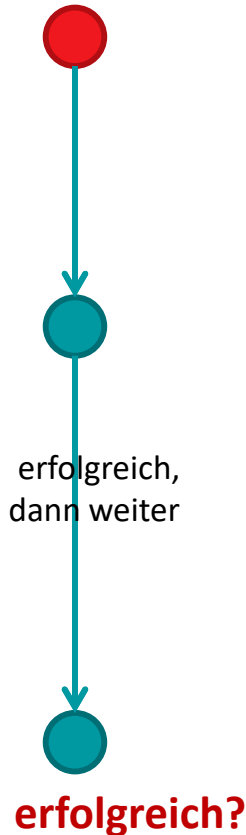
## Mit Transaktion:

### Beginn Transaktion

```
UPDATE Konten  
SET Saldo =  
    Saldo - 100  
WHERE KtoNr = 2345;
```

```
UPDATE Konten  
SET Saldo =  
    Saldo + 100  
WHERE KtoNr = 4567;
```

### Abschluss Transaktion



## Vorher:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

Summe: 800 € + 500 € = 1300 €

## Zwischenzeitlich:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	500

Summe: 700 € + 500 € = 1200 €



# Anwendung von Transaktionen

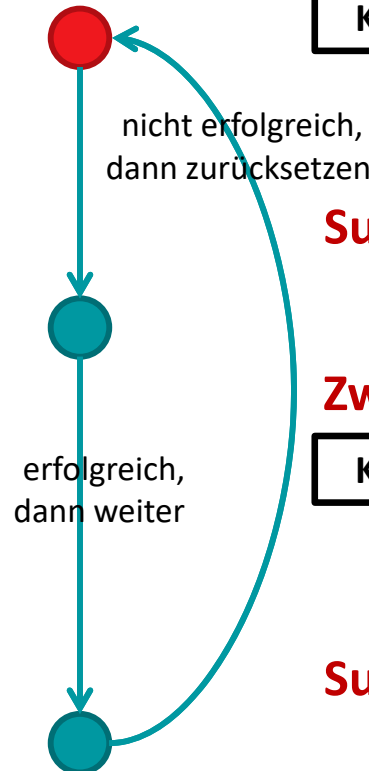
Mit Transaktion:

Beginn Transaktion

```
UPDATE Konten  
SET Saldo =  
    Saldo - 100  
WHERE KtoNr = 2345;
```

```
UPDATE Konten  
SET Saldo =  
    Saldo + 100  
WHERE KtoNr = 4567;
```

Zurückrollen Transaktion



Vorher:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

Summe: 800 € + 500 € = 1300 €

Zwischenzeitlich:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	500

Summe: 700 € + 500 € = 1200 €



# Anwendung von Transaktionen

Mit Transaktion:

Beginn Transaktion

```
UPDATE Konten
SET Saldo =
    Saldo - 100
WHERE KtoNr = 2345;
```

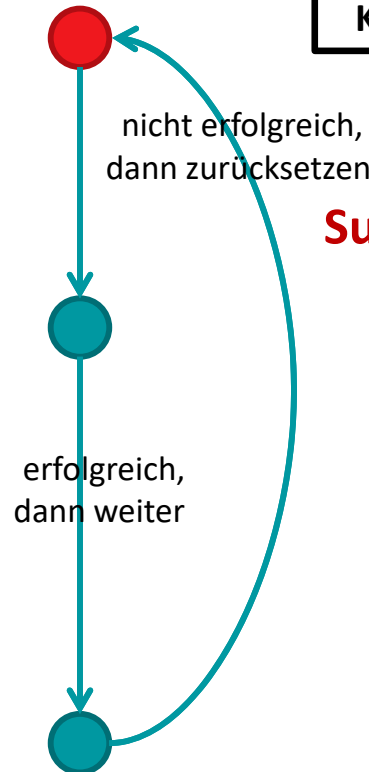
```
UPDATE Konten
SET Saldo =
    Saldo + 100
WHERE KtoNr = 4567;
```

Zurückrollen Transaktion

Vorher:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

Summe: 800 € + 500 € = 1300 €





# Anwendung von Transaktionen

Mit Transaktion:



Beginn Transaktion

```
UPDATE Konten
SET Saldo =
    Saldo - 100
WHERE KtoNr = 2345;
```

```
UPDATE Konten
SET Saldo =
    Saldo + 100
WHERE KtoNr = 4567;
```

Zurückrollen Transaktion



Vorher:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

Summe: 800 € + 500 € = 1300 €

# Anwendung von Transaktionen

## Mit Transaktion:

### Beginn Transaktion

```
UPDATE Konten  
  SET Saldo =  
      Saldo - 100  
  WHERE KtoNr = 2345;
```

```
UPDATE Konten  
  SET Saldo =  
      Saldo + 100  
  WHERE KtoNr = 4567;
```

### Abschluss Transaktion



## Vorher:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

**Summe: 800 € + 500 € = 1300 €**

# Anwendung von Transaktionen

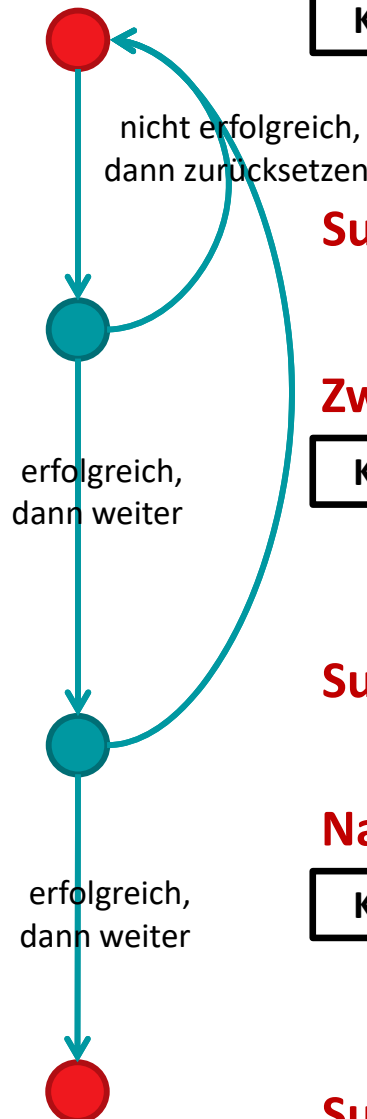
## Mit Transaktion:

### Beginn Transaktion

```
UPDATE Konten  
SET Saldo =  
    Saldo - 100  
WHERE KtoNr = 2345;
```

```
UPDATE Konten  
SET Saldo =  
    Saldo + 100  
WHERE KtoNr = 4567;
```

### Abschluss Transaktion



## Vorher:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	800
	4567	Yilmaz	500

Summe: 800 € + 500 € = 1300 €

## Zwischenzeitlich:

Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	500

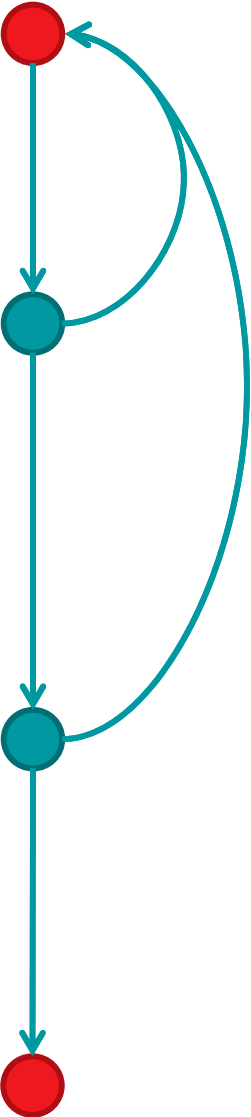
Summe: 700 € + 500 € = 1200 €

## Nacher:

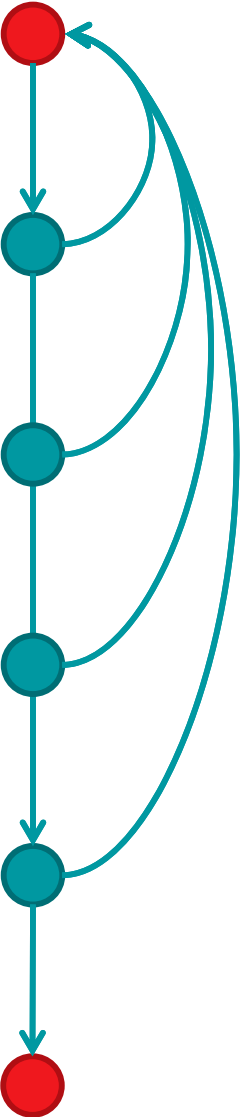
Konten	<u>KtoNr</u>	Name	Saldo
	2345	Müller	700
	4567	Yilmaz	600

Summe: 700 € + 600 € = 1300 €

# Anwendung von Transaktionen

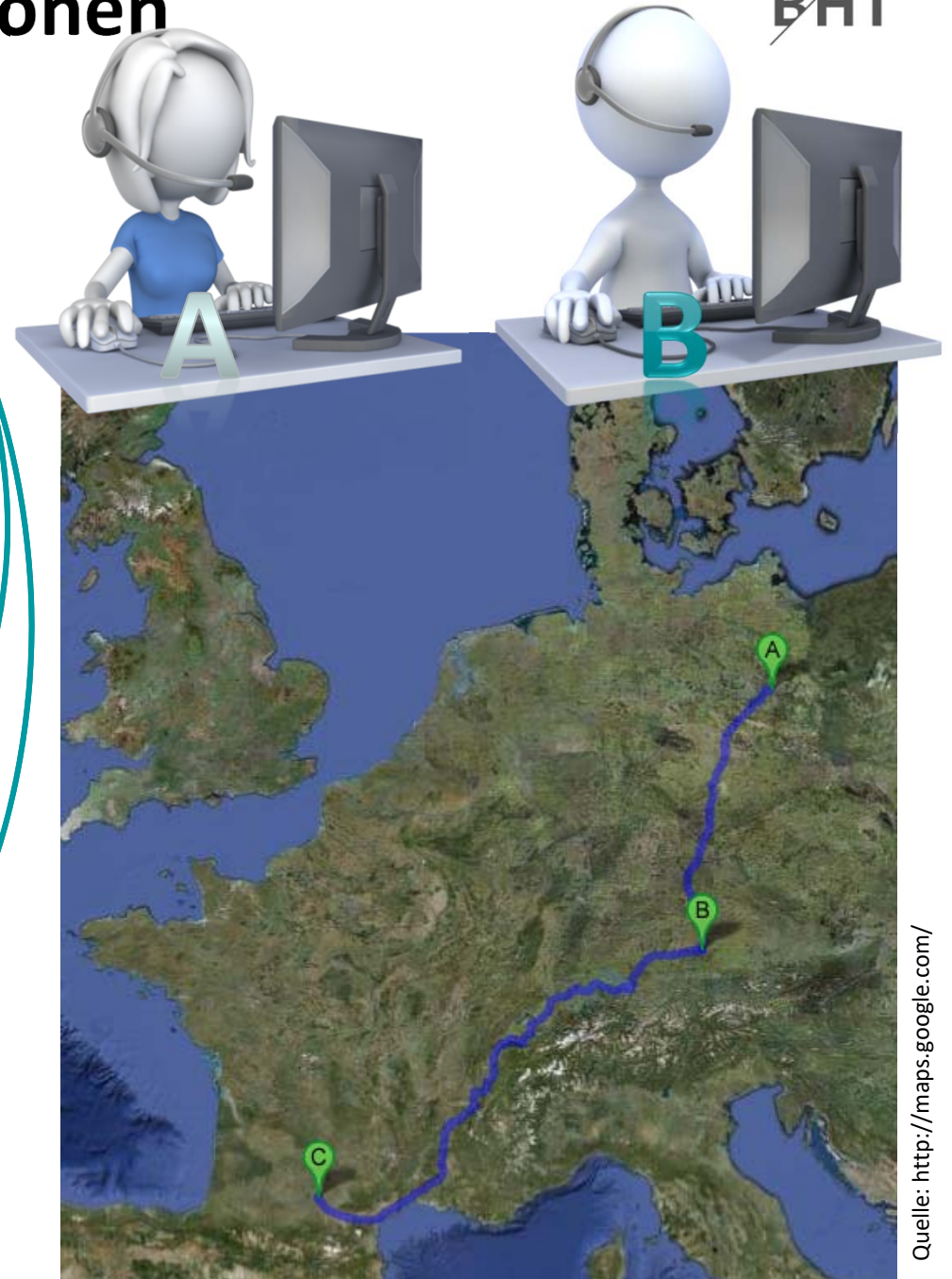
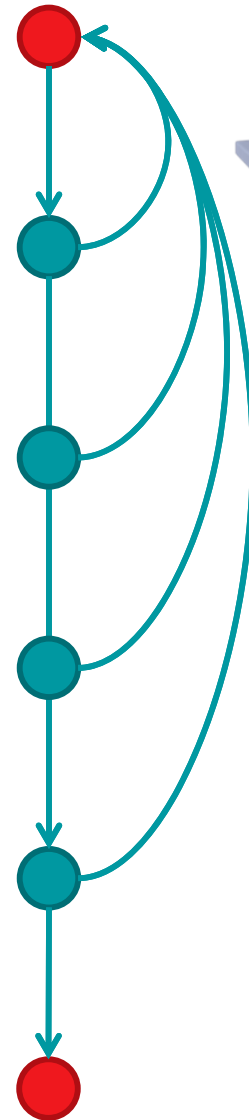


# Anwendung von Transaktionen



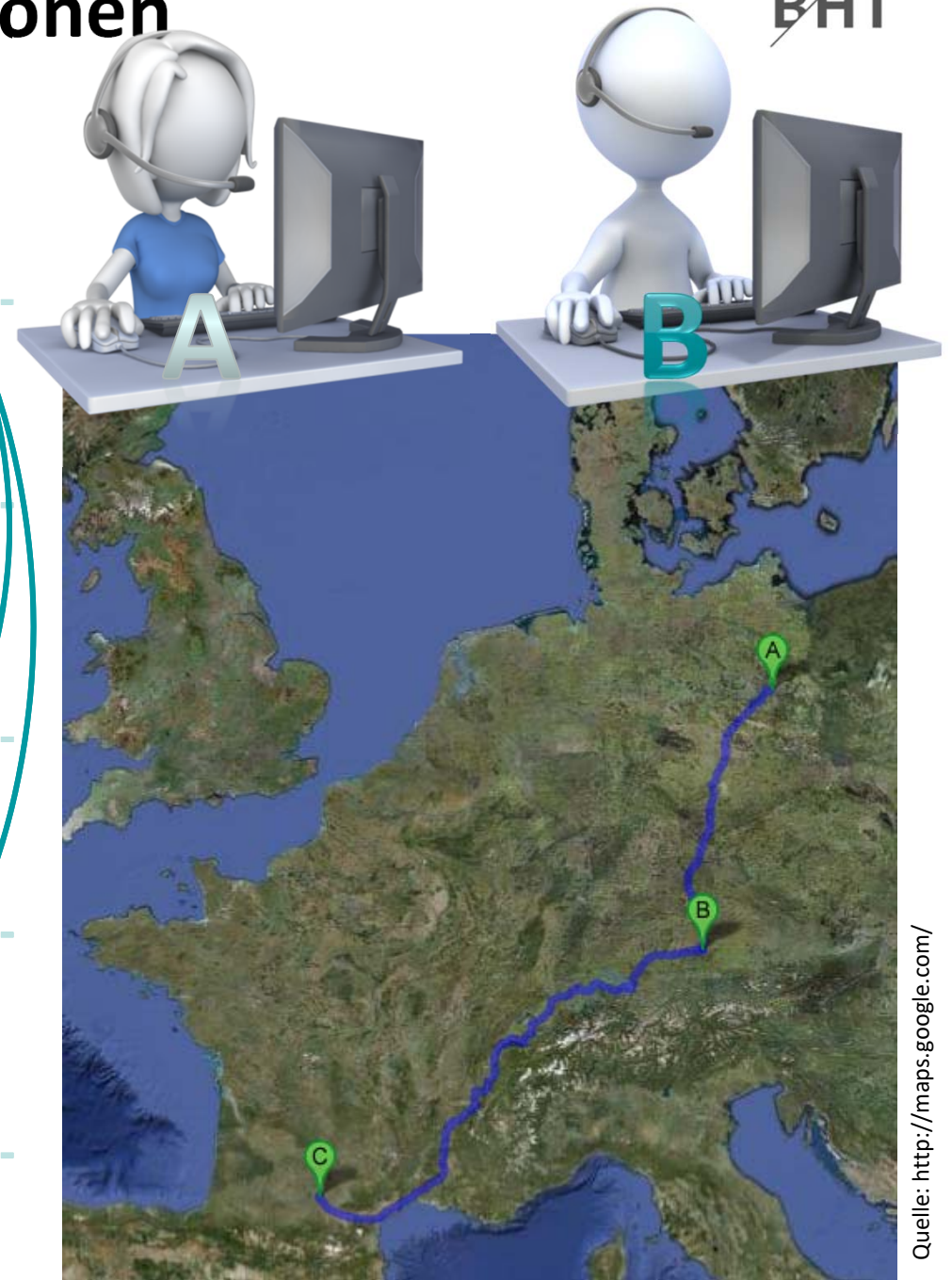
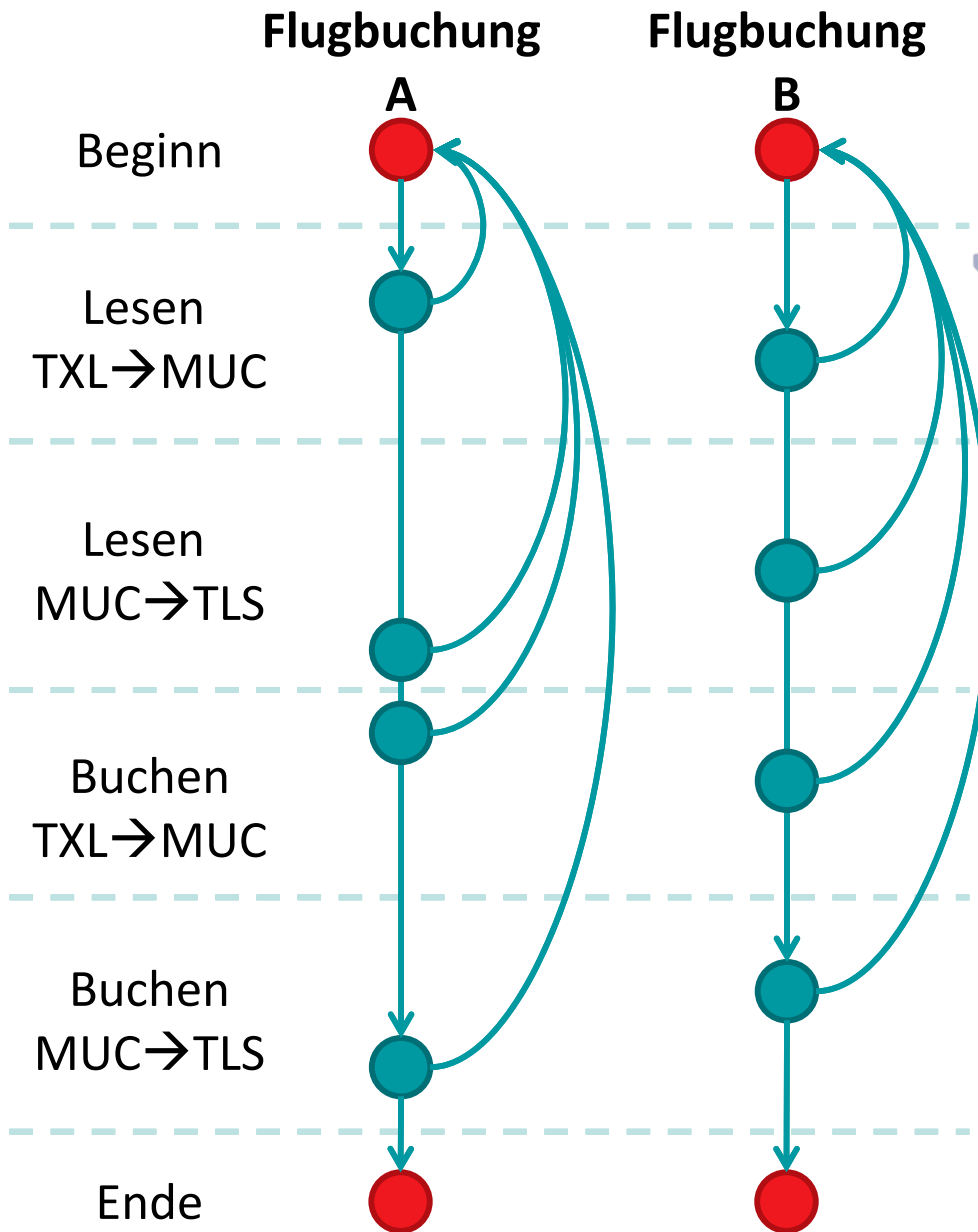
# Anwendung von Transaktionen

