Verteilte Systeme

R. Kaiser, R. Kröger, O. Hahm

(HTTP: http://www.cs.hs-rm.de/~kaiser
E-Mail: robert.kaiser@hs-rm.de)
Kai Beckmann
Sebastian Flothow

Sommersemester 2022

8. Verteilte Transaktionen

http://statuesquo.blogspot.fr/2017/07/kreislauf-des-geldes-circuit-de-largent.html

Inhalt

Hochschule RheinMain

8. Verteilte Transaktionen
- 8.1 Einführung
- 8.2 Transaktionskonzept
- 8.3 Stellenlokale Commit-Verfahren
- 8.4 Das 2-Phasen Commit-Protokoll
- 8.5 X/Open Distributed Transaction Processing

Einführung

Hochschule RheinMain

Neues Problem bei der Entwicklung verteilter Anwendungen

- Ausfall von einzelnen Komponenten des verteilten Systems (partial failure property)

$\Rightarrow$ komplexe Fehlersituationen in verteilten Anwendungsprogrammen

Motivation für Transaktionen

- *Atomare Aktionen* als Erweiterung des DB-Tansaktionskonzepts zu allgemeinem Programmierkonstrukt
- Reduktion der Komplexität der Anwendungsprogrammierung in Gegenwart von Fehlern und Nebenläufigkeit
- Bessere Einsicht in die Wirkungsweise eines Programms
- Automatisches Backward Error Recovery, Kombination mit Forward Error Recovery-Verfahren möglich

Transaktionskonzept

Eine *Transaktion* ist die Zusammenfassung einer Menge von Aktionen (Operationen auf log. Betriebsmitteln) mit folgenden Eigenschaften (ACID) (Härder/Reuter, 1983):

- *Atomicity = Atomarität im Fehlerfall (Alles-oder-Nichts)*:
  - Transaktion entweder vollständig ausgeführt oder erscheint, als ob nie begonnen
  - Kein Zwischenzustand zwischen Anfangs- und Endzustand sichtbar
- *Consistency = Konsistenz*:
  - Eine Transaktion überführt das System von einem konsistenten Zustand in einen konsistenten Zustand
  - Allgemein: Erfüllung aller Anwendungs-Constraints
- *Isolation*:
  - Verhalten jeder Transaktion als ob „allein auf der Welt“
- *Durability = Dauerhaftigkeit (Permanenz der Wirkung)*:
  - Wirkung abgeschlossener Transaktionen geht nicht verloren (selbst bei Auftreten aller erlaubten Fehler eines Fehlermodells)

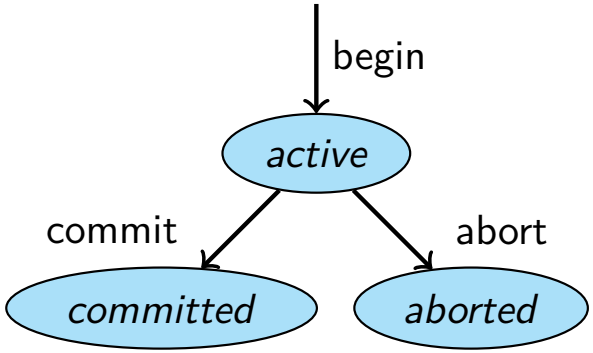
Lokale/Verteilte Transaktionen

Hochschule RheinMain

Lokale Transaktion

- Wirkung auf einzelnes Rechensystem beschränkt

Zustandsdiagramm



Verteilte Transaktion

- Wirkung auf mehreren Stellen eines verteilten Rechensystems

?

im Folgenden zu entwickeln

Fehlermodell

Das Fehlermodell beschreibt alle vorhergesehenen Fehler, auf die ein System geordnet reagiert, alle anderen heiSSen Katastrophen.

Transaktionsfehler (transaction abort):

- Zurücksetzen einzelner Transaktionen, z.B. durch
  - Expliziter Abbruch durch Benutzer
  - Fehler im Applikationsprogramm
  - als Folge einer Deadlockauflösung

Stellenfehler (site failure):

- Ausfall eines beteiligten Rechensystems, z.B. durch
  - Transiente oder permanente Hardware-Fehler (auch Stromausfall)
  - Betriebssystemabsturz mit reboot
- Alle laufenden Prozesse stürzen ab
- Alle Transaktionen im Zustand *active* gehen in den Zustand *aborted* über

Fehlermodell (2)

Hochschule RheinMain

Medienfehler:

- Nicht-behebbarer Fehler auf nicht-flüchtigem Speichermedium, das von Transaktionsverarbeitung genutzt wird, z.B.
  - Magnetplatte (genutzt für Speicherung der log. Daten)
  - Magnetband (genutzt für Logging)
- Standardbehandlung durch stabilen Speicher (redundante Speicherung auf mehreren Medien, z.B. Spiegelplatten, RAID, ...)
- hier nicht weiter betrachtet

Kommunikationsfehler:

- Fehler im Nachrichtensystem führen zum Verlust von Nachrichten
- Partitionierung: Zerfall des Netzes in mehrere isolierte Teilnetze

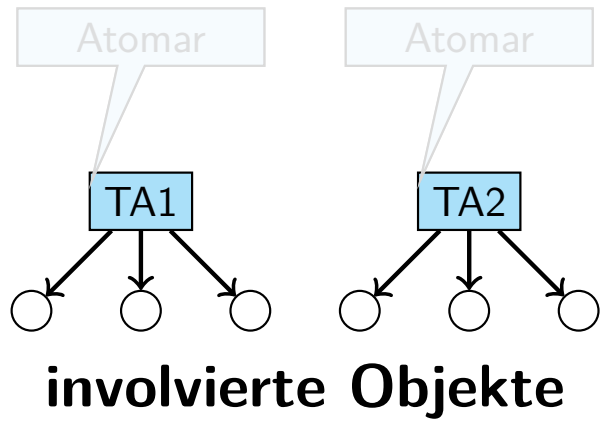
Flache / geschachtelte Transaktionen

Hochschule RheinMain

Flache Transaktionen

- klassisches Modell aus DB-Kontext
- Transaktion involviert Menge von Objekten (log. Betriebsmitteln)
- Transaktionen können Objekte gemeinsam nutzen (geregelt durch Concurrency Control- Mechanismen)
- Transaktionen können nicht geschachtelt werden

⇒ Nachteil: Keine Möglichkeit, Teilergebnisse festzuschreiben

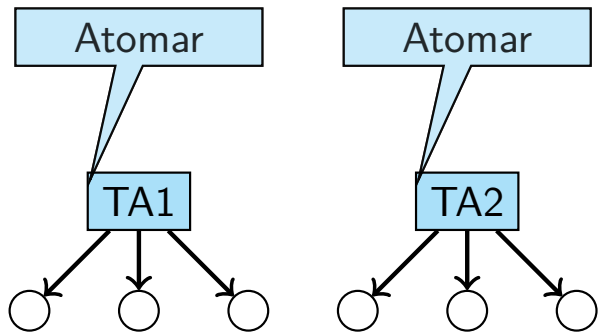


Flache / geschachtelte Transaktionen

Hochschule RheinMain

Flache Transaktionen

- klassisches Modell aus DB-Kontext
- Transaktion involviert Menge von Objekten (log. Betriebsmitteln)
- Transaktionen können Objekte gemeinsam nutzen (geregelt durch Concurrency Control- Mechanismen)
- Transaktionen können nicht geschachtelt werden



involvierte Objekte

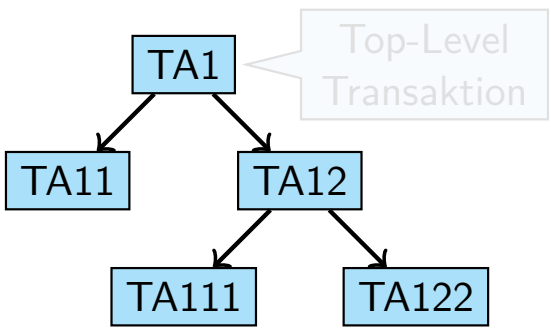
⇒ Nachteil: Keine Möglichkeit, Teilergebnisse festzuschreiben

Flache / geschachtelte Transaktionen (2)

Hochschule RheinMain

Geschachtelte Transaktionen (Nested Transactions)

- Eine Transaktion kann (auch zu einem Zeitpunkt) innere Transaktionen (Subtransaktionen) besitzen
- isolierte Zurücksetzbarkeit innerer Transaktionen: Abort einer inneren TA führt zu Exception (nicht: Abbruch!) bei übergeordneter TA
- Abort einer TA führt zum Abort aller inneren TAen
- Commitment relativ zum Parent (endgültig erst mit Commitment der Top-Level TA)
- Nebenläufigkeitskontrolle auf jeder Schachtelungsebene

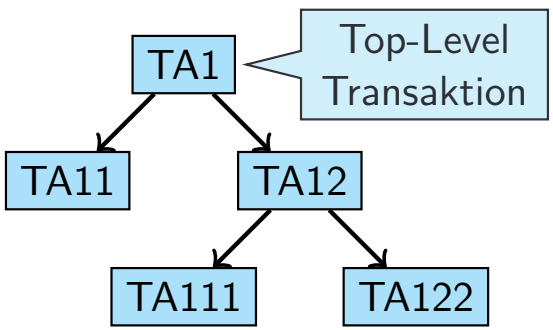


Flache / geschachtelte Transaktionen (2)

Hochschule RheinMain

Geschachtelte Transaktionen (Nested Transactions)

- Eine Transaktion kann (auch zu einem Zeitpunkt) innere Transaktionen (Subtransaktionen) besitzen
- isolierte Zurücksetzbarkeit innerer Transaktionen: Abort einer inneren TA führt zu Exception (nicht: Abbruch!) bei übergeordneter TA
- Abort einer TA führt zum Abort aller inneren TAen
- Commitment relativ zum Parent (endgültig erst mit Commitment der Top-Level TA)
- Nebenläufigkeitskontrolle auf jeder Schachtelungsebene



Stellenlokale Commit-Verfahren

Verfahren zur lokalen Bereitstellung von Atomarität im Fehlerfall und Permanenz der Wirkung:

- Intention Lists (Lampson 1981)
- Shadowing (Gray 1981)
- Write-Ahead Logging

Intention lists

Vorgehensweise

- Beabsichtigte Veränderungen auf Datenobjekten werden in Liste gesammelt (d.h. Arbeiten auf Kopien der Originaldaten)
- Liste wird in stabilen Speicher geschrieben
- Entscheidung (committed-aborted) treffen
- Falls aborted: Liste verwerfen
- Falls committed: Originale der Objekte in nicht-flüchtigem Speicher aktualisieren

Im Verteilten System

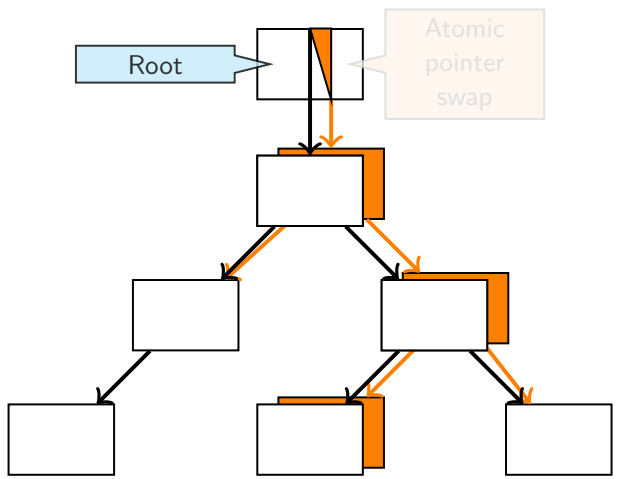
- Jeder Knoten führt eine vorläufige Liste und kennt den Koordinator
- Ein Koordinator schreibt alle Knoten in eine Liste und benachrichtigt diese
- Die benachrichtigten Knoten aktualisieren die Objekte gemäSS der Liste und löschen diese

Shadowing

Hochschule RheinMain

Vorgehensweise

- Nichtflüchtiger Speicher als Baumstruktur mit Verweisblöcken (entsprechend UNIX- Dateisystem) angenommen
- After Images aller Blöcke als Shadow-Version bis zur Wurzel anlegen
- Entscheidung (committed-aborted) treffen durch atomaren Pointer-Swap in Root-Block (Schreiben eines Blockes)
- Falls aborted: Shadow-Struktur verwerfen
- Falls committed: ehemalige Original-Blöcke freigeben, Shadow-Blöcke sind jetzt die Originale



nach Stellenfehler

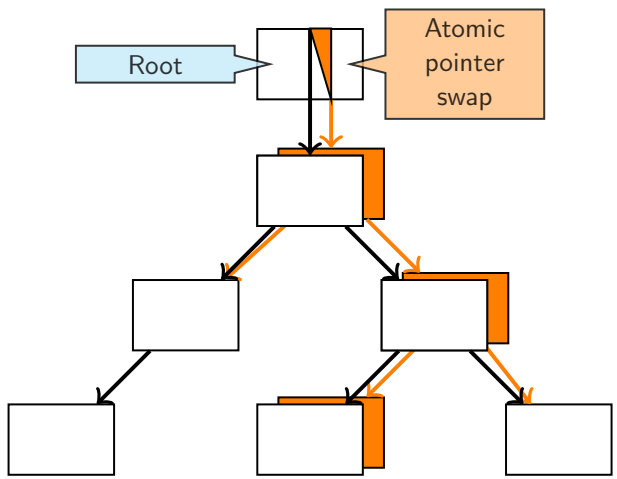
- Entweder alter Zustand oder neuer Zustand ist etabliert

Shadowing

Hochschule RheinMain

Vorgehensweise

- Nichtflüchtiger Speicher als Baumstruktur mit Verweisblöcken (entsprechend UNIX- Dateisystem) angenommen
- After Images aller Blöcke als Shadow-Version bis zur Wurzel anlegen
- Entscheidung (committed-aborted) treffen durch atomaren Pointer-Swap in Root-Block (Schreiben eines Blockes)
- Falls aborted: Shadow-Struktur verwerfen
- Falls committed: ehemalige Original-Blöcke freigeben, Shadow-Blöcke sind jetzt die Originale



nach Stellenfehler

- Entweder alter Zustand oder neuer Zustand ist etabliert

Shadowing (2)

Hochschule RheinMain

Vorteile

- Atomarer Zustandswechsel durch Schreiben eines/des Root-Blockes

Nachteile

- keine Nebenläufigkeit von Commit-Vorgängen
- Physische Anordnung der Datenblöcke zueinander geht durch Commit-Vorgänge verloren, da Shadow-Blöcke aus der Freiliste genommen werden müssen.

