



## Chapter 12

---

# Long Transactions

# Special problems

## Three broad application classes where long transactions occur:

- Data analysis and maintenance tasks for ,conventional‘ databases (example of maintenance: index reconstruction) ⇒ more efficient recovery needed in case of failure.
- Workflow systems with humans in the loop ⇒ execution times become unpredictable, exclusive locks impede collaborative progress.
- Design systems ⇒ exclusive lock for objects checked out impedes concurrent, collaborative progress; efficient recovery needed.

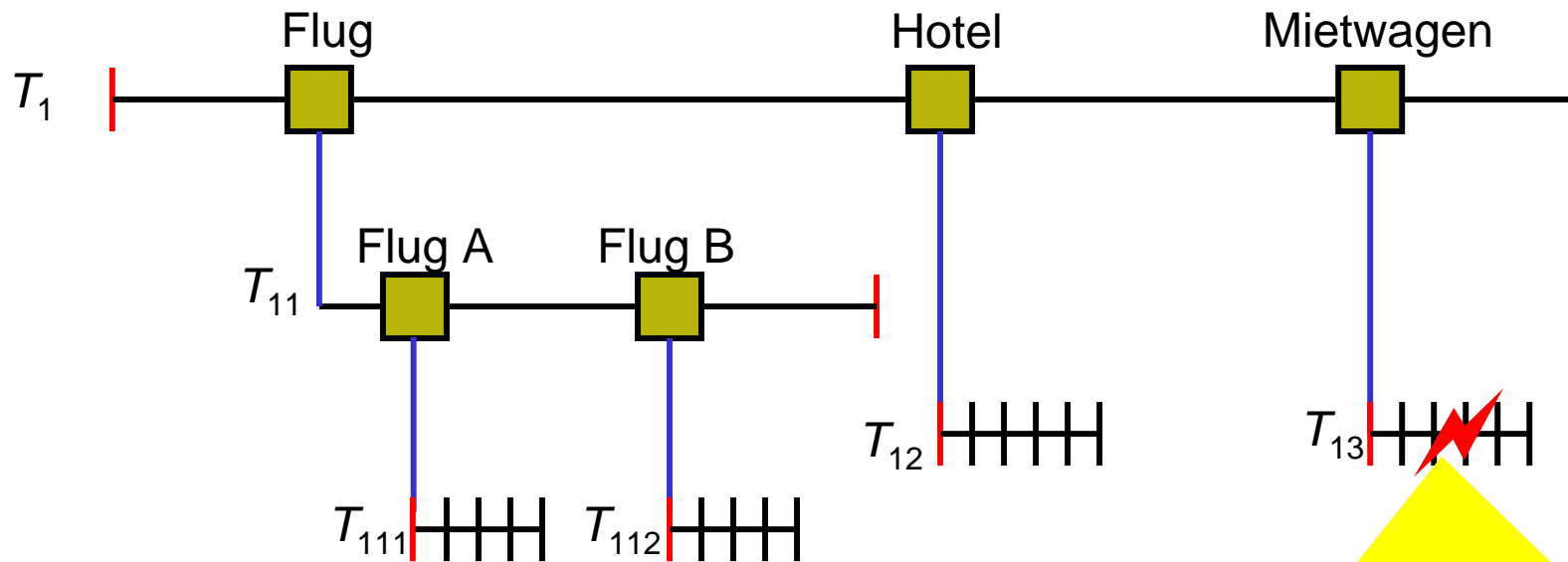
# Efficient recovery

# Motivation

- Outside appearance: ACID
  - Internal structure:
    - ◆ Finer-grained recovery.
    - ◆ Parallelism.
  - Approach: Organize long transaction as a tree of subtransactions.
- ⇒ **Closed nested transaction**

# Beispiel (1)

Buchung einer Reise: Hierzu seien z.B. die Buchung unterschiedlicher Flugabschnitte, eines Hotels und eines Mietwagens notwendig.



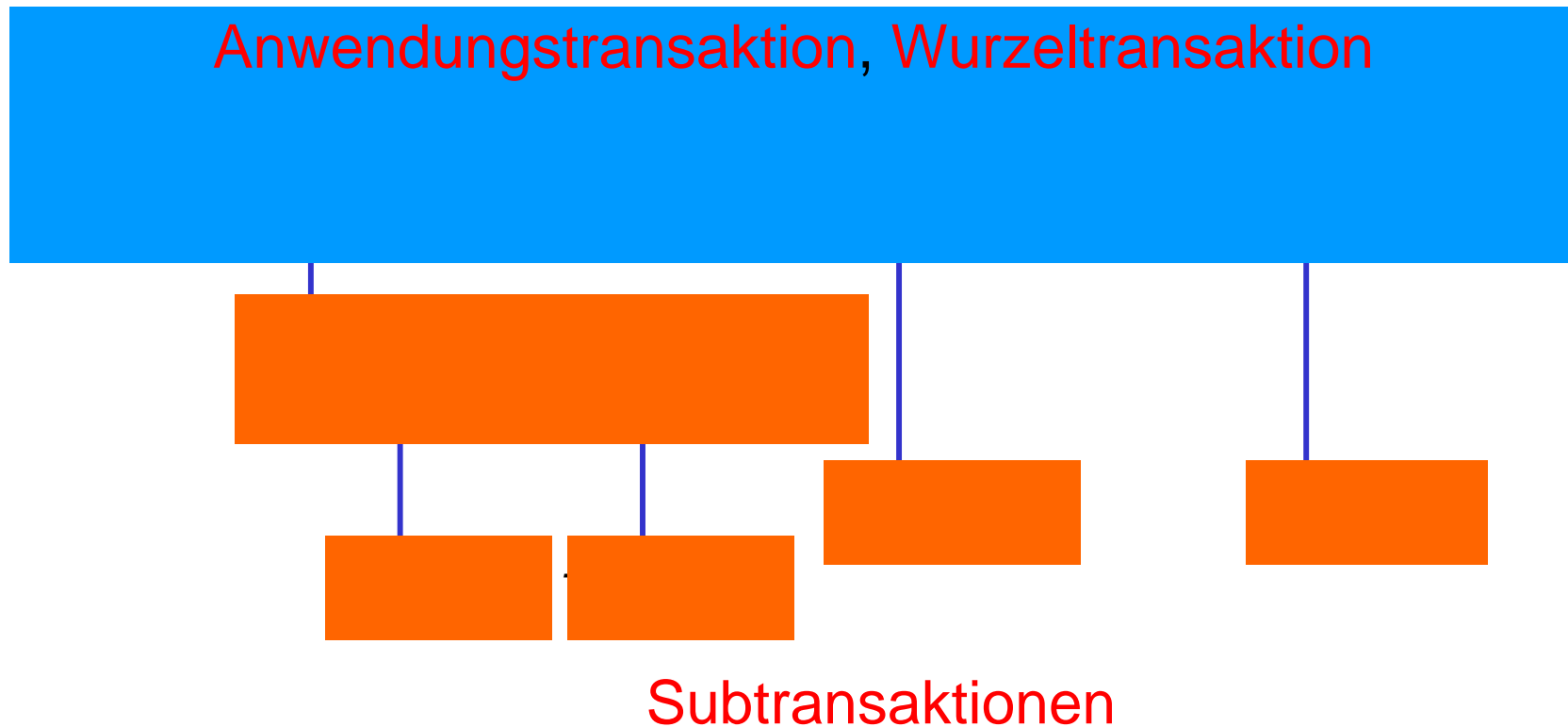
Sinnvoller wäre es, lediglich die Mietwagenbuchung (evtl. modifiziert) zu wiederholen.