BHT



Quelle: <http://maps.google.com/>

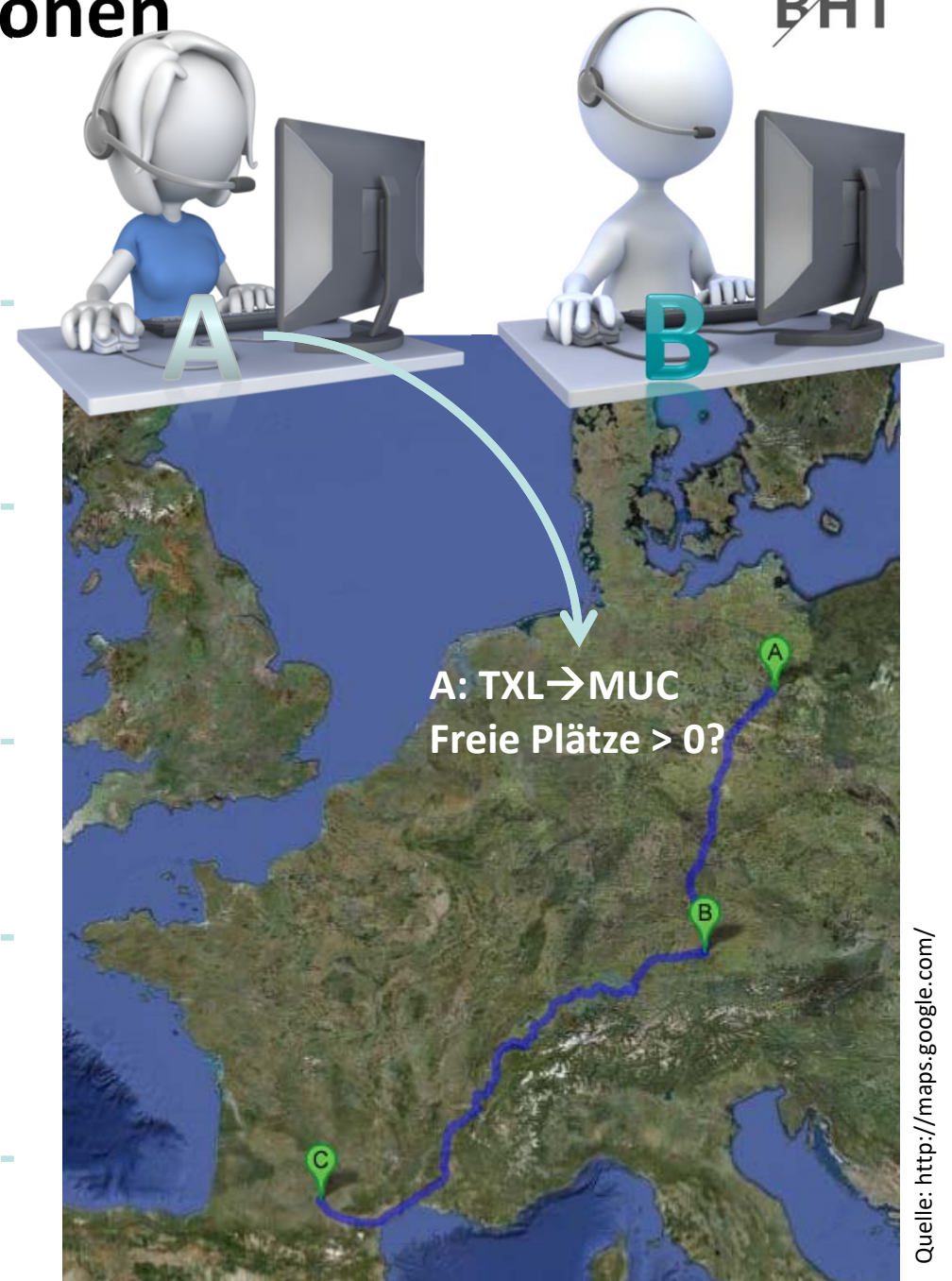
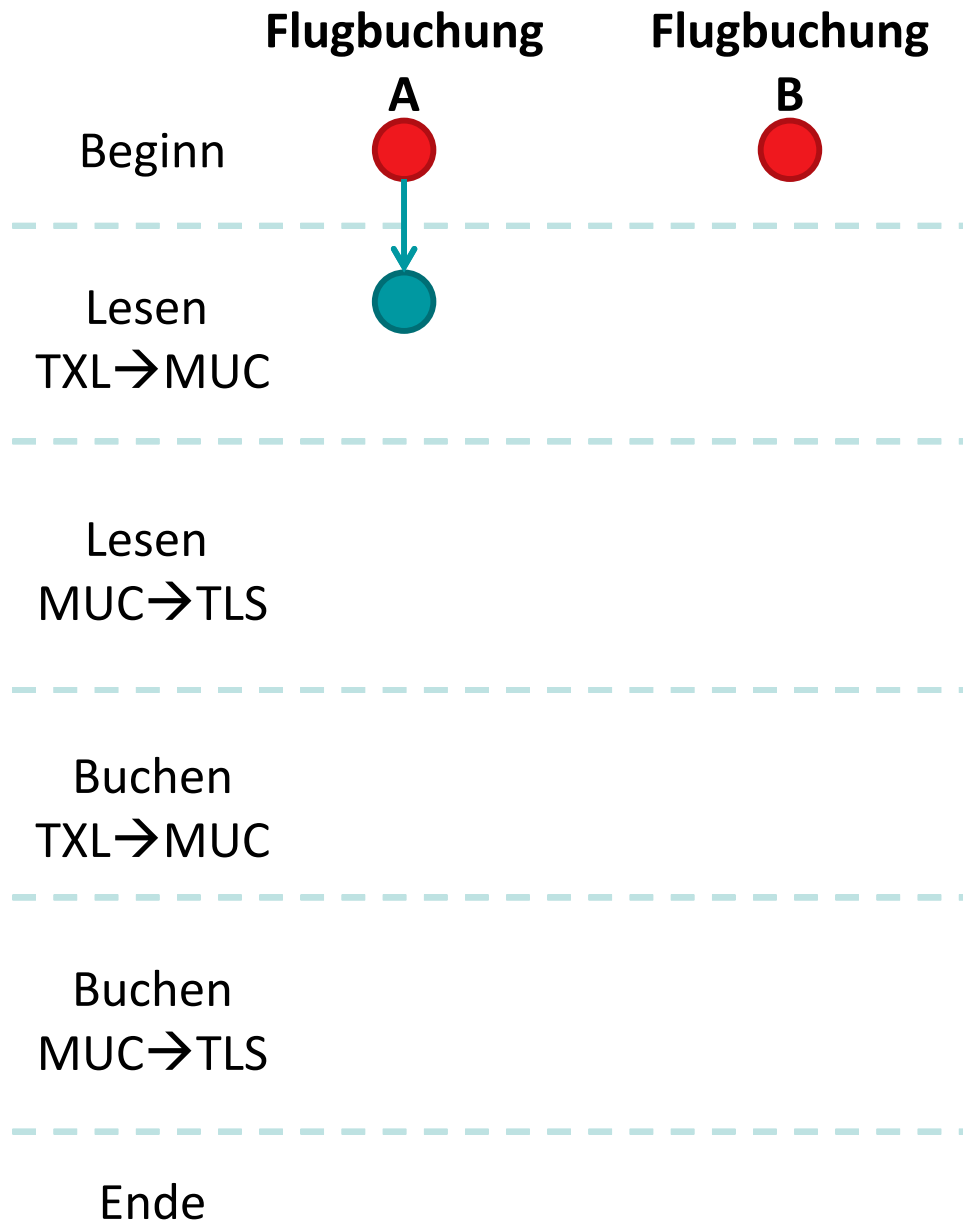
# Anwendung von Transaktionen



Quelle: <http://maps.google.com/>

# Anwendung von Transaktionen

BHT

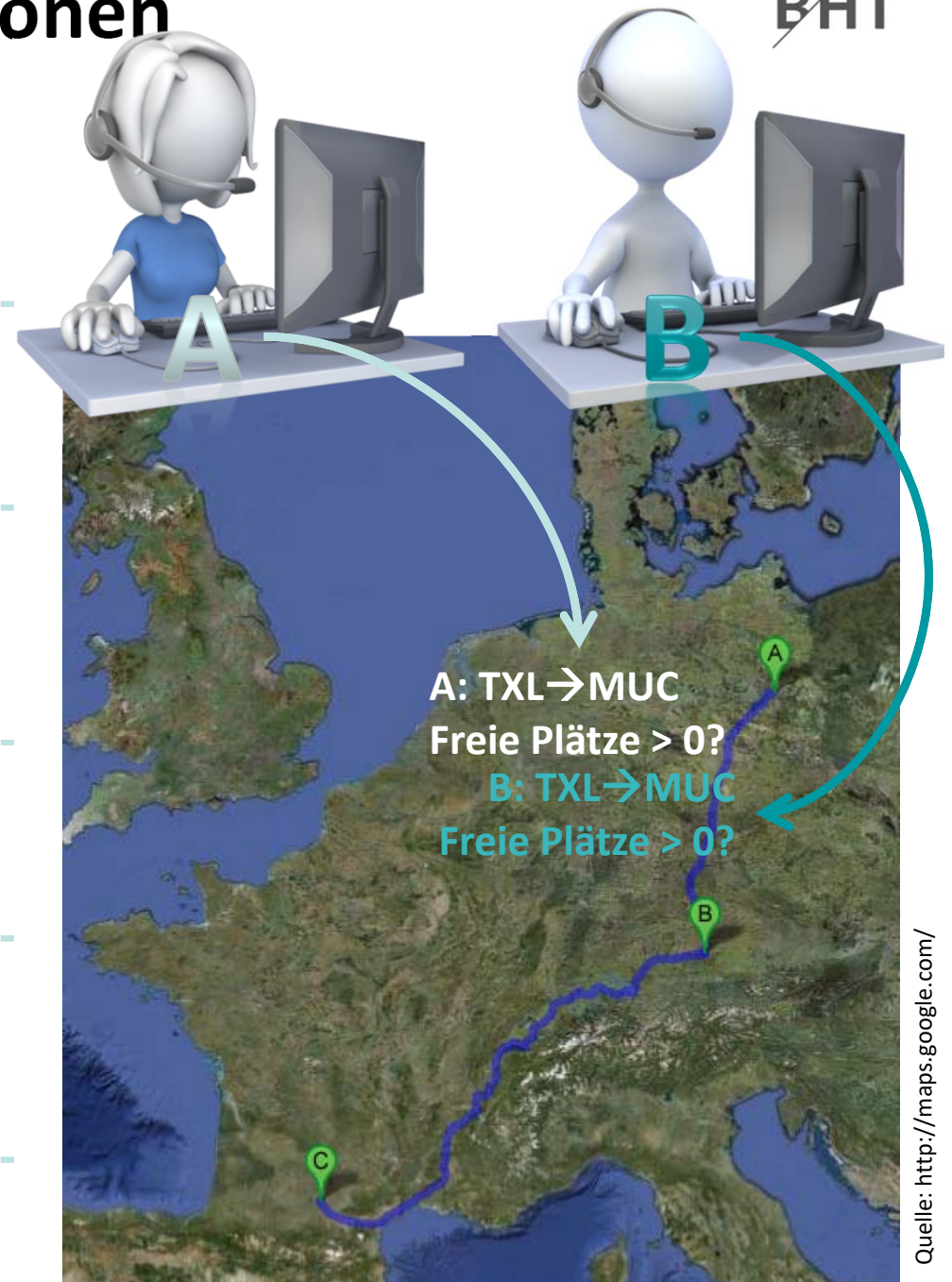
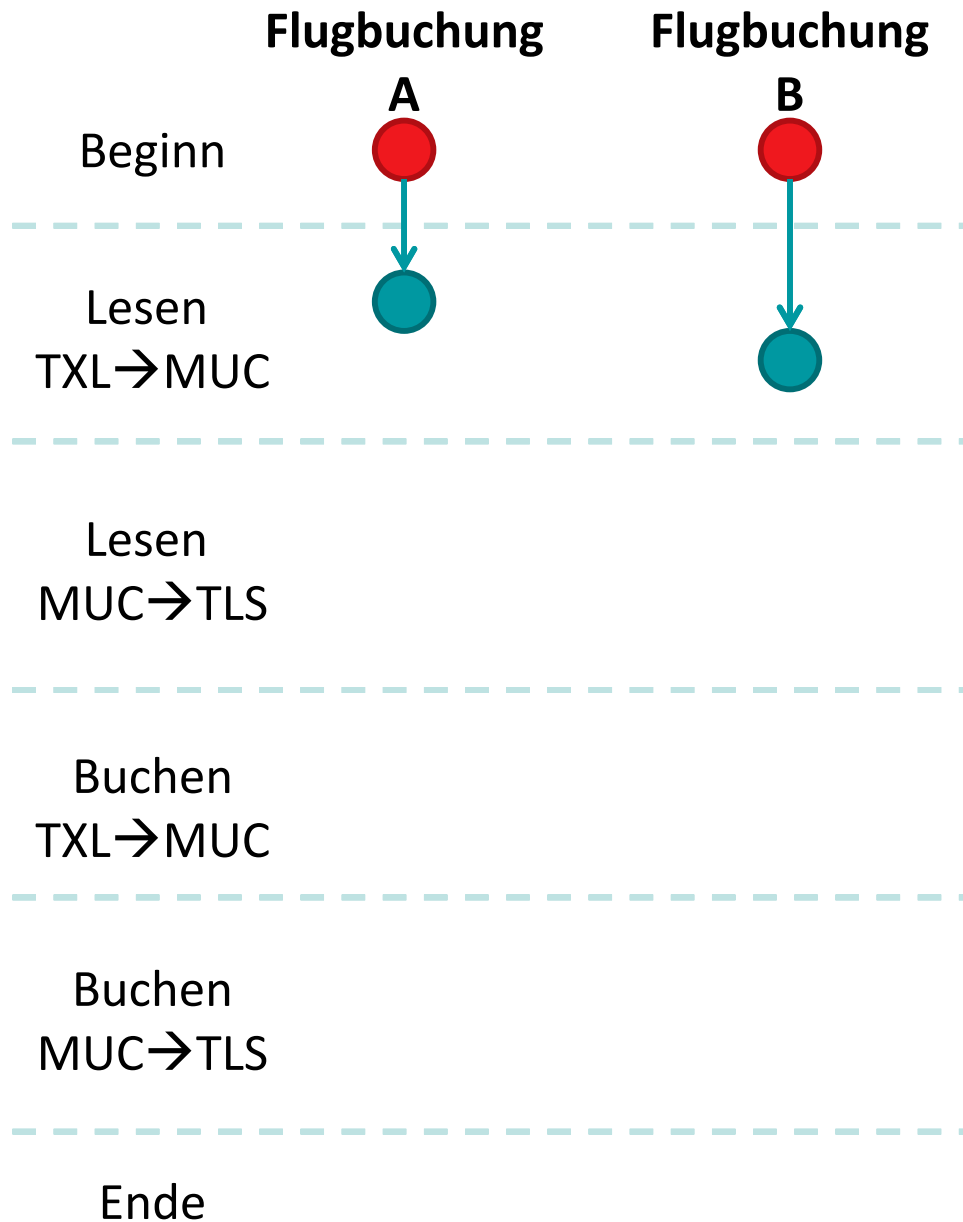