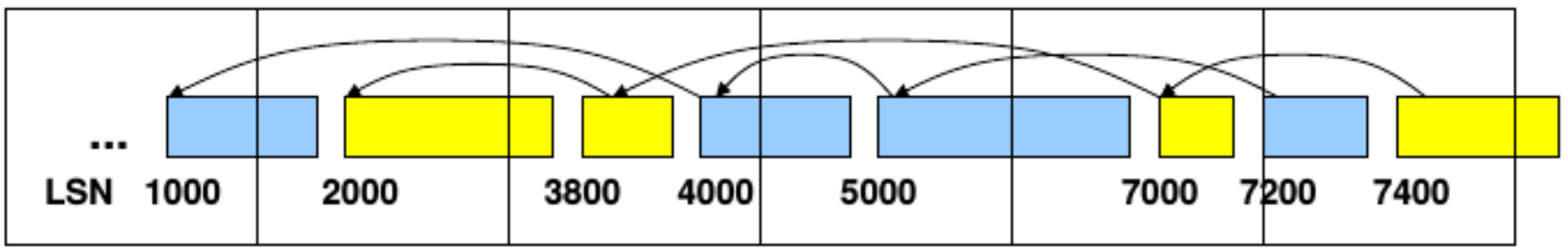
Write-Ahead Logging

Hochschule RheinMain

Log-Grundlagen

- Log besteht logisch aus Folge von sogenannten Log Records variabler Länge identifiziert durch Log Sequence Number (LSN)
  $\rightarrow$ Byte Offset im Log-Strom analog TCP Sequenznummer
- Log ist gemeinsam genutzt innerhalb eines Knotens
- Verkettung von zusammengehörigen Log-Einträgen eines Commit-Vorgangs
- Realisierung des Logs in replizierten Dateien (stabiler Speicher)
- Übergeordnete Tabelle mit Log-Einträgen und SQL-Zugriff in Datenbanken üblich

Blockstruktur des Mediums



älter Schreibrichtung $\longrightarrow$ jünger

Write-Ahead Logging(2)

Schreiben von Log-Einträgen

- Nur sequentielles Schreiben der Log Files (hohe Performance)
- Gepuffertes Schreiben von Log-Einträgen im Arbeitsspeicher
- *Forced Write* eines Log-Eintrags erzwingt Ablage aller vorangegangenen Log-Einträge (mit kleinerer LSN) und Rückkehr aus der Operation, nachdem Block physisch auf dem Medium steht

Lesen von Log-Einträgen

- Nur im Fall von Stellenfehlern und evtl. bei Transaktionsfehlern (s.u.)

Log-Verkürzung

- Log kann logisch beliebig lang werden, aber Restart-Dauer nach Stellenfehler muss begrenzt werden
- Nutzung von Checkpoints

Write-Ahead Logging(3)

Nutzung von Log-Einträgen im Rahmen des lokalen Commitments

- Log Records einer Transaktion sind untereinander verkettet
- Schreiben von Before Images (Undo Records) für alle Objekte, deren persistenter Originalzustand durch die Transaktion im Zustand active geändert werden (Update in Place).
- Schreiben von After Images (Redo Records) für alle erarbeiteten finalen Objektzustände der Transaktion
- Ablegen eines finalen Outcome Records mittels Forced Write:
  - erzwingt Ablage aller zugehörigen Log-Einträge
  - erscheint dieser im Log, ist das Commitment vollzogen
  - kann auch als Flag im letzten Block realisiert sein

Write-Ahead Logging Protocol

- Gesichertes Schreiben von Log Records vor Modifikation des originalen persistenten Zustands

Write-Ahead Logging(4)

Behandlung von Transaktionsfehlern

- (Eventuell) „Aborted“ Outcome Record erzeugen und ablegen
- Falls Before Records geschrieben wurden, deren Inhalte wieder als persistenten Objektzustand etablieren

Behandlung von Stellenfehlern

- Lesen des Logs
- Für alle nicht abgeschlossenen Transaktionen evtl. vorhandene Before Images etablieren
- Für alle Transaktionen mit vorhandenem „Committed“ Outcome Record das jeweils letzte After Image aller Objekte (irgendwann) als persistenten Objektzustand etablieren (Idempotenz)

Write-Ahead Logging(5)

Vorteile

- Mehrere ineinander verzahnte Commit-Vorgänge können gleichzeitig stattfinden
- Hohe I/O-Performance durch Buffering und sequentielles Schreiben des Logs
- Anordnung der Datenblöcke des persistenten Zustands bleibt erhalten (Update in Place)
- Parallelisierung des Logs möglich
- Nach Stellenfehler und anschlieSSendem Neustart muss nur der Log analysiert werden
- Log kann auch von Commit-Protokollen mitgenutzt werden (s.u.)

Das 2-Phasen Commit-Protokoll

Commit-Protokolle

- dienen dazu, in einer verteilten Umgebung die Commit/Abort-Entscheidung einer Menge von Prozessen zu koordinieren
- z.B. für Durchsetzung von Atomarität im Fehlerfall und Dauerhaftigkeit für verteilte Transaktionsumgebung
- sind Spezialfälle sog. Konsensus-Protokolle (hier: ja/nein)

2-Phasen-Commit-Protokoll

- ist das am weitesten verbreitete Protokoll
- von hoher praktischer Bedeutung und auch in zahlreichen Produkten enthalten
- im weiteren besprochen

Allgemeinere Theorie

- Mehrphasen-Commit-Protokolle (z.B. Skeen, 1981)
- schwache / starke Terminierungsbedingungen führen zu blockierenden / nichtblockierenden Commit-Algorithmen

Eigenschaften eines Commit-Algorithmus

Ein Commit-Algorithmus für eine Menge von Prozessen besitzt folgende Eigenschaften:

1. Alle Prozesse, die eine Entscheidung treffen, treffen die gleiche Entscheidung
2. Eine einmal getroffene Entscheidung ist bindend für jeden Prozess und kann nicht widerrufen werden
3. Nur wenn alle Prozesse für Commit entscheiden, lautet die gemeinsame Entscheidung auf Commit, d.h. auch: sobald ein Prozess auf Abort entscheidet, muss die gemeinsame Entscheidung auf Abort lauten
4. Wenn zu einem Zeitpunkt alle aufgetretenen Fehler repariert sind und hinreichend lange keine neuen Fehler auftreten, erreichen die Prozesse eine gemeinsame Entscheidung.

Begriffe

Fenster der Verwundbarkeit (des Commit-Protokolls)

- Zeitspanne zwischen der lokalen Commit-Enscheidung eines Prozesses und Kenntnis der Gesamtentscheidung
- wird auch Unsicherheitsperiode des Prozesses (Uncertaincy Period) genannt

Blockierendes Protokoll

- Protokoll sieht vor, dass ein Prozess die Reparatur eines Fehlers abwarten muss

Unabhängiges Recovery

- Prozess kann nach einem Fehler eine Entscheidung treffen, ohne mit anderen Prozessen zu kommunizieren

Teilhaber (Participants)

- Prozesse, die Commit-Protokoll abwickeln

Background

Hochschule RheinMain

Sätze:

- Wenn Kommunikationsfehler oder Gesamtsystemfehler möglich sind, gibt es kein Commit-Protokoll, das keinen Prozess blockiert
  Bemerkung: nicht ausgeschlossen ist die Existenz eines nicht-blockierenden Commit-Protokolls, wenn nur einzelne Stellen ausfallen
- Kein Commit-Protokoll kann unabhängiges Recovery ausgefallener Prozesse garantieren
  Bemerkung: Es gibt kein Commit-Protokoll ohne Unsicherheitsperiode für Teilhaber

(Anmerkung: Nicht prüfungsrelevant)

Grundlagen des 2-P-C-Protokolls

J. Gray, 1978
Blockierender Commit-Algorithmus mit schwacher Terminierungseigenschaft ($\Rightarrow$ treten keine Fehler auf, treffen alle Prozesse irgendwann eine Entscheidung).
Rollen

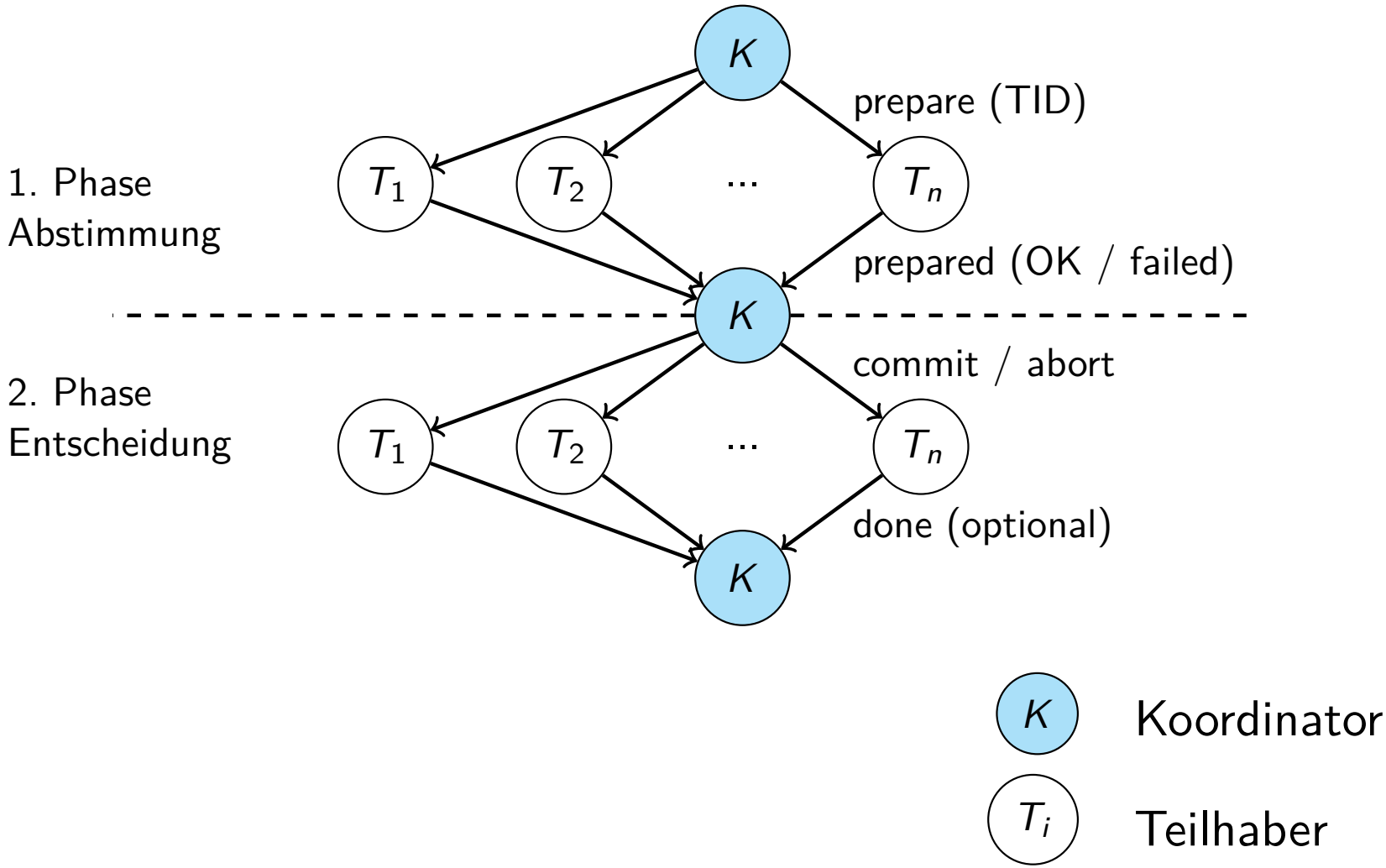
- Teilhaber /Teilnehmer
- Koordinator als ausgezeichneter Teilhaber, der Abwicklung des Protokolls steuert Bemerkung: i.d.R. Teilhaber der Ursprungsstelle der Transaktion, der Commit-Vorgang initiiert
- Koordinator kennt alle Teilhaber
- Teilhaber kennen nur ihren Koordinator, aber sich nicht gegenseitig