Konventioneller TM: Es muss die gesamte Transaktion zurückgesetzt werden

# Beispiel (2)

## Lösung:

Die Teilaktivitäten werden als eigene **Subtransaktionen** betrachtet. Es entsteht ein **Transaktionsbaum**.



# Vorgehen

7

## Ziel:

Weitgehende Verwendung des konventionellen TM.

## Aufgabe:

Bestimme notwendige Erweiterungen.


# Recovery (1)

Differenziertere Recovery: Schrittweises, hierarchisches Vorgehen soll sich widerspiegeln.

- Blatt-Transaktionen sind ACID.
- Zusammenhänge zwischen Transaktionen entlang eines Pfades:
  - ◆ **Fehlschlag einer Subtransaktion** → Störung, aber Fehlschlag der Vatertransaktion nicht zwingend.  
Vatertransaktion bestimmt *Stelle* in ihrer Ausführung, *ab der* Fortführung möglich ist.
  - ◆ **Störung in der Vatertransaktion** → Fehlschläge aller Subtransaktionen *ab einer* von der Vatertransaktion zu bestimmenden *Stelle*.
  - ◆ **Fehlschlag der Vatertransaktion** → Fehlschlag *aller* bereits ausgeführten Subtransaktionen.

# Recovery (2)

Differenziertere Recovery: Schrittweises, hierarchisches Vorgehen soll sich widerspiegeln.

- Blatt-Transaktionen sind ACID.  konventioneller TM
- Zusammenhänge zwischen Transaktionen entlang eines Pfades:
  - ◆ **Fehlschlag einer Subtransaktion** → Störung, aber Fehlschlag

1. Jede Subtransaktion bildet eine eigene **Commit-Sphäre**: Sie gibt Änderungen an ihrem Transaktionsende, d.h. vor der Beendigung der umgebenden Transaktion, frei.
2. Fehlschlag einer Subtransaktion: Zurücksetzen.
3. Bei Fehlschlag der umgebenden Transaktion kein Zurücksetzen möglich (Subtransaktion ist abgeschlossen!), stattdessen Aufruf von **Kompensationstransaktionen**.

# Nebenläufigkeit (1)

## Nebenläufigkeit innerhalb der Transaktion:

- Subtransaktionen innerhalb einer Anwendungstransaktion lassen sich parallel bearbeiten (Intratransaktions-Parallelität).
- Im Beispiel könnte etwa die Hotel- und die Mietwagenbuchung u.U. gleichzeitig geschehen.
- Es gilt unverändert die Konfliktregelung:
  - ◆ Serialisierbarkeit innerhalb der Anwendungstransaktion
  - ◆ Sperrenfreigabe nach außen erst am Ende der Anwendungstransaktion

# Nebenläufigkeit (2)

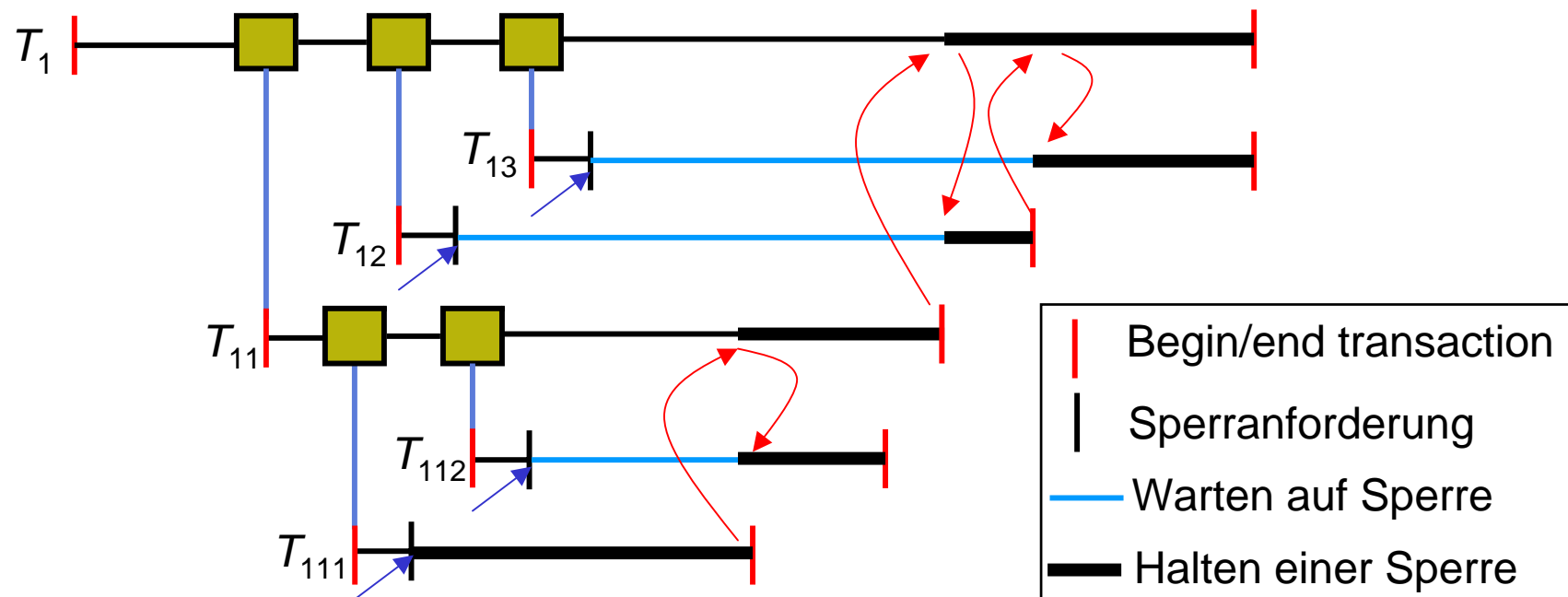
- **Annahme:** Verwendung des 2PL-Protokolls.
- **Folge:** Jede Subtransaktion gibt ihre Sperren beim Abschluss frei. Aber: Vatertransaktion ist noch nicht abgeschlossen, darf Sperren noch nicht freigeben.
- **Lösung:** Bei Beendigung einer Subtransaktion werden die Sperren an die Vatertransaktion vererbt. Damit wird die Lebensdauer der Sperren bis zum Ende der Anwendungstransaktion zugesichert.
- **Zusätzliche Regel:** Sperranforderungen einer Subtransaktion sind mit den gehaltenen Sperren ihrer Vorfahren (nicht jedoch mit denen ihrer möglicherweise parallelen Geschwister) verträglich.