Quelle: <http://maps.google.com/>



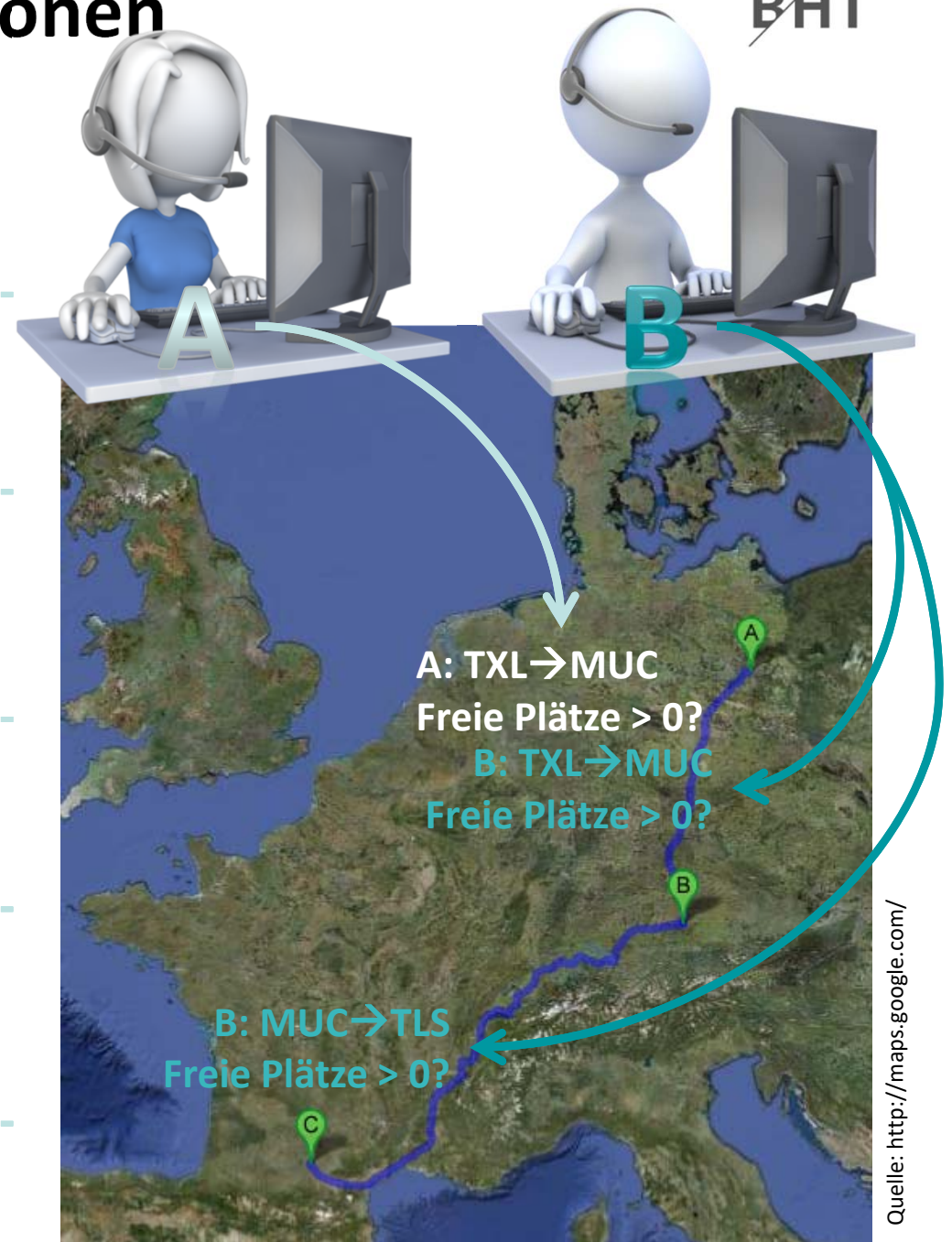
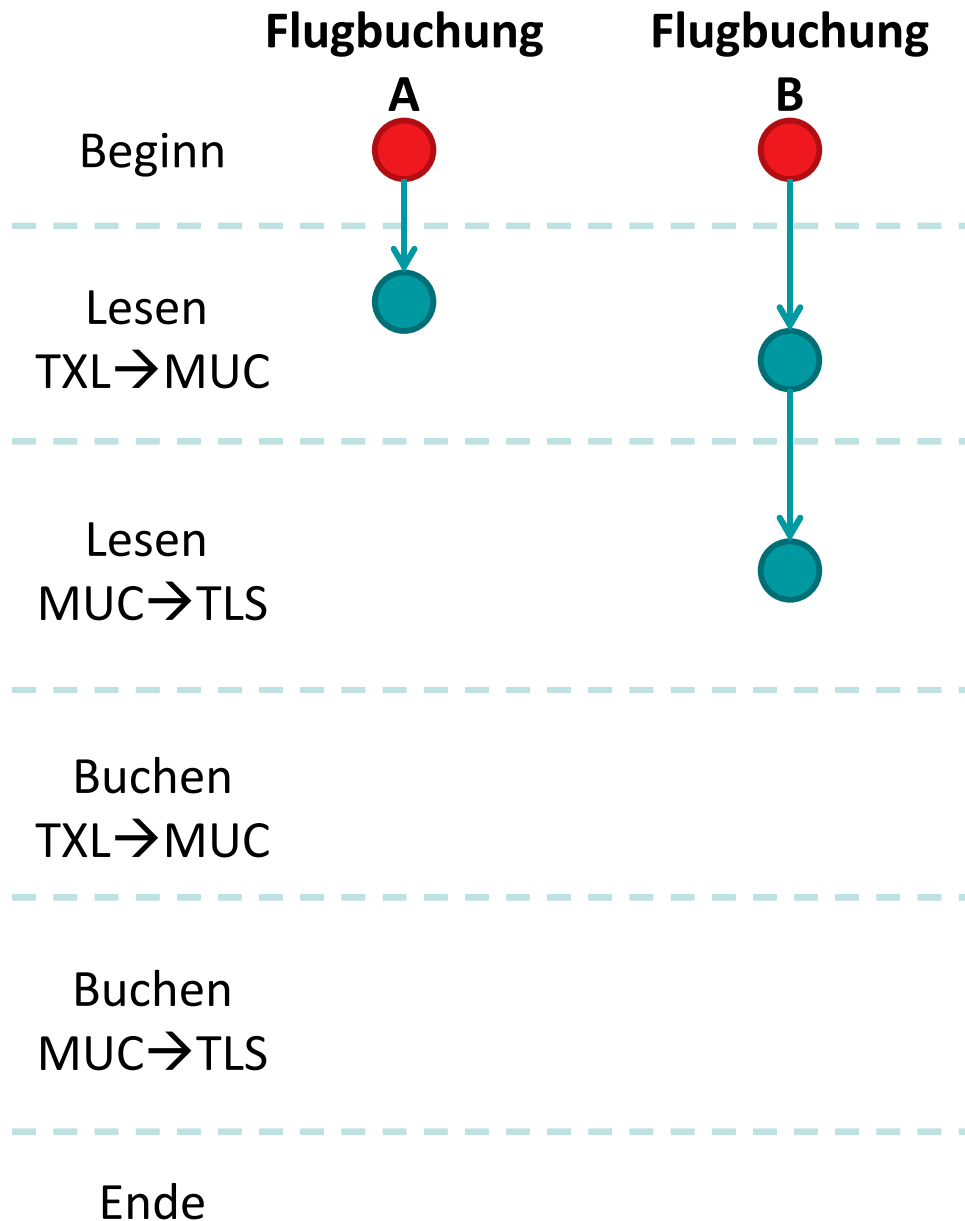
# Anwendung von Transaktionen

~~BHT~~

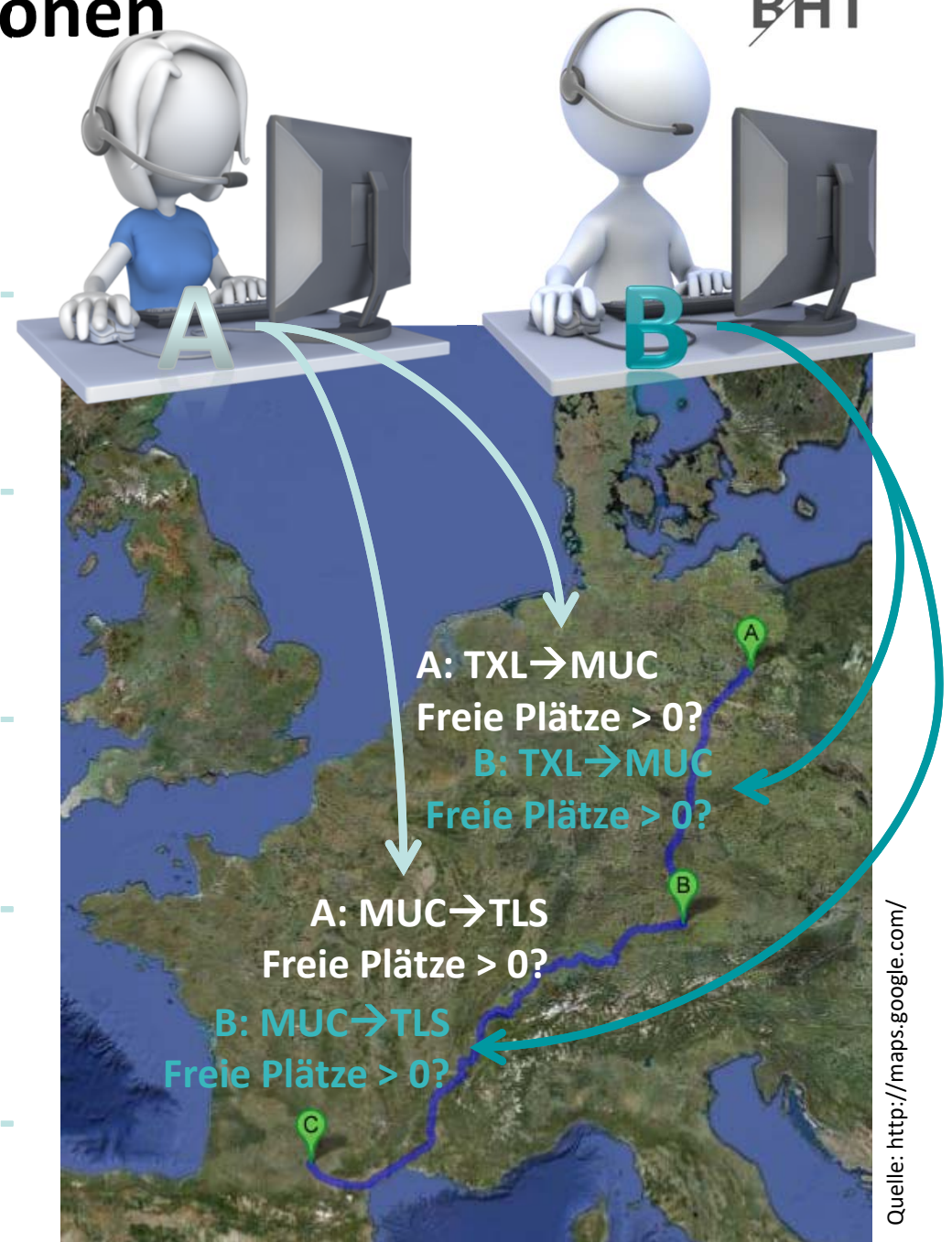
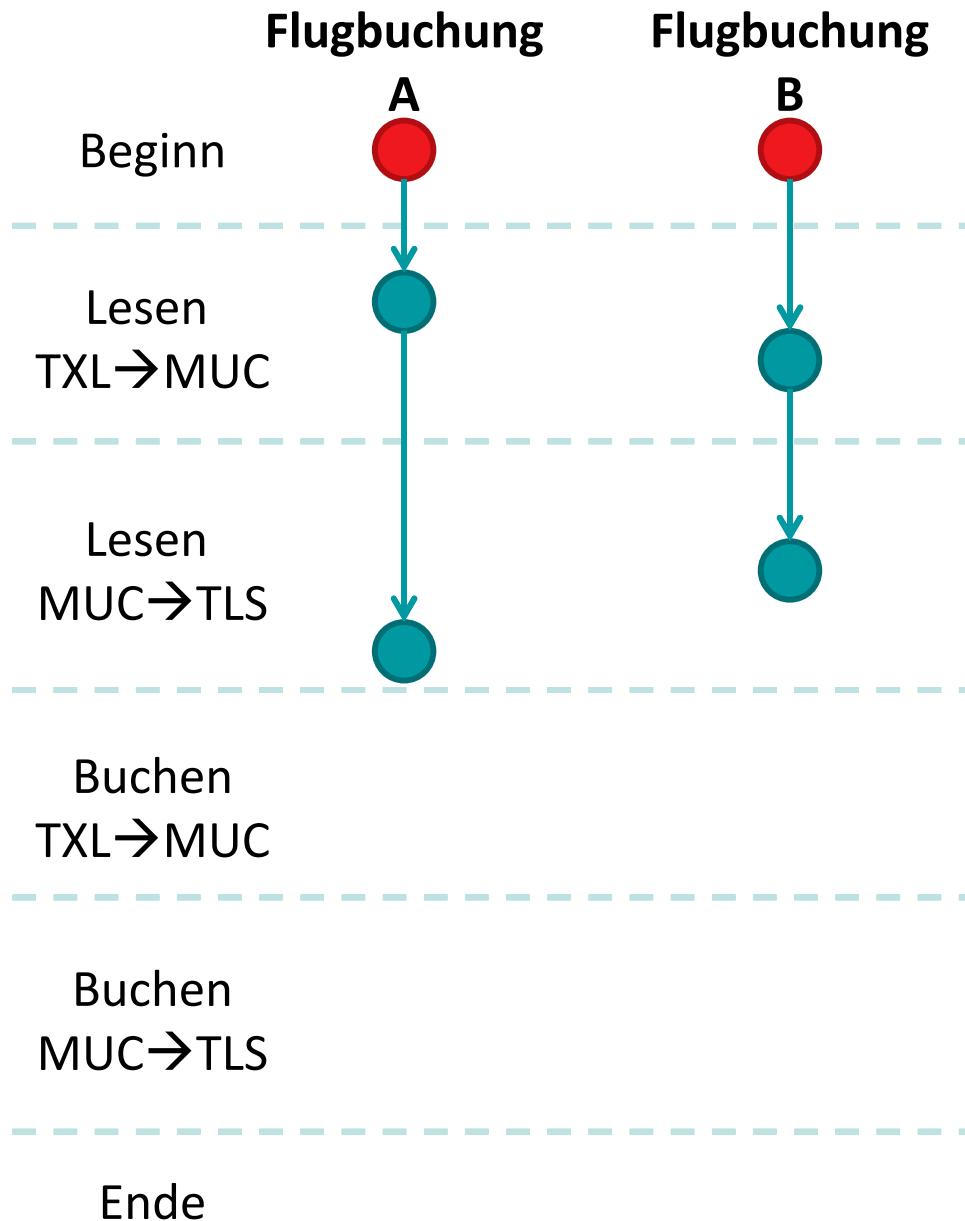


# Anwendung von Transaktionen

~~BHT~~

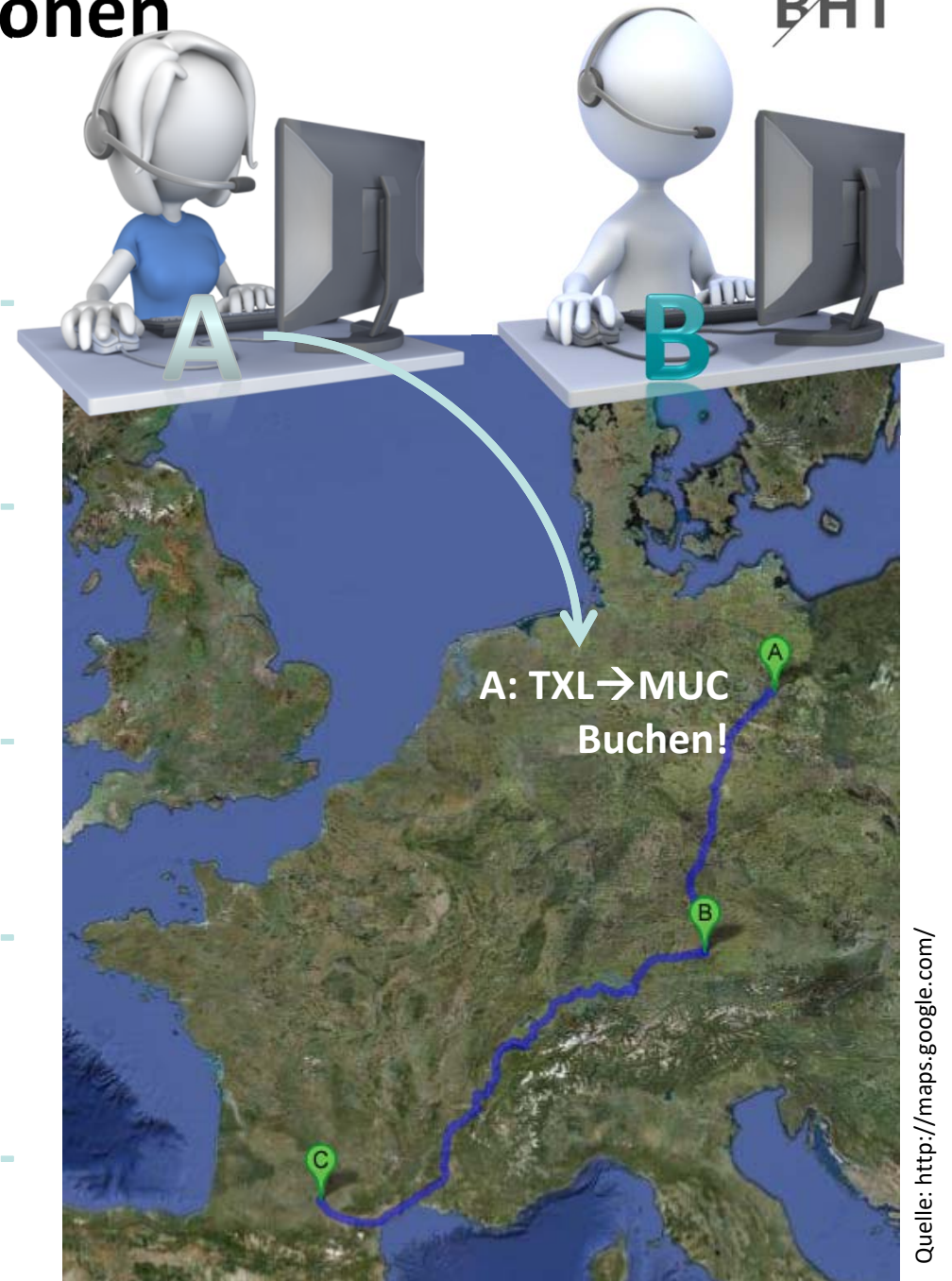
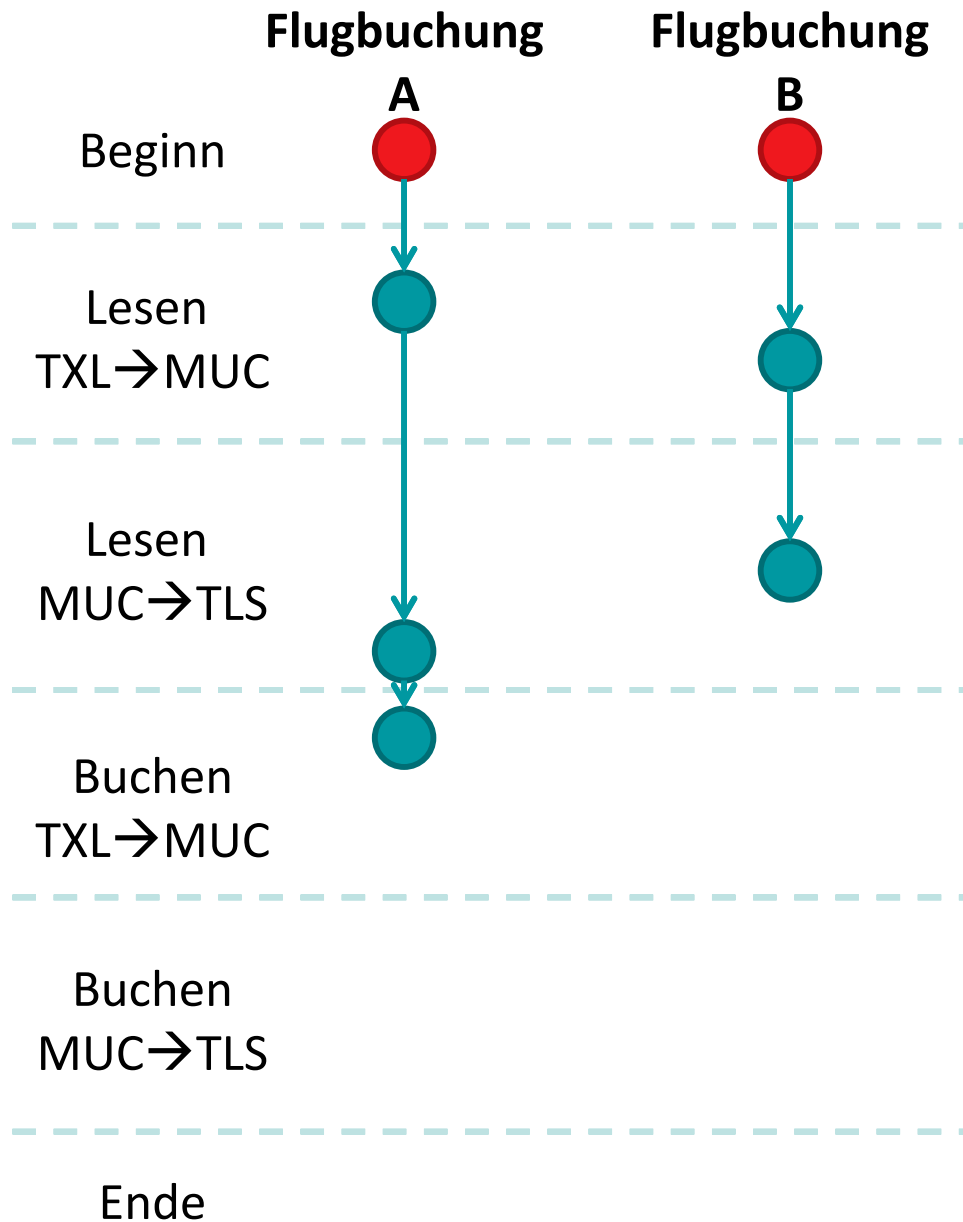