Nutzung von jeweils lokalem Log jedes Teilhabers zur Fortschreibung des Zustands des Commit-Vorgangs

2-P-C-Protokoll

Hochschule RheinMain

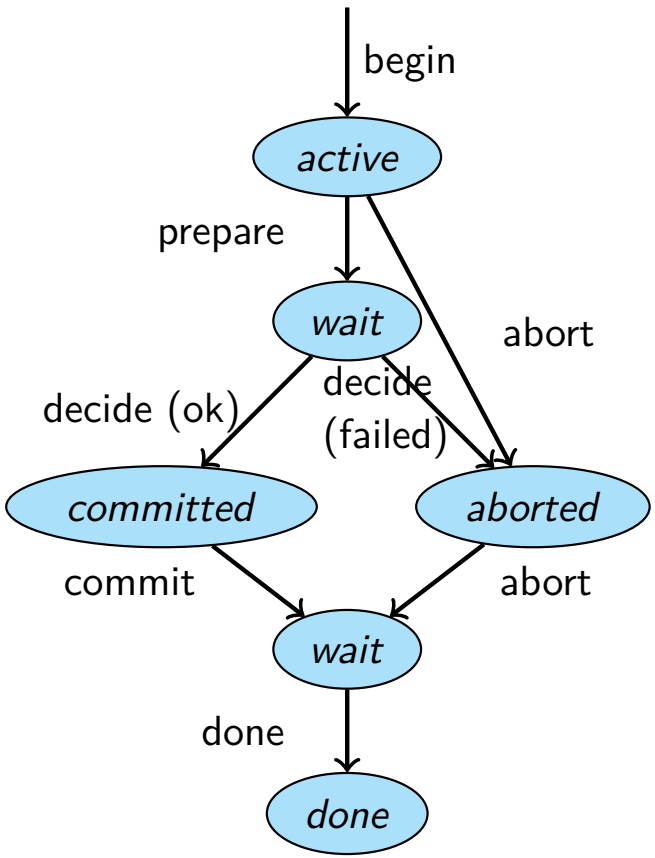
Nachrichtenfluss (Normalfall ohne Fehler)



2-P-C-Protokoll (2)

Hochschule RheinMain

Zustandsdiagramm des Koordinators



Auffordern

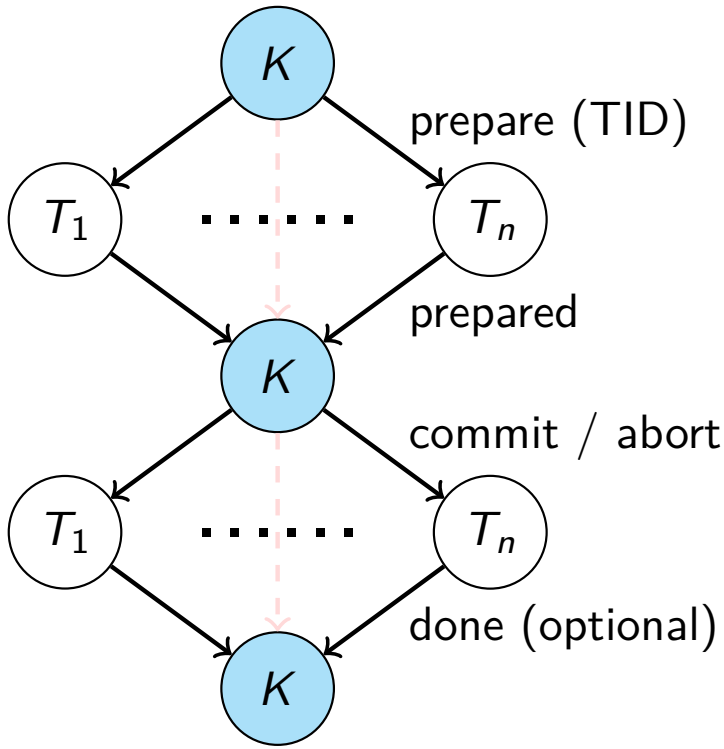
Warten auf Antworten

lokale Entscheidung des Koordinators

Verbreiten

Warten auf Bestätigung

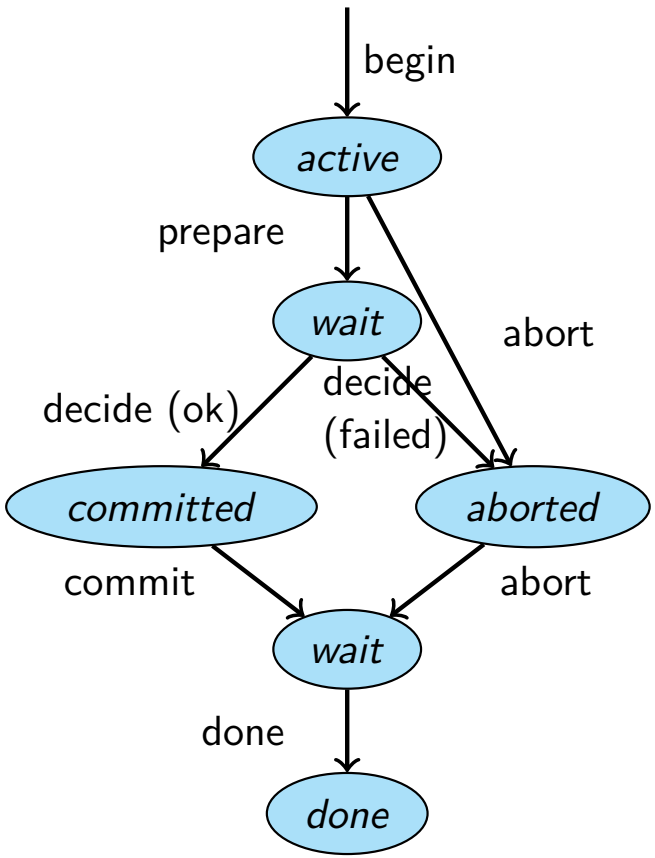
Vergessen



2-P-C-Protokoll (2)

Hochschule RheinMain

Zustandsdiagramm des Koordinators



Auffordern

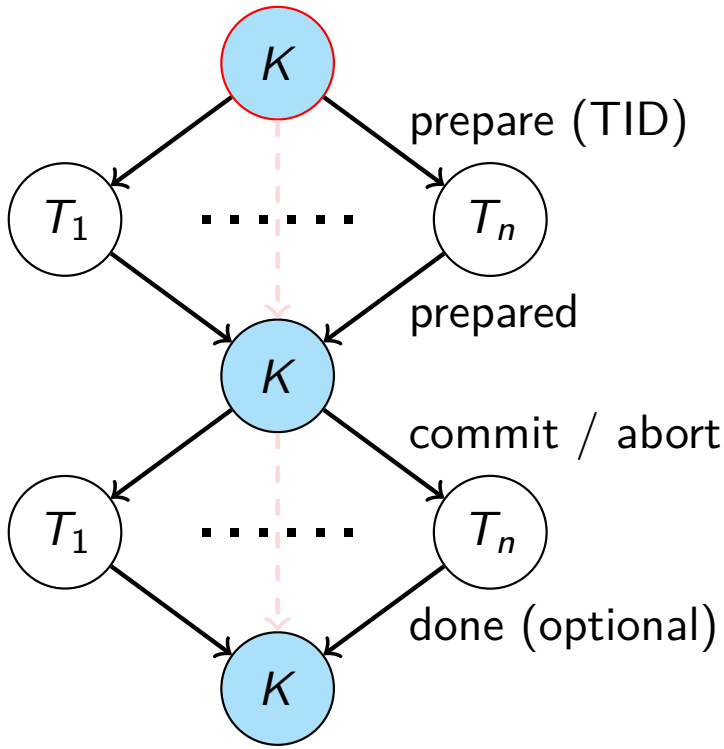
Warten auf Antworten

lokale Entscheidung des Koordinators

Verbreiten

Warten auf Bestätigung

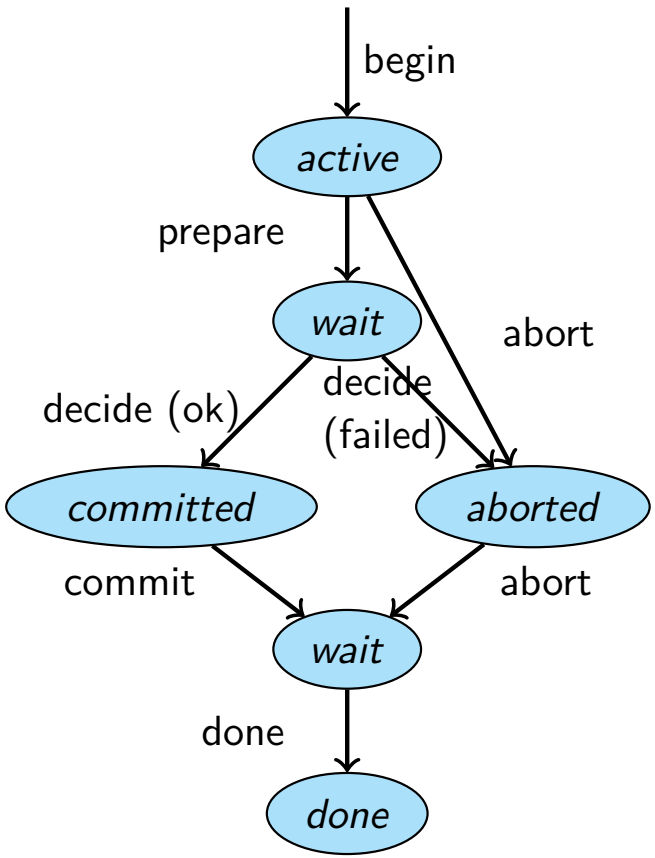
Vergessen



2-P-C-Protokoll (2)

Hochschule RheinMain

Zustandsdiagramm des Koordinators



Auffordern

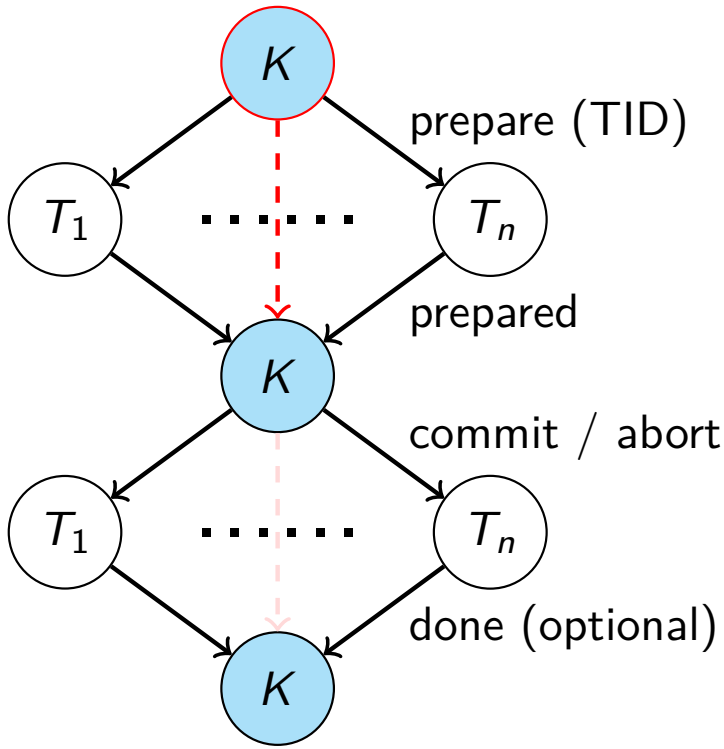
Warten
auf Antworten

lokale
Entscheidung
des Koordinators

Verbreiten

Warten
auf Bestätigung

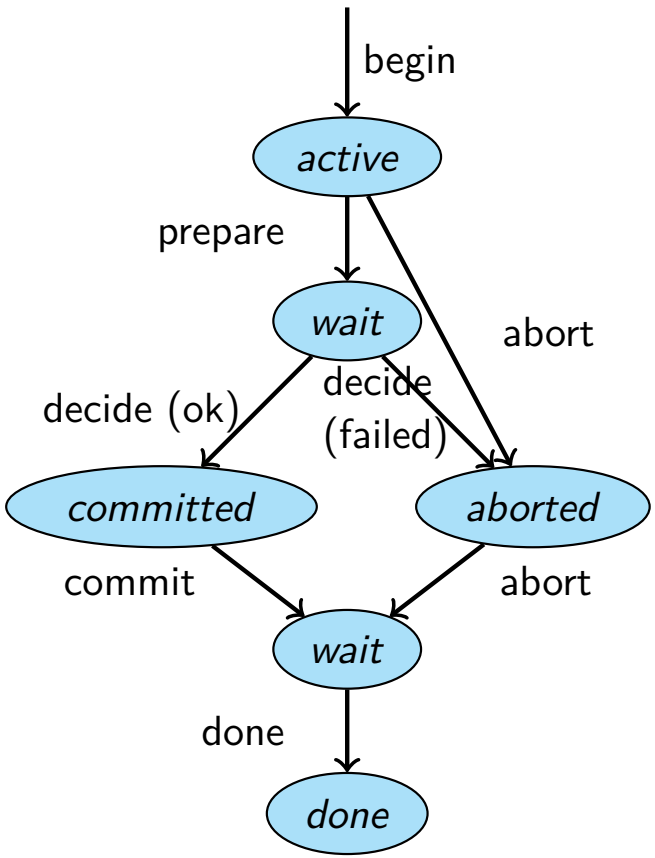
Vergessen



2-P-C-Protokoll (2)

