Kapitel 2

Transaktionsmodelle

Inhalt

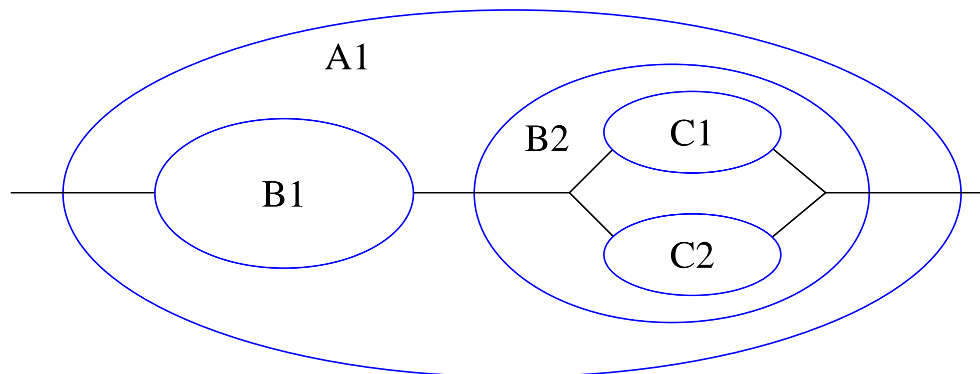
- ❑ Kontrollbereiche
- ❑ Beschreibung von Transaktionsmodellen
  - ➯ Regelsprache
  - ➯ Beispiel: flache Transaktionen (mit Sicherungspunkten)
- ❑ Transaktionsmodelle
  - ➯ gekettete Transaktionen
  - ➯ verteilte Transaktionen
  - ➯ geschachtelte Transaktionen
  - ➯ Mehrebenentransaktionen
  - ➯ Mini-Batches
  - ➯ Sagas
- ❑ Zusammenfassung

Kontrollbereiche (1)

❑ Idee

➪ Ausgangspunkt

- Kontrolle von Programmausführungen in verteilten Mehrbenutzerumgebungen bedeutet primär:
  - Auswirkungen beliebiger Operationen so lange begrenzen, bis sie mit Sicherheit nicht mehr zurückgesetzt werden müssen
  - Abhängigkeiten der Operationen voneinander aufzeichnen, um die Ausführungsreihenfolge verfolgen zu können, falls zu irgendeinem Zeitpunkt fehlerhafte Daten vorgefunden werden
- *Spehres of Control* (SoC): Bjork, Davis; Anfang der 70er



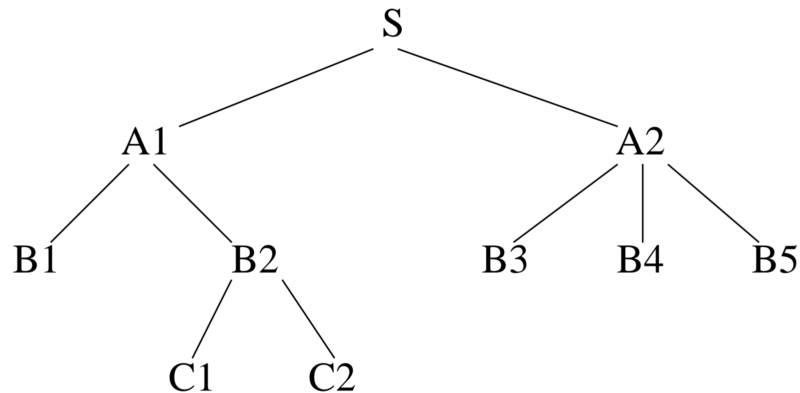
  - unteilbare Prozesse: aus Sicht der aufrufenden Schicht ununterbrechbare Operation
  - zentrale Eigenschaft: Nebenwirkungsfreiheit unteilbarer Prozesse
  - kontrollierte Übergabe von Änderungen (aus Kontrollbereich)
  - Informationen zugänglich für alle gleichzeitig ablaufenden Prozesse des nächstgrößeren Kontrollbereichs (Verzicht auf einseitiges Rücksetzen)

Kontrollbereiche (2)

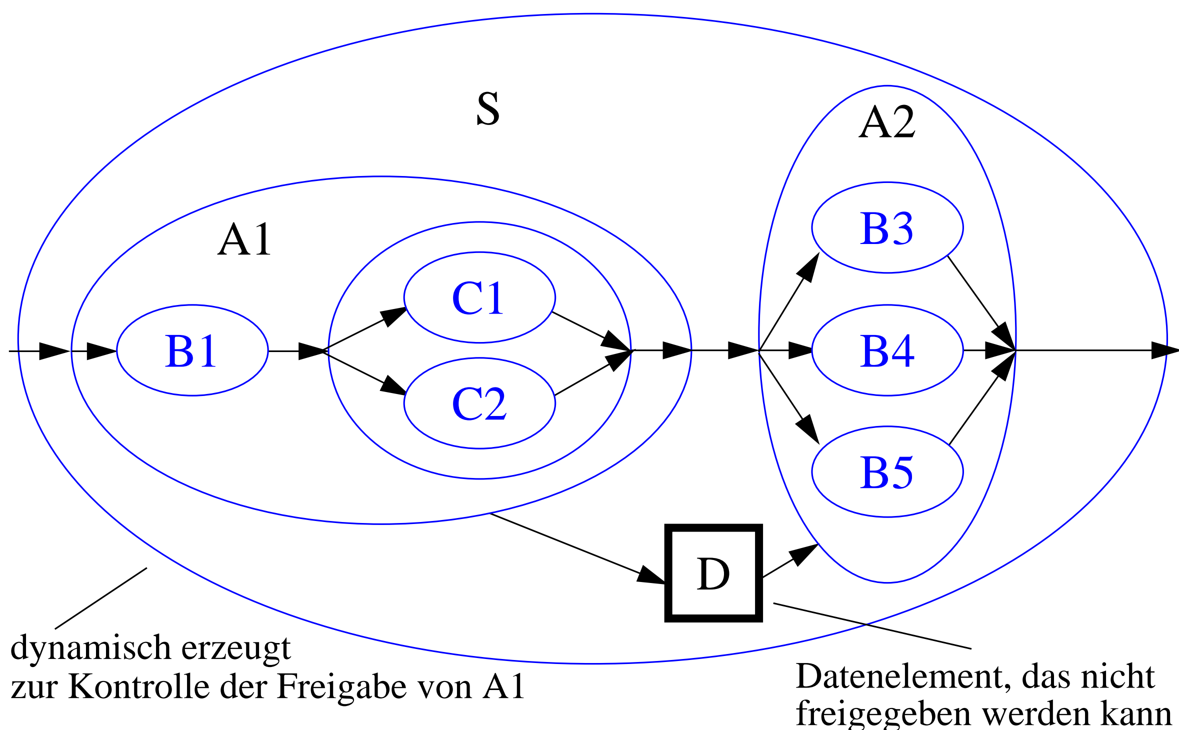
- Definitionen
  - Prozesskontrolle
    - gewährleistet, dass Information, die von einem atomaren Prozess benötigt wird, nicht von anderen modifiziert wird
    - beschränkt Abhängigkeiten von Änderungen dieses Prozesses für andere Prozesse
  - Prozess-Atomizität
    - Verarbeitungsgranulat, als Einheit zu betrachten
    - von ihr eingeführte Verarbeitungskontrolle gestattet es, Operator als atomar auf einer Abstraktionsebene zu betrachten, auch wenn seine Implementierung aus vielen parallelen und/oder sequentiellen Operationen auf nächst tieferer Ebene zusammengesetzt ist
  - Prozessfreigabe
    - solange Funktion noch abgewickelt wird, bleiben ihre Zustandsänderungen lokal
    - solange Prozessfreigabe noch nicht erfolgt ist und Ergebnisse zurückgehalten (kontrolliert) werden, kann einseitiges Zurücksetzen (process undo) über viel größere Einheiten von Prozessen durchgeführt werden
    - einseitiges Rücksetzen (unilateral abort) heißt hier: nicht von jedem Teilnehmer eines Prozesses muss OK vorliegen
    - Kontrolle der Prozessfreigabe: Zurückhalten von Auswirkungen eines Prozesses, selbst über Prozessende hinaus

Kontrollbereiche (3)

- Beispiel
  - Hierarchie von SoC

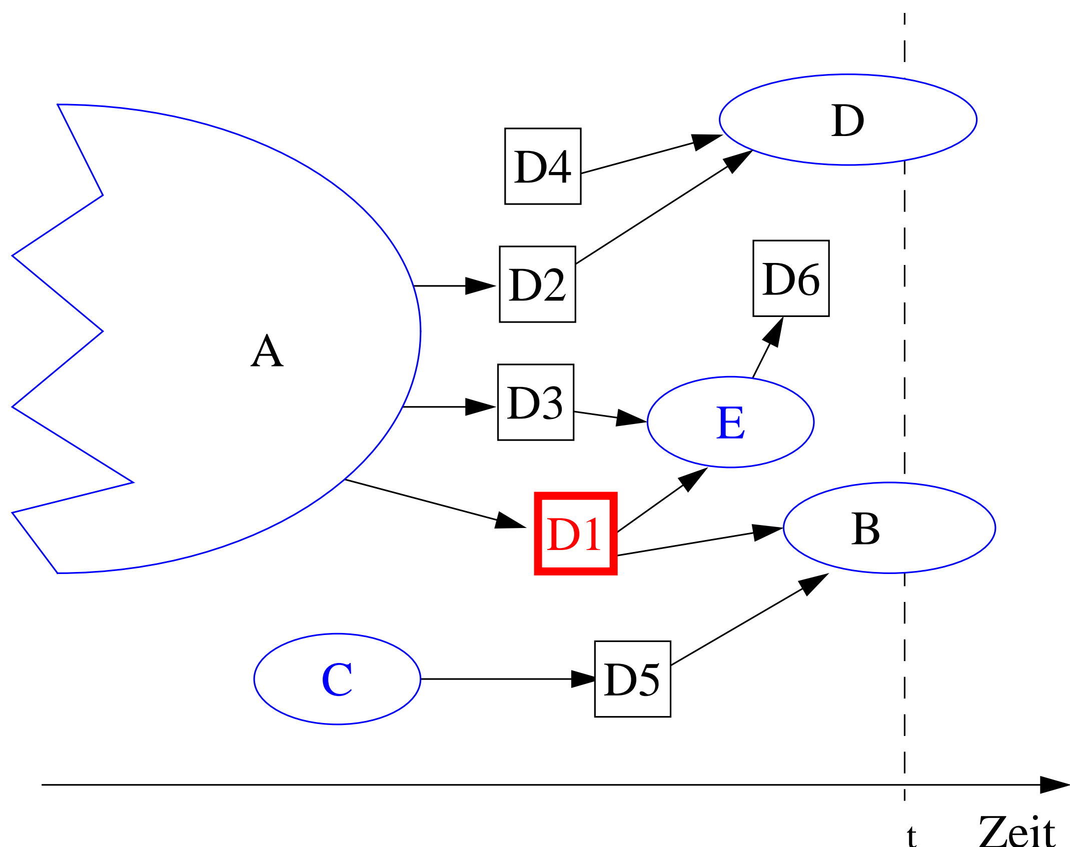


  - SoC-Darstellung: jede SoC aus Sicht aller Prozesse auf gleicher oder höherer Kontrollebene atomare Aktion



Kontrollbereiche (4)