# Anwendung von Transaktionen

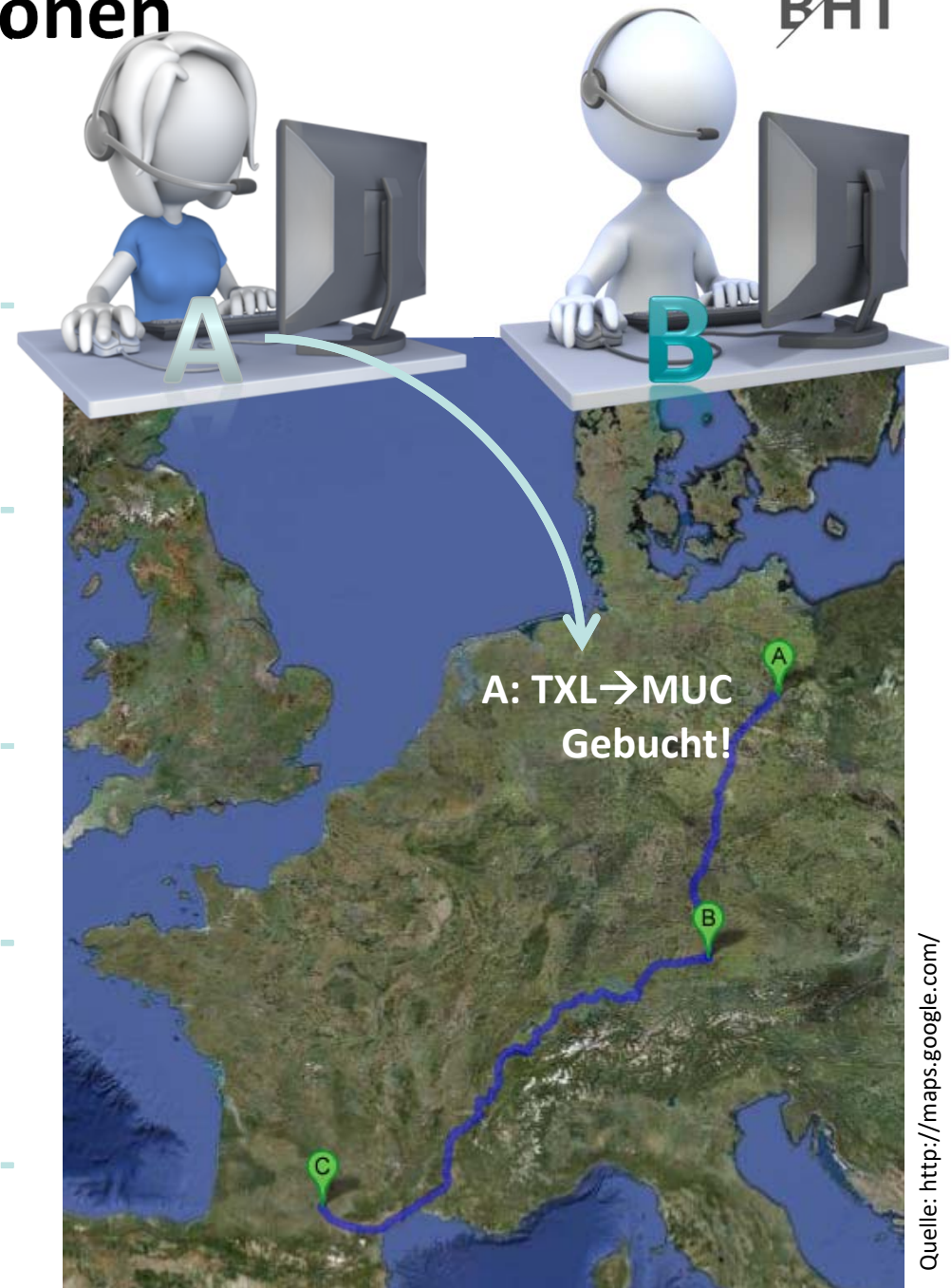
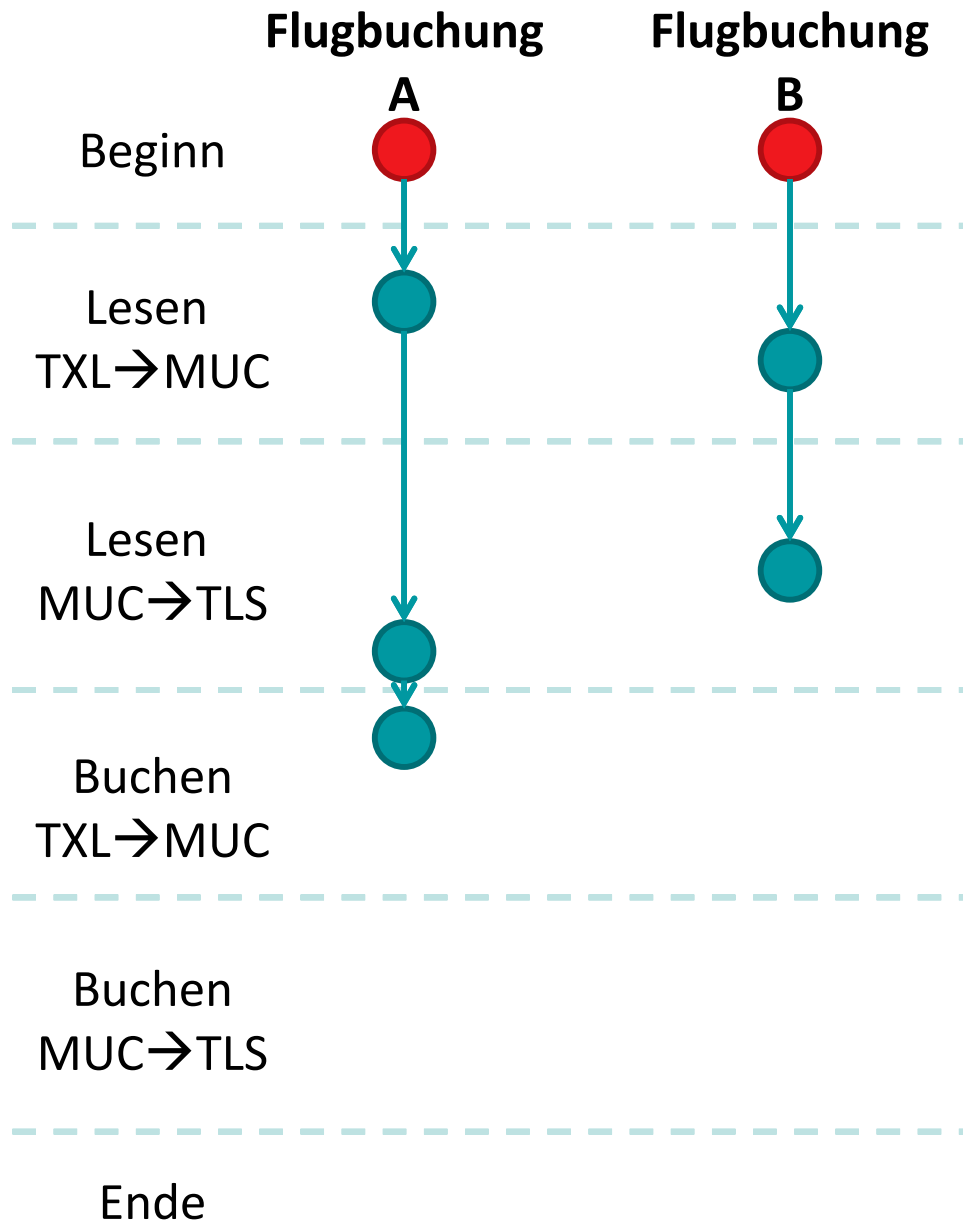


# Anwendung von Transaktionen

BHT



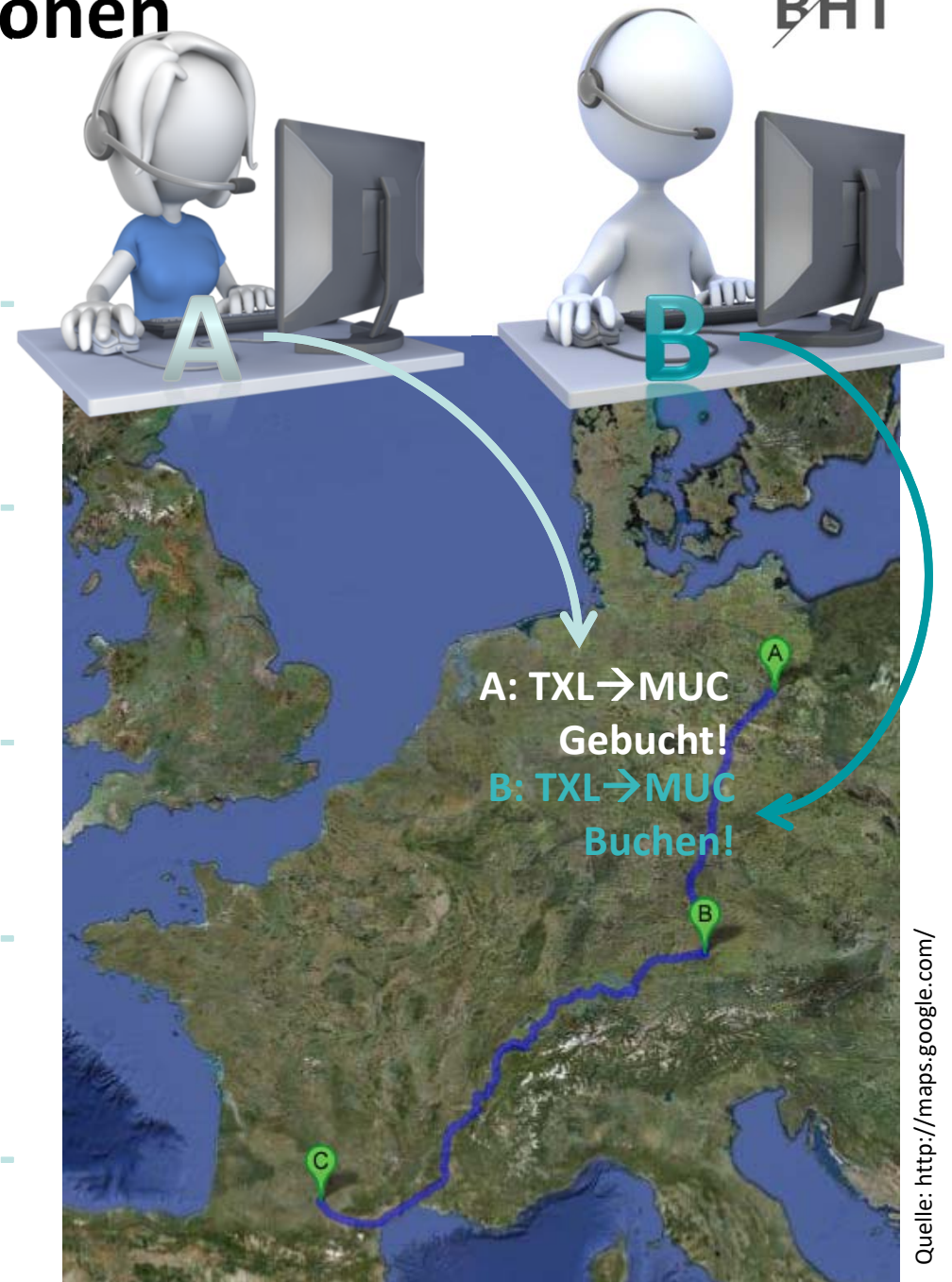
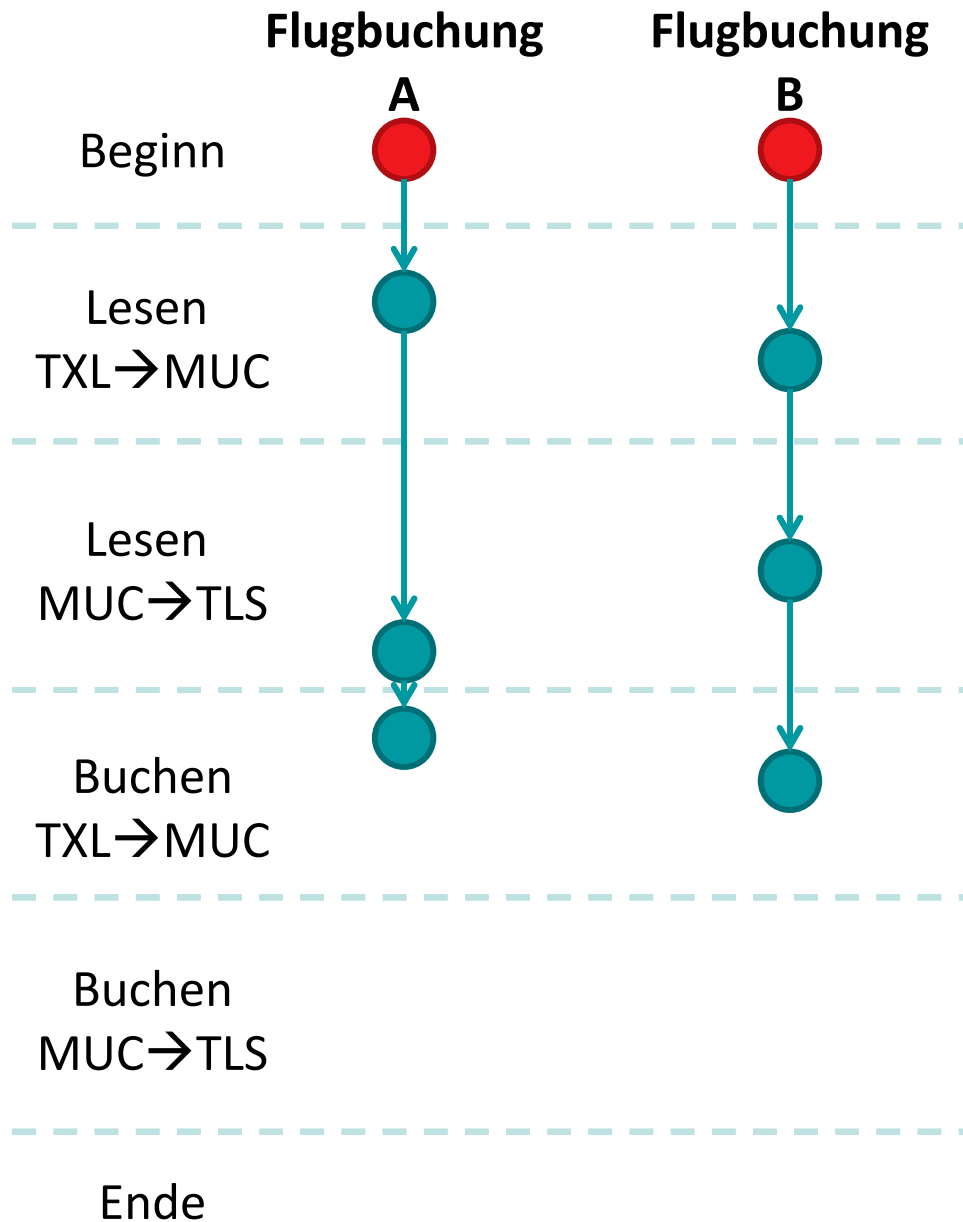
# Anwendung von Transaktionen



Quelle: <http://maps.google.com/>

# Anwendung von Transaktionen

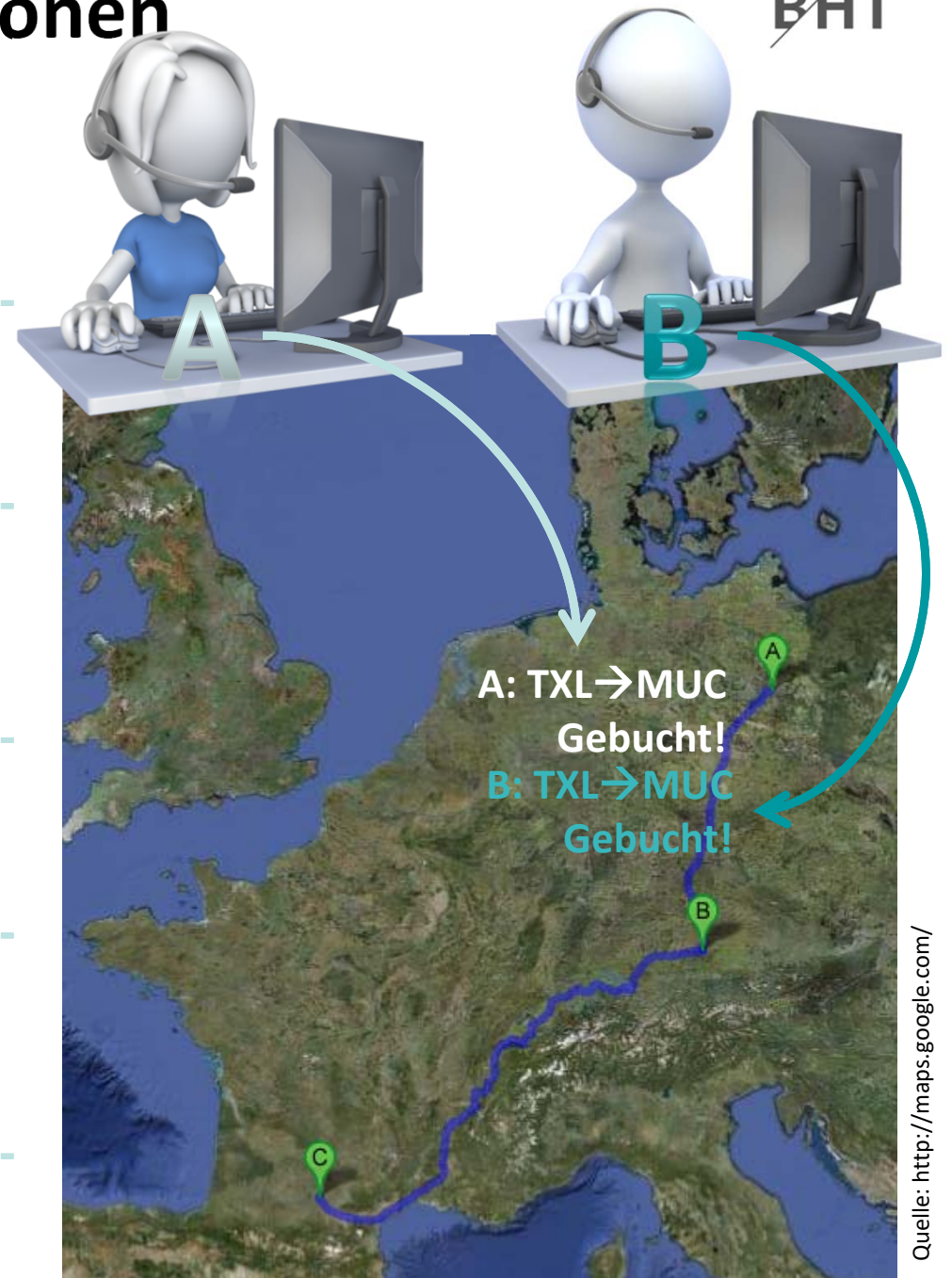
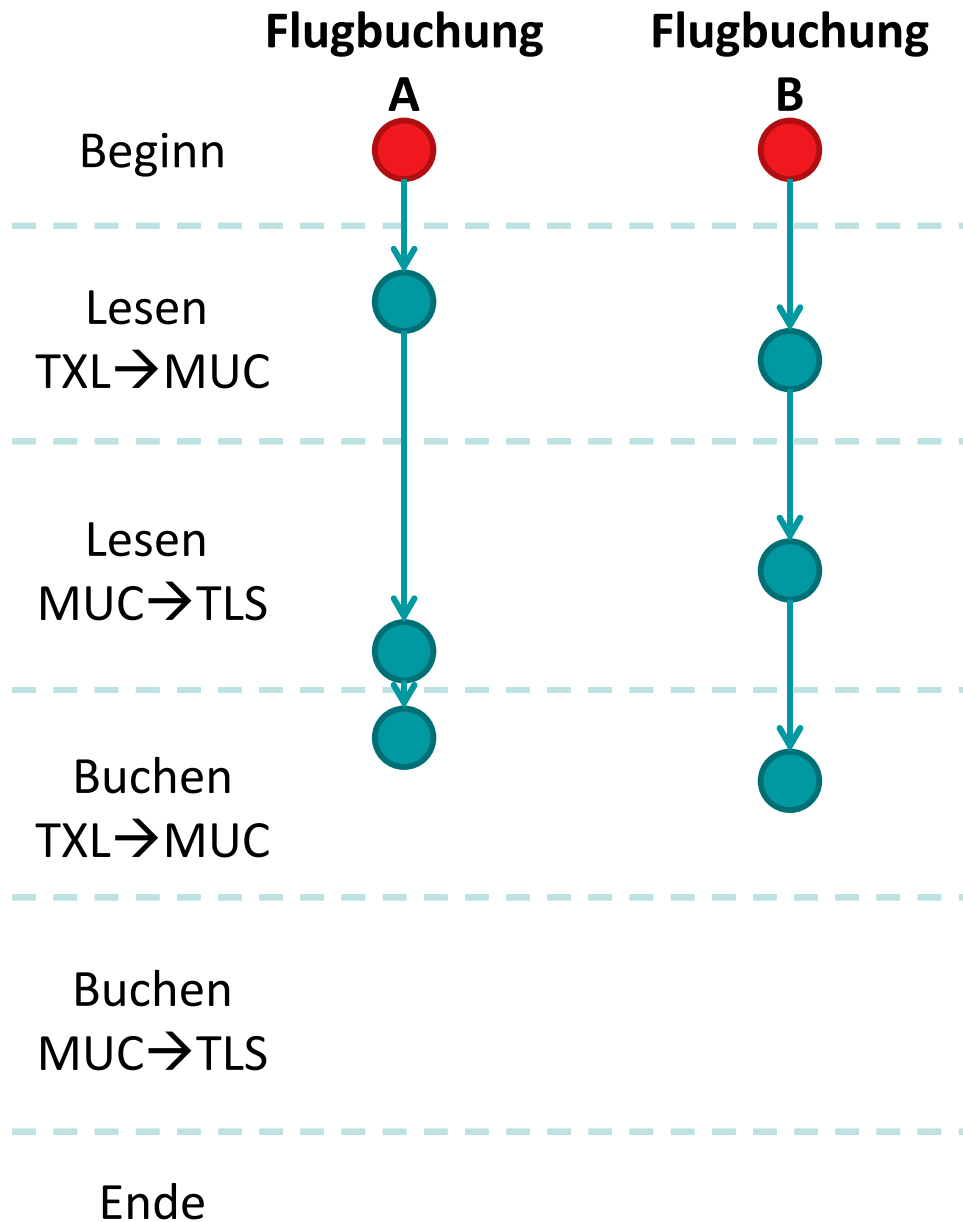
~~BHT~~