# Nebenläufigkeit (3)

12

**Ergebnis:** Maximale Nebenläufigkeit unter der Einschränkung, dass Subtransaktionen derselben Vatertransaktion nur nacheinander auf dasselbe Objekt zugreifen.

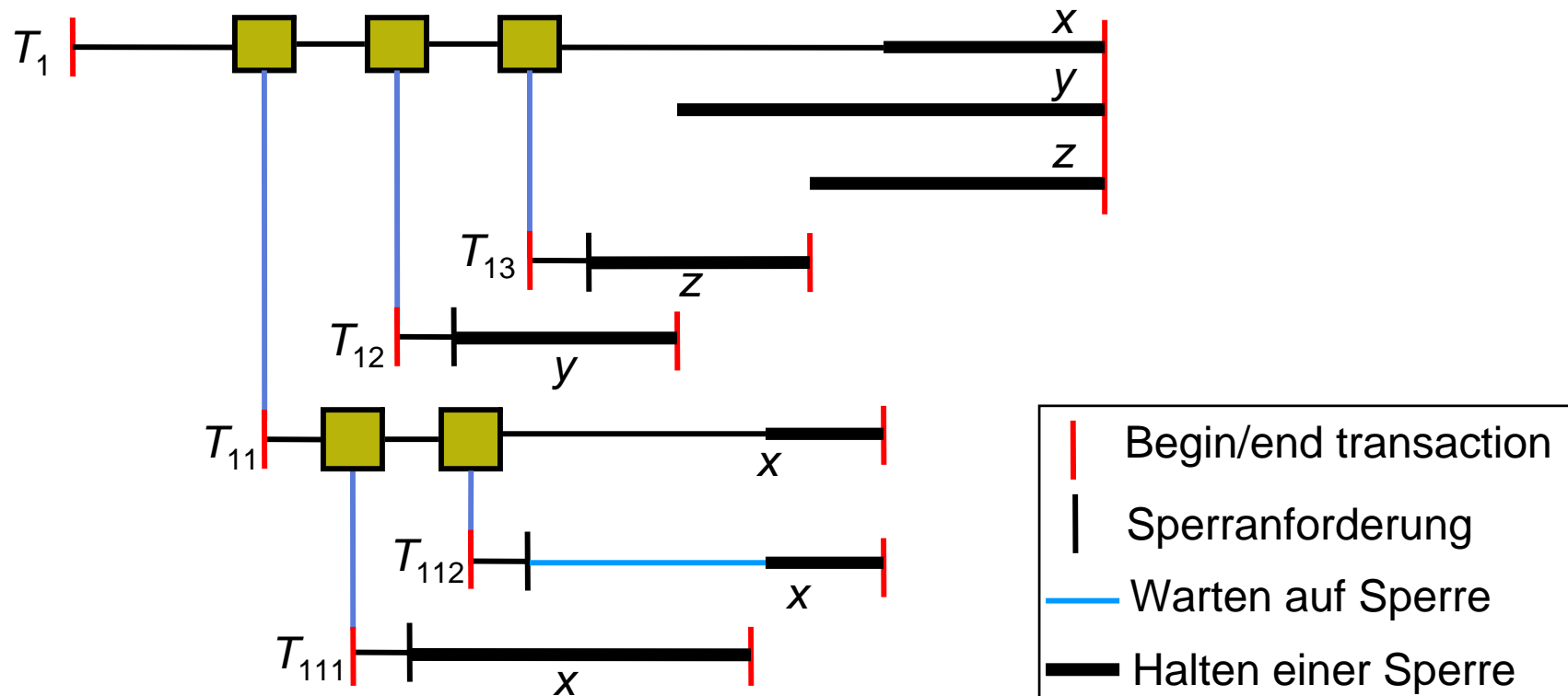
**Beispiel:** Alle Blatt-Transaktionen schreiben dasselbe Objekt x.



# Nebenläufigkeit (4)

**Ergebnis:** Maximale Nebenläufigkeit unter der Einschränkung, dass Subtransaktionen derselben Vatertransaktion nur nacheinander auf dasselbe Objekt zugreifen.