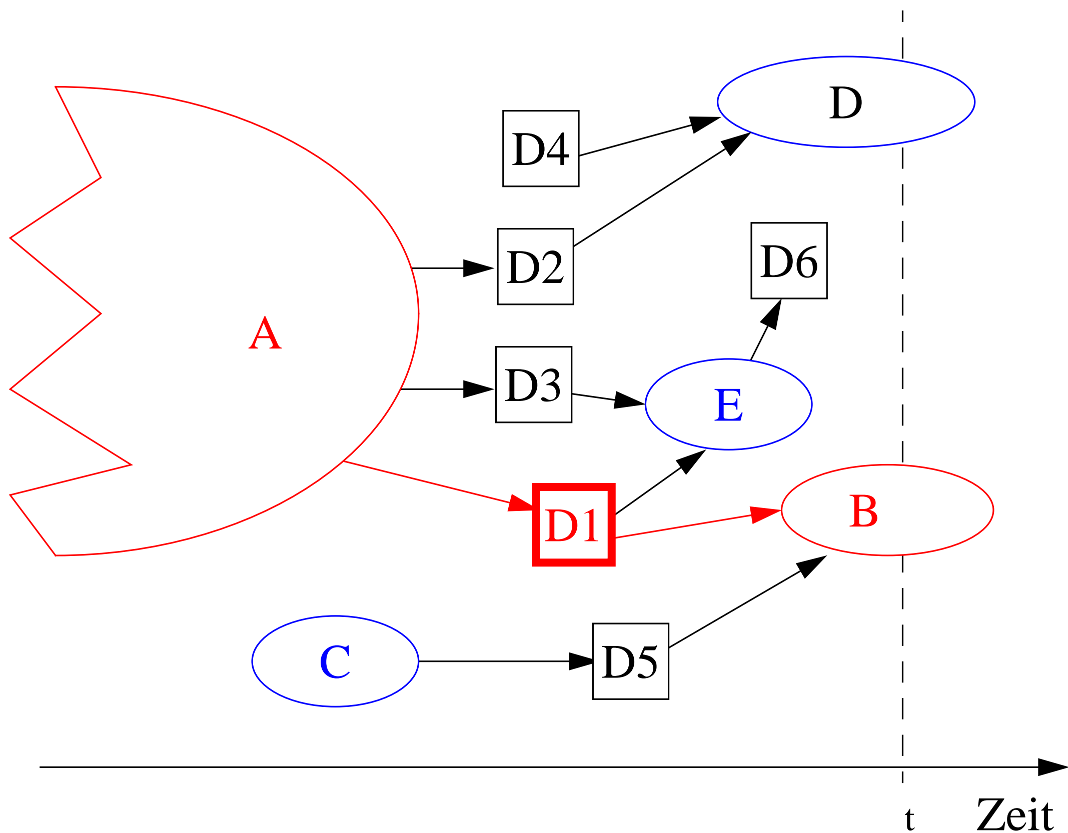
- 2 Arten von Kontrollbereichen
  - ergibt sich durch *statische Strukturierung* des Systems in Hierarchie abstrakter Datentypen
  - resultiert aus *dynamischen Interaktionen* von SoCs auf gemeinsamen Daten, die noch nicht freigegeben werden können
- Szenario
  - 1. Beginn: Austausch fehlerhafter Daten



- SoC A hat D1 erzeugt
- Annahme: in B wird Problem entdeckt, das von D1 herrührt

Kontrollbereiche (5)

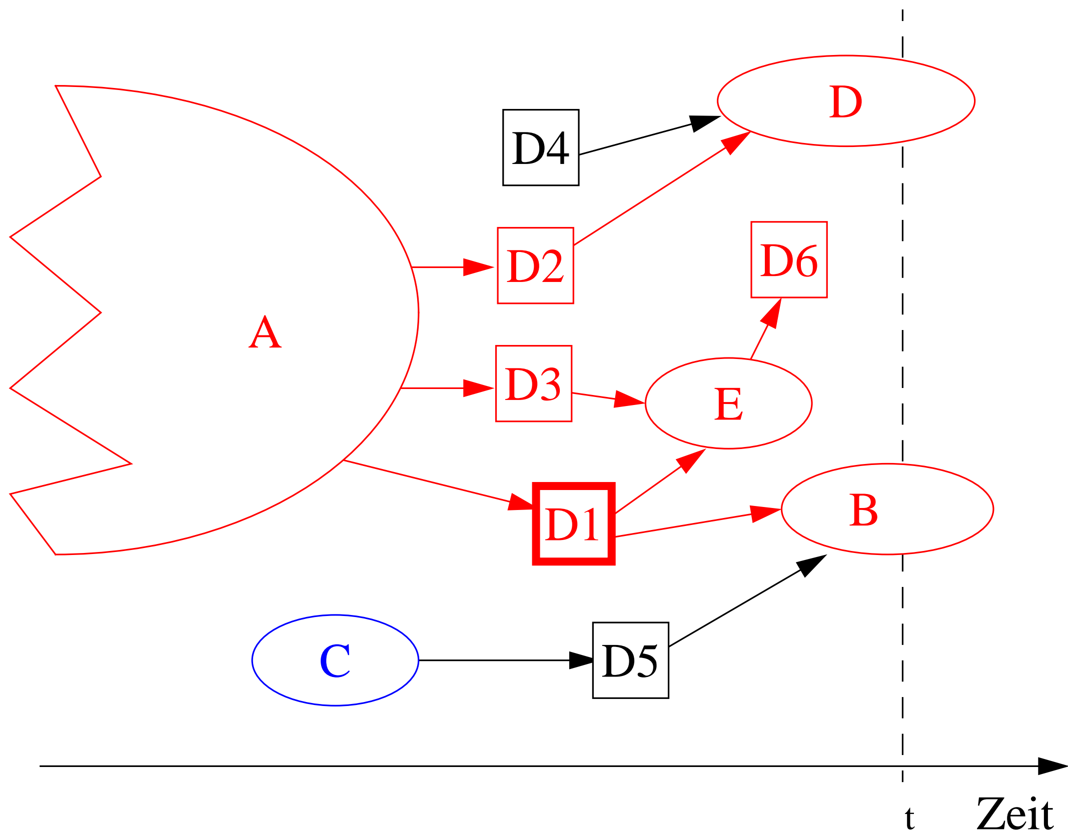
- Szenario (Forts.)
  - 2. zeitliches Zurückverfolgen der Abhängigkeiten



- dynamische SoC wird rückwärts aufgebaut, bis die SoC enthalten ist, die fehlerhaftes D1 erzeugt hat

Kontrollbereiche (6)

- Szenario (Forts.)
  - 3. Erzeugen aller dynamischen Abhängigkeiten, um alle Prozesse zu erfassen, die von der Fehlerkorrektur betroffen sind

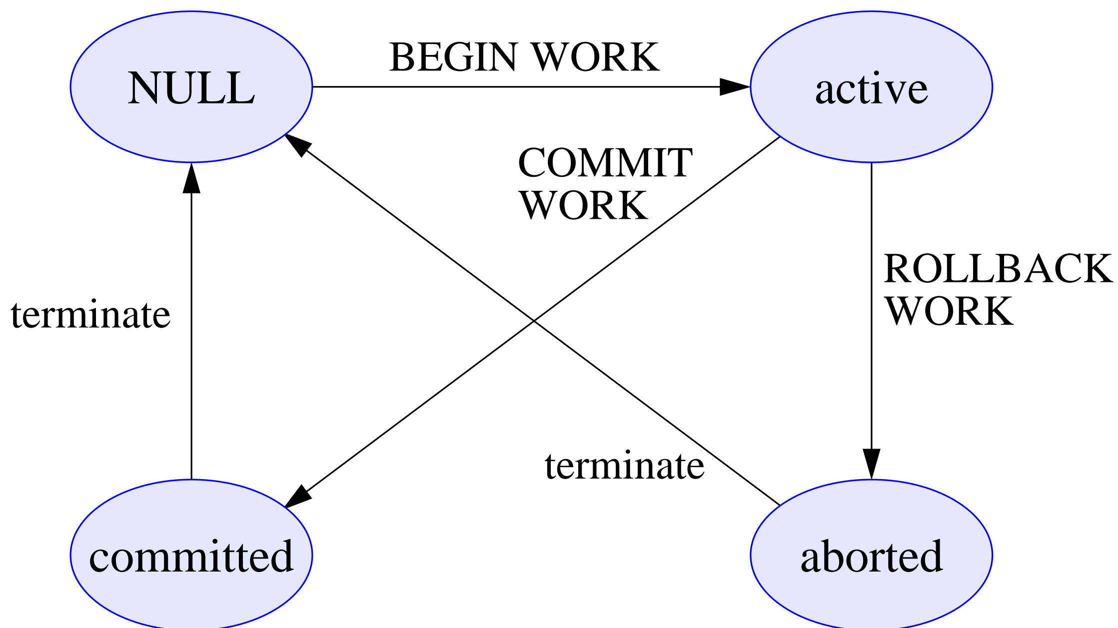


- Recovery-SoC wird zeitlich gesehen ‘vorwärts’ expandiert; muss alle Prozesse umfassen, die *potentiell* von der Nutzung des fehlerhaften D1 betroffen sind

Beschreibung von Transaktionsmodellen (1)

❑ Notation

➩ Zustands-/Übergangsdiagramm für einzelne, flache Transaktionen aus Sicht der Anwendung



➩ Abhängigkeiten zwischen Transaktionen

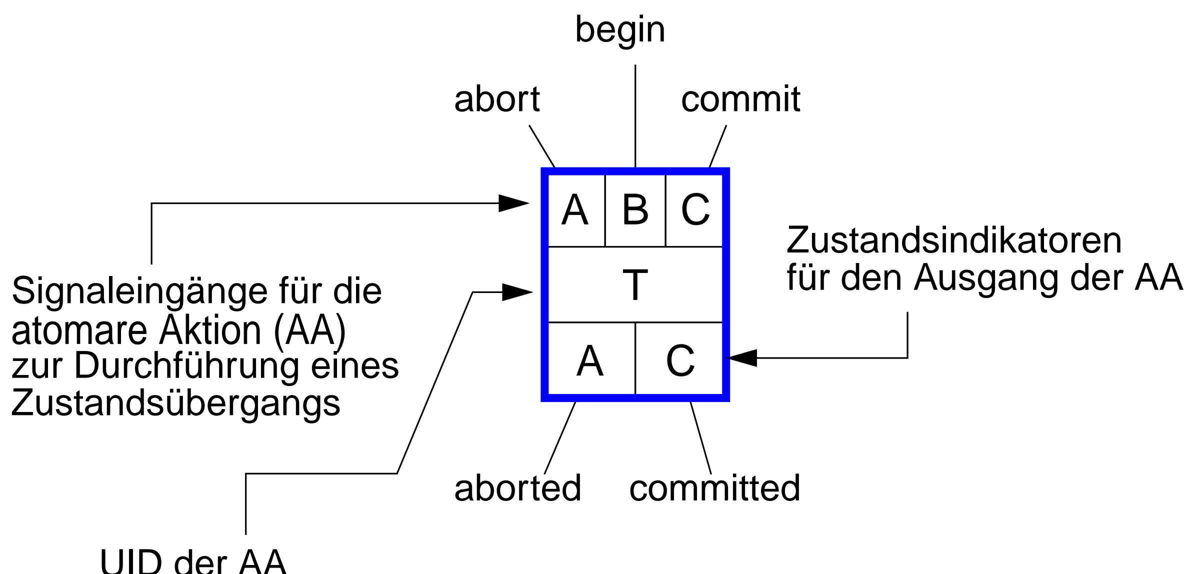
- strukturelle Abhängigkeiten
  - hierarchische Organisation
  - festgelegt, da ‘Aufrufhierarchie im Code vorgegeben’
- dynamische Abhängigkeiten
  - Nutzung gemeinsamer Daten
  - können beliebige Anzahl sonst unabhängiger atomarer Aktionen einhüllen

➩ strukturelle und dynamische Abhängigkeiten als Regeln beschreibbar!