Quelle: <http://maps.google.com/>

# Anwendung von Transaktionen

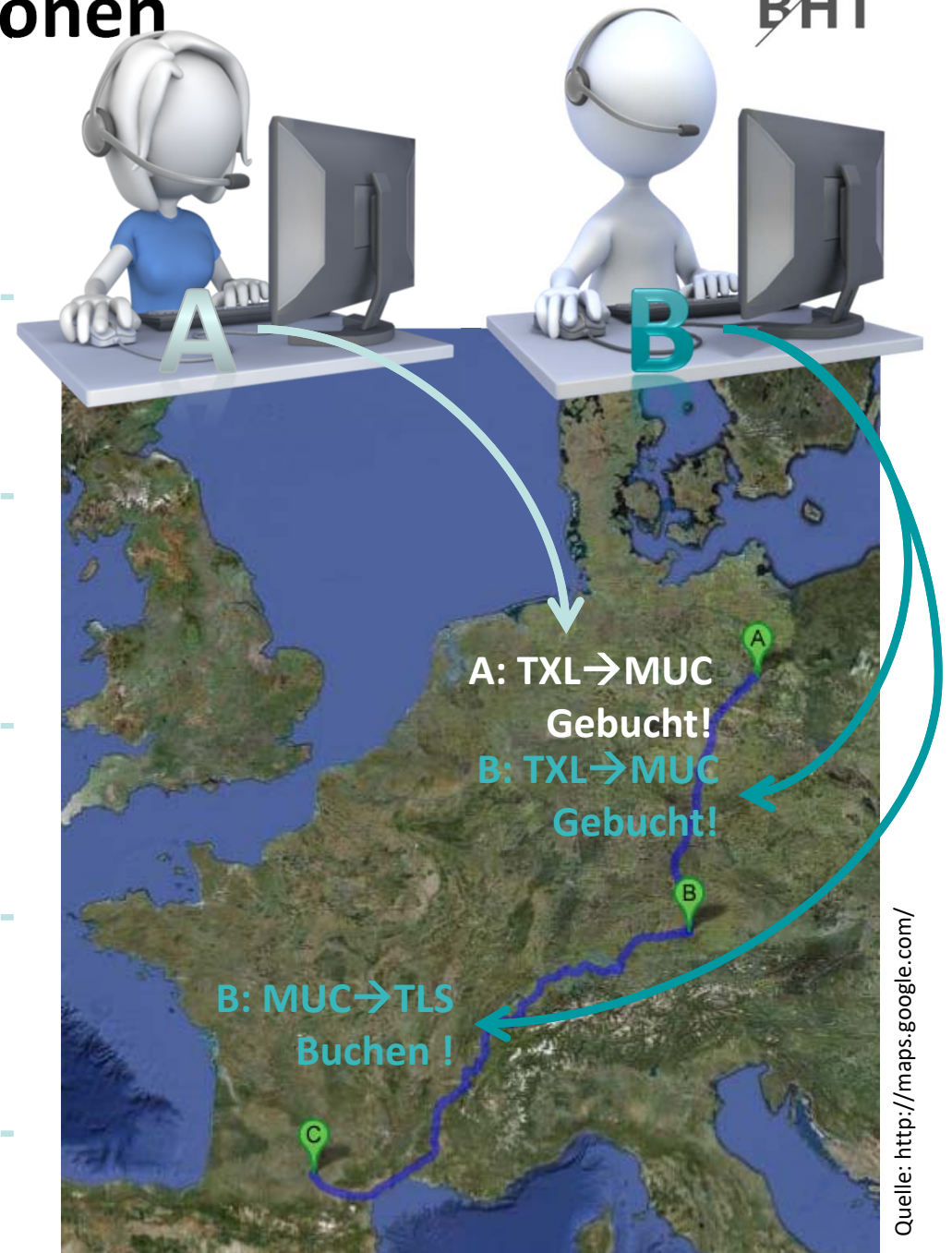
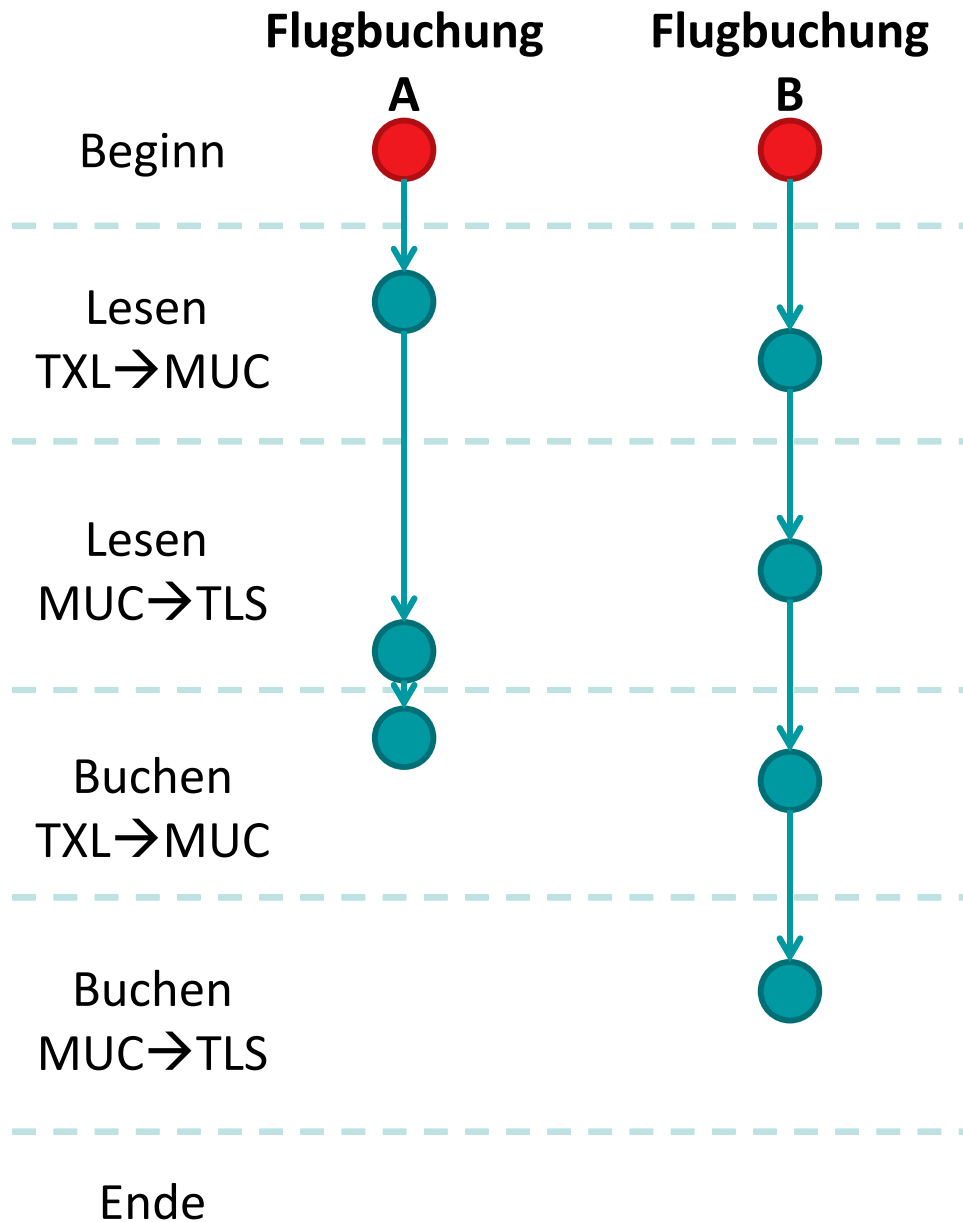
BHT



Quelle: <http://maps.google.com/>

# Anwendung von Transaktionen

BHT

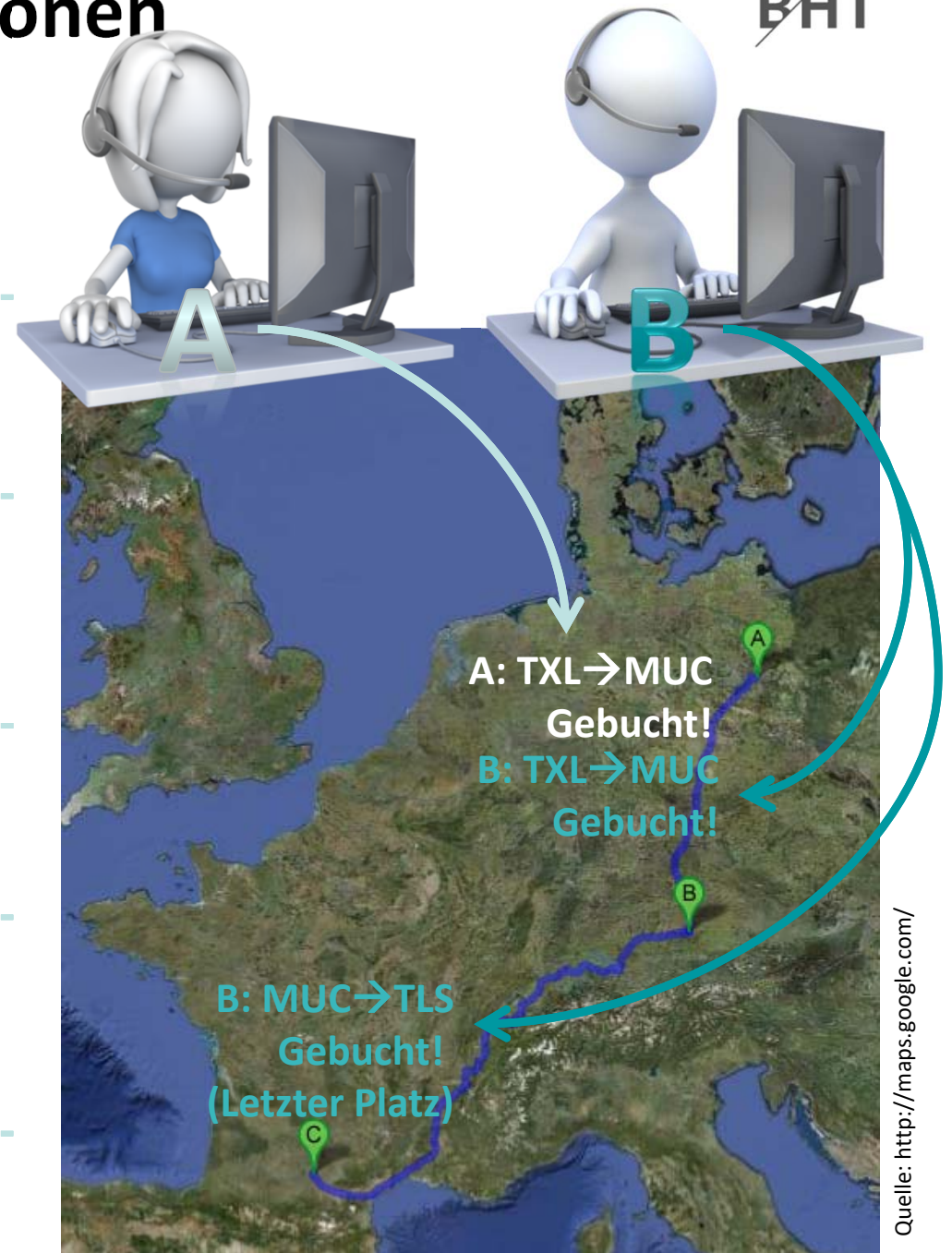
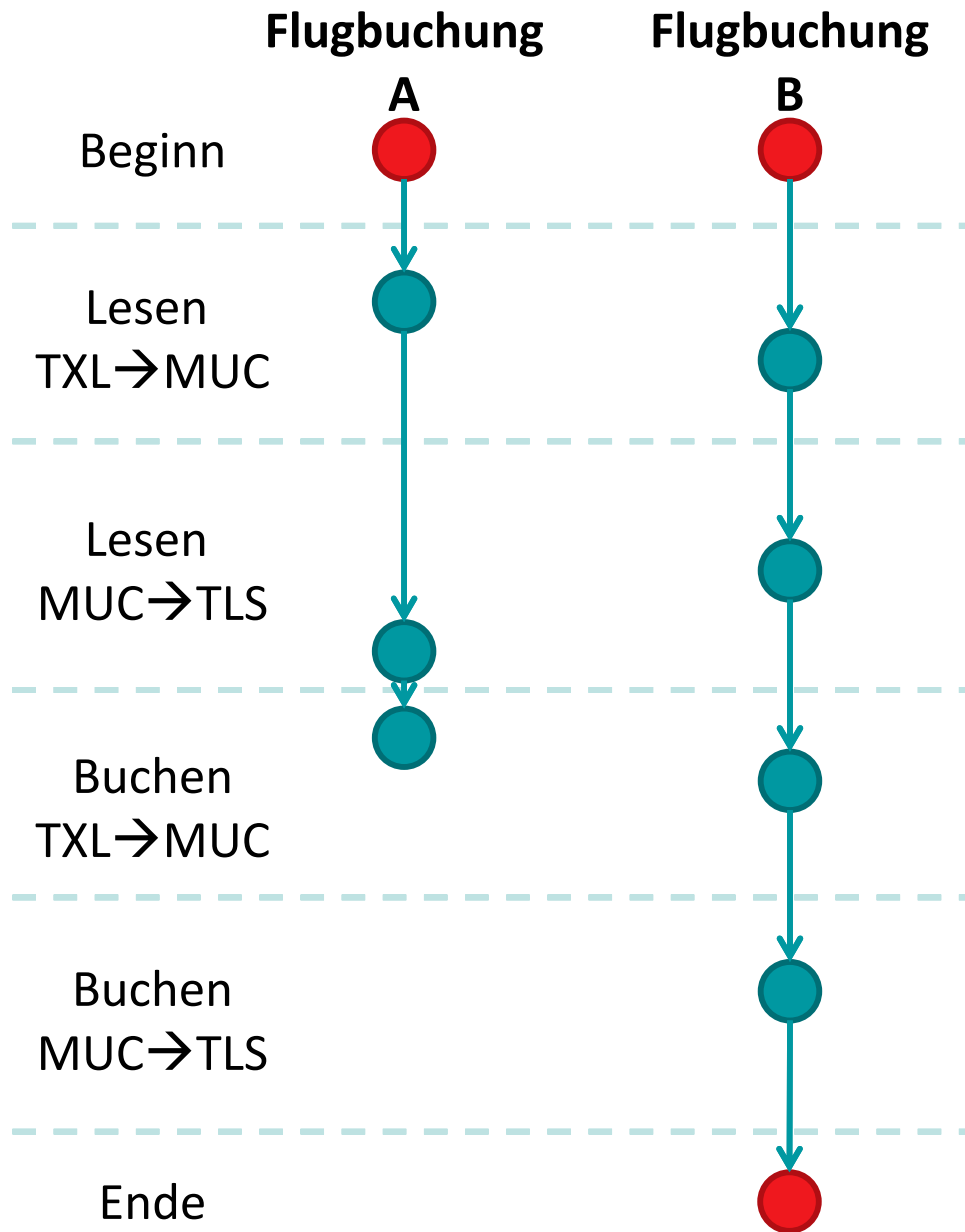