**Beispiel:**  $T_{11}$  schreibt Objekt  $x$ ,  $T_{12}$  Objekt  $y$  und  $T_{13}$  Objekt  $z$ .



# Rücksetzpunkte (1)

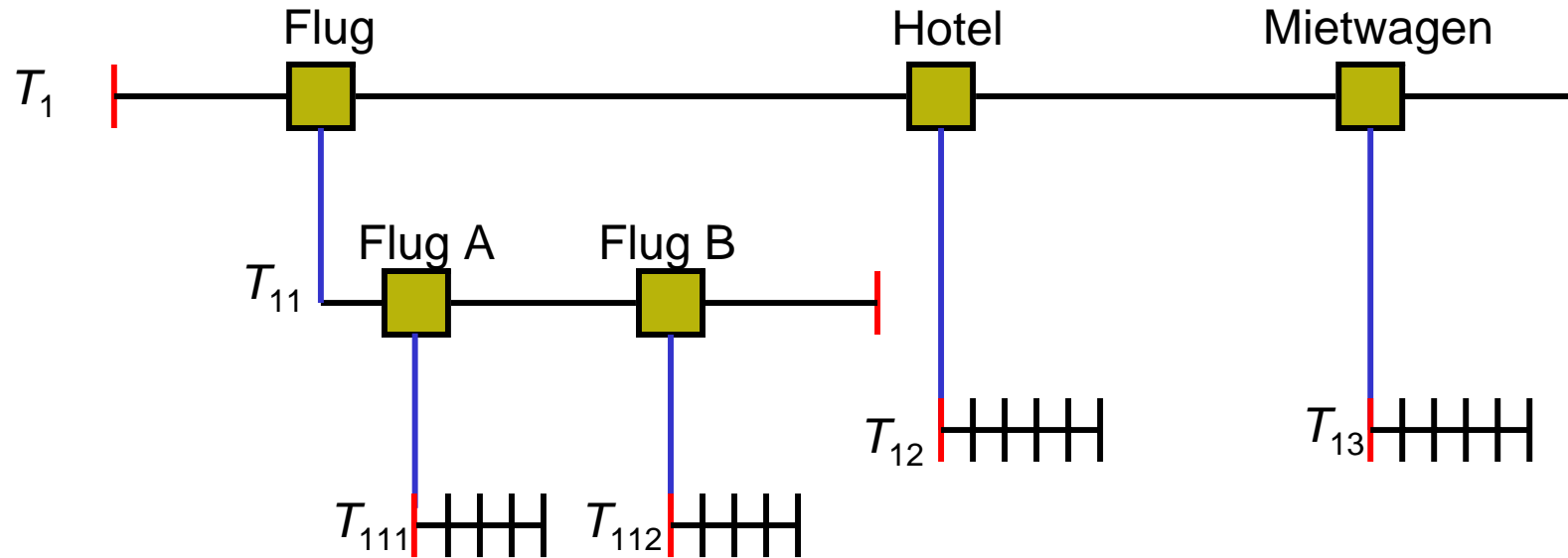
**Kompensationstransaktionen:** Wegen vollständiger Isolation aller Subtransaktionen und der Wurzeltransaktion ist die einfachste Lösung das Rücksetzen auf den Beginn der Transaktion.

Damit ist einheitliche Behandlung von Kompensation und Rücksetzen (fehlgeschlagener Transaktionen) möglich.

⇒ **Regel:** Bei geschlossen geschachtelten Transaktionen stellt der Beginn jeder Subtransaktion (sowie der Beginn der Wurzeltransaktion) einen **Rücksetzpunkt** dar.

# Rücksetzpunkte (2)

15



- Scheitert die Buchung von Flug B, so muss u.U. auch die Buchung von Flug A zurückgenommen werden. Ein geeigneter Rücksetzpunkt ist dann der Beginn von Transaktion  $T_{11}$ .
- Soll der Effekt von Transaktion  $T_{111}$  hingegen aufrechterhalten werden, so ist der Beginn von  $T_{112}$  und damit das Ende von  $T_{111}$  der geeignete Rücksetzpunkt.

# Rücksetzpunkte (3)

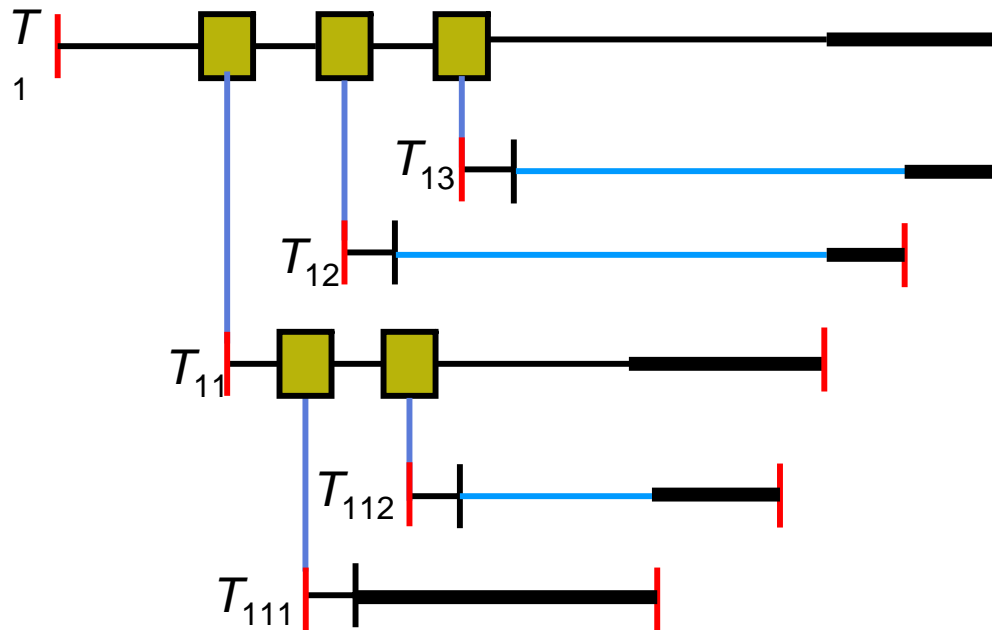
## Vorgehen:

- Für jedes von einer Subtransaktion veränderte Objekt wird ein **Before-Image** angelegt.
- Beim **commit** einer Subtransaktion werden diese Before-Images an die Vatertransaktion vererbt.
- Die (ererbten) Before-Images (evtl. mehrere!) einer (Vater-) Transaktion müssen so lange gehalten werden, bis die (Vater-)Transaktion beendet ist.

Grund: Welcher von i.d.R. mehreren möglichen Rücksetzpunkten gewählt wird, ist abhängig von der Anwendungssemantik.

# Rücksetzpunkte (4)

All Subtransaktionen  $T_{111}$ ,  $T_{112}$ ,  $T_{12}$  und  $T_{13}$  schreiben das Objekt  $x$ . Jeweilige Before-Images:  $x^{111}$ ,  $x^{112}$ ,  $x^{12}$  und  $x^{13}$ .