Beschreibung von Transaktionsmodellen (2)

- Regelaufbau
  - aktiver Teil:
    - Begin Work (B), Rollback Work (Abort, A), und Commit Work (C) verursachen Zustandsänderungen bei einer atomaren Aktion
    - Transaktionsmodelle können Bedingungen definieren, die diese Events auslösen
  - passiver Teil
    - abhängige atomare Aktionen dürfen bestimmte Zustandsübergänge nicht von sich aus durchführen
    - Regeln erforderlich, die Bedingungen für diese Zustandsübergänge spezifizieren

- Grafische Notation



Beschreibung von Transaktionsmodellen (3)

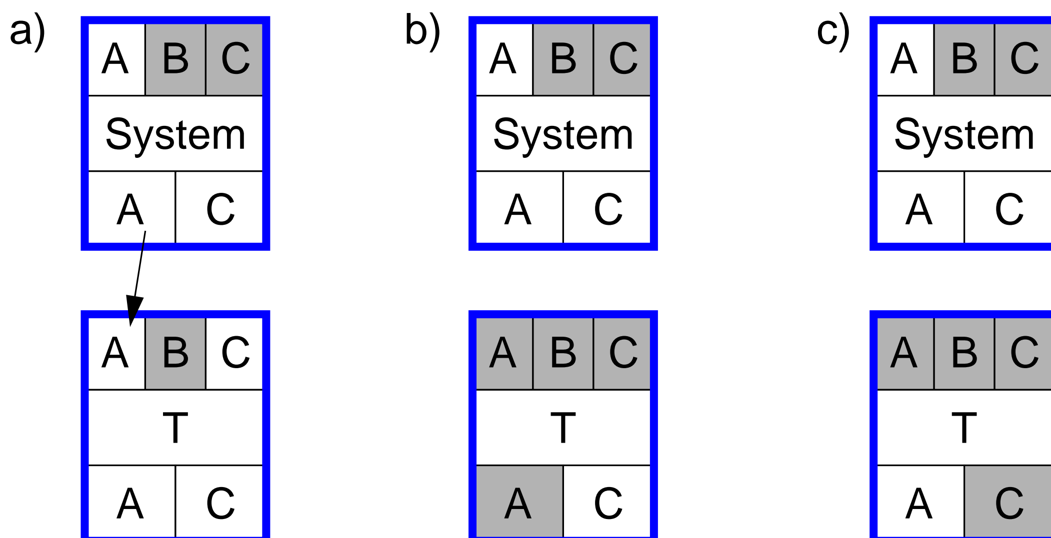
❑ Flache Transaktionen

➪ **System-TA**

- beschreibt Ablaufumgebung
- immer aktiv, solange System läuft
- führt kein Commit durch
- führt Abort nur als Ergebnis eines System-Crashs durch

➪ **flache TA**

- nur eine strukturelle Abhängigkeit:
  - wenn System-TA Abort durchführt, muss sie es auch



- Beobachtungen
  - beendete TA kann nicht mehr auf Events reagieren oder ihren Endzustand ändern
  - flache TA taugen nicht zu Modellierung komplexer Berechnungen; sie sind vollständig unabhängig von ihrer Umgebung (bis auf Abort durch System-Crash); können sich zu beliebigen Zeitpunkten zurücksetzen oder Commit durchführen

Beschreibung von Transaktionsmodellen (4)

- Regelsprache
  - Regelstruktur:

```
<rule identifier> „:"
  [<preconditions>] "→"
    [<rule modifier list>] ","
    [<signal list>] ","
    <state transition>
```

  - Regeln für:
    - Begin Work: $S_B(T)$
    - Rollback Work: $S_A(T)$
    - Commit Work: $S_C(T)$

S:	Signal
T:	UID der AA
Subskript:	<state transition>

- Regelsemantik
  - Regel, die einem Signal zugeordnet ist, wird aktiviert, wenn entsprechendes Event auftritt; sie wird aber erst ausgeführt, wenn `<preconditions>` erfüllt
  - `<preconditions>` ist einfacher Prädikatausdruck; oft Prädikate über den Zustand anderer TA, z. B. A(T) oder C(T)
  - Regelidentifikator spezifiziert zu welchem Signaleingang Pfeil (in der grafischen Darstellung) zeigt; `<preconditions>` unterscheiden, woher Signal kommt
  - `<signal list>` beschreibt, welche Signale bei der `<state transition>` generiert werden: Identifikatoren von Regeln, die durch das Signal aktiviert werden (in der grafischen Notation: Endpunkt des Pfeils)

Beschreibung von Transaktionsmodellen (5)

❑ Regelsemantik (Forts.)

➩ `<rule modifier list>` fügt zusätzliche Signale in (andere!) Regeln ein oder löscht sie (entsprechen Pfeilen in der grafischen Notation; delete blockiert eine Regel mit allen ihren Signalen)

```
<rule modifier> ::=
    „+“ „(„ <rule id> „|“ <signal> „)“
|   „-“ „(„ <rule id> „|“ <signal> „)“
|   „delete(„ <rule id> „)“
```

❑ Regelausführung

➩ wenn Event auftritt, identifizierte Regel überprüfen

➩ `<preconditions>` nicht erfüllt: Regel nur markieren, um anzuzeigen, dass Event aufgetreten

➩ andernfalls rechte Seite der Regeln ausführen (erzeugt i. allg. weitere Signale)

➩ alle Aktionen, die von einem Event ausgelöst werden, *atomar* ausführen, d. h., Kette abhängiger Events vollständig abwickeln, bevor unabhängiger (von aussen kommender) Event betrachtet wird

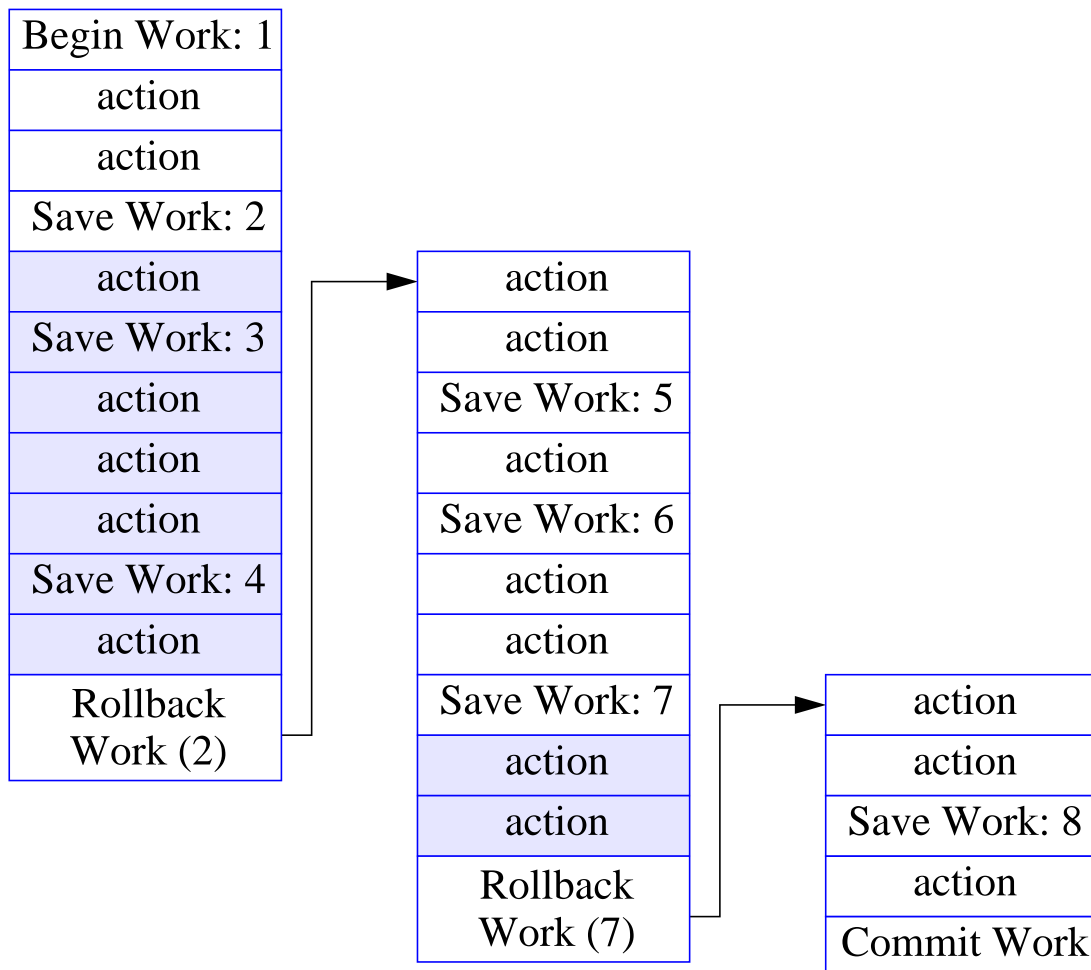
➩ wenn AA ihren Endzustand erreicht hat: alle Regeln löschen, alle ihre belegten Signaleingänge entfernen

Beschreibung von Transaktionsmodellen (6)

- Regeln für das Modell flacher Transaktionen
  - **$S_B(T)$: →**
    (+(S_A(system)|S_A(T)), delete(S_B(T))), , Begin Work
  - **$S_A(T)$: →**
    (delete(S_A(T)), delete(S_C(T))), , Rollback Work
  - **$S_C(T)$: →**
    (delete(S_A(T)), delete(S_C(T))), , Commit Work

Beschreibung von Transaktionsmodellen (7)

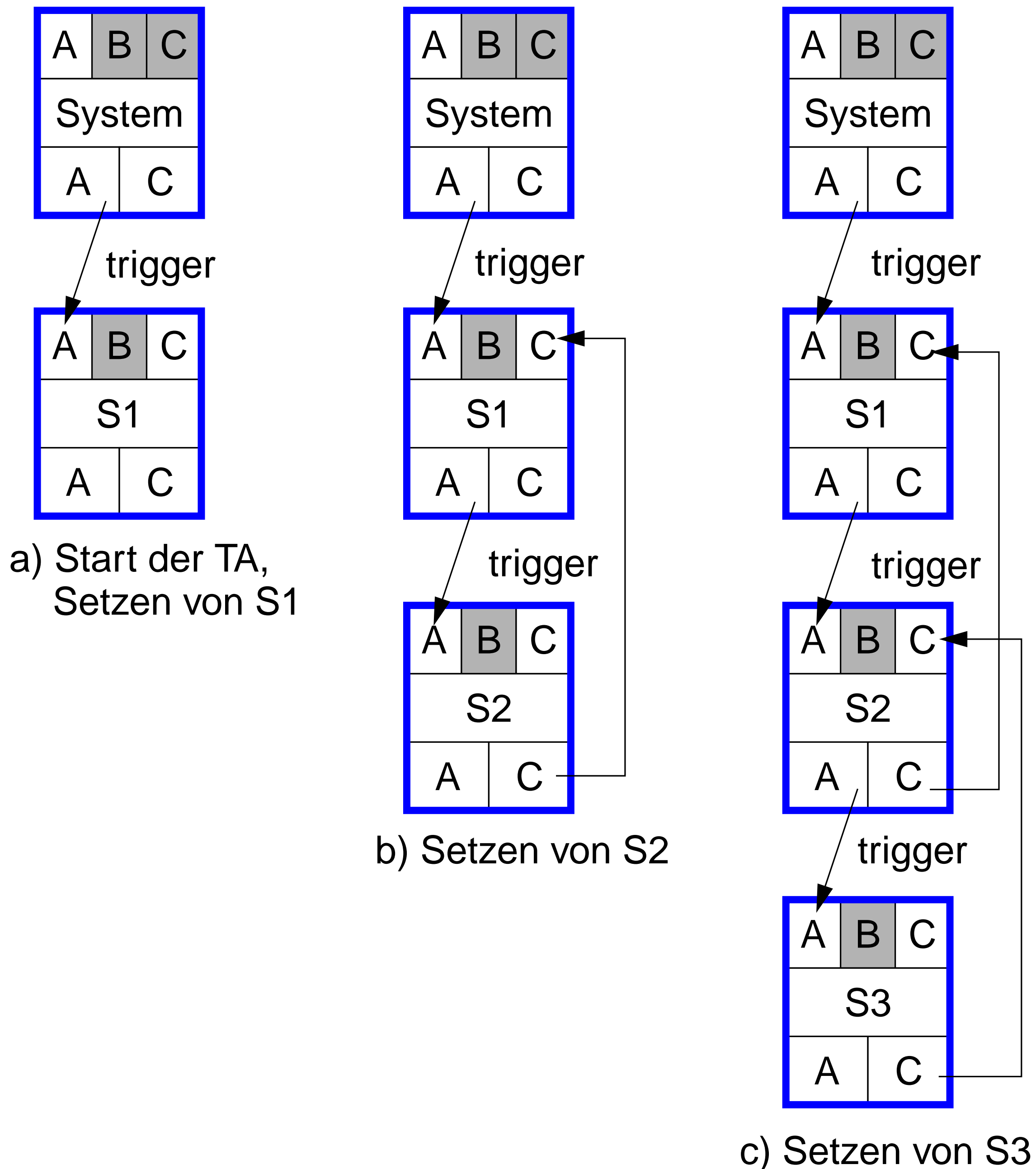
- Flache Transaktionen mit Sicherungspunkten
  - Sicherungspunkte innerhalb einer Transaktion
    - werden explizit durch Anwendungsprogramm gesetzt
    - modifiziertes Rollback benennt einen Sicherungspunkt



Beschreibung von Transaktionsmodellen (8)

❑ Flache Transaktionen mit Sicherungspunkten (Forts.)