Quelle: <http://maps.google.com/>



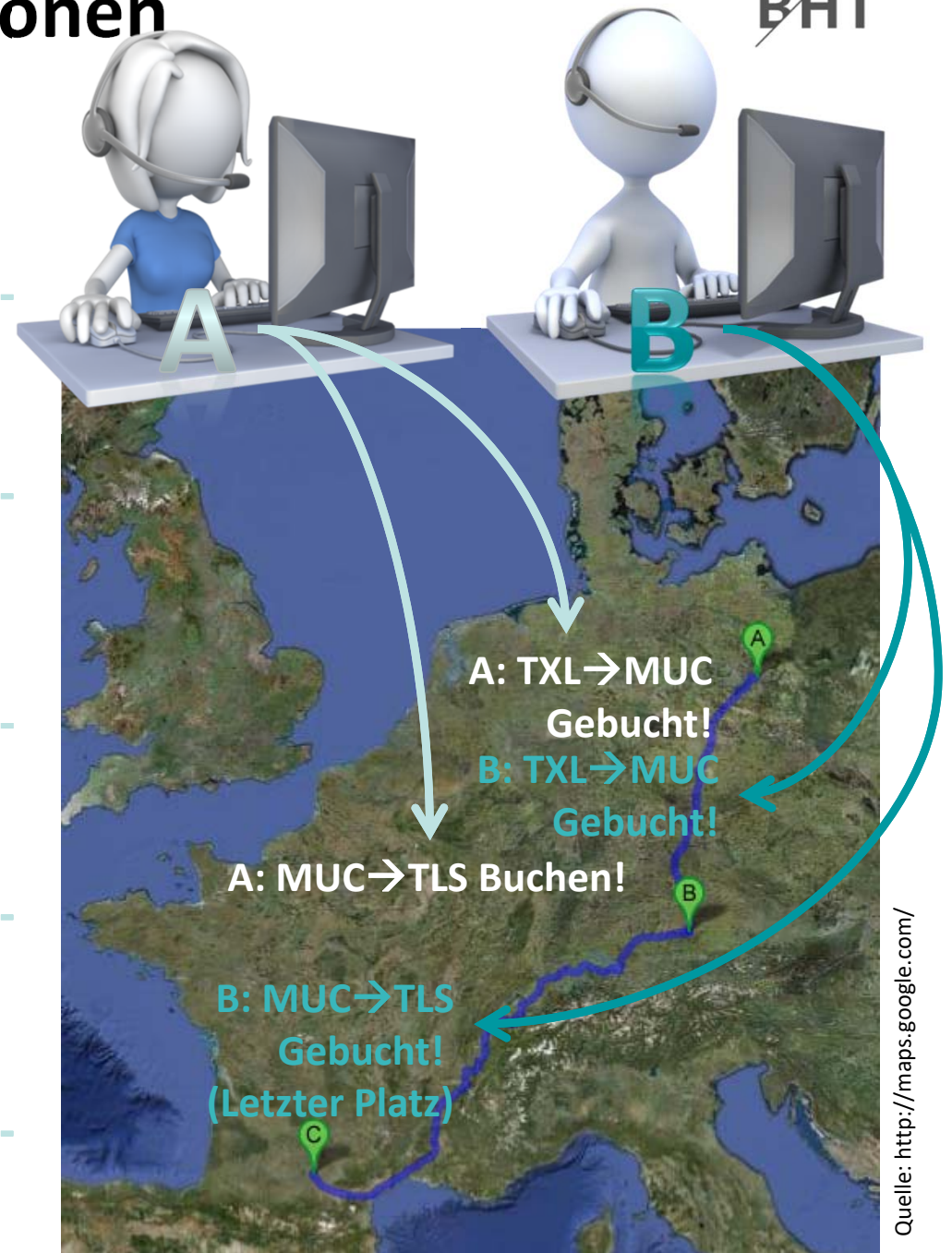
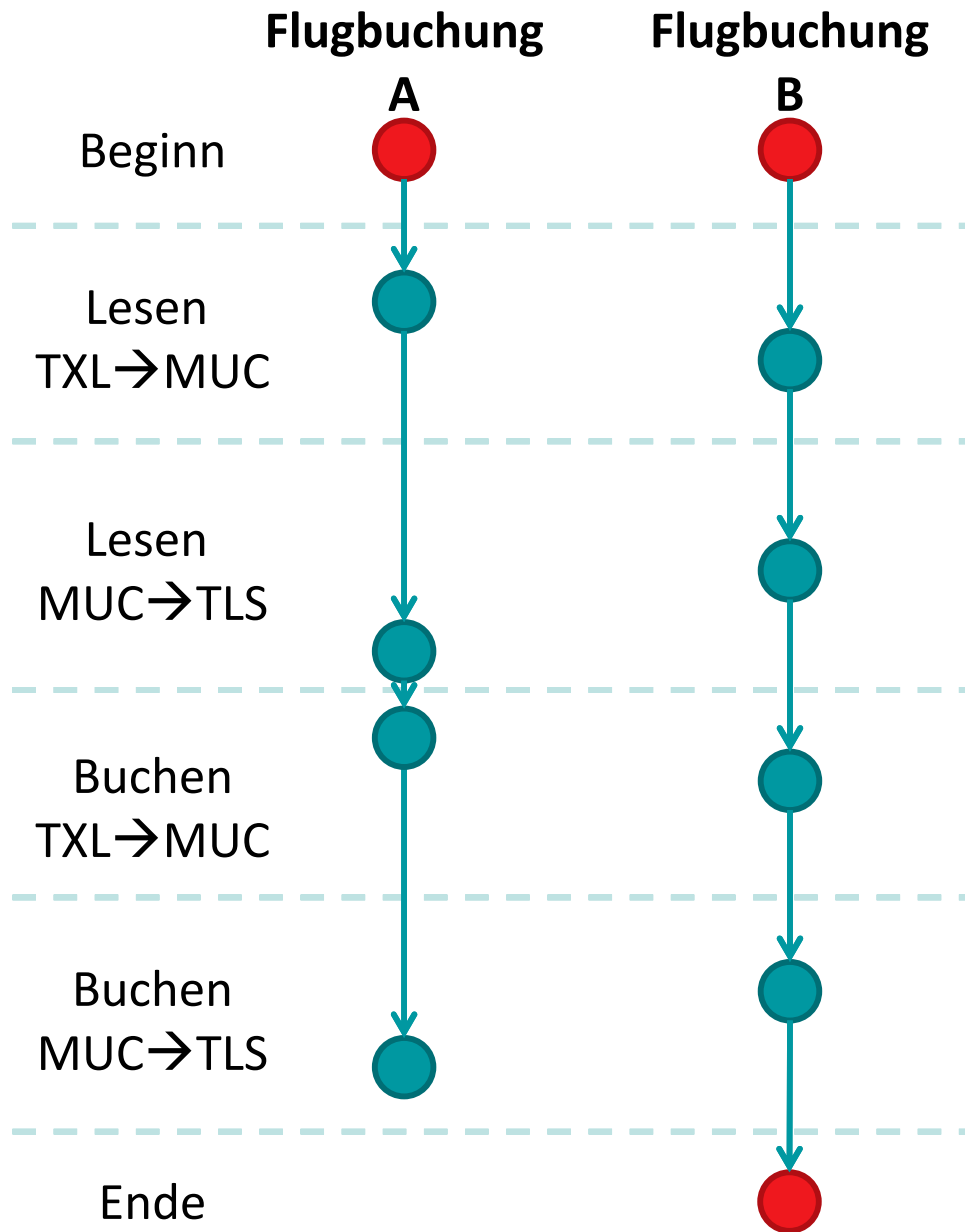
# Anwendung von Transaktionen

BHT

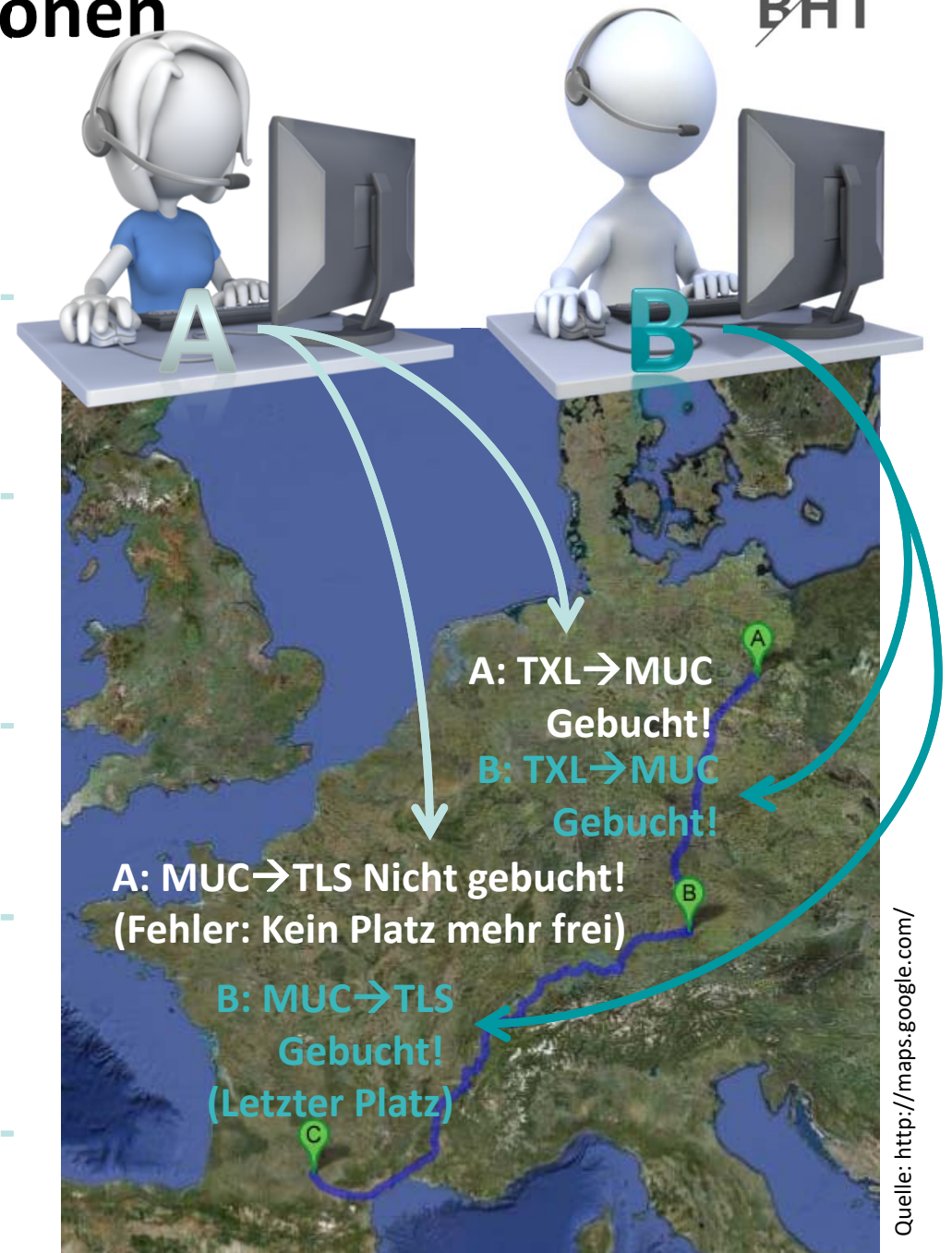
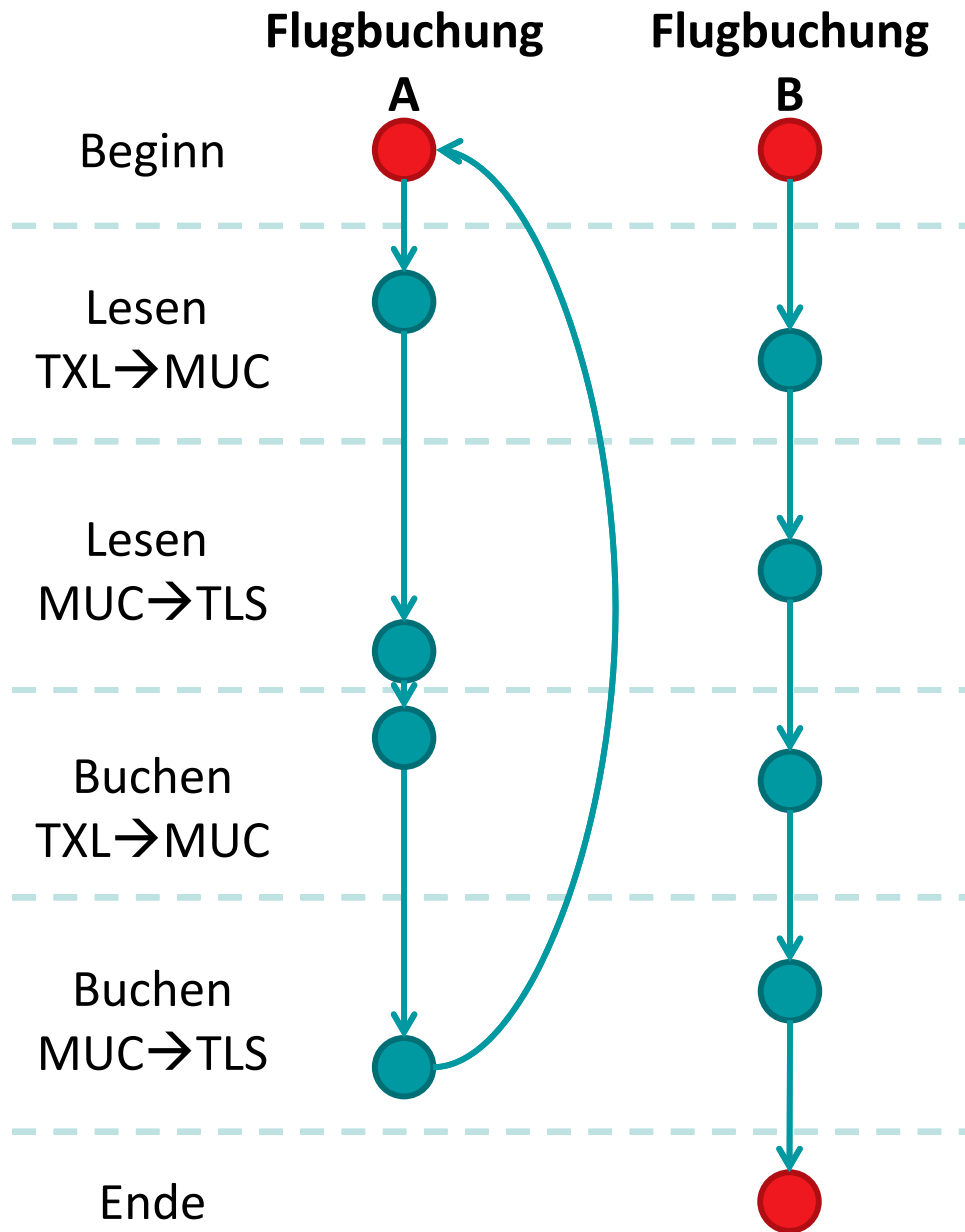


# Anwendung von Transaktionen

~~BHT~~

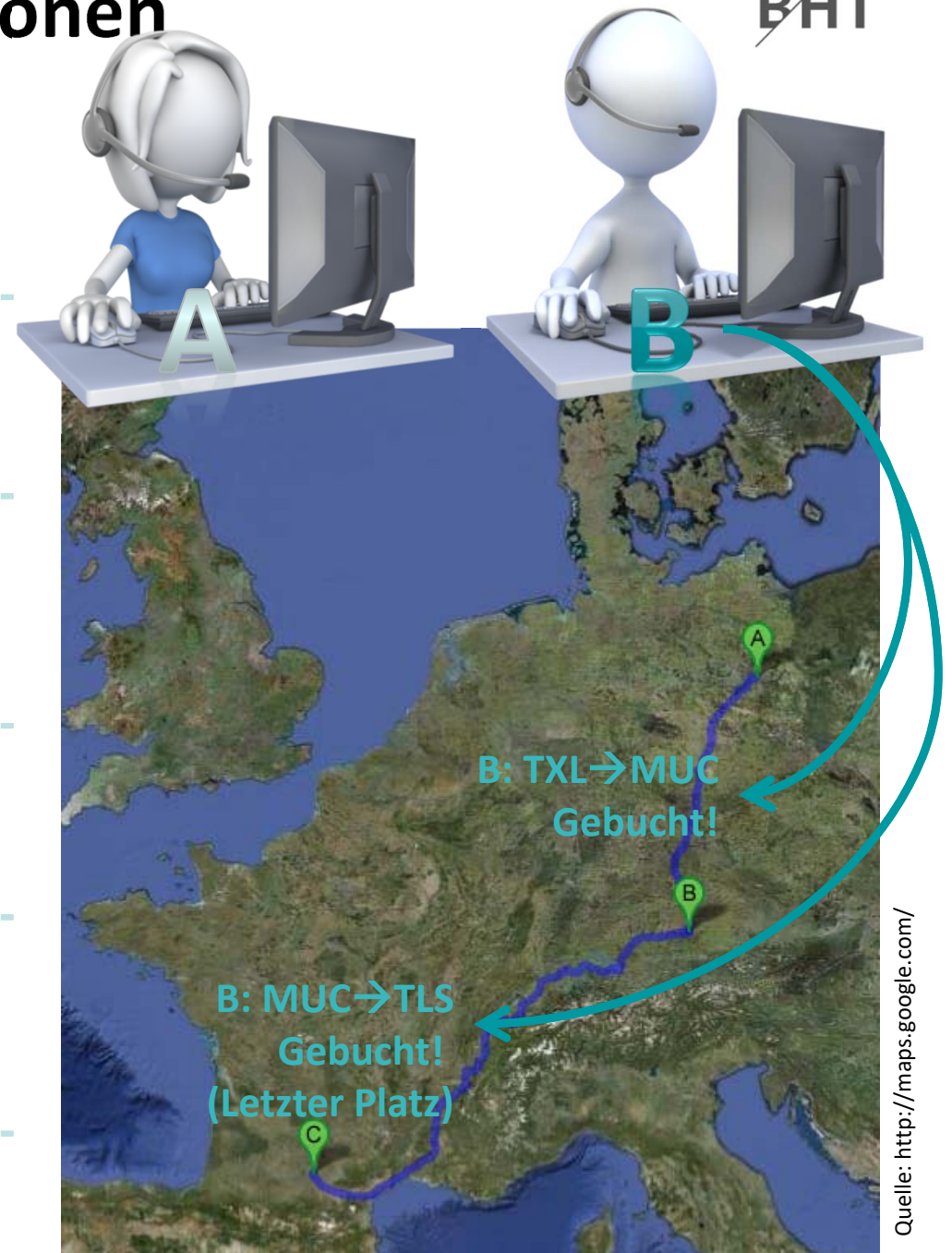
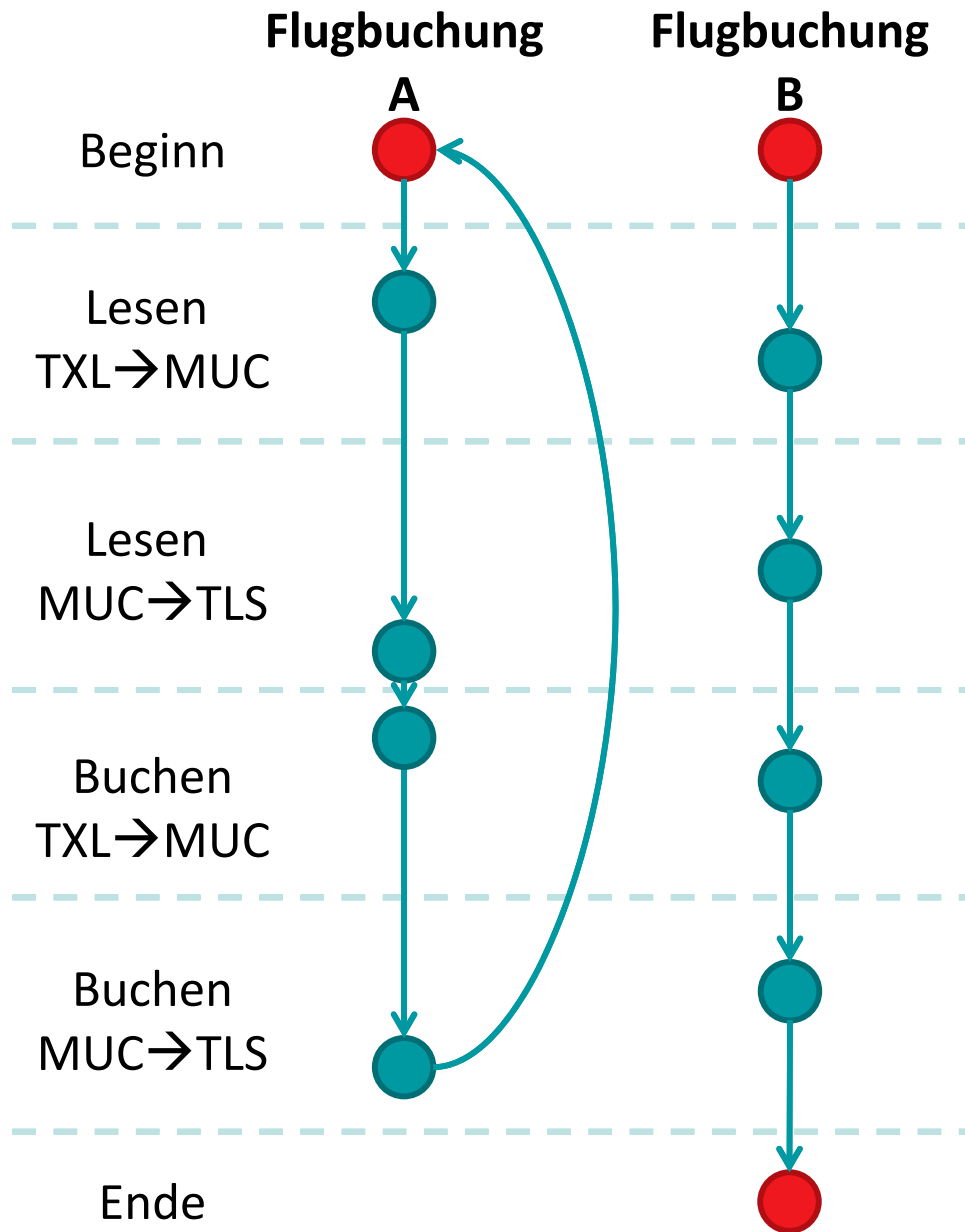