Hochschule RheinMain

Zustandsdiagramm des Koordinators



Auffordern

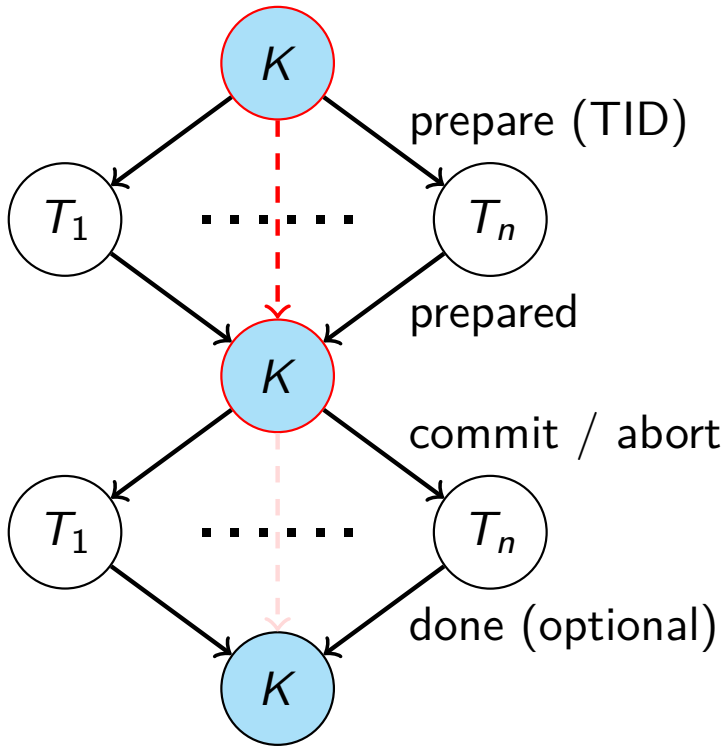
Warten
auf Antworten

lokale
Entscheidung
des Koordinators

Verbreiten

Warten
auf Bestätigung

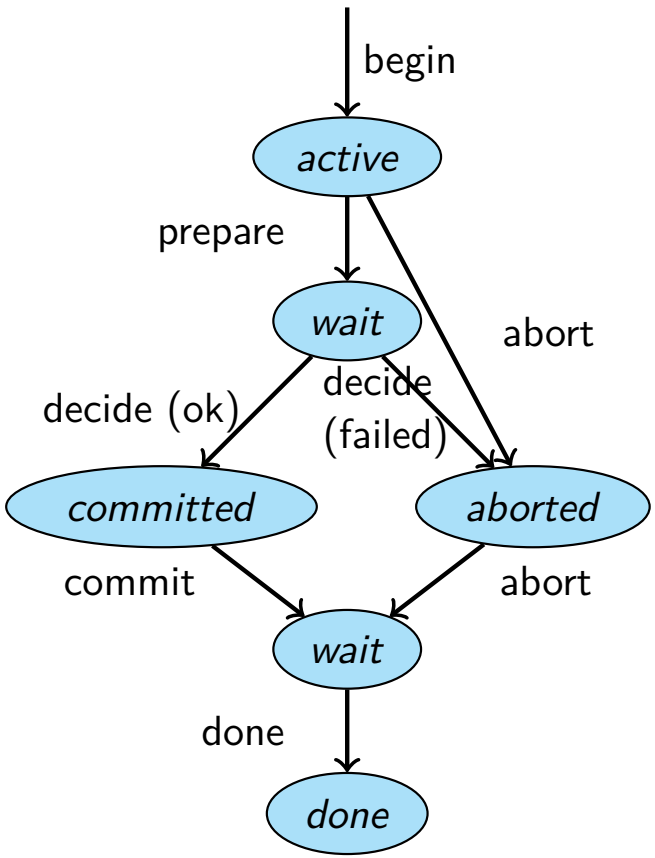
Vergessen



2-P-C-Protokoll (2)

Hochschule RheinMain

Zustandsdiagramm des Koordinators



Auffordern

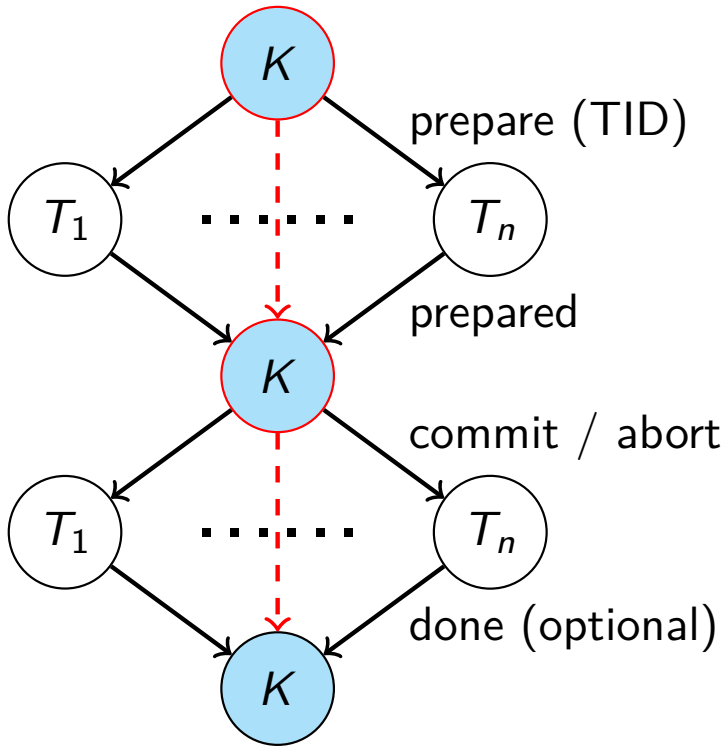
Warten
auf Antworten

lokale
Entscheidung
des Koordinators

Verbreiten

Warten
auf Bestätigung

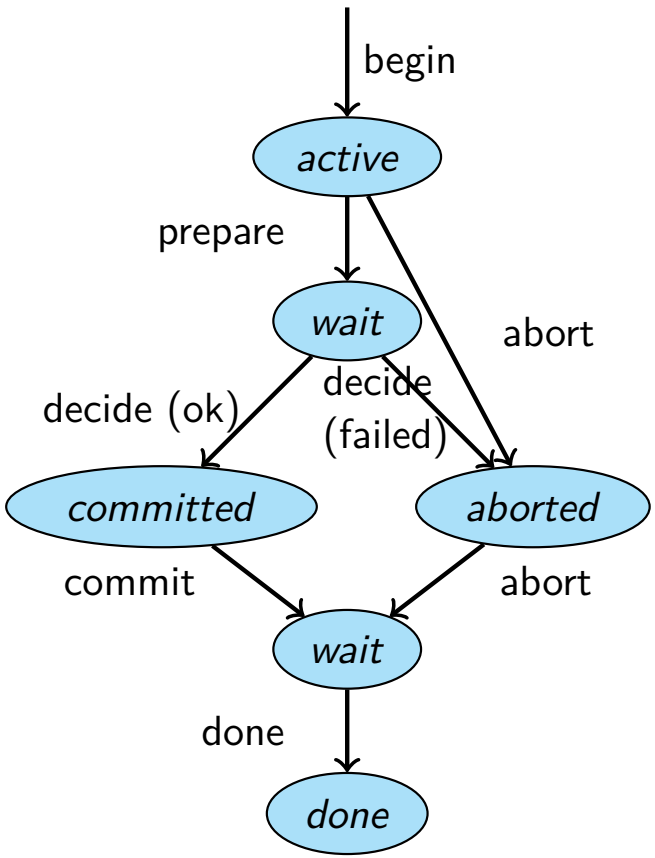
Vergessen



2-P-C-Protokoll (2)

Hochschule RheinMain

Zustandsdiagramm des Koordinators



Auffordern

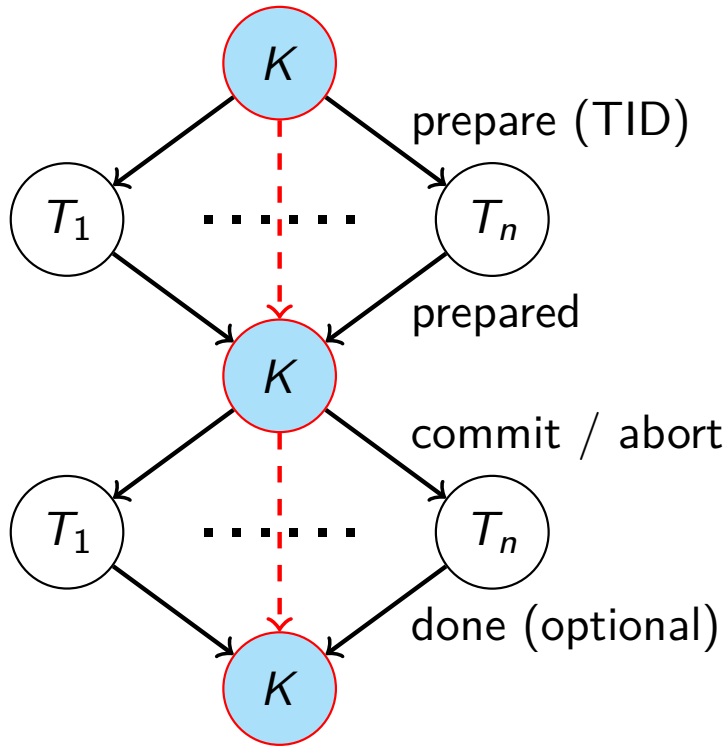
Warten
auf Antworten

lokale
Entscheidung
des Koordinators

Verbreiten

Warten
auf Bestätigung

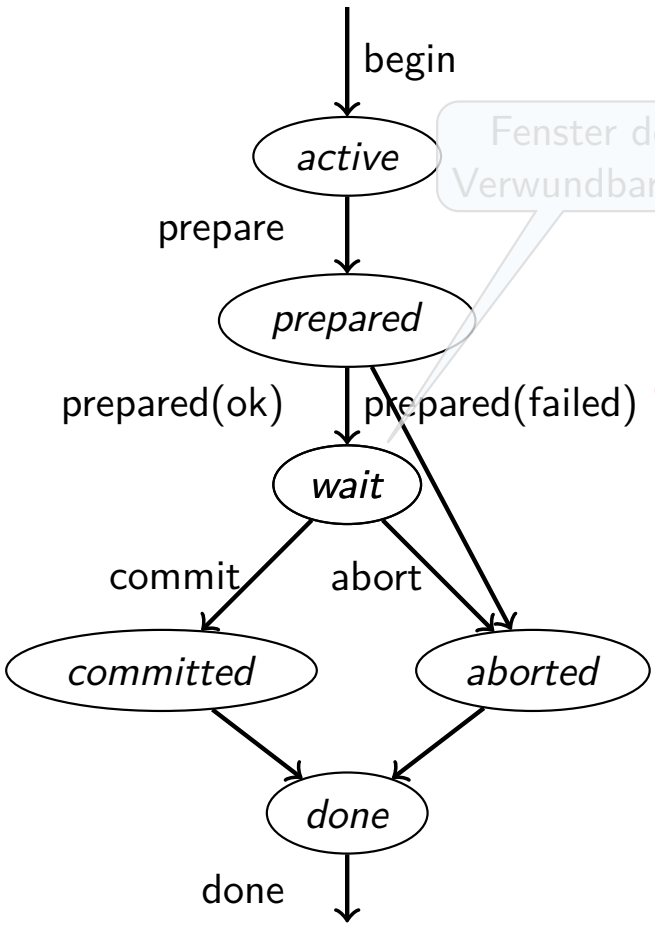
Vergessen



2-P-C-Protokoll (3)