➪ grafische Darstellung



Beschreibung von Transaktionsmodellen (9)

- Flache Transaktionen mit Sicherungspunkten (Forts.)
  - Modell
    - Folge von AA
    - verknüpft durch Folge von Commits (von der momentanen Position zurück zum TA-Beginn) und
    - verknüpft durch Folge von Aborts, ausgehend von System-TA als Folge eines Crashs
  - Regelmenge
    - S1 = UID der TA
    - delete wie oben, zur Vereinfachung weggelassen
    - **S_B(S1): $\rightarrow$ +(S_A(system)|S_A(S1)), , Begin Work**
S_A(S1): $\rightarrow$, , Rollback Work
S_C(S1): $\rightarrow$, , Commit Work
    - Save Work (i) erzeugt Signal S_B(Si) bei einer neu eingerichteten AA Si, die den Sicherungspunkt repräsentiert
    - Regelmenge der AA Si:
S_B(Si): $\rightarrow$ +(S_A(Si-1)|S_A(Si)), , Begin Work
S_A(Si): $\rightarrow$, , Rollback Work
S_C(Si): $\rightarrow$, S_C(Si-1), Commit Work
    - Rollback Work (i) erzeugt Signal S_A(Si)

Transaktionsmodelle (1)

- Beschränkungen flacher Transaktionen
  - auf *kurze* Transaktionen zugeschnitten, Probleme mit 'langlebigen' Aktivitäten
  - Alles-oder-Nichts-Eigenschaft oft inakzeptabel: hoher Arbeitsverlust
    - keine Binnen-Kontrollstruktur
    - keine Möglichkeit der Kapselung oder Zerlegung in Teilabläufe
    - keine abgestufte Kontrolle für Synchronisation und Recovery
  - Isolation
    - Leistungsprobleme durch lange Sperren
    - keine Unterstützung von Kooperation
  - keine Unterstützung von Parallelisierung
  - keine Benutzerkontrolle

- Anwendungsbeispiele
  - lange Batch-Vorgänge
    - Beispiel: Zinsberechnung
    - 'Alles-oder-Nichts' führt zu hohem Verlust an Arbeit
    - denkbar: Zerlegung in viele unabhängige TA - dann jedoch manuelle Recovery nach Systemfehler

Transaktionsmodelle (2)

❑ Anwendungsbeispiele (Forts.)

➪ Mehrschritt-Transaktionen, langlebige Aktivitäten

- Beispiel: mehrere Reservierungen
- lange Sperrdauer (Isolation) führt zu katastrophalem Leistungsverhalten (Konflikte, Deadlocks)
- Rücksetzen der gesamten Aktivität im Fehlerfall i. allg. nicht akzeptabel

➪ Entwurfsvorgänge (CAD, CASE, ...)

- lange Dauer (Wochen, Monate)
- kontrollierte Kooperation zwischen mehreren Entwerfern (*vor* Commit)
- Versionen

➪ Aktive DBS

- DBS reagiert eigenständig auf bestimmte Ereignisse
- Spezifikation durch ECA-Regeln

➪ Echtzeit-Anwendungen

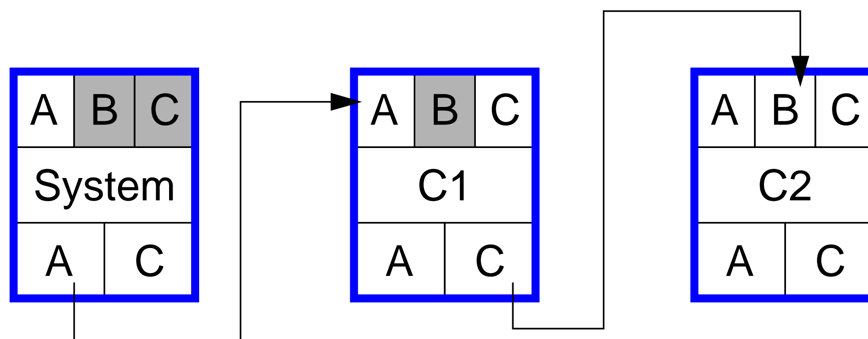
- zeitbezogene Konsistenzanforderungen (Deadlines)
- häufige irreversible Interaktionen mit der Außenwelt

➪ Föderative DBS / Multi-DBS

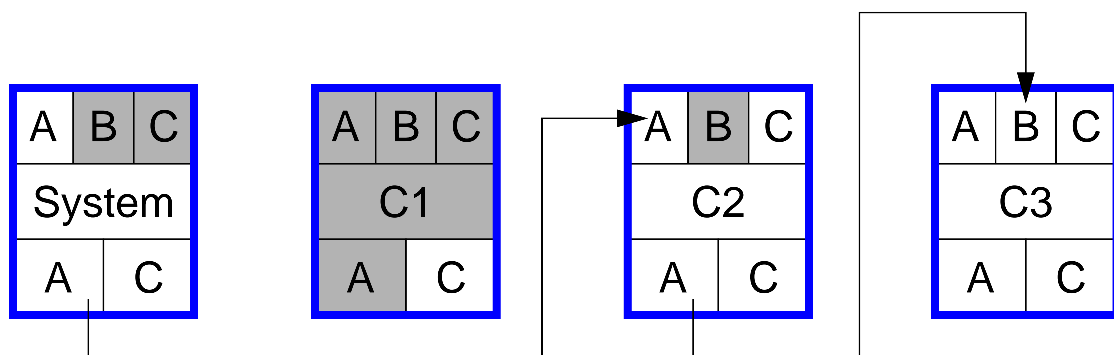
- Unterstützung lokaler Knotenautonomie
- unterschiedliche Synchronisations- und Recovery-Protokolle

Transaktionsmodelle (3)

- Gekettete Transaktionen
  - Variation der Anwendung von Sicherungspunkten
  - Konzept:
    - Folge von atomaren Aktionen (AA), die sequentiell ausgeführt werden
    - Teil der Commit-Verarbeitung: Signal generieren, das Begin Work bei der nächsten AA auslöst
    - Zustandsübergang atomar: Chain Work (Commit + Begin)
  - Graphische Darstellung:



- erste TA der Kette wurde gestartet; Start der zweiten später durch Commit der ersten ausgelöst



- erste TA der Kette hat Commit ausgeführt; zweite nun strukturell abhängig von „System“

Transaktionsmodelle (4)

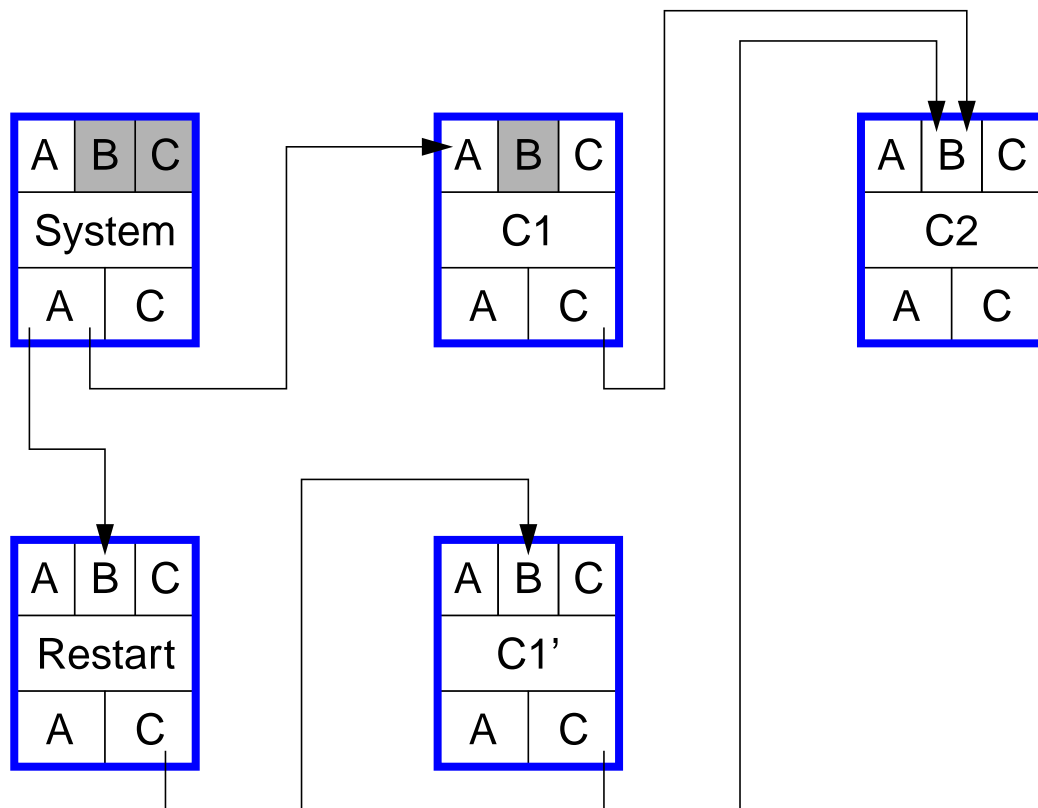
- Gekettete Transaktionen (Forts.)
  - Regelmenge
    - **$S_B(Cn) : \rightarrow +(S_A(system)|S_A(Cn)), ,$ Begin Work**
    - **$S_A(Cn) : \rightarrow , ,$ Rollback Work**
    - **$S_C(Cn) : \rightarrow , S_B(Cn+1),$ Commit Work**
  - Gekettete Transaktionen vs. Sicherungspunkte
    - Ablaufstruktur: gekettete TA erlauben wie Sicherungspunkte Substruktur, die langer Aktivität aufgeprägt werden kann (DB-Kontext bleibt erhalten)
    - Commit vs. Sicherungspunkt: Zurücksetzen nur möglich zum letzten SP (= Commit) - vorher: zu beliebigen SP
    - Sperrbehandlung: Commit erlaubt Freigabe der Sperren, die später nicht mehr benötigt werden
    - Verlust von Arbeit: Sicherungspunkte erlauben flexibleres Zurücksetzen nur, solange System normal arbeitet
    - Restart-Behandlung: bei geketteten TA wird Zusand des jüngsten Commit wiederhergestellt (Problem der Wiederherstellung des Programmzustands wie bei SP)

Transaktionsmodelle (5)

❑ Gekettete Transaktionen (Forts.)