Zeitpunkt	Before-Images
$b(T_1)$	-
$b(T_{11})$	-
$b(T_{111})$	$x^{111}(T_{111})$
$e(T_{111})$	$x^{111}(T_{11})$
$b(T_{112})$	$x^{111}(T_{11})x^{112}(T_{112})$
$e(T_{112})$	$x^{111}(T_{11})x^{112}(T_{11})$
$e(T_{11})$	$x^{111}(T_1)$
$b(T_{12})$	$x^{111}(T_1)x^{12}(T_{12})$
$e(T_{12})$	$x^{111}(T_1)x^{12}(T_1)$
$b(T_{13})$	$x^{111}(T_1)x^{12}(T_1)x^{13}(T_{13})$
$e(T_{13})$	$x^{111}(T_1)x^{12}(T_1)x^{13}(T_1)$
$e(T_1)$	-

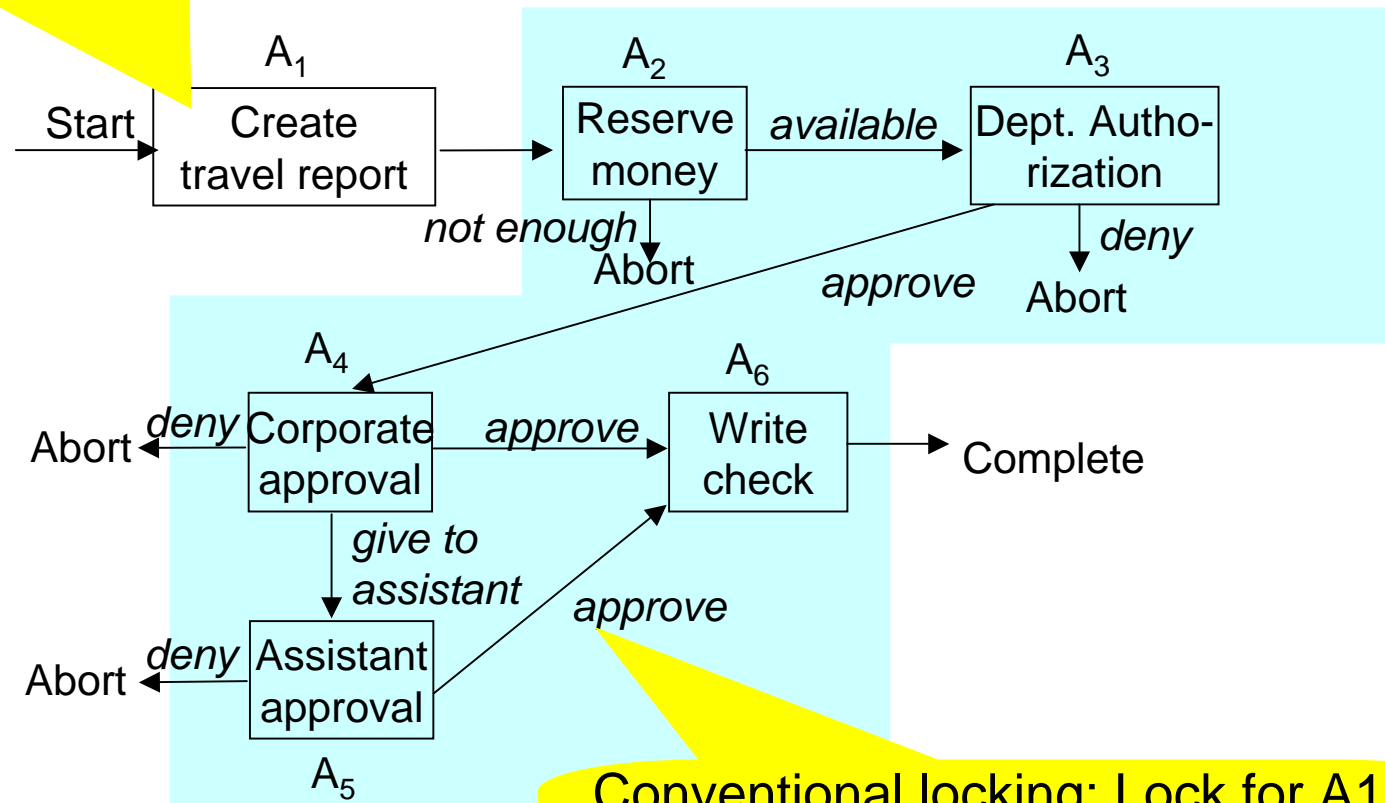
# Controlled interaction

# Motivation

- Usually, design activities are a collaborative effort with well-defined points of interaction.
- Isolation can then only be maintained over parts of a transaction.
  - ◆ Also permits higher degree of internal parallelism.
  - ◆ Threatens global and local consistency.
  - ◆ Complicates recovery.
- Several known approaches that differ in how well they are suited to different application scenarios.
- We study two general approaches:
  - ◆ Open nested transactions.
  - ◆ Sagas.

# Example

Traveler requests expense reimbursement (USD 1000,-) from Account A123.



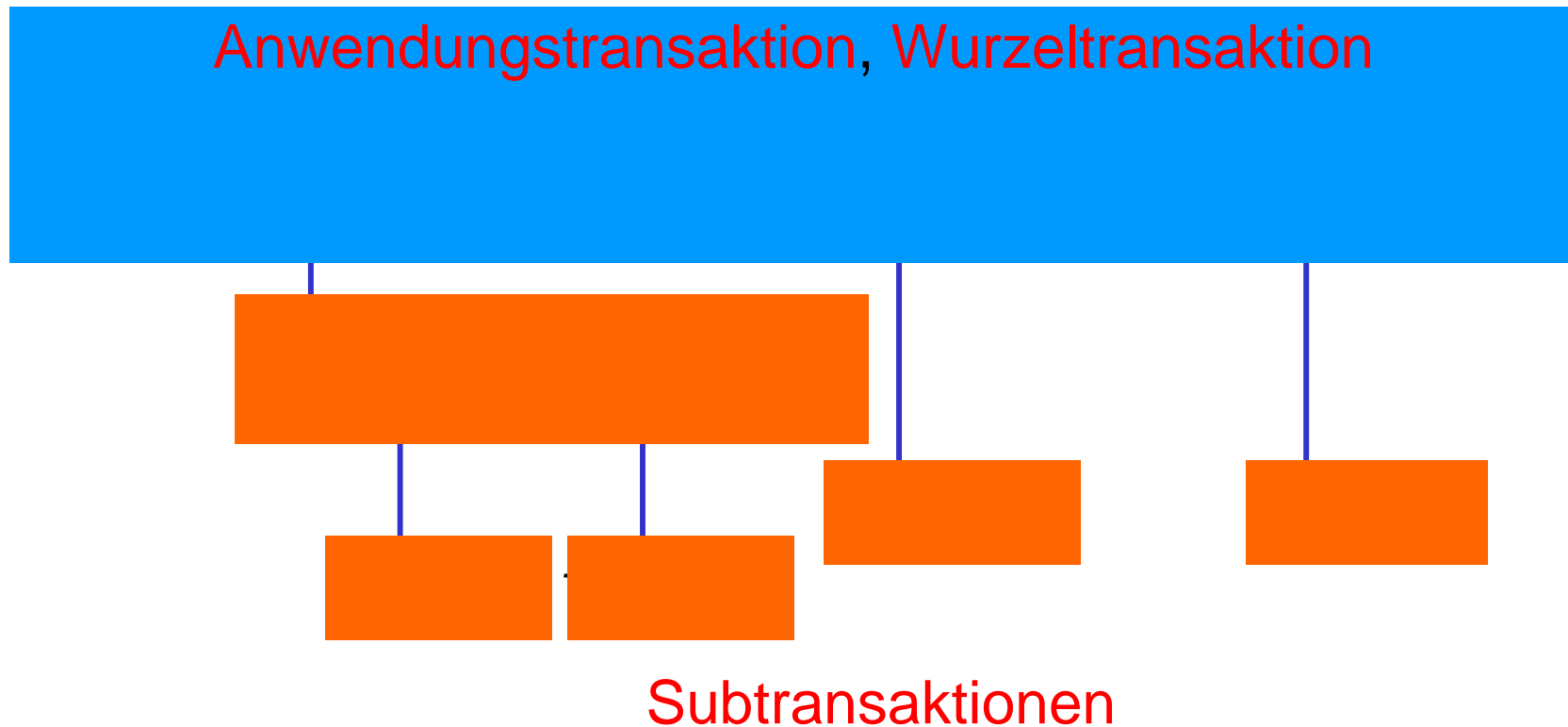
But: No concurrency control → overdraft.