Hochschule RheinMain

Zustandsdiagramm der Teilhaber



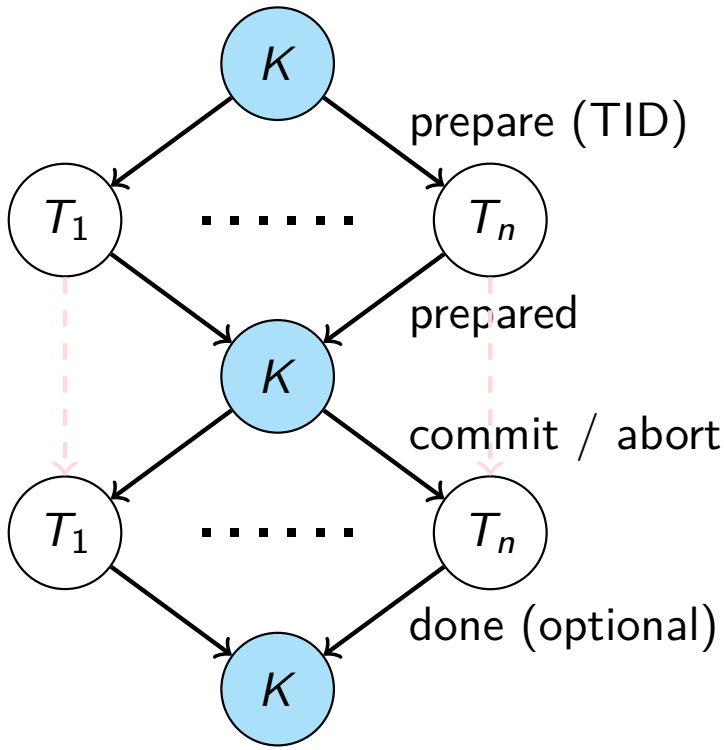
Fenster der Verwundbarkeit

lokales Vorbereiten

Warten auf Entscheidung des Koordinators

Aktionen entsprechend Entscheidung ausführen

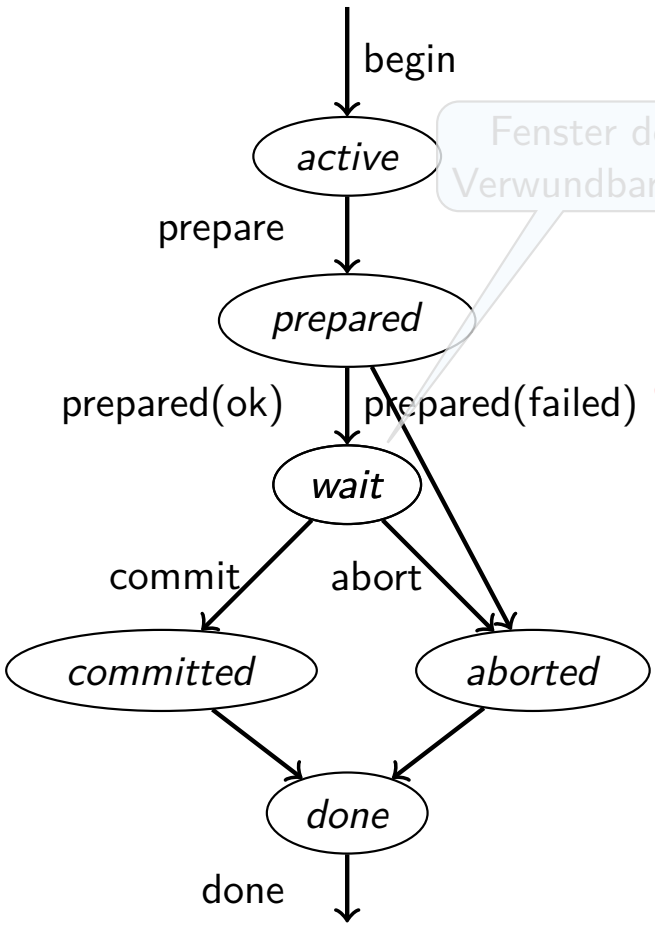
Ende mitteilen



2-P-C-Protokoll (3)

Hochschule RheinMain

Zustandsdiagramm der Teilhaber



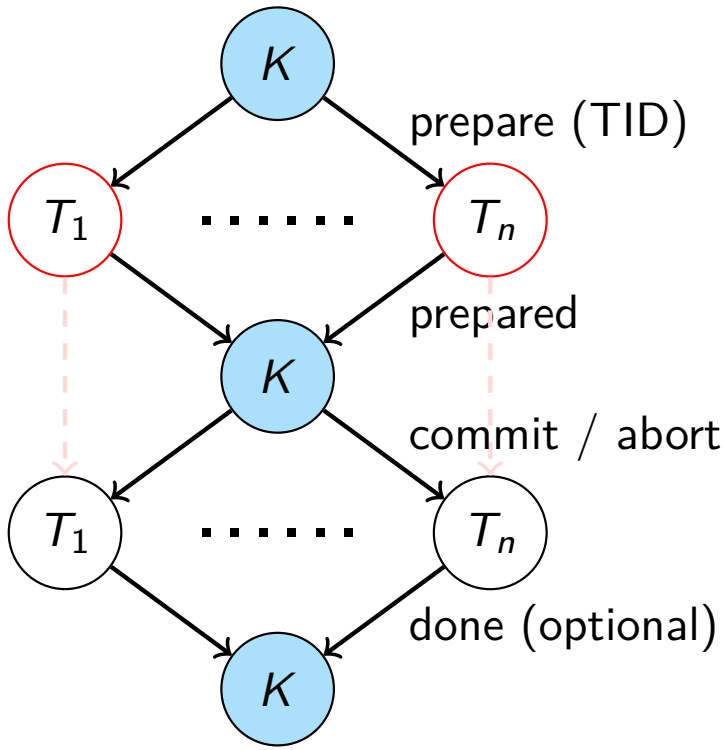
Fenster der Verwundbarkeit

lokales Vorbereiten

Warten auf Entscheidung des Koordinators

Aktionen entsprechend Entscheidung ausführen

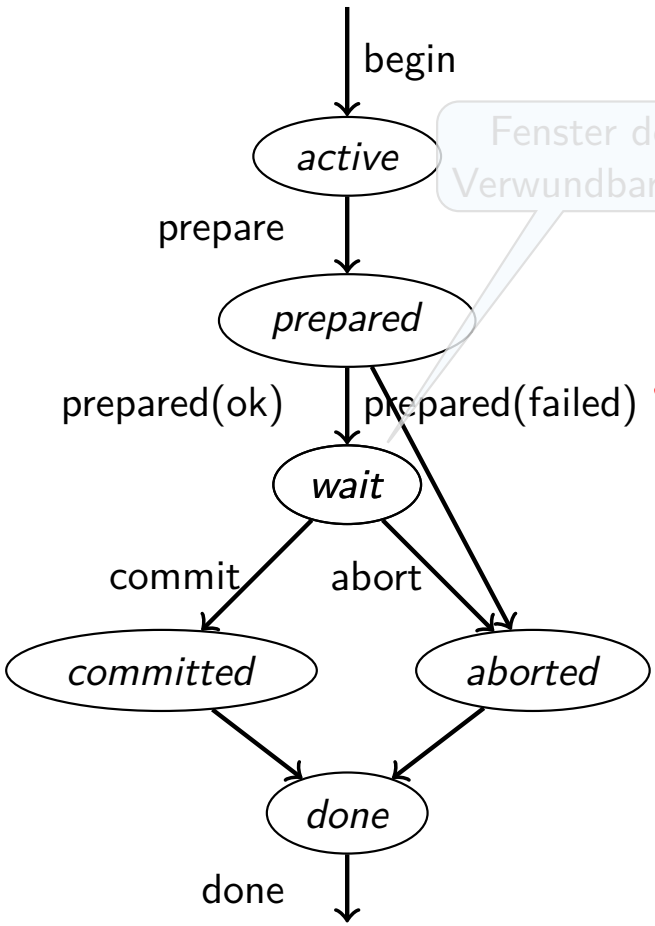
Ende mitteilen



2-P-C-Protokoll (3)

Hochschule RheinMain

Zustandsdiagramm der Teilhaber



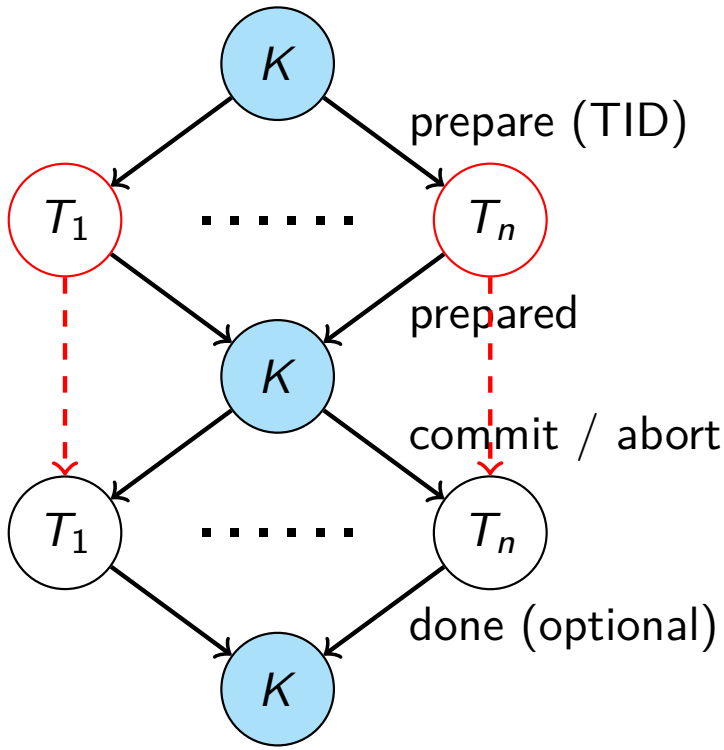
Fenster der Verwundbarkeit

lokales Vorbereiten

Warten auf Entscheidung des Koordinators

Aktionen entsprechend Entscheidung ausführen

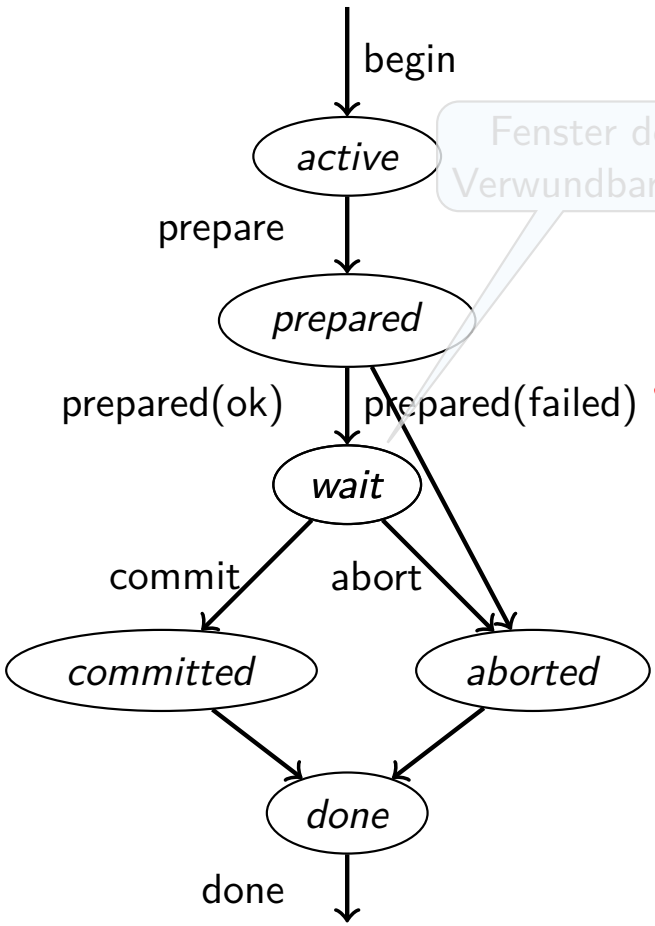
Ende mitteilen



2-P-C-Protokoll (3)

Hochschule RheinMain

Zustandsdiagramm der Teilhaber



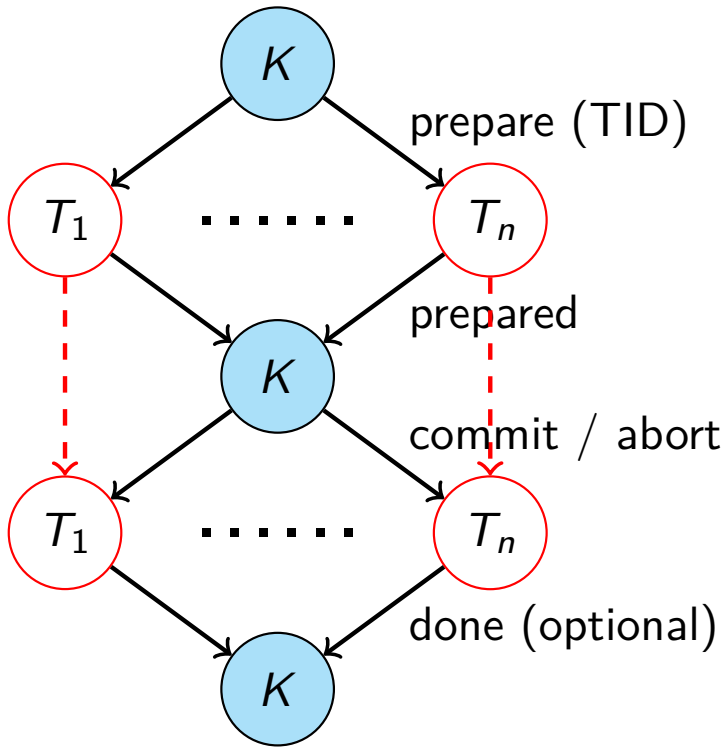
Fenster der Verwundbarkeit

lokales Vorbereiten

Warten auf Entscheidung des Koordinators

Aktionen entsprechend Entscheidung ausführen

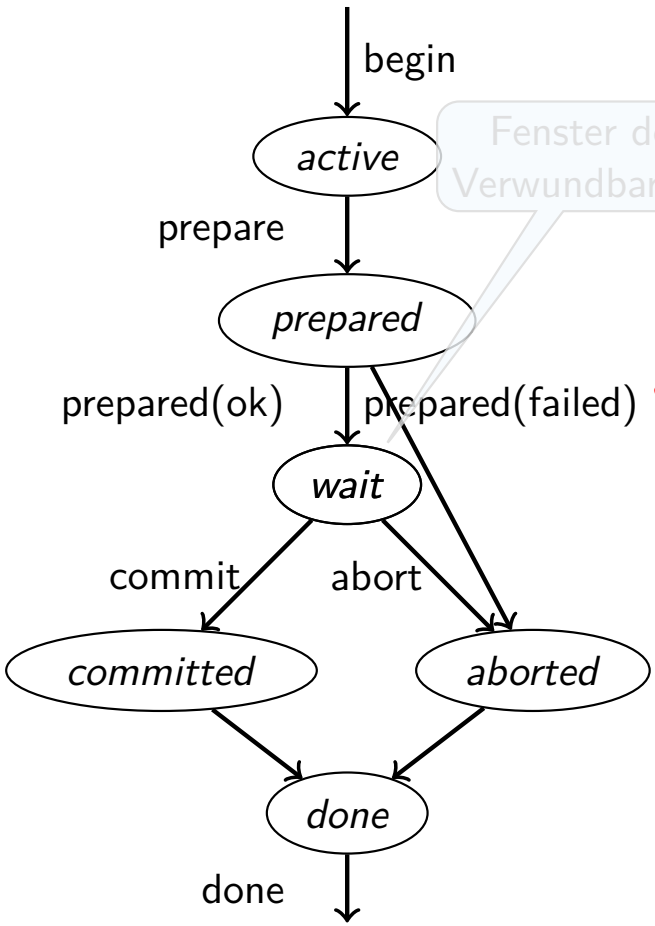
Ende mitteilen



2-P-C-Protokoll (3)