➪ Restart

- Einführung einer Restart-Aktion
- wird aktiviert als Folge des Systemausfalls; übernimmt die Rolle von „System“ und startet C1’ (mit denselben Abhängigkeiten, die C1 hatte)



UNIVERSITÄT KAISERSLAUTERN

DBIS Datenbanken und Informationssysteme

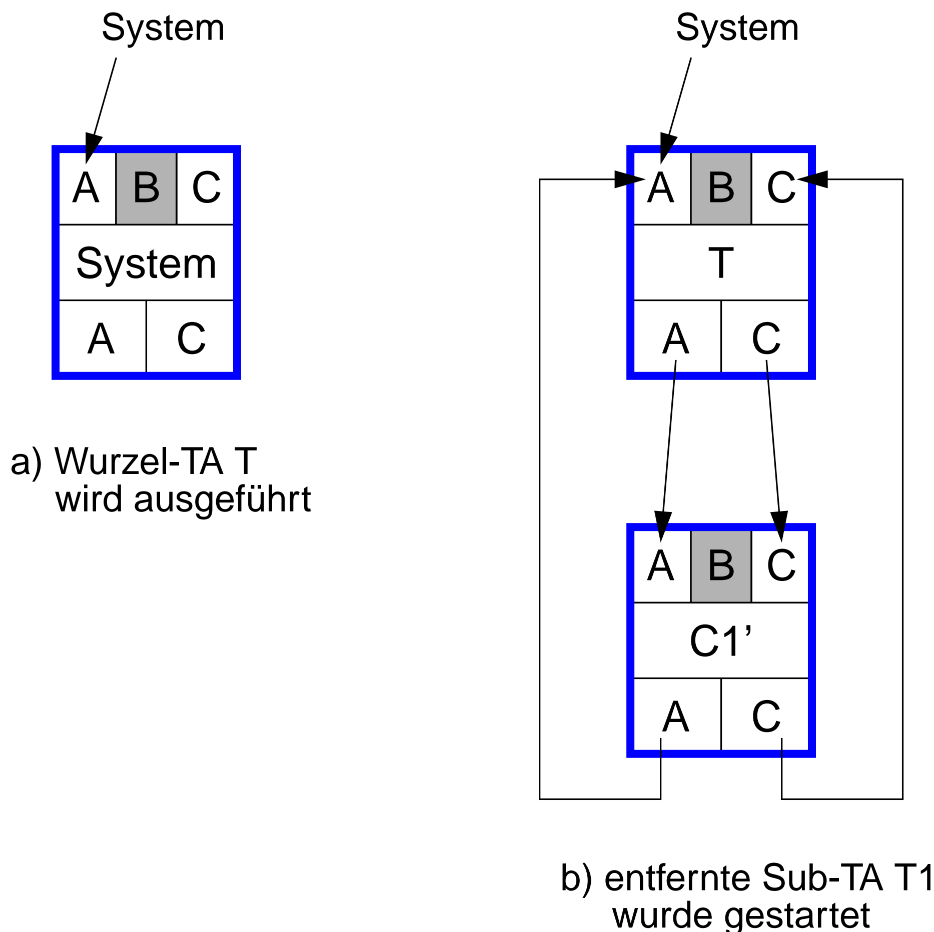
Transaktionsmodelle (6)

❑ Verteilte Transaktionen

➪ verteilte TA ist typischerweise flache TA,

- die in verteilter Umgebung abläuft und deshalb mehrere Knoten im Netz aufsucht
- deren Verteilung von den Daten abhängt
- die in „Scheiben" derselben Top-Level-TA aufgeteilt ist

➪ Graphische Darstellung:



a) Wurzel-TA T wird ausgeführt

b) entfernte Sub-TA T1 wurde gestartet

- Kopplung zwischen Sub-TA und ihrer übergeordneten TA in diesem Modell viel stärker als bei geschachtelten TA

Transaktionsmodelle (7)

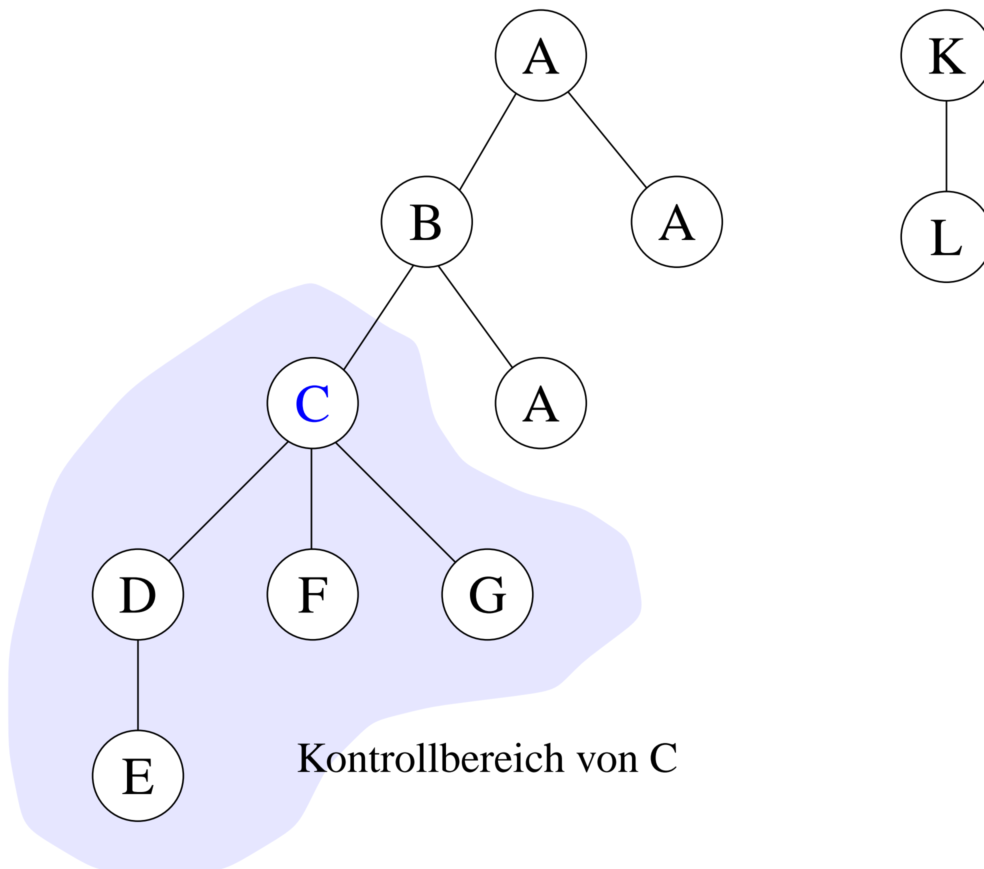
- Geschachtelte Transaktionen
  - Ziele
    - dynamische Zerlegung einer TA in eine Hierarchie von Sub-TA
    - Bewahrung der ACID-Eigenschaften für die (äußere) TA
    - Gewährleistung von Ununterbrechbarkeit und Isolation für jede Sub-TA
  - Zerlegung eines Vorgangs (unit of work) in Teilaufgaben; Verteilung und Bearbeitung in einem Rechensystem
  - Vorteile
    - Parallelverarbeitung innerhalb einer Transaktion
      - Ausnutzung anwendungsspezifischer Parallelität
      - Abbildung auf mehrere Prozessoren
    - feinere Recovery-Kontrolle innerhalb einer Transaktion
      - Rücksetzen einer Sub-TA betrifft nur sie und ihre Kinder
      - weitere Verfeinerung durch Sicherungspunkt-Konzept möglich
    - explizite Kontrollstruktur
      - einfachere Programmierung paralleler Abläufe
      - Alles-oder-Nichts-Eigenschaft von Sub-TA reduziert Komplexität

Transaktionsmodelle (8)

- Geschachtelte Transaktionen (Forts.)
  - Vorteile (Forts.)
    - Modularität des Gesamtsystems
      - einfache und sichere Zerlegung eines TA-Programms in Sub-TA
      - unabhängiger Entwurf / Implementierung von Moduln
      - Unterstützung von Kapselung und Fehlerisolation
    - verteilte Systemimplementierung
      - Einsatz verteilter Algorithmen
      - Erhöhung von Verfügbarkeit und Leistung
  - Basis-Konzept von Moss entwickelt (1981)
    - Transaktionsbaum veranschaulicht statische Aspekte der Aufrufhierarchie
    - ausgezeichnete Transaktion = Top-Level-Transaktion (TL) bildet äußersten Kontrollbereich

Transaktionsmodelle (9)

❑ Geschachtelte Transaktionen (Forts.)



➪ ACID-Prinzip gilt für TL-Transaktionen, da kein umhüllender Kontrollbereich mehr existiert

Transaktionsmodelle (10)

❑ Geschachtelte Transaktionen (Forts.)

➪ Dynamisches Verhalten

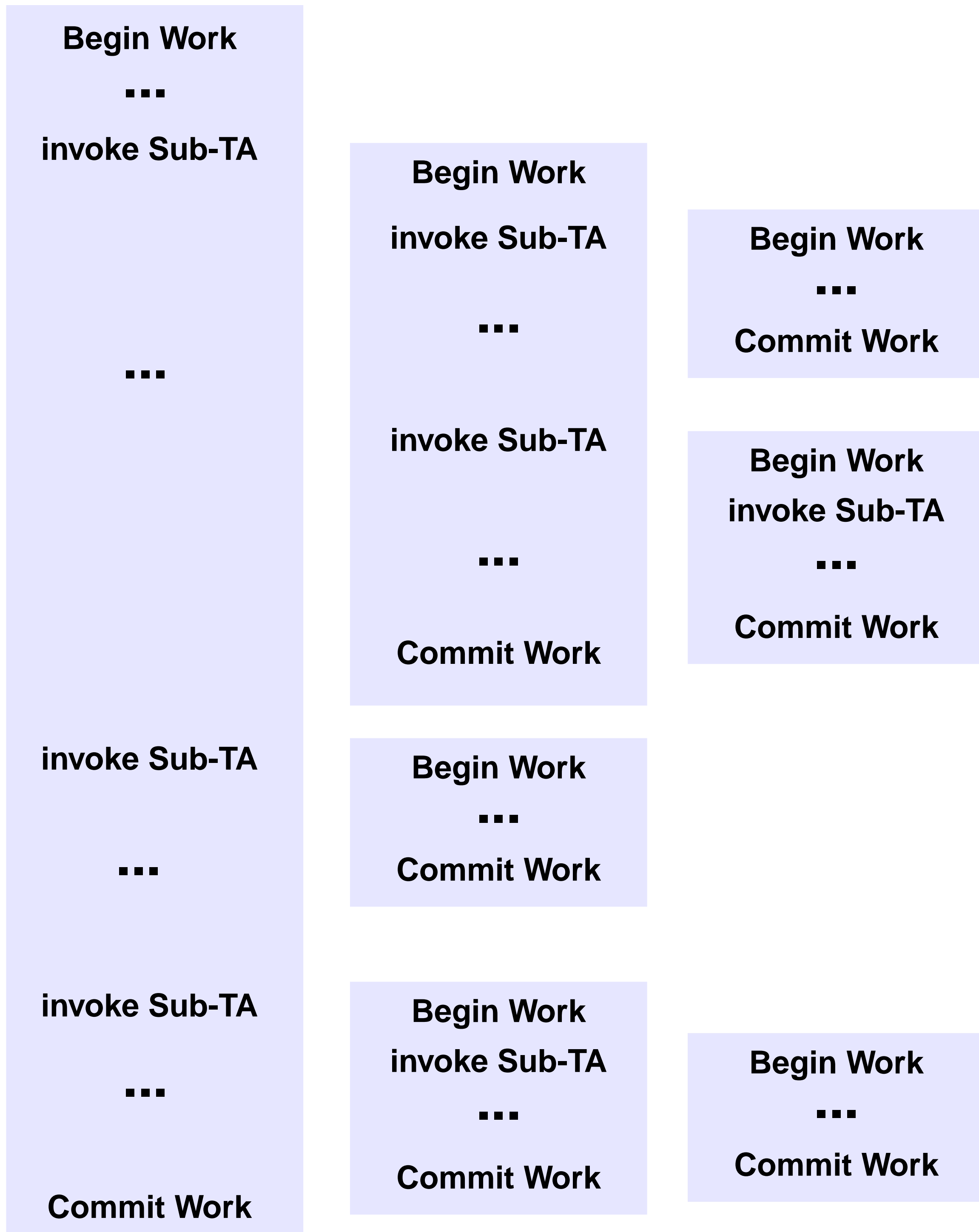
- Commit-Regel: (lokales) Commit einer Sub-TA macht ihre Ergebnisse nur der Erzeuger-TA zugänglich; endgültiges Commit einer Sub-TA dann und nur dann, wenn für alle Vorfahren bis hin zur TL-TA endgültiges Commit erfolgreich
- Rücksetz-Regel: wird (Sub-)TA auf irgendeiner Schachtelungsebene zurückgesetzt, werden alle ihre Sub-TA unabhängig von ihrem lokalen Commit-Status ebenso zurückgesetzt (rekursiv anwenden)
- Sichbarkeits-Regel: alle Änderungen einer Sub-TA werden bei ihrem Commit für ihre Erzeuger-TA sichtbar; alle Objekte, die Erzeuger-TA hält, können den Sub-TAs zugänglich gemacht werden; Änderungen einer Sub-TA sind für Geschwister-TA nicht sichtbar