Conventional locking: Lock for A123 may be held for days, blocking everything else.

# Geschachtelte Transaktionen

Lösung:

Unverändert: **Transaktionsbaum**.



# Isolation / Nebenläufigkeit (1)

22

## Nebenläufigkeit innerhalb der Transaktion:

- Subtransaktionen innerhalb einer Anwendungstransaktion lassen sich parallel bearbeiten (Intratransaktions-Parallelität).
- Es gilt unverändert die Konfliktregelung:

Keine Isolation über Blatt-Transaktionen hinaus.

Anwendungstransaktion

⇒ Keine Isolation der Gesamttransaktion

⇒ Reduzierte Konsistenz und Isolation schon innerhalb der Transaktion

⇒ Hohe Nebenläufigkeit über alle Transaktionen

# Recovery

Wirkung kann bereits sichtbar sein falls nicht Blatt!

Wirkung des gestörten oder fehlgeschlagenen Teils können bereits sichtbar sein!

- Blatt-Transaktionen sind ACID.
- Zusammenhänge zwischen Transaktionen entlang eines Pfades:
  - ◆ **Fehlschlag einer Subtransaktion** → Störung, aber Fehlschlag der Vatertransaktion nicht zwingend.  
Vatertransaktion bestimmt *Stelle* in ihrer Ausführung, *ab der* Fortführung möglich ist.
  - ◆ **Störung in der Vatertransaktion** → Fehlschläge aller Subtransaktionen *ab einer* von der Vatertransaktion zu bestimmenden *Stelle*.
  - ◆ **Fehlschlag der Vatertransaktion** → Fehlschlag *aller* bereits ausgeführten Subtransaktionen.

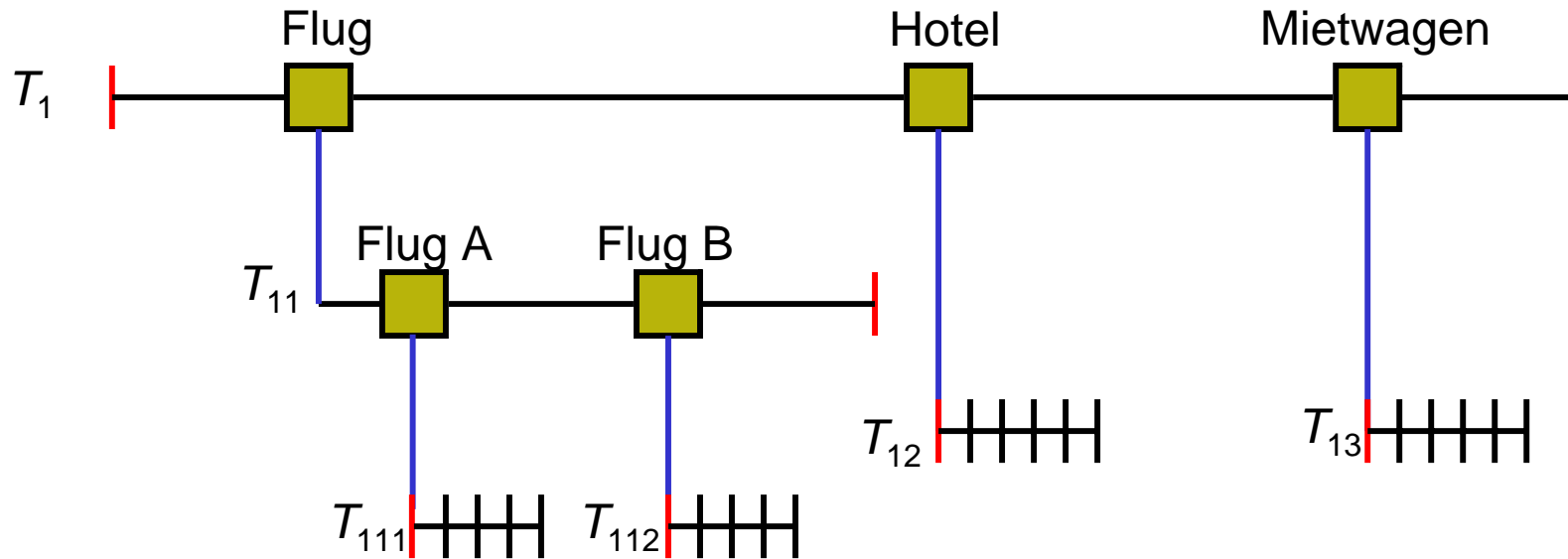
# Summary

24

- Outside appearance: ACID

⇒ **Open nested transaction**

# Beispiel (1)



## Beispiel (2)

26

- Angenommen Transaktion  $T_1$  buche einen Platz in Flug A. Dieser sei damit ausgebucht. Sie buche dann Flug B.
- Die parallel ablaufende Transaktion  $T_2$  versuche ebenfalls einen Platz in Flug A zu buchen. Dieser erscheint jedoch ausgebucht.  $T_2$  kommt also nicht weiter.
- Wird jetzt allerdings  $T_1$  zurückgesetzt, so stellt sich im Nachhinein heraus, dass Flug A doch nicht ausgebucht gewesen wäre.

### Fazit:

Die Serialisierbarkeit der Transaktionen  $T_1$  und  $T_2$  ist also nicht gewährleistet. Der Ablauf der einzelnen Blatt-Transaktionen ist zwar serialisierbar, nicht aber der Ablauf der Wurzeltransaktionen  $T_1$  und  $T_2$ .