Hochschule RheinMain

Zustandsdiagramm der Teilhaber

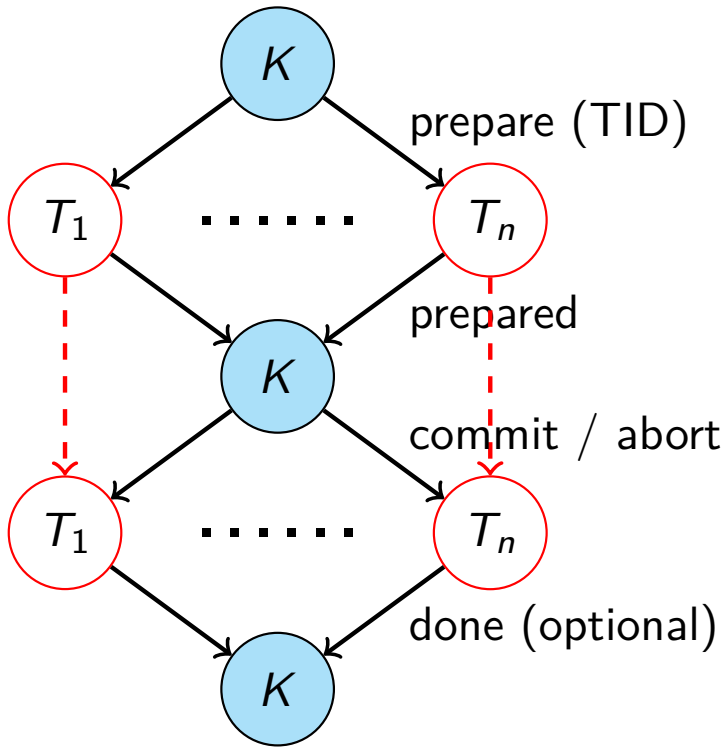


lokales Vorbereiten

Warten auf Entscheidung des Koordinators

Aktionen entsprechend Entscheidung ausführen

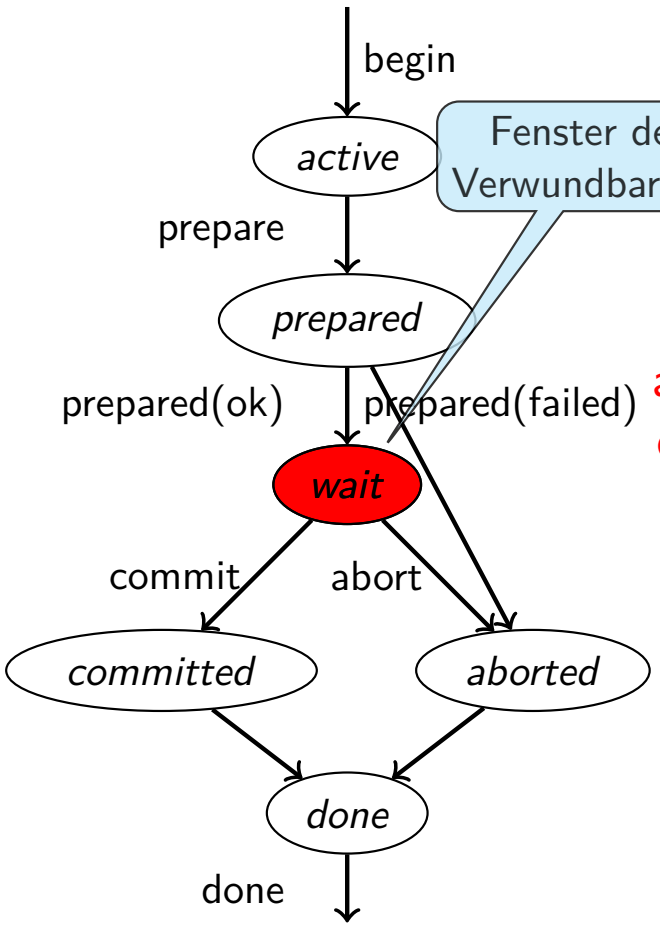
Ende mitteilen



2-P-C-Protokoll (3)

Hochschule RheinMain

Zustandsdiagramm der Teilhaber

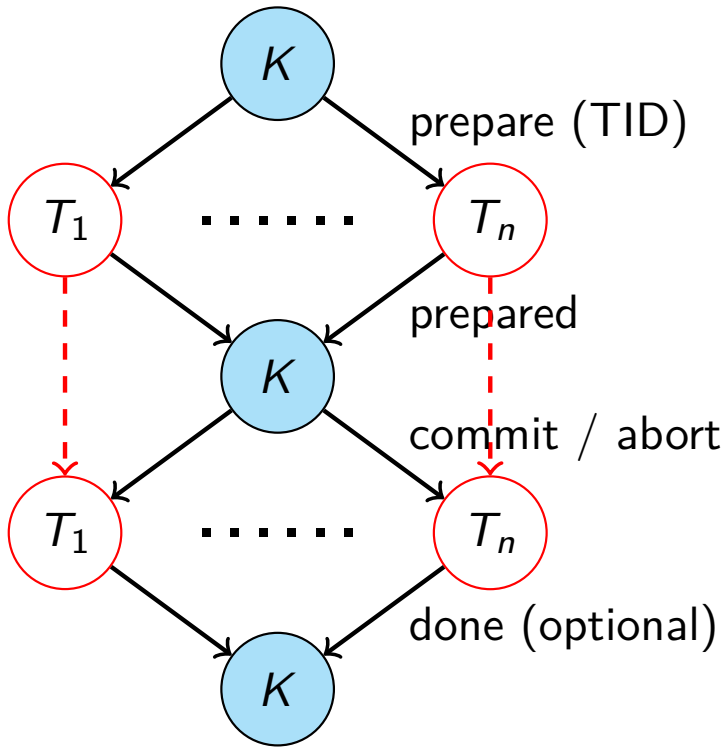


lokales Vorbereiten

Warten auf Entscheidung des Koordinators

Aktionen entsprechend Entscheidung ausführen

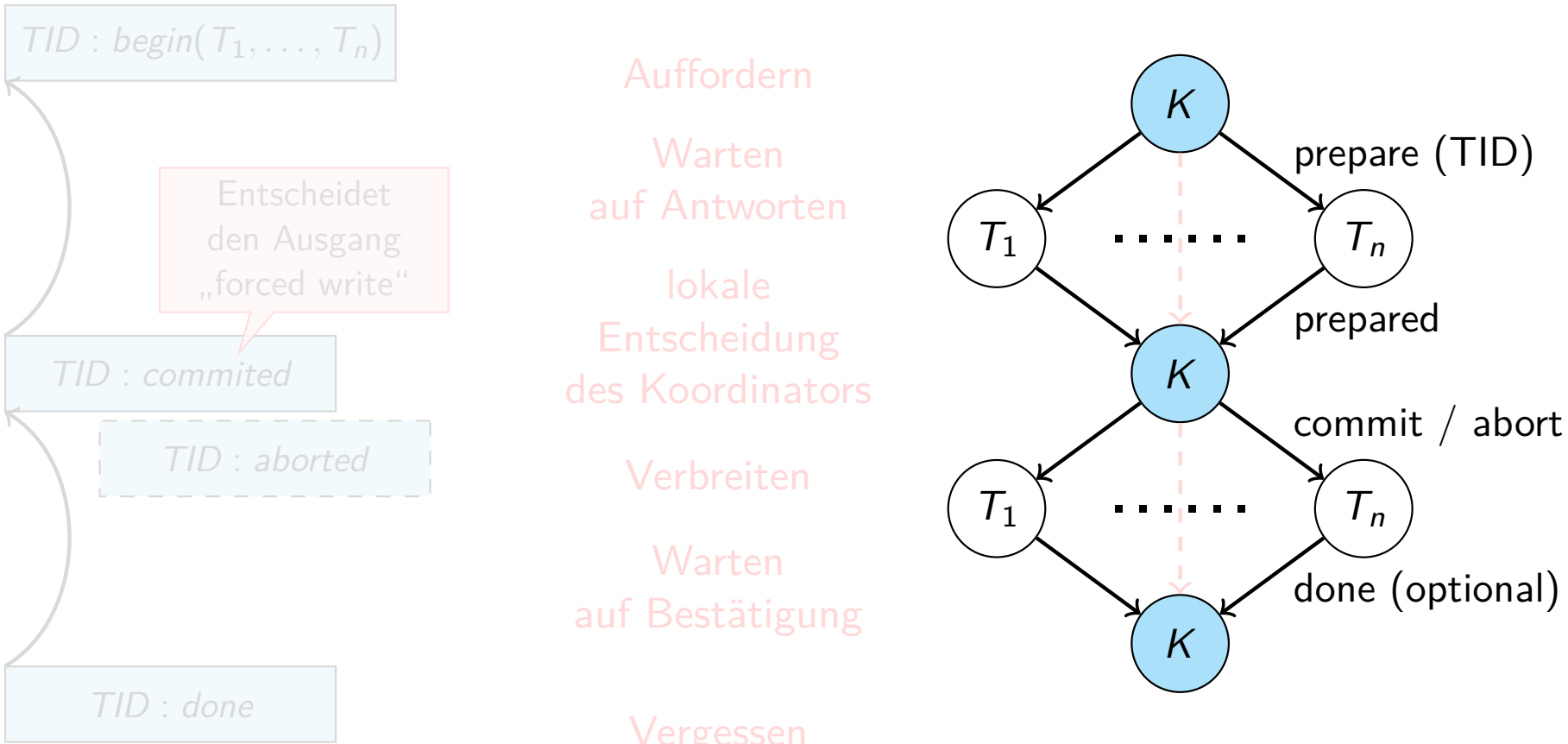
Ende mitteilen



2-P-C-Protokoll (4)