➪ Generalisierung von Sicherungspunkten

➪ Geschachtelte TA sind das Äquivalent der Modularisierung auf der Ebene des Kontrollflusses

Transaktionsmodelle (11)

- Geschachtelte Transaktionen (Forts.)

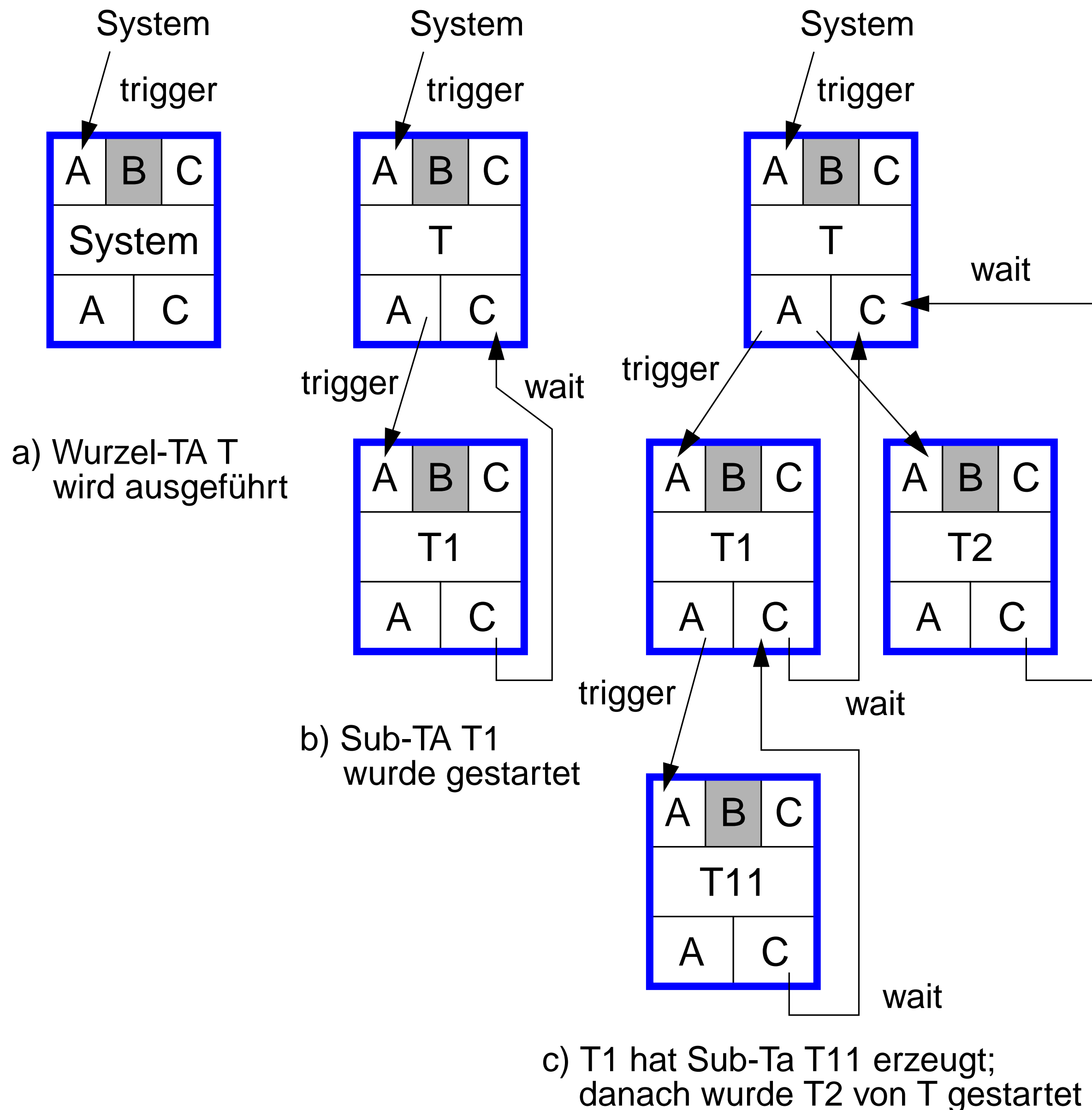


➪ jede Sub-TA ist eingebettet in die SoC der Erzeuger-TA; vollständige ACID-Eigenschaften nur für die TL-TA

Transaktionsmodelle (12)

❑ Geschachtelte Transaktionen (Forts.)

➩ Graphische Darstellung



- Abort-Signale von oben nach unten durchgereicht; Übergang in Commit-Zustand hängt davon ab, ob Erzeuger-TA ihn schon vollzogen hat

Transaktionsmodelle (13)

- Geschachtelte Transaktionen (Forts.)
  - Regelmenge
    für Sub-TA Tkn mit Erzeuger-Ta Tk:
    - **S_B(Tkn) : → +(S_A(Tk)| S_A(Tkn)), , Begin Work**
    - **S_A(Tkn) : → , , Rollback Work**
    - **S_C(Tkn) : C(Tk) → , , Commit Work**
  - neu: Bedingung für Commit
  - in der graphischen Darstellung Pfeil zurück von Commit-Zustand einer Sub-TA zum Commit-Zustand ihrer Erzeuger-TA (mit „wait“ markiert)

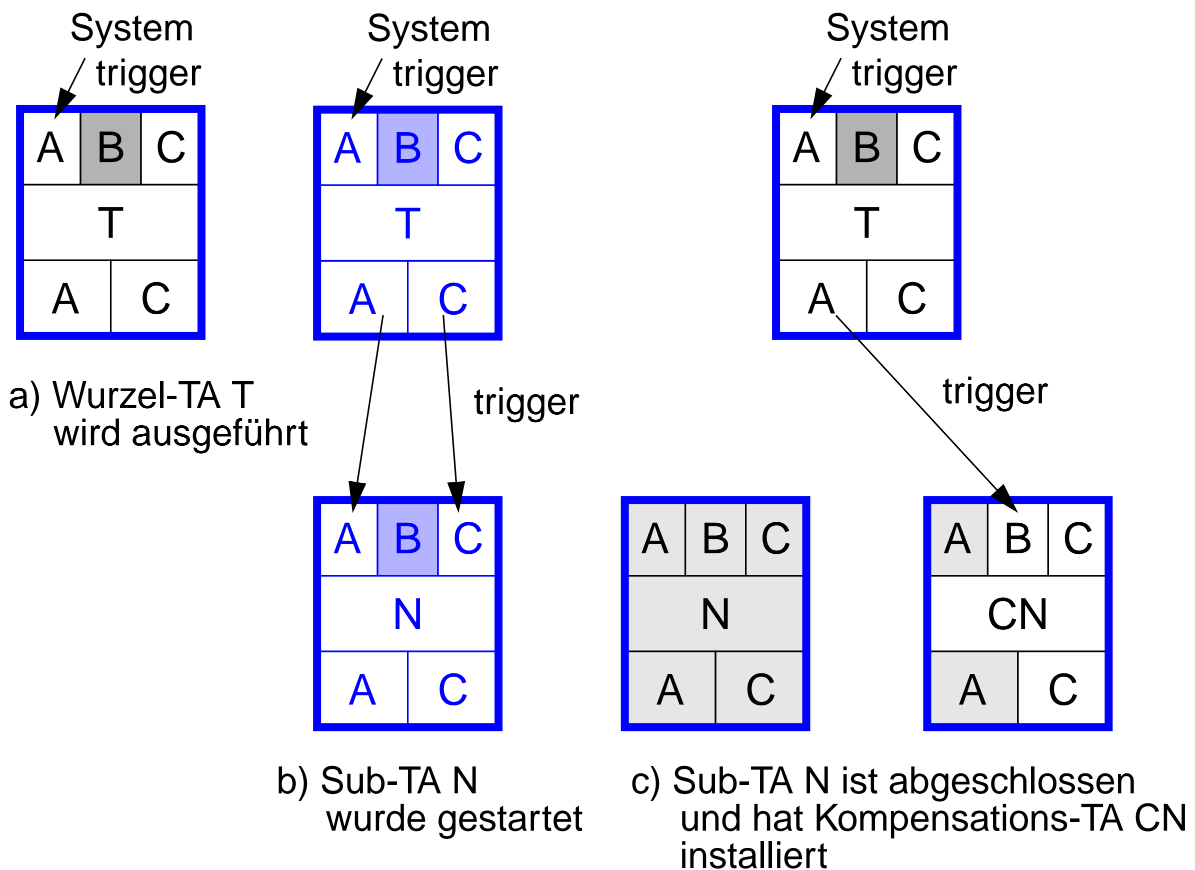
Transaktionsmodelle (14)

❑ Mehrebenen-Transaktionen

➩ Verallgemeinerung geschachtelter Transaktionen

➩ Sub-TA dürfen vor Ende der Erzeuger-TA Commit ausführen ('pre-commit')

- Rollback also nicht mehr möglich
- aber: **Kompensation**
  - Wirkung der Sub-TA rückgängig machen, wenn Erzeuger-TA scheitert
  - Kompensations-TA selbst wieder geschachtelte oder Mehrebenen-TA



Transaktionsmodelle (15)

- Mehrebenen-Transaktionen (Forts.)
  - Kompensations-TA
    - installieren, wenn Sub-TA Commit ausführt
    - aufbewahren, solange Erzeuger-TA noch läuft
    - nach Abschluss der Erzeuger-TA wegwerfen
    - wichtig: Kompensations-TA muss Commit erreichen!
  - Regeln für Sub-TA N:
    - **$S_B(N) : \rightarrow (+(S_A(T) \mid S_A(N)), +(S_C(T) \mid S_C(N)))$, , Begin Work**
    - **$S_A(N) : \rightarrow$, , Rollback Work**
    - **$S_C(N) : \rightarrow (-(S_A(T) \mid S_A(N)), +(S_A(T) \mid S_B(CN)))$, , Commit Work**
  - Regeln für Kompensations-TA CN:
    - Kompensations-TA im Falle des Scheiterns neu starten als CN' (vgl. gekettete TA)
    - **$S_B(CN) : \rightarrow +(S_C(restart) \mid S_B(CN'))$, , Begin Work**
    - **$S_A(CN) : \rightarrow$, $S_B(CN')$, Rollback Work**
    - **$S_C(CN) : \rightarrow delete(S_B(CN'))$, , Commit Work**