# Anwendung von Transaktionen

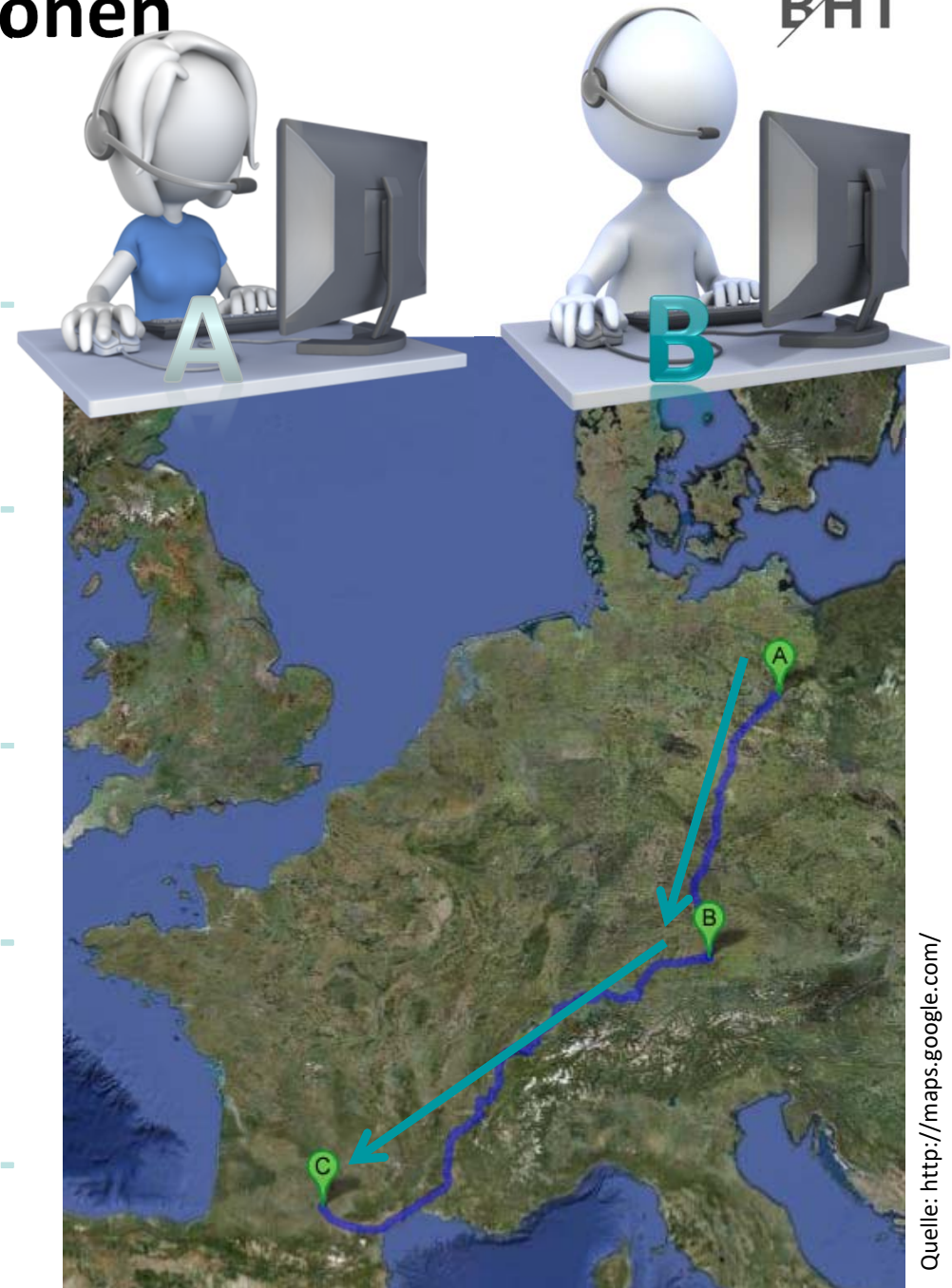
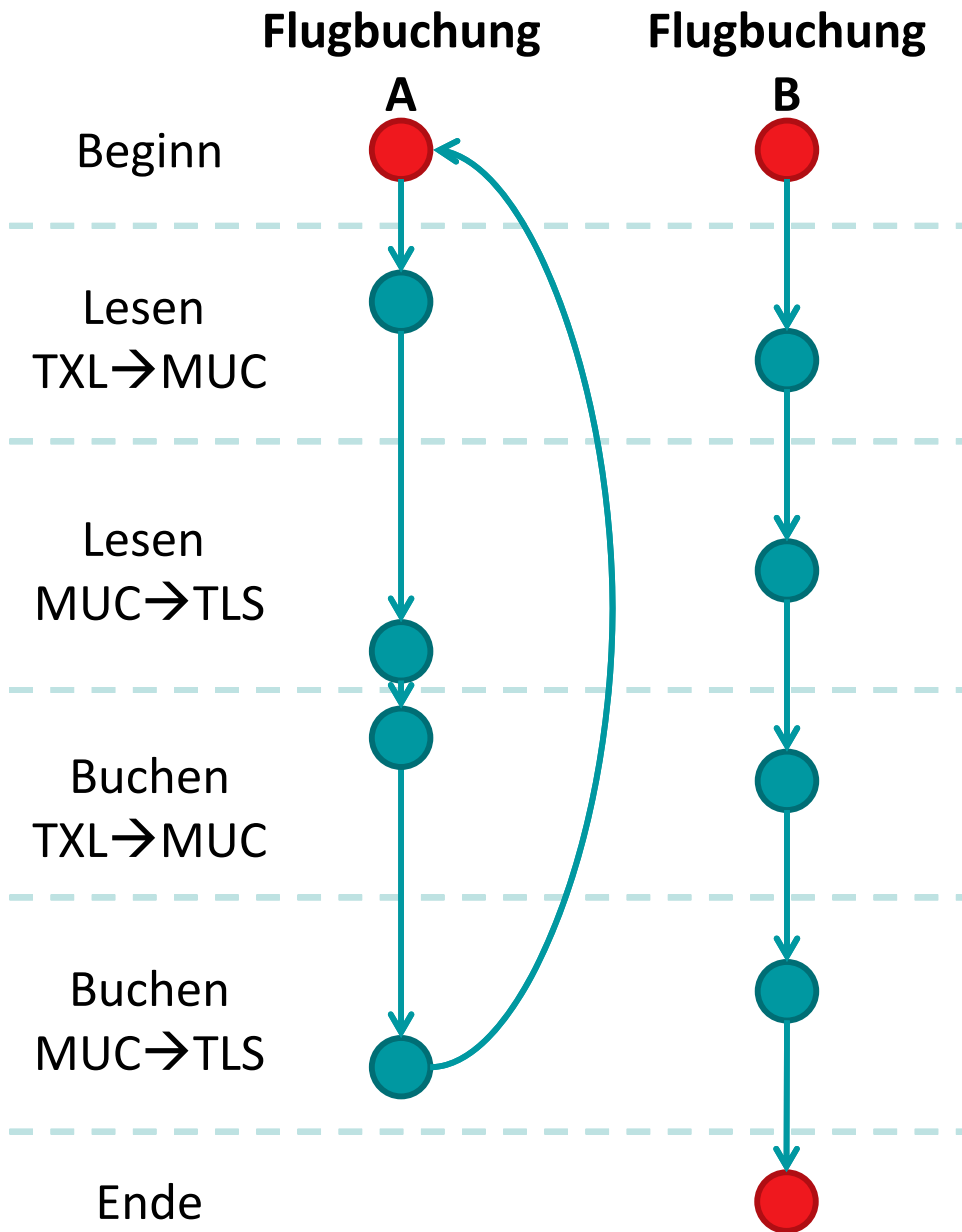


# Anwendung von Transaktionen

BHT



# Anwendung von Transaktionen



Quelle: <http://maps.google.com/>

# Transaktionen in SQL

## SQL kennt besondere Befehle, mit denen Transaktionen

- begonnen
  - **BEGIN TRANSACTION** bzw. **START TRANSACTION**
  - Führt alle nachfolgenden SQL-Befehle innerhalb einer Transaktion aus
- erfolgreich abgeschlossen
  - **COMMIT** bzw. **COMMIT TRANSACTION**
  - schließt eine Transaktion ab, alle durchgeführten Operationen werden dauerhaft in der Datenbank wirksam
- nicht erfolgreich beendet
  - **ROLLBACK** bzw. **ROLLBACK TRANSACTION**
  - schließt die Transaktion ab, verwirft alle durchgeführten Operationen

**werden können.**

# Transaktionen in SQL

## Macht es aus Sicht des Anwenders Sinn ein einzelnes SELECT-, INSERT-, UPDATE- oder DELETE-Kommando in einer Transaktion auszuführen?

- einzelne Kommandos sind per Definition für sich genommen bereits atomar, konsistent, isoliert und dauerhaft
- sie müssen deshalb vom Programmierer nicht in einer Transaktion ausgeführt werden
- in einigen DBMS (z.B. Postgres) wird intern dennoch eine Transaktion auch für "atomare" Kommandos verwendet, weil diese auch viele Datensätze betreffen können (z.B. UPDATE)



# Transaktionen in MS Access

## Laut Dokumentation kennt MS Access die SQL-Anweisungen

- Beginn Transaction
- Commit Transaction
- Rollback Transaction

Es ist mir nicht gelungen, mit den Standard SQL-Befehlen in MS Access Transaktionen zu erzeugen.

Home Library Learn Downloads Support Community Sign in | United States - English |

Search MSDN with Bing

- MSDN Library
- Office Development
- 2007 Microsoft Office System
- Access 2007
- Access 2007 Developer Reference
- Microsoft Access SQL Reference
  - Data Manipulation Language
    - SELECT Statement (Microsoft Access SQL)
    - SELECT...INTO Statement (Microsoft Access SQL)
    - INSERT INTO Statement (Microsoft Access SQL)
    - UPDATE Statement (Microsoft Access SQL)
    - DELETE Statement (Microsoft Access SQL)
    - EXECUTE Statement (Microsoft Access SQL)

**TRANSACTION Statement (Microsoft Access SQL)**

Office 2007 | 1 out of 10 rated this helpful - Rate this topic

Access Developer Reference

Used to initiate and conclude explicit transactions.

**Syntax**

Initiate a new transaction.

BEGIN TRANSACTION

**TRANSACTION Statement (Microsoft Access SQL)**

Office 2007 | 1 out of 10 rated this helpful - Rate this topic

Access Developer Reference

Used to initiate and conclude explicit transactions.

**Syntax**

Initiate a new transaction.

BEGIN TRANSACTION

Conclude a transaction by committing all work performed during the transaction.

COMMIT [TRANSACTION | WORK]

Conclude a transaction by rolling back all work performed during the transaction.

ROLLBACK [TRANSACTION | WORK]

**Remarks**

Transactions are not started automatically. To start a transaction, you must do so explicitly using BEGIN TRANSACTION.

Transactions can be nested up to five levels deep. To start a nested transaction, use BEGIN TRANSACTION within the context of an existing transaction.

Transactions are not supported for linked tables.



## Stattdessen

- Workspace-Objekt
  - Beginnen einer Transaktion
  - erfolgreichen Abschließen (Commit)
  - erfolglosem Beenden (Rollback)
- Datenbankfehler nicht verschlucken, sondern behandeln, insbesondere
  - Rollback
  - Fehlermeldung anzeigen

```
Sub demoTrans()  
  On Error GoTo fehler  
  'Deklaration  
  Dim db As Database  
  Dim wks As Workspace  
  'Initialisierung  
  Set db=CurrentDb  
  Set wks=DBEngine.Workspaces(0)  
  'Datenbankoperationen ausführen  
  wks.BeginTrans  
  
  db.Execute "<Irgendein SQL>",_  
            dbFailOnError  
  db.Execute "<Noch ein SQL>",_  
            dbFailOnError  
  
  '...  
  wks.CommitTrans  
  wks.Close  
  Exit Sub  
  
fehler:  
  wks.Rollback  
  wks.Close  
End Sub
```

```
' Kontoführungsgebühr für alle Kunden
Sub kontofuehrung()
  On Error GoTo datenbankfehler

  Dim rcsKto As Recordset
  Dim db As Database
  Dim wks As Workspace

  Set db = CurrentDb
  Set rcsKto = db.OpenRecordset("tblKonten", dbOpenDynaset, dbFailOnError)
  Set wks = DBEngine.Workspaces(0)

  wks.BeginTrans ' Beginn der Transaktion

  Do Until rcsKto.EOF
    ' Kontoführungsgebühr in Höhe von 7 EUR abbuchen
    rcsKto.Edit
    rcsKto.Fields("ktoSaldo") = rcsKto.Fields("ktoSaldo") - 7
    rcsKto.Update

    rcsKto.MoveNext
  Loop

  wks.CommitTrans ' erfolgreicher Abschluss der Transaktion
  wks.Close
Exit Sub

fehler:
  MsgBox Err.Description
  wks.Rollback , Zurückrollen im Falle eines Fehlers
  wks.Close

End Sub
```

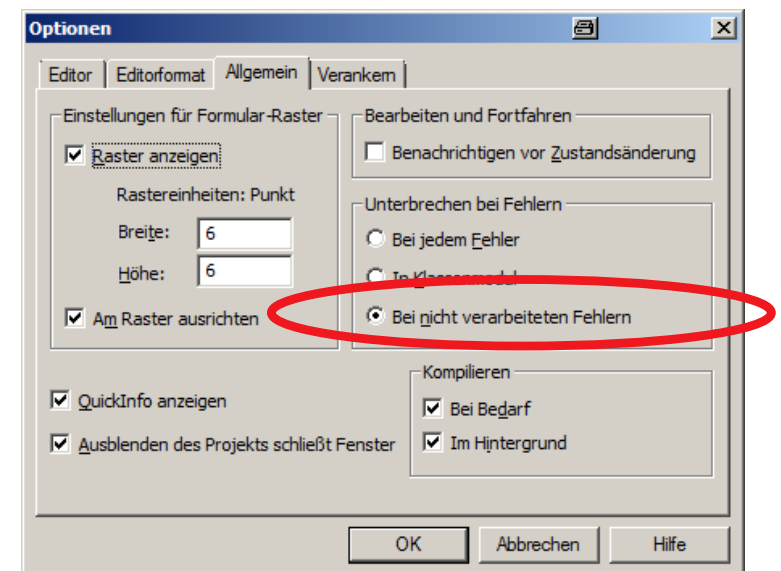
## Fehlerbehandlung

- bei Auftreten eines Fehlers Sprung zu einer Sprungmarke
- Hinter der Sprungmarke Befehle zur Behandlung des Fehlers ausführen, z.B.
  - Zurückrollen der Transaktion
  - Details zum Fehler dem Benutzer anzeigen, dazu kann das Err-Objekt verwendet werden

```
' Einschalten der Fehlerbehandlung durch Sprung
On Error GoTo fehler
' ...
fehler:
wks.Rollback ' Transaktion auf wks-Objekt
' Meldungsfenster mit Fehlerinformation
MsgBox Err.Description
' ...
```

## Wichtig! Stolperfalle in Zusammenhang mit Fehlerverarbeitung:

- Symptom: On Error GoTo scheint ohne Wirkung zu bleiben, d.h. Fehler führen immer zur Unterbrechung des Programms, selbst wenn eine Fehlerbehandlung programmiert wurde
- Lösung: VBA-Editor > Menü "Extras" > Menüeintrag "Optionen" > Dialog "Optionen" > Registerkarte "Allgemein" > Gruppe "Unterbrechen bei Fehlern" > Option "Bei nicht verarbeiteten Fehlern" aktivieren



## D08.01: Tabelle und Formular

- Datenbank mit einer Tabelle "Konten", die Spalte für Saldo darf nicht negativ werden (Jugendkonto)
- In einem Formular
  - können ein Ausgangskonto und ein Zielkonto gewählt werden
  - ein Betrag kann erfasst werden und eine Überweisung getätigt werden

The screenshot shows the 'tblKonten' table in MS Access. The table has three columns: 'ktoNr', 'ktoInhaber', and 'ktoSaldo'. The data is as follows:

ktoNr	ktoInhaber	ktoSaldo
12345	Müller	700,00 €
45678	Yilmaz	800,00 €
78901	Schmidt	200,00 €
90123	Meier	5,00 €
*		

The screenshot shows the 'frmUeberweisung' form in MS Access. The form is titled 'Überweisung' and contains the following fields and controls:

- 'Von Konto' dropdown menu with '90123' selected.
- 'Nach Konto' dropdown menu with '45678' selected.
- 'Betrag' text box with '1,00 €' entered.
- 'Überweisung durchführen' button.
- A list box showing the account details for the selected 'Nach Konto' (45678):