Hochschule RheinMain

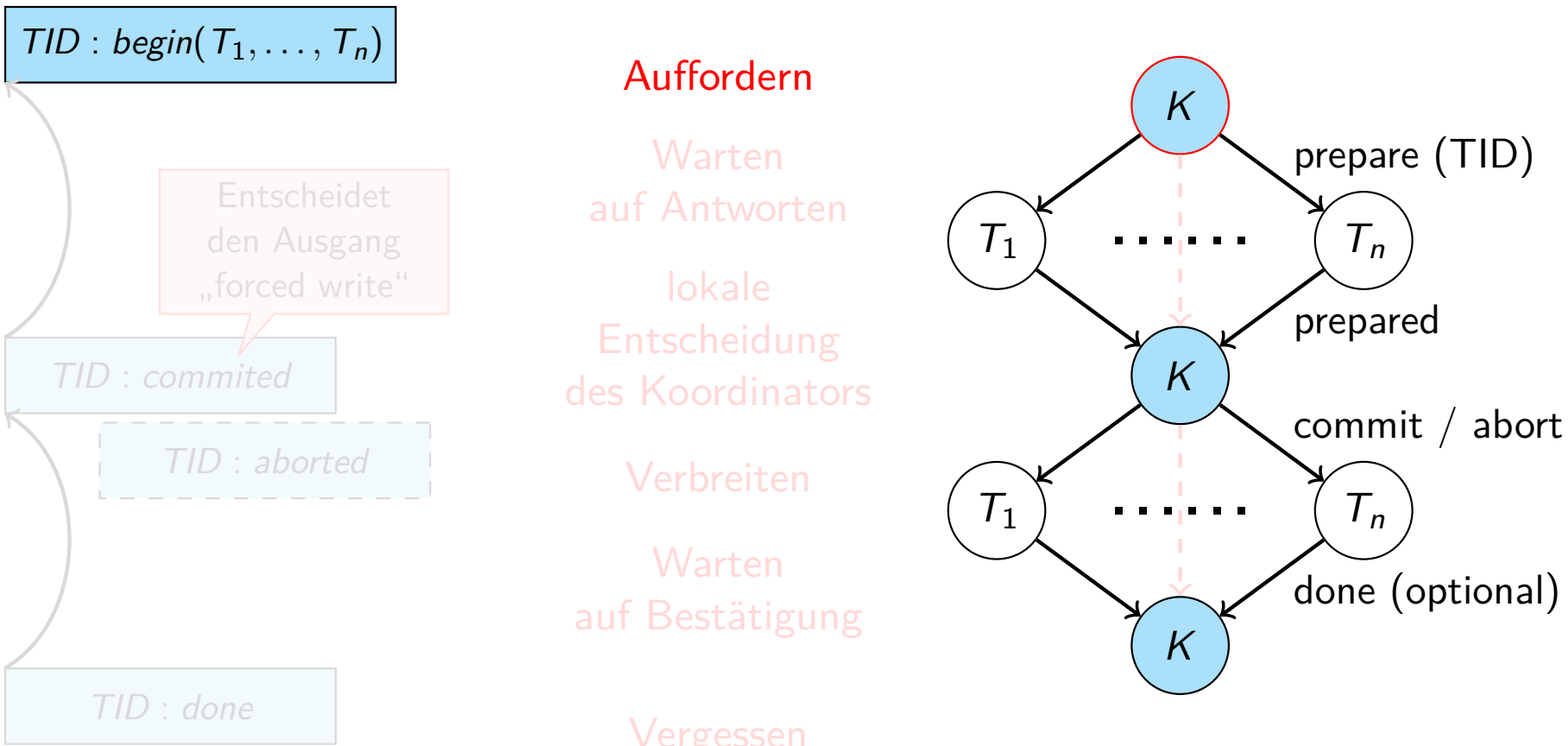
Logging des Koordinators



2-P-C-Protokoll (4)

Hochschule RheinMain

Logging des Koordinators



2-P-C-Protokoll (4)

Logging des Koordinators

$TID : begin(T_1, \ldots, T_n)$

Auffordern

Warten
auf Antworten

Entscheidet
den Ausgang
„forced write“

$TID : commited$

lokale
Entscheidung
des Koordinators

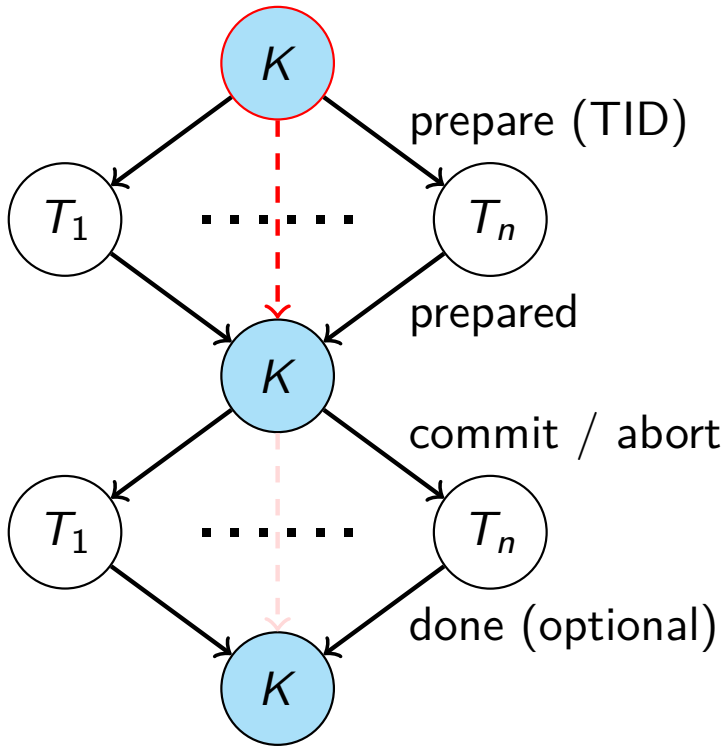
$TID : aborted$

Verbreiten

Warten
auf Bestätigung

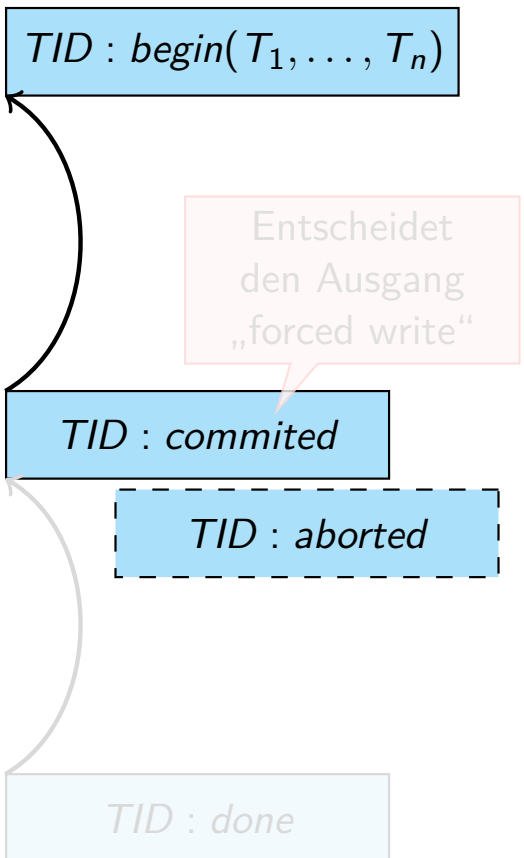
$TID : done$

Vergessen



2-P-C-Protokoll (4)

Logging des Koordinators



Auffordern

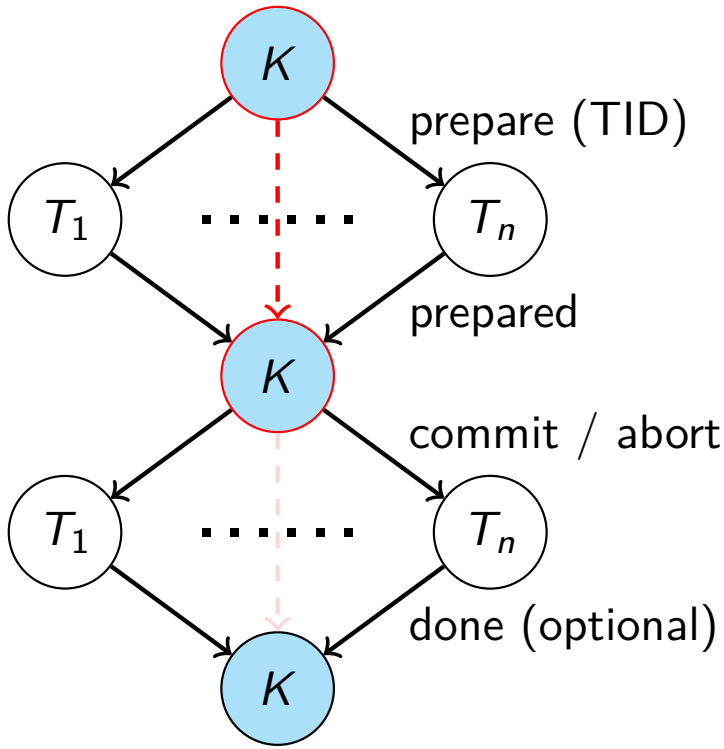
Warten
auf Antworten

lokale
Entscheidung
des Koordinators

Verbreiten

Warten
auf Bestätigung

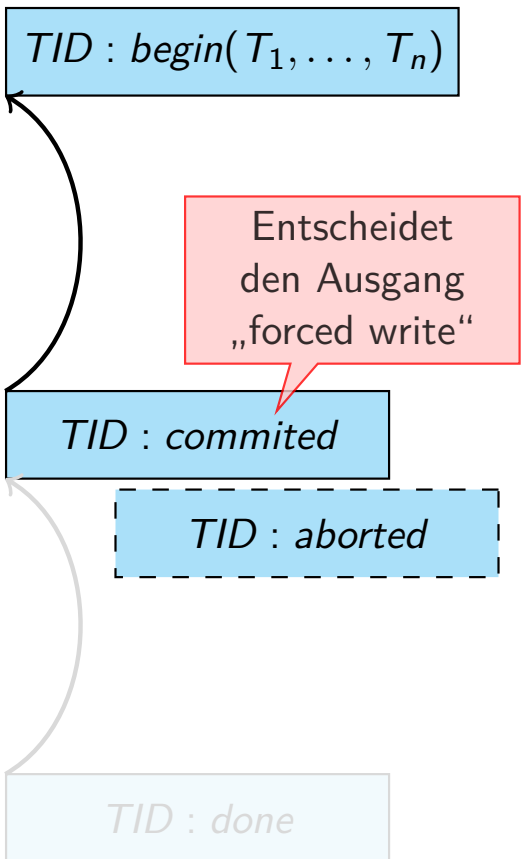
Vergessen



2-P-C-Protokoll (4)

Hochschule RheinMain

Logging des Koordinators



Auffordern

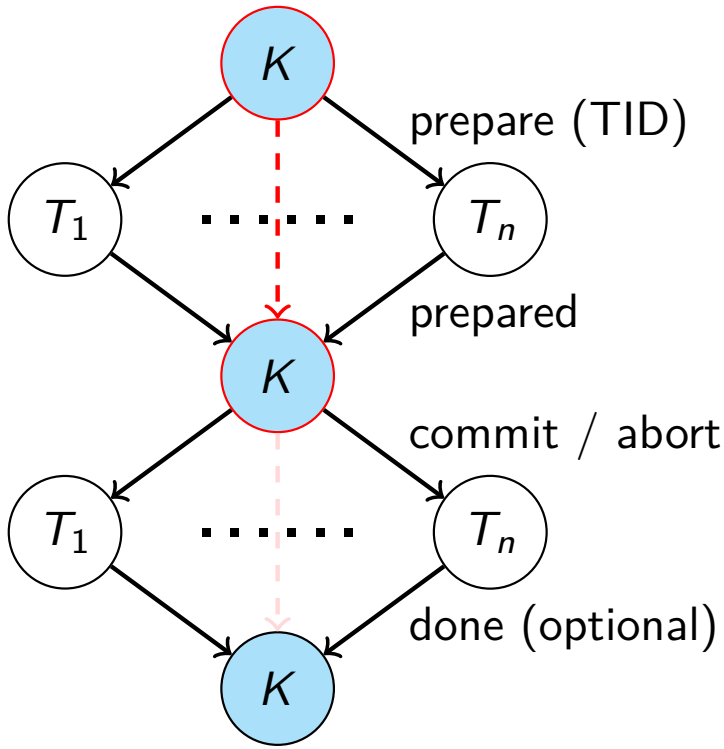
Warten auf Antworten

lokale Entscheidung des Koordinators

Verbreiten

Warten auf Bestätigung

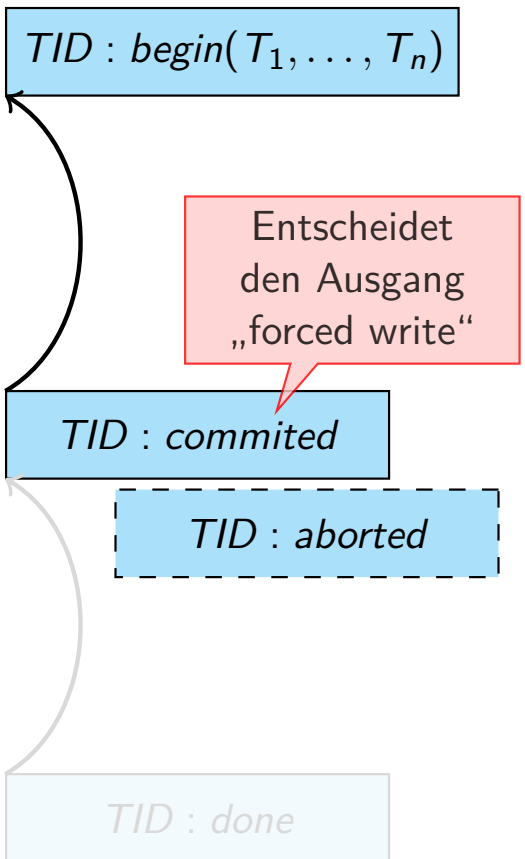
Vergessen



2-P-C-Protokoll (4)