Transaktionsmodelle (16)

- Mehrebenen-Transaktionen (Forts.)
  - Nutzung

T

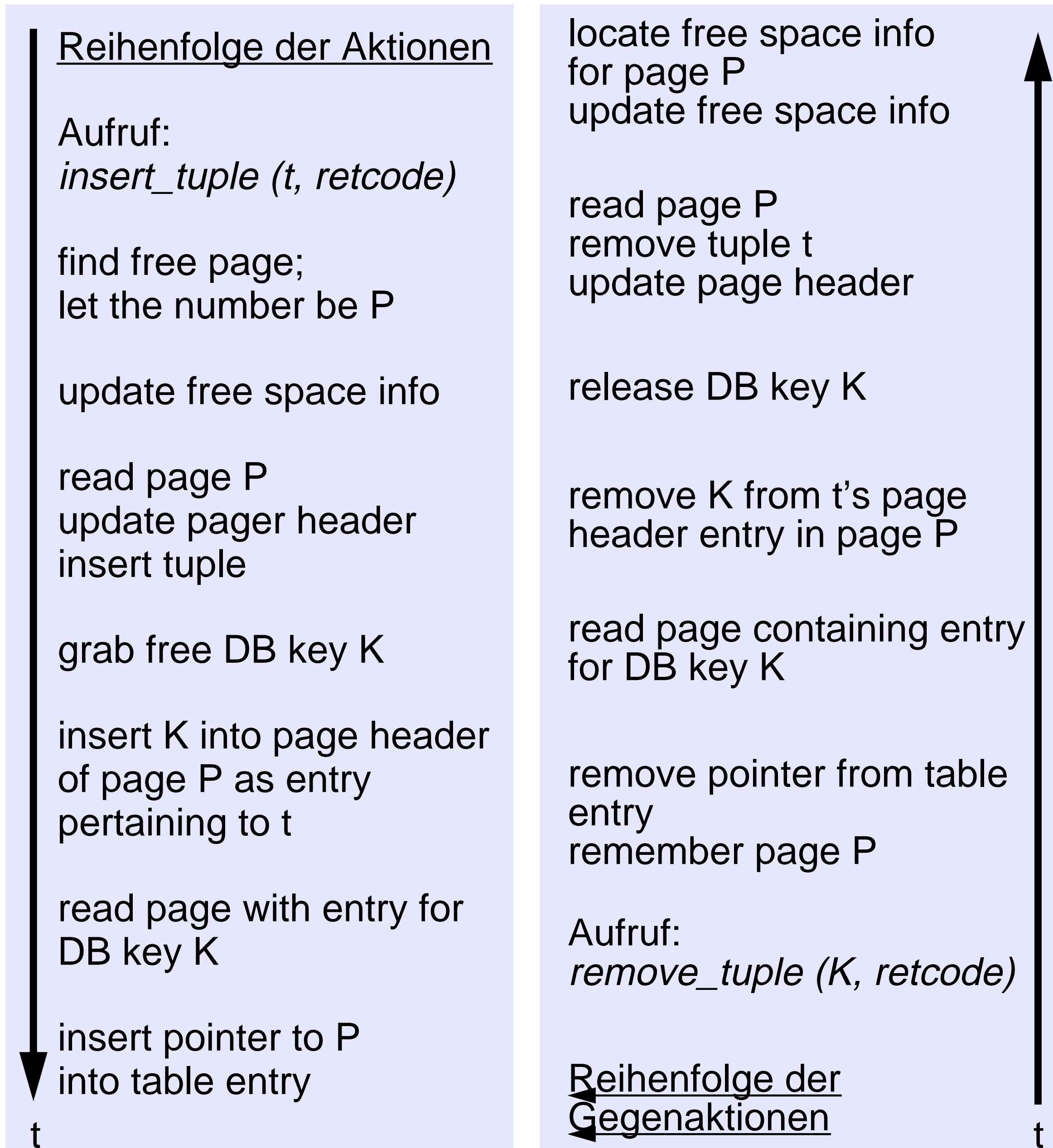
SELECT ...	INSERT ...	UPDATE ...	SELECT ...
	insert tuple	insert B-tree entry	
update page	insert address table entry	locate position	insert entry
			split page

- SQL-Anweisungen können als Sub-TA aufgefasst werden
- ebenso die internen Modulaufrufe
- bei geschachtelten TA: Sperren auf Seiten, Adresstabellen, B-Baum-Seiten (!) bleiben erhalten (bei der Erzeuger-TA)
- bei Mehrebenen-TA: Sperren werden freigegeben!
  z.B. Seite für andere Sub-TA zugänglich;
  Kompensation: Tupel wieder aus Seite löschen

Transaktionsmodelle (17)

❑ Mehrebenen-Transaktionen (Forts.)

➪ Nutzung (Forts.)



- Entfernung des Tupels funktioniert auch, wenn Seite inzwischen von anderen TAs geändert wurde

Transaktionsmodelle (18)

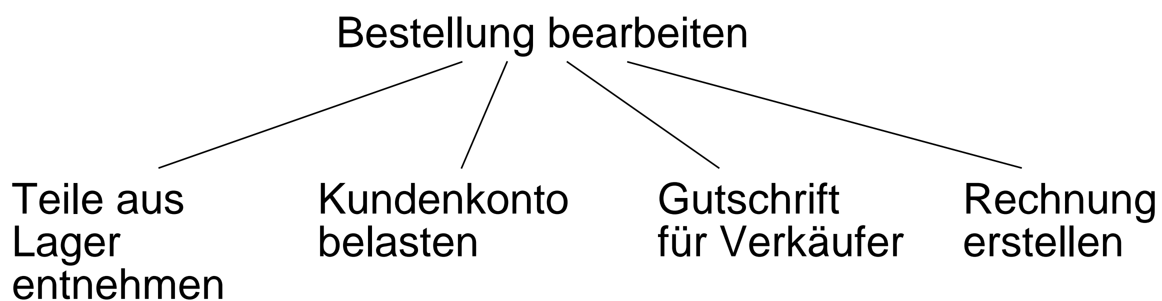
- ❑ Mehrebenen-Transaktionen (Forts.)
  - ➩ Nutzung (Forts.)
    - Isolation?
      - verhindern, dass andere Sub-TA Tupel aus der Seite löscht, bevor einfügende TA Commit ausgeführt hat
      - entscheidend: auf der höheren Ebene (Ebene der Tupel) ist zu erkennen, dass einfügende TA noch arbeitet
      - Mehrebenen-TA benötigen eine bestimmte Struktur der Objekte, mit denen TA arbeiten
    - Hierarchie von ADTs
      - Abstraktionshierarchie: gesamtes System besteht aus strikter Hierarchie von Objekten mit ihren Operationen
      - Schichtenbildung: Objekte der Schicht n implementiert unter Verwendung von Operationen der Schicht n-1
      - Disziplin: Schicht n darf nur auf Objekte der Schicht n-1 zugreifen
    - vorgenannte Anforderungen im Fall von DBVS erfüllt:
      Tupel - Sätze - Seiten - Blöcke
    - Schachtelung der TA
      - kann auf beliebige Schichten (beliebige Operationen) verallgemeinert werden
      - theoretisch fundierter Ansatz

Transaktionsmodelle (19)

- Mehrebenen-Transaktionen (Forts.)
  - Nutzung (Forts.)
    - Transaktionsverwaltung in jeder Schicht
      - Ausnutzung von Anwendungssemantik zur Synchronisation möglich
      - Wahl unterschiedlicher Synchronisationstechniken pro Schicht möglich
      - Nutzung der Basisdienste zur TA-Verwaltung des BS
      - potentiell hoher Aufwand zur TA-Verwaltung, insbesondere für Logging und Recovery (Protokollierung in mehreren Schichten)
    - reduzierte Konfliktgefahr zwischen TA unter Wahrung der Serialisierbarkeit
    - vorzeitiges Commit (Freigabe von Änderungen/Sperren), aber Schutzschirm auf höherer Ebene bleibt erhalten (z. B. S2PL für TL-TA)
    - für Gesamt-TA gelten weiterhin ACID-Eigenschaften

Transaktionsmodelle (20)

- Offen geschachtelte Transaktionen
  - ‘Anarchische’ Variante der Mehrebenen-TA
    - keine Restriktionen bzgl. semantischer Beziehung zwischen Sub-TA und Erzeuger-TA
    - insbesondere auch keine Objekthierarchie
  - Beispiel



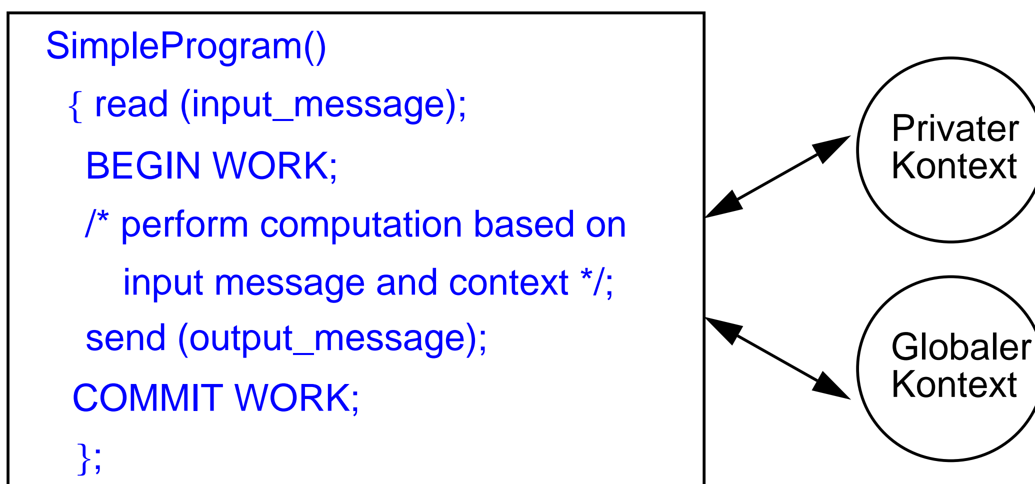
    - Sub-TA arbeiten of auf verschiedenen Datenbeständen
  - Prinzipien
    - Sperren werden vor Beendigung der TL-TA freigegeben
    - für jede ändernde Sub-TA T wird Kompensations-TA CT bereitgestellt
    - im Fehlerfall: Ausführung von CT (kein UNDO(T))
    - keine Serialisierbarkeit

Transaktionsmodelle (21)

- Langlebige Transaktionen
  - Verarbeitung langer Batch-Anwendungen
  - Einsatz von flachen Transaktionen ?
    - kontextfreie Verarbeitung: output_msg = f (input_msg)
    - "Exactly once"-Semantik wird eingehalten
    - Kosten im Restart-Fall
    - Sicherungspunkt oder geschachtelte TA helfen bei Crash nicht
  - Zerlegung in Mini-Batches
    - Nutzung von Verarbeitungskontexten
    - Hintereinander-Ausführung der TA-Folge unter Beibehaltung des TA-Verarbeitungskontextes
    - output_msg = f (input_msg, context)
  - Kontextinformationen
    - *Transaktion:* Cursorpositionen, . . .
    - *Programm:* letzte TA, die erfolgreich Commit durchgeführt hat, . . .
    - *Terminal:* Liste der Funktionen, die aufgerufen werden können; letztes ausgegebenes Fenster; Liste der Benutzer, die das Terminal benutzen dürfen, . . .
    - *Benutzer:* letzter Auftrag, den der Benutzer bearbeitet hat; nächstes zu benutzendes Paßwort, . . .

Transaktionsmodelle (22)

- Langlebige Transaktionen
  - Einsatz von Verarbeitungskontexten
    - Die Ausführung eines kontext-sensitiven TA-Programms beruht auf
      - den **Parametern in der Eingabe-Nachricht**
      - **und Zustandsinformationen als Kontext**



  - Ergebnis:
    - eine **Ausgabenachricht**
    - und ein **geänderter Kontext**
  - Eigenschaften langlebiger Transaktionen
    - Minimierung der verlorengegangenen Arbeit im Fehlerfall
    - wiederherstellbarer Verarbeitungszustand
    - expliziter Kontrollfluß
    - Einhaltung der ACID-Eigenschaften

Transaktionsmodelle (23)

- Langlebige Transaktionen (Forts.)
  - Beispiel: Zinsberechnung

```
ComputeInterest (interest_rate) {

#include <string.h>
#include <sqlca.h>
#define max_account_no 999999

exec sql begin declare section;
  long last_account_done;
  double interest_rate;
  int logsize;
exec sql end declare section;

exec sql define stepsize 1000;

/* Annahme: Kontonr. 1 bis 1000000,
Relation batchcontext enthält id des
letzten Mini-Batches */

logsize = 0;
exec sql select count (*) into :logsize
         from batchcontext;

if (sqlca.sqlcode != 0 | | logsize = = 0)   {
/* batchcontext entweder nicht vorhanden oder
leer → Anfang der Kette */
  exec sql begin work;
  exec sql drop table batchcontext;
  exec sql create table batchcontext
      (last_account_done integer);
  last_account_done = 0;
  exec sql insert into batchcontext
      values (: last_account_done);
  exec sql commit work;
}
```

Transaktionsmodelle (24)

- Langlebige Transaktionen (Forts.)
  - Beispiel: Zinsberechnung (Forts.)

```
else    {
  /* Restart */
  exec sql select last_account_done
    into :last_account_done
    from batchcontext;
}
while (last_account_done < max_account_no)  {
  /* ein Mini-Batch: */
  exec sql begin work;
  exec sql update accounts
    set account_total = account_total
        * (1 + :interest_rate)
    where account_no between
      : last_account_done + 1 and
      : last_account_done + :stepsize;
  exec sql update batchcontext
    set last_account_done =
    last_account_done + :stepsize;
  exec sql commit work;
  last_account_done =
    last_account_done + stepsize;
}
/* letzter Mini-Batch ausgeführt */
exec sql begin work;
exec sql drop table batchcontext;
exec sql commit work;
return;
}
```

Transaktionsmodelle (25)

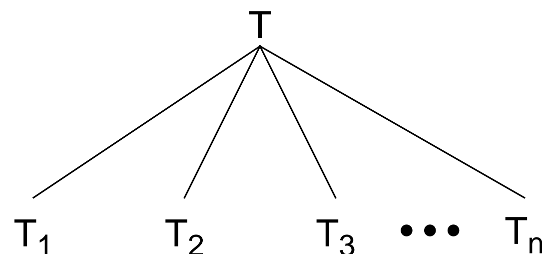
❑ SAGAS

➪ Linderung der LLT-Probleme bei speziellen Anwendungen

- vorzeitige Freigabe von Ressourcen
- LLT nicht mehr atomar - jedoch keine Preisgabe der DB-Konsistenz
- Koordinierte Fehlerbehandlung bzw. Rücksetzung erforderlich

➪ spezielle Art von zweistufigen geschachtelten TA

- **Saga** ≡ LLT, die in eine Sammlung von Subtransaktionen aufgeteilt werden kann



➪ Aspekte der Ablaufsteuerung und -kontrolle

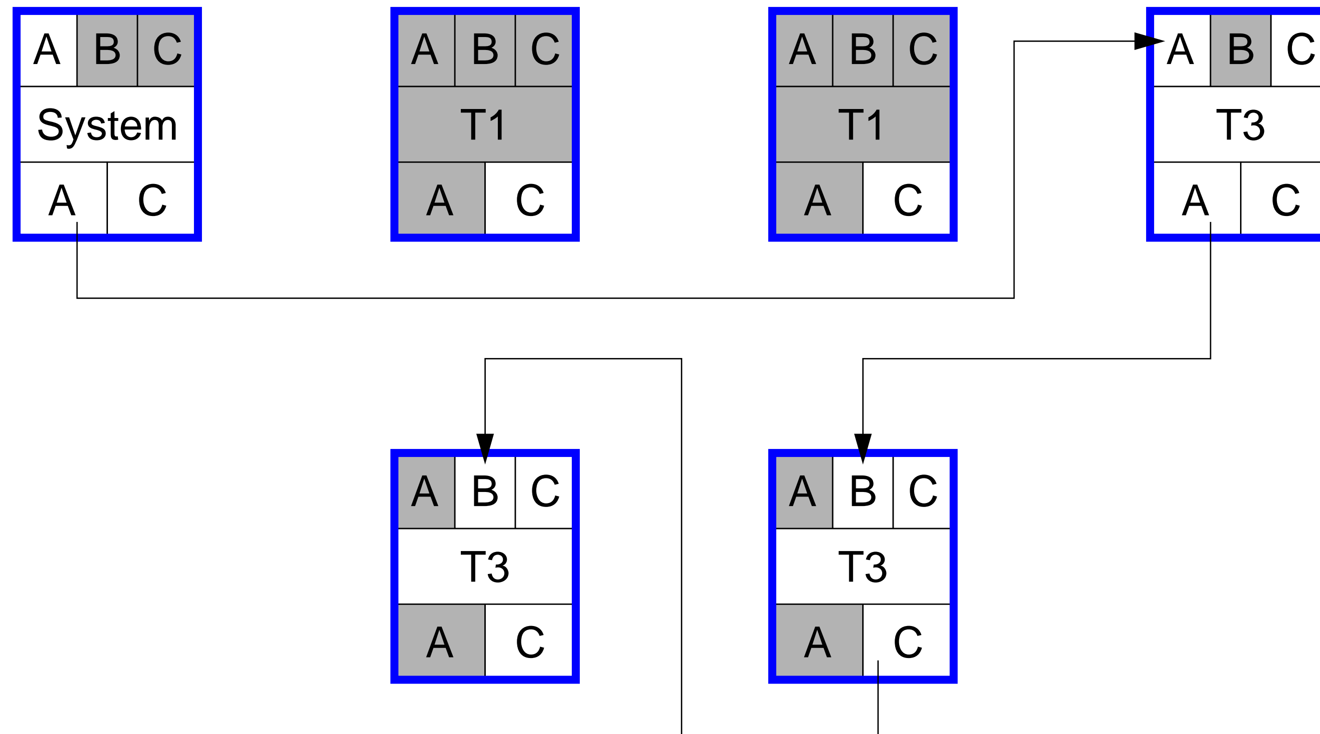
- T_i geben alle Ressourcen (z. B. Sperren) frei
- expliziter Kontrollfluß zwischen den T_i (Sequenz) im Anwendungsprogramm
- Synchronisation verlangt Serialisierbarkeit der T_i, jedoch nicht von T
- Konfliktbehandlung durch Warten oder Abbruch von T_i oder von T
- Fehlerbehandlung von T_i durch UNDO(T_i) und Kompensation $CT_{i-1},...,CT_1$

Transaktionsmodelle (26)

- SAGAS (Forts.)
  - ➩ Es muß für Kompensation gesorgt werden !
    - alle T_i gehören zusammen
    - Bereitstellung von Kompensationstransaktionen CT_i für jede T_i
    - keine teilweise Ausführung von T
  - ➩ Eine Saga ist eine Menge flacher Transaktionen T_1, T_2, ..., T_n, die im einfachsten Fall sequentiell verarbeitet wird
  - ➩ Zusicherung des DBS
    - Das Endergebnis einer Saga ist entweder die Ausführungsfolge T_1, T_2, ..., T_i, ..., T_{n-1}, T_n
    - oder bei einem Fehler im Schritt j
T_1, T_2, ..., T_j (abort), CT_{j-1}, ..., CT_2, CT_1
    - DBS garantiert LIFO-Ausführung der Kompensationen im Fehlerfall
  - ➩ T_j, $CT_j \neq UNDO(T_j)$ im allgemeinen Fall
    - für CT_j müssen Ressourcen wieder angefordert werden
    - Deadlock-Gefahr

Transaktionsmodelle (27)

- SAGAS (Forts.)
  - **Graphische Darstellung**



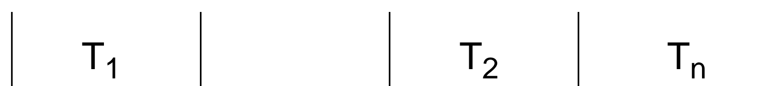
Transaktionsmodelle (28)

❑ SAGAS (Forts.)

➩ Struktur

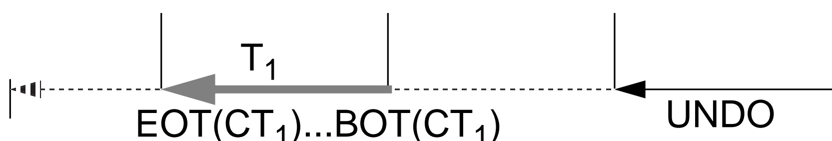
- BEGIN-SAGA, ABORT-SAGA, END-SAGA, BOT, ABORT, EOT wie bei flachen TA
- Ablauf

a) BS ... BOT(T_1) ... EOT(T_1) ... BOT(T_2) ... EOT(T_2) ... EOT(T_n) ... ES



b) BS ... BOT(T_1) ... EOT(T_1) ... BOT(T_2) ... ABORT

c) BS ... BOT(T_1) ... EOT(T_1) ... BOT(T_2) ... ABORT-SAGA



d) Unterbrechung durch Systemfehler: wie Fall c

➩ Zusammenfassung:

- ACID für jede Sub-TA T_i
  (D kann durch Kompensation aufgehoben werden)
- CD für umfassende TA T

Transaktionsmodelle (29)

- SAGAS (Forts.)
  - Verfeinerung: Reduktion des Aufwandes beim Scheitern einer Saga oder bei Systemfehler durch Nutzung von Savepoints
    - Sicherung des AP-Zustandes
    - Übergabe der Savepoint-ID an ABORT-SAGA
    - Partielles Rücksetzen möglich
      Scenario:
      - Savepoint nach T_1 und T_3
      - Crash nach T_2 und T_5
      - Ablauf:
        BS, T_1, T_2, CT_2, T_2, T_3, T_4, T_5, CT_5, CT_4, T_4, T_5, T_6, ES
    - C_i ist Anwendungsprogramm (vordef. TA)
      - Speicherung in DB vorteilhaft
      - keine unkontrollierten Programme
      - aktuelle Parameter für CT_i aus DB
      - automatische Recovery

Zusammenfassung

- Vergleich der Transaktionsmodelle

	geschachtelte TA	Mehrebenen-TA offen geschacht. mit Disziplin	offene geschachtelte TA	Batch-TA (langlebige TA)	Sagas	Entwurfs-TA (Ausblick)
expliziter Kontrollfluß (außerhalb TA)	-	-	-	einfache Sequenz	Verkettung, einfache Sequenz	-
frühzeitige Freigabe von Änderungen	-	nur relativ zur selben Schicht	ja	frühestens am Ende einer Teil-TA	frühestens am Ende einer Teil-TA	im Kooperationsmodus
stabile Ergebnisse nach Systemausfall	-	-	für die TA, die unabhängig Commit gemacht haben	für die schon beendete Teil-TA	-	nur wenn Objektversion in Gruppen-DB eingebracht
Begrenzung der dynamischen Rücksetzung (Rollback)	ja	ja	ja	-	-	-
Korrektheit	Serialisierbarkeit	Serialisierbarkeit	-	Serialisierbarkeit der Einzel-TA	Serialisierbarkeit der Einzel-TA	Kooperation auf vorläufigen Objekten
Kontext-Verwaltung für zusammenhängende Abläufe	-	-	-	teilweise	-	-
explizte Konflikt-behandlung	-	-	-	-	-	Grant-/Return-Protokoll