# Isolation / Nebenläufigkeit (2)

27

## Lösungen:

- Subtransaktionen sollten statt der vollständigen Sperrfreigabe Sperren mit einer differenzierteren Bedeutung setzen, um die Isolation wieder zu gewährleisten (**semantische Concurrency-Control**) und trotzdem die **Inter-Transaktions-Parallelität** nicht unnötig einzuschränken.

**Kooperation** durch  
**semantische** Konfliktrelationen

- Kompensationstransaktionen setzen nicht mehr einfach zurück.

**Kooperation** durch **globale**  
Informierung

# Vergleich

## Geschlossen geschachtelte Transaktionen

- garantieren Isolation, verhindern somit Erhöhung der Intertransaktionsparallelität,
- haben alle ACID-Eigenschaften, bieten aber intern differenzierte Rücksetzmöglichkeiten.

## Offen geschachtelte Transaktionen

- bieten intern differenzierte Rücksetzmöglichkeiten,
- schränken die Isolation und damit die Serialisierbarkeit zu Gunsten einer Kooperation ein,
- bieten eine Grundlage für unterschiedliche Kooperationsmodelle,
- stellen hohe und situationsabhängige Ansprüche an semantische Sperren oder Kompensationstransaktionen.

Systematisches Vorgehen für Kompensationstransaktionen durch einfacheres Modell  $\Rightarrow$  Sagas

# Sagas (1)

- **Saga:** Anwendungstransaktion  $s_i$  bestehend aus einer linearen Folge von Subtransaktionen  $t_{ij}$ :
  - ◆ Die Subtransaktionen besitzen ACID-Eigenschaften.
  - ◆ Finalzustände der  $t_{ij}$  werden nicht nur innerhalb  $s_i$ , sondern auch den  $t_{kl}$ ,  $k \neq i$ , sichtbar.

⇒ Die Anwendungstransaktion besitzt keine ACID-Eigenschaften.
- **Folge für Recovery:**
  - ◆ Atomizität bleibt erhalten: Saga  $s_i$  wird bei Fehlschlag einer  $t_{ij}$  in ihrer Gesamtheit zurückgesetzt.
  - ◆ Dabei wird  $t_{ij}$  wie üblich zurückgesetzt.
  - ◆ Die früheren Subtransaktionen bedürfen eines differenzierteren Rücksetzens, da ihr Ergebnisse sichtbar waren.

# Sagas (2)

## ■ Lösung:

- ◆ Jede Subtransaktion  $t_{ij}$  wird mit einer Kompensationstransaktion  $c_{ij}$  ausgestattet („Best effort“).
- ◆ Über die  $c_{ij}$  werden keine Annahmen gemacht. Insbesondere wird also nicht garantiert, dass der Zustand zu Beginn von  $t_{ij}$  wiederhergestellt wird.
- ◆ Bei Rücksetzen werden die  $c_{ij}$  in umgekehrter Reihenfolge der  $t_{ij}$  ausgeführt.
- ◆ Sei  $s = ((t_1, c_1), \dots, (t_n, c_n))$ . Dann wird entweder die Folge  
 $t_1, t_2, \dots, t_n$   
oder die Folge  
 $t_1, t_2, \dots, t_j, c_j, \dots, c_2, c_1 \quad (0 \leq j < n)$   
ausgeführt.

# Scheduling

## Saga-Verwalter (SM)

- Notiere **begin-saga** in SM-Log
- Für jede Untertransaktion  $t_j$ , notiere Ausführung in SM-Log und reiche  $t_j$  weiter an Transaktions-Verwalter (TM).
- Falls Abort-Meldung vom TM, schreibe **abort-saga** in SM-Log und beende normale Ausführung. Sonst vermerke erfolgreichen Abschluss von  $t_j$ .
- Bei erfolgreichem Ende notiere **end-saga** in SM-Log.

## Transaktions-Verwalter (TM)

- Menge der  $t_{ij}$  für eine Menge von Sagas  $s_i$  bilden wieder eine Historie wie gewohnt.
- TM arbeitet konventionell wie bisher beschrieben. Führt insbesondere sein eigenes TM-Log.

# Rücksetzen

TM melde Abort von  $t_j$ :

- TM hat bereits  $t_j$  konventionell zurückgesetzt.
- Daher anschließend schrittweise Anweisung an TM zur Durchführung von  $c_{j-1}, \dots, c_2, c_1$  als konventionelle Transaktionen mit entsprechendem Notieren in SM-Log.
- Problem: Falls eine der  $c_i$  einen Fehler aufweist, kann die Saga nicht abgeschlossen werden. Erfordert besondere Zusatzmaßnahmen.

# Restart (1)

## Backward Recovery:

1. TM führt seine (konventionelle) Recovery durch.
2. SM inspiziert die Einträge im SM-Log
  - (a) Es existiert zu *begin-saga* ein *end-saga*: Keine Aktion.
  - (b) Es existiert zu *begin-saga* ein *abort-saga*: Sei  $c_j$  die Kompensationstransaktion, die vom TM noch nicht als beendet gemeldet war. Dann schrittweise Anweisung an TM zur Durchführung von  $c_j, \dots, c_2, c_1$ .
  - (c) Andernfalls: Letzte im SM-Log vermerkte Untertransaktion  $t_j$  war an TM gegangen. Falls noch nicht als abgeschlossen vermerkt, wurde  $t_j$  von TM zurückgesetzt, an den TM sind der Reihe nach  $c_{j-1}, \dots, c_2, c_1$  zu senden. Falls als abgeschlossen vermerkt, gehen  $c_j, \dots, c_2, c_1$  an TM.