Hochschule RheinMain

Logging des Koordinators



Auffordern

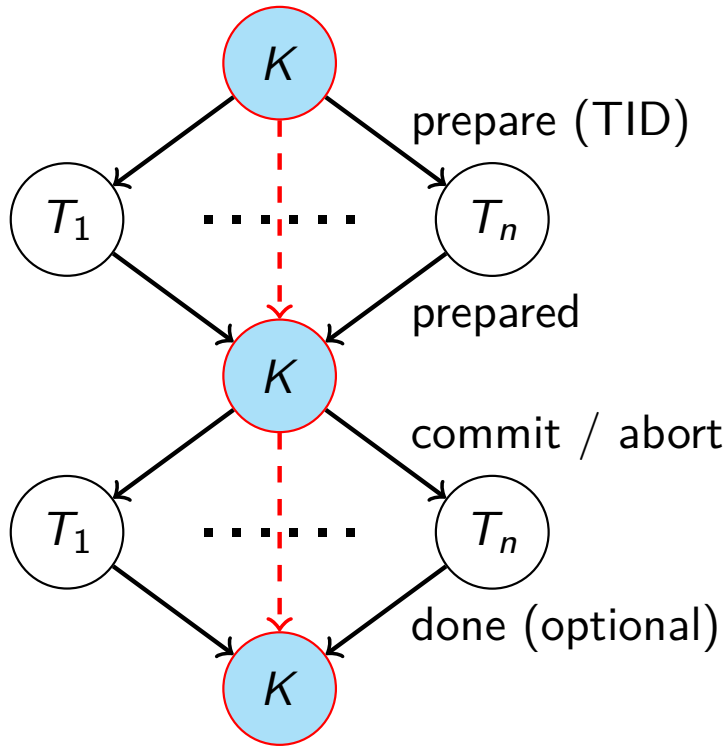
Warten auf Antworten

lokale Entscheidung des Koordinators

Verbreiten

Warten auf Bestätigung

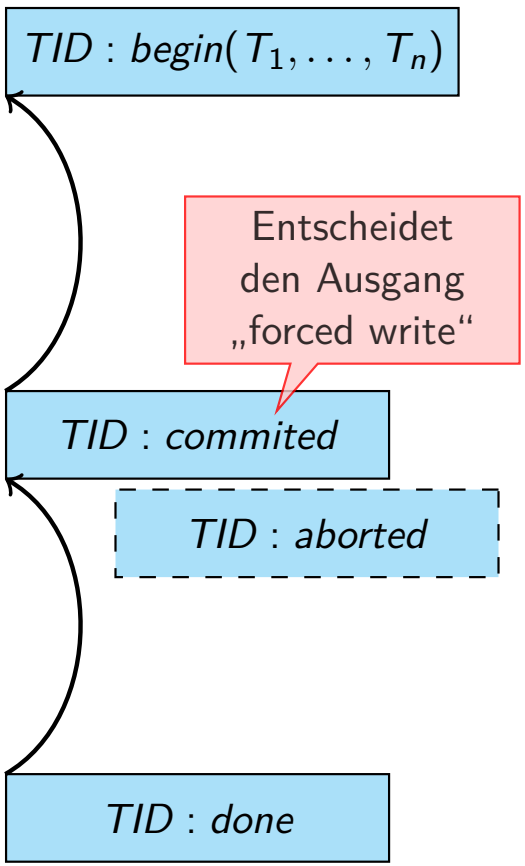
Vergessen



2-P-C-Protokoll (4)

Hochschule RheinMain

Logging des Koordinators



Auffordern

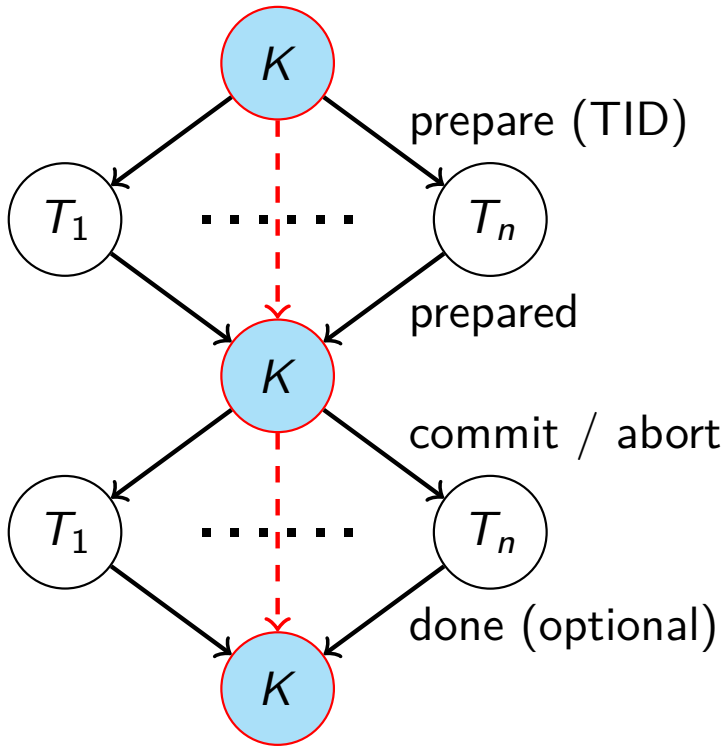
Warten auf Antworten

lokale Entscheidung des Koordinators

Verbreiten

Warten auf Bestätigung

Vergessen



2-P-C-Protokoll (5)

Hochschule RheinMain

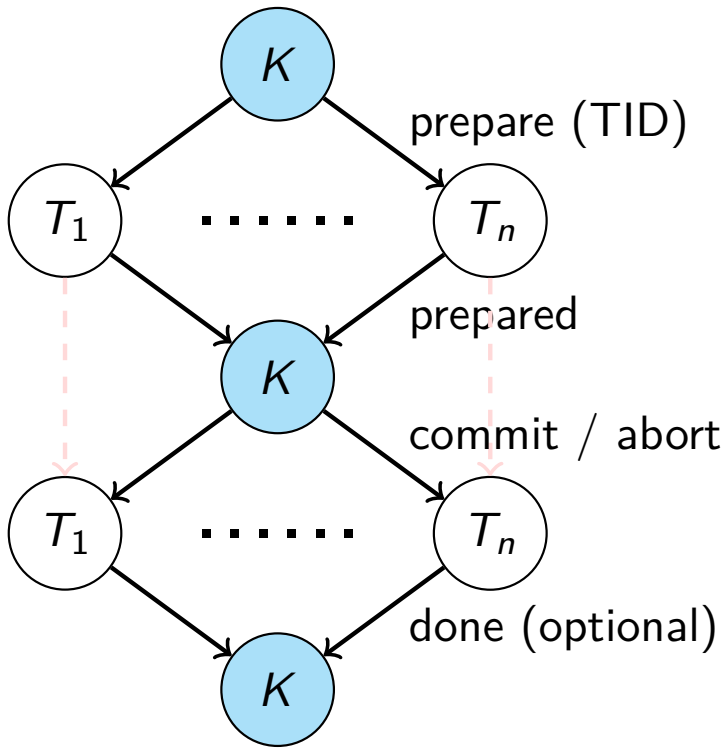
Logging der Teilhaber

lokales Vorbereiten

Warten auf Entscheidung des Koordinators

Aktionen entsprechend Entscheidung ausführen

Ende mitteilen



entsprechend bei einseitigem Abort

2-P-C-Protokoll (5)

Hochschule RheinMain

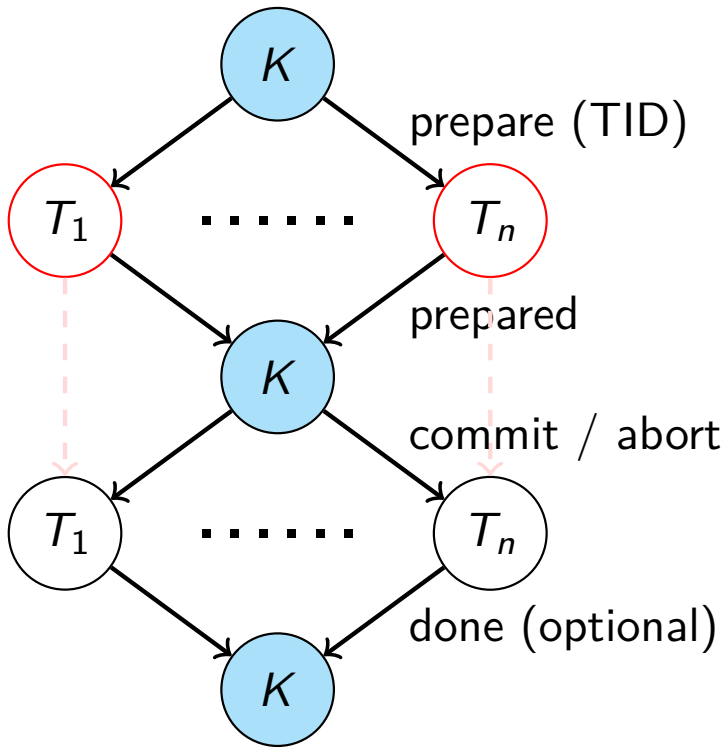
Logging der Teilhaber

lokales Vorbereiten

Warten auf Entscheidung des Koordinators

Aktionen entsprechend Entscheidung ausführen

Ende mitteilen

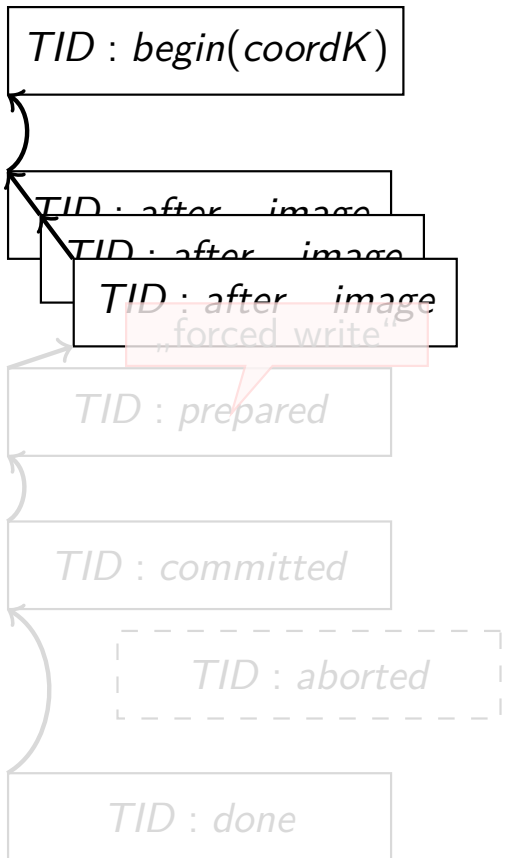


entsprechend bei einseitigem Abort

2-P-C-Protokoll (5)

Hochschule RheinMain

Logging der Teilhaber

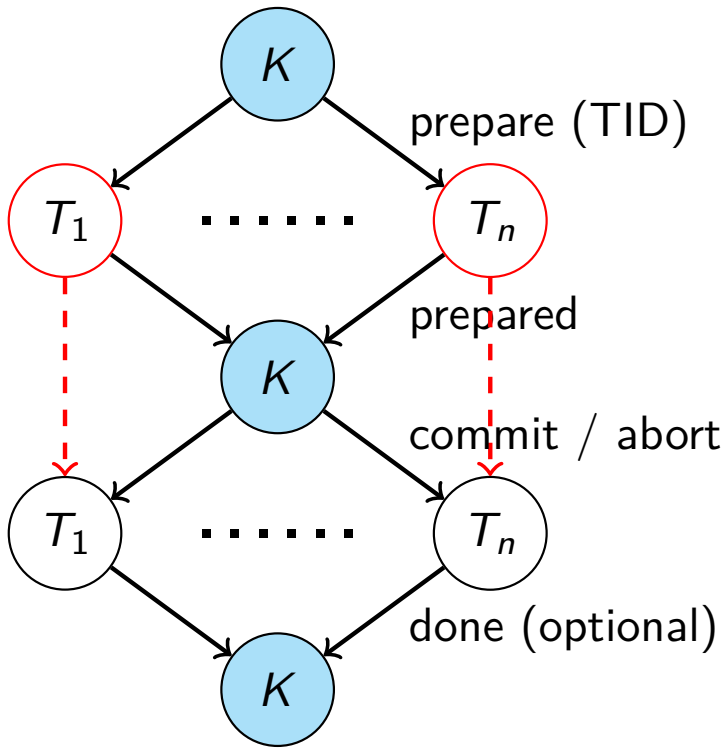


lokales Vorbereiten

Warten auf Entscheidung des Koordinators

Aktionen entsprechend Entscheidung ausführen

Ende mitteilen

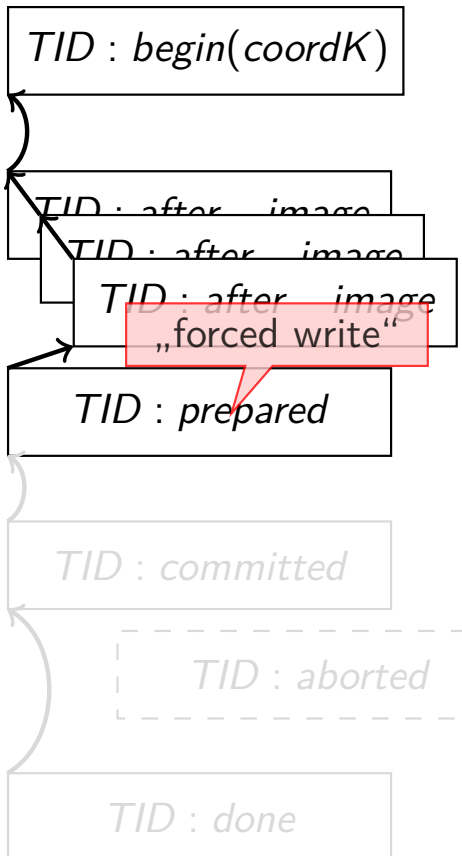


entsprechend bei einseitigem Abort

2-P-C-Protokoll (5)

Hochschule RheinMain

Logging der Teilhaber