12345	Müller
45678	Yilmaz
78901	Schmidt
90123	Meier

## D08.01: Modul "Konten"

- Gezeigt wird die Implementierung der Überweisung
  - zunächst ohne Transaktionen und dabei möglicherweise auftretende Probleme
  - dann mit Transaktionen und Vermeidung der Fehler
- Um den Effekt des Jugendkontos beobachten zu können, müssen wir erst Gutschreiben, dann Abbuchen!

```
btnUeberweisen Click
Option Compare Database
Option Explicit

Private Sub btnUeberweisen_Click()
    ' Zuerst ohne Transaktion versuchen
    Konten.ueberweisenOhneTransaktion Me.cmbQuellKonto, _
                                       Me.cmbZielKonto, Me.txtBetrag

    ' Dann mit Transaktion versuchen
    'Konten.ueberweisen Me.cmbQuellKonto, Me.cmbZielKonto, Me.txtBetrag
End Sub
```

```
(Allgemein) ueberweisenOhneTransaktion
Option Compare Database
Option Explicit

Sub ueberweisen(lngVonKto As Long, _
                lngNachKto As Long, _
                intBetrag As Integer)

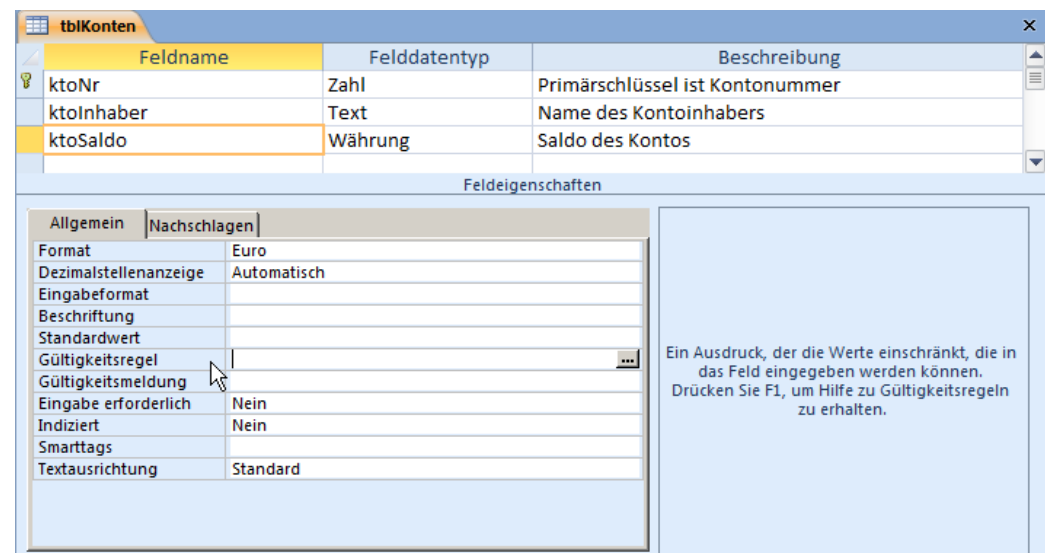
    ' TODO
End Sub

Sub ueberweisenOhneTransaktion(lngVonKto As Long, _
                                lngNachKto As Long, _
                                intBetrag As Integer)

    ' TODO
End Sub
```

## D08.01: Tabelle "Kunden"

- Legen Sie eine Tabelle "tblKonten" an mit den Spalten
  - ktoNr: Typ Zahl (Long Integer), Primärschlüssel
  - ktoInhaber: Text
  - ktoSaldo: Währung
- Legen Sie als Gültigkeitsbedingung für die Spalte Währung fest, dass der Betrag nicht negativ sein darf
- Erfassen Sie Testdaten für die Tabelle



## D08.01: Umsetzung im Modul "Konten"

### – Variante 1

- Prozedur **ueberweisenOhneTrans** (siehe nächste Folie, im PDF-Format per Copy und Paste übernehmen) anlegen und analysieren
- Einbinden der Prozedur in die Ereignisprozedur, die die Überweisung auslöst
- führen Sie durch diesen Aufruf mehrere Transaktionen durch, die auch Fehlerfälle enthalten können

### – Variante 2

- Prozedur mit dem Namen **ueberweisen** implementieren, die Transaktionen nutzt und die Simulation eines Fehlers ermöglicht
- Einbinden der Prozedur in die Ereignisprozedur, die die Überweisung auslöst
- führen Sie durch diesen Aufruf mehrere Transaktionen durch, die auch Fehlerfälle enthalten können



# Inhalt

## Ziel und Einordnung

## Rückblick

## Transaktionen

- Konsistenz und Integrität
  - Konsistenzsicherung als Ziel relationaler Datenbanken
  - Gefahren für die Konsistenz von Datenbanken
- Transaktionen und ACID
- Anwendung von Transaktionen
  - Anwendungsszenarien
  - Transaktionen in SQL
  - Transaktionen mit MS Access
- Technik der Transaktionsverarbeitung
  - Parallele Transaktionen und deren Probleme
  - Transaktionslevel
  - Umsetzung paralleler Transaktionen
  - Protokollierung
- Zusammenfassung

## Ausblick

# Parallele Transaktionen

## Grundsätzlich gilt die ACID-Eigenschaft "Isolation"

- nach der sich parallele Transaktionen gegenseitig nicht beeinflussen dürfen
- DBS steht einem Benutzer nicht exklusiv zur Verfügung, deshalb
  - Antwortzeiten für jeden Benutzer sollen möglichst kurz sein
  - möglichst viele Benutzeraktionen sollen pro Zeiteinheit verarbeitet werden

## Zur Leistungsoptimierung bringt DBMS die Einzeloperationen von parallelen Transaktionen in eine geeignete Reihenfolge (Schedule)

- Ergebnis paralleler Ausführung muss so sein, als wären Transaktionen nach einander ausgeführt worden
- auch neu eintreffende Transaktionen müssen mit bereits laufenden verzahnt werden
- nicht erfolgreich endende Transaktionen müssen beachtet werden

## Mangelnde Isolation kann zu folgenden Problemen führen<sup>1</sup>

- Lost Updates: Zwei Transaktionen modifizieren parallel denselben Datensatz und nach Ablauf dieser beiden Transaktionen wird nur die Änderung von einer von ihnen übernommen
- Dirty Read: Daten einer noch nicht abgeschlossenen Transaktion werden von einer anderen Transaktion gelesen. Wird noch nicht abgeschlossene Transaktion anschließend zurückgesetzt, wurden von der anderen Transaktion falsche Daten gelesen.
- Non-Repeatable Read: Wiederholte Lesevorgänge liefern unterschiedliche Ergebnisse, weil zwischenzeitlich Änderungen committet wurden.
- Phantom Read: Suchkriterien treffen während einer Transaktion auf unterschiedliche Datensätze zu, weil eine (während des Ablaufs dieser Transaktion laufende) andere Transaktion Datensätze hinzugefügt oder entfernt hat.

---

1) Quelle: [http://de.wikipedia.org/wiki/Isolation\\_%28Datenbank%29](http://de.wikipedia.org/wiki/Isolation_%28Datenbank%29)  
Wirtschaftsinformatik 2 - LE 08 - Transaktionen

# Transaktionslevel

## SQL-Standard definiert stufenartige Bedingungen (Level),

- mit denen Leistung und Isolation abgewogen werden können
- die Probleme mit Datenänderungen in parallelen Transaktionen unterschiedlich stark vermeiden helfen
  - Read Uncommitted
  - Read Committed
  - Repeatable Read
  - Serializable

# Transaktionslevel

## SQL-Standard definiert stufenartige Bedingungen (Level),

- mit denen Leistung und Isolation abgewogen werden können
- die Probleme mit Datenänderungen in parallelen Transaktionen unterschiedlich stark vermeiden helfen
  - Read Uncommitted
  - Read Committed
  - Repeatable Read
  - Serializable

### Read Uncommitted

- jede Änderung wird sofort für alle anderen Transaktionen sichtbar (unabhängig davon, ob COMMIT zur Bestätigung bereits erfolgt ist)
- kann Dirty Read, Non-Repeatable Read und Phantom-Read verursachen
- kann bei sehr große Leseabfrage zur Reporterzeugung die Abfragegeschwindigkeit verbessern, wenn präzise Konsistenz nicht zwingend
- wird nicht von allen DBMS implementiert (z.B. nicht von MS Access, Postgres, aber von MySQL)

# Transaktionslevel

## SQL-Standard definiert stufenartige Bedingungen (Level),

- mit denen Leistung und Isolation abgewogen werden können
- die Probleme mit Datenänderungen in parallelen Transaktionen unterschiedlich stark vermeiden helfen
  - Read Uncommitted
  - Read Committed
  - Repeatable Read
  - Serializable

### Read Committed

- lässt Änderungen für andere Transaktionen sichtbar werden, die vor der aktuell auszuführenden Operation committet wurden
- kann beim wiederholten Lesen der gleichen Daten ein anderes Ergebnis liefern (Non-Repeatable Reads oder Phantom Read)
- Standard-Level in MS Access

# Transaktionslevel

## SQL-Standard definiert stufenartige Bedingungen (Level),

- mit denen Leistung und Isolation abgewogen werden können
- die Probleme mit Datenänderungen in parallelen Transaktionen unterschiedlich stark vermeiden helfen
  - Read Uncommitted
  - Read Committed
  - **Repeatable Read**
  - Serializable

### Repeatable Read

- wiederholte Leseoperationen auf gleichen Daten liefern immer die gleichen Ergebnisse
- hinzugefügte, gelöschte oder geänderte Datensätze bleiben unberücksichtigt (Phantom Read ist möglich)

# Transaktionslevel

## SQL-Standard definiert stufenartige Bedingungen (Level),

- mit denen Leistung und Isolation abgewogen werden können
- die Probleme mit Datenänderungen in parallelen Transaktionen unterschiedlich stark vermeiden helfen
  - Read Uncommitted
  - Read Committed
  - Repeatable Read
  - Serializable

## Serializable

- parallel ablaufende Transaktionen liefern genau das gleiche Ergebnis, als würden sie nacheinander ablaufen
- obwohl es von außen so aussieht, dass Transaktionen nacheinander ablaufen, werden ihre Operationen intern tatsächlich parallel ausgeführt
- es können keine Probleme auftreten



## Verschiedene Mechanismen zur Umsetzung von Transaktionen

- Sperrverfahren: setzen während Schreib- und ggf. Leseoperationen eine Sperre auf Daten, so dass diese von parallelen Transaktionen nicht geändert werden können
  - einfache Sperrverfahren: sperren jeweils vor den Einzeloperationen und geben nach Einzeloperation die Sperre frei
  - 2-Phasen-Sperrverfahren: besorgen sich erst alle notwendigen Sperren, führen dann alle Operationen durch und geben anschließend alle Sperren wieder frei
- optimistische Verfahren: Transaktion wird durchgeführt und an ihrem Ende wird geprüft, ob es zu einem Konflikt gekommen sein könnte. In diesem Fall wird die Transaktion zurückgerollt andernfalls comittet
- Zeitmarkenverfahren: Zugriff auf Daten wird nur dann zugelassen, wenn eine bestimmte Verarbeitungsreihenfolge eingehalten wurde, andernfalls könnte ein Konflikt auftreten und deshalb wird die Transaktion zurückgesetzt

## Dauerhaftigkeit von Transaktionen erfordert, dass

- bei vollständiger oder teilweise Zerstörung der Datenbank der letzte konsistente Zustand wieder hergestellt werden kann
- bei Unterbrechung der Verarbeitung und damit u.U. laufender Transaktion (z.B durch Stromausfall) und anschließendem Neustart des Systems der letzte konsistente Zustand wieder hergestellt werden kann
- beim Zurücksetzen von Transaktionen der Zustand vor der Änderung wiederhergestellt werden kann

## Dauerhaftigkeit wird erreicht, indem Änderungen in einem Datenbank-Log protokolliert werden

Nr.	Schritt	Beschreibung
1	Lesen der Daten	Daten werden von der Festplatte gelesen und im Arbeitsspeicher bereitgestellt, sofern sie sich nicht bereits dort befinden
2	Merken bisheriger Daten	Zu ändernde Daten werden im Arbeitsspeicher zwischengespeichert (Before Image)
3	Ändern der Daten	Änderungen (Update, Insert, Delete) der Daten erfolgt im Arbeitsspeicher
4	Merken geänderter Daten	Geänderte Daten werden im Arbeitsspeicher zwischengespeichert (After Image)
5	Logdaten sichern	Before- und After Image werden vom Arbeitsspeicher in die Datei des Datenbank-Logs auf der Festplatte geschrieben
6	Geänderte Daten speichern	geänderten Daten werden vom Arbeitsspeicher in die Datenbank geschrieben
7	Transaktionsende	Commit-Eintrag wird in Log-Datei geschrieben. Anschließend wird der erfolgreiche Abschluss (dem Programm/Benutzer) bekannt gegeben

# Protokollierung

## Datenbank-Log dient zur (Re-)Konstruktion eines konsistenten Datenbankzustands

- wird eine Transaktion zurück gerollt, so kann anhand des Before-Image im Datenbank-Log der Ausgangszustand vor der Transaktion rekonstruiert werden
- wird die Datenbank vollständig oder teilweise vernichtet,
  - kann nach Einspielen der letzten Datenbanksicherung
  - das Datenbank-Log mit der Datenbank verglichen und
  - jede im Datenbank-Log abgeschlossene Transaktion in der Datenbank nachvollzogen werden (nicht abgeschlossene Transaktionen sind dann zurückgerollt)
  - dadurch entsteht der letzte konsistente Datenbankzustand
- darf nicht auf gleicher Festplatte gespeichert werden, wie Datenbank

# Inhalt

## Organisation

## Rückblick

## Einordnung

## Transaktionen

- Konsistenzsicherung als Ziel relationaler Datenbanken
- Gefahren für die Konsistenz von Datenbanken
- Transaktionen und ACID
- Anwendung in SQL und mit MS Access
- Weitere Details der Transaktionsverarbeitung
  - Parallele Transaktionen und deren Probleme
  - Transaktionslevel
  - Umsetzung paralleler Transaktionen
  - Protokollierung
- Zusammenfassung

## Ausblick

## Transaktionen

- sind zusammengehörige Abfolgen von Datenbankoperationen
- führen die Datenbank von einem konsistenten Zustand in einen konsistenten Zustand (nicht notwendigerweise ein anderer)
- weisen die ACID-Eigenschaften auf, d.h. sie stellen sicher
  - Atomarität
  - Konsistenz (Consistency)
  - Isoliertheit
  - Dauerhaftigkeit

## Transaktionen in MS Access

- Umsetzung basierend auf Workspace-Objekt des Benutzers mit

```
DBEngine.Workspaces(0).BeginTrans
```

```
DBEngine.Workspaces(0).CommitTrans
```

```
DBEngine.Workspaces(0).Rollback
```

- und der Option `dbFailOnError` beim Ausführen von, z.B.

```
CurrentDb.Execute "<einSQL>", dbFailOnError
```

```
Dim db AS Database
```

```
Set db = CurrentDb
```

```
Set rcs = db.OpenRecordSet ("<einSQL>", dbOpenDynaSet,  
dbFailOnError)
```

- und einem Code-Block, in dem auf Fehler reagiert werden kann

```
On Error GoTo fehler
```

```
fehler:
```

- optional der Möglichkeit eigene Fehler (ab Fehlernummer 513) zu erzeugen und Fehler ausgeben

```
Err.Raise vbObjectError + 513, , "Eigene Fehlermeld!"
```

```
MsgBox "Huch... " & Err.Description
```

# Zusammenfassung

## Technisch werden

- möglicherweise auftretende Probleme (z.B. Lost Update, Dirty Read, Unrepeatable Read und Phantom Read)
- durch Transaktionslevel (Read Uncommitted, Read Committed, Repeatable Read, Serializable) werden Probleme unterschiedlich konsequent ausgeschlossen

## Umsetzung erfordert u.a.

- Synchronisationsverfahren (hier bspw. Sperrverfahren) und
- Datenbank-Log (hier insb. Before- und After-Image)



# Inhalt

## Ziel und Einordnung

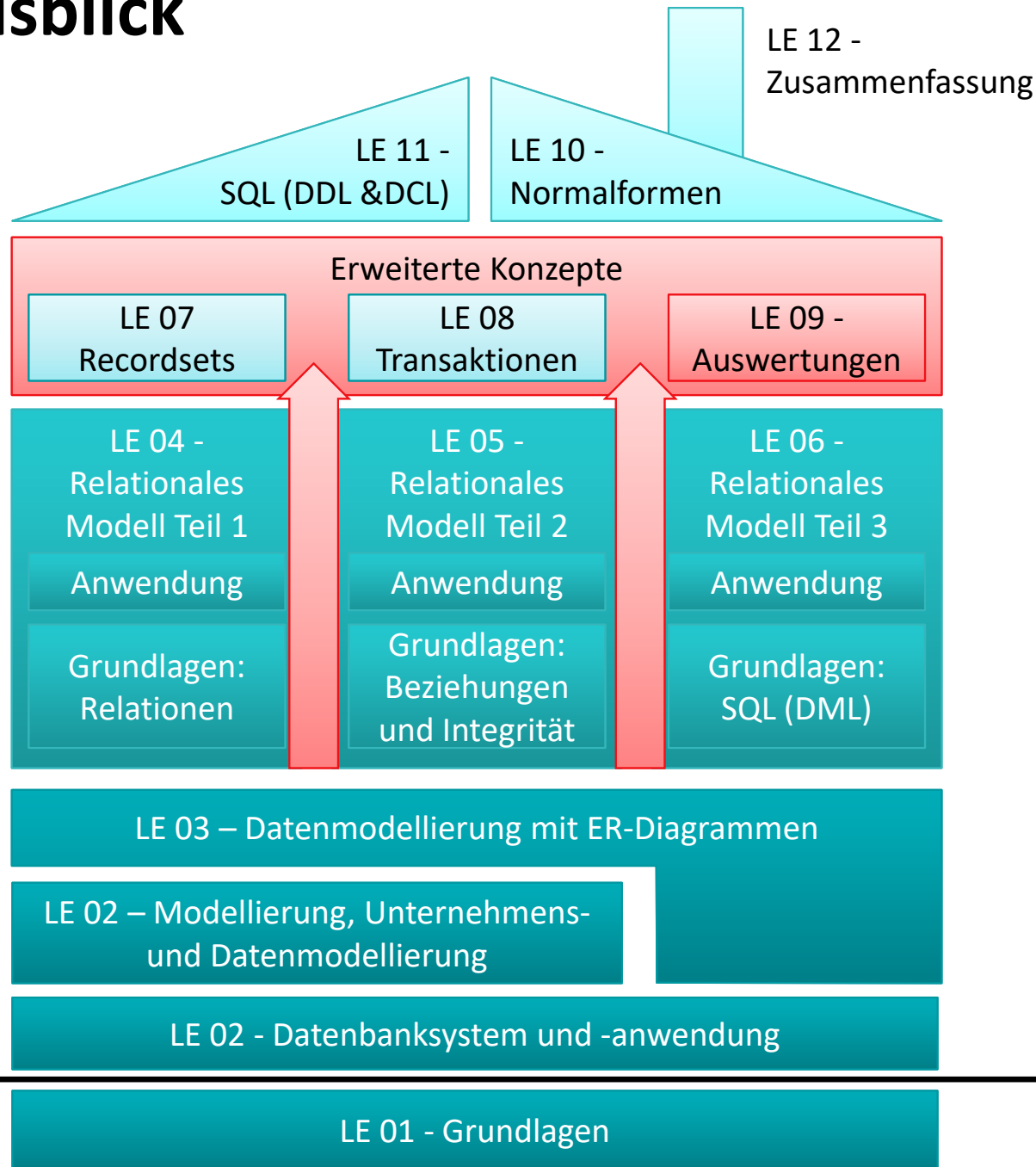
## Rückblicke

## Transaktionen

- Konsistenzsicherung als Ziel relationaler Datenbanken
- Gefahren für die Konsistenz von Datenbanken
- Transaktionen und ACID
- Anwendung in SQL und mit MS Access
- Weitere Details der Transaktionsverarbeitung
  - Parallele Transaktionen und deren Probleme
  - Transaktionslevel
  - Umsetzung paralleler Transaktionen
  - Protokollierung
- Zusammenfassung

## Ausblick

# Ausblick



# Inhalt

## Ziel und Einordnung

## Rückblicke

## Transaktionen

- Konsistenzsicherung als Ziel relationaler Datenbanken
- Gefahren für die Konsistenz von Datenbanken
- Transaktionen und ACID
- Anwendung in SQL und mit MS Access
- Weitere Details der Transaktionsverarbeitung
  - Parallele Transaktionen und deren Probleme
  - Transaktionslevel
  - Umsetzung paralleler Transaktionen
  - Protokollierung
- Zusammenfassung

## Ausblick

# **Wirtschaftsinformatik 2**

## **LE 08 – Transaktionen**

Prof. Dr. Thomas Off

<http://www.ThomasOff.de/lehre/beuth/wi2>