# Restart (2)

## Backward/Forward Recovery:

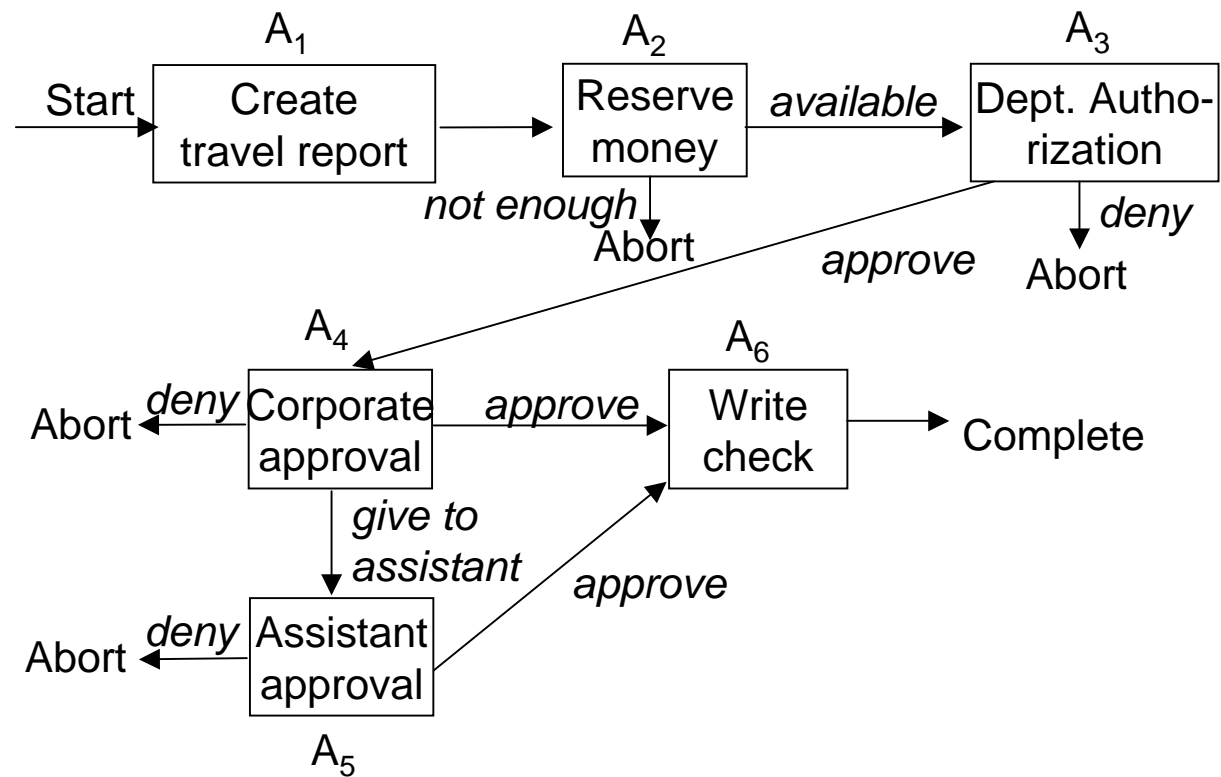
Abwandlung à la geschachtelte Transaktionen für den Fall 2.(c) (weder *end-saga* noch *abort-saga*).

- Es gibt es keinen eigentlichen Grund zum Rücksetzen (eingeschränkte Isolation und Atomizität!), man könnte die Saga auch fortsetzen.
- Erforderlich: Savepoints zwischen einigen oder allen der  $t_{ij}$ .
- Sei  $s = t_1, t_2, t_3, t_4, \dots, t_n$  und das System während der Ausführung von  $t_4$  abgestürzt. Existiere Savepoint zwischen  $t_2$  und  $t_3$ . Dann Restart-Abfolge (über das SM-Log sicherzustellen):

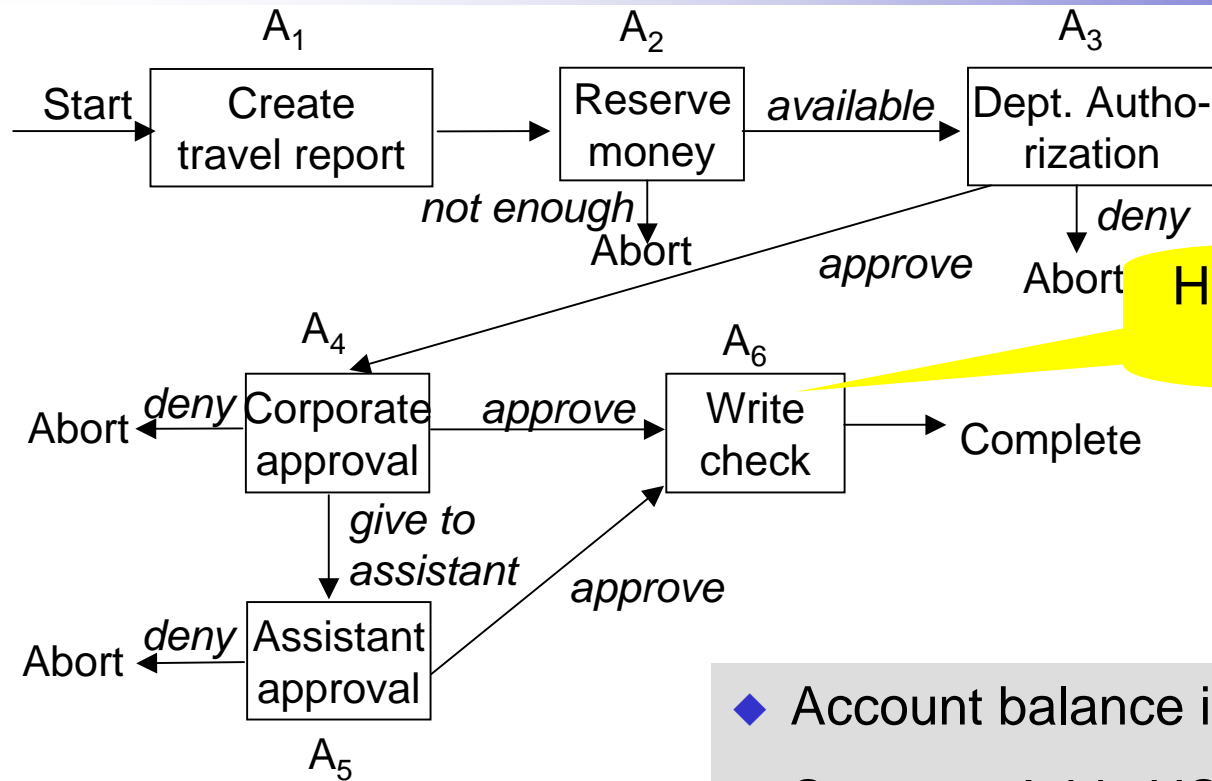
$c_3, \text{restore-savepoint}, t_3, t_4, \dots, t_n$

# Sagas (3)

Generalization to graphs is possible:



# Compensation



How would one compensate for a real-world action?

- ◆ Account balance is USD 2000,--.
- ◆ Saga  $s_1$ : Adds USD 1000,-- to account.
- ◆ Saga  $s_2$ : Deletes USD 2500,-- from account.
- ◆ Rollback of  $s_1$ : **Account goes negative.**
- ◆ Compensation should know of  $s_2$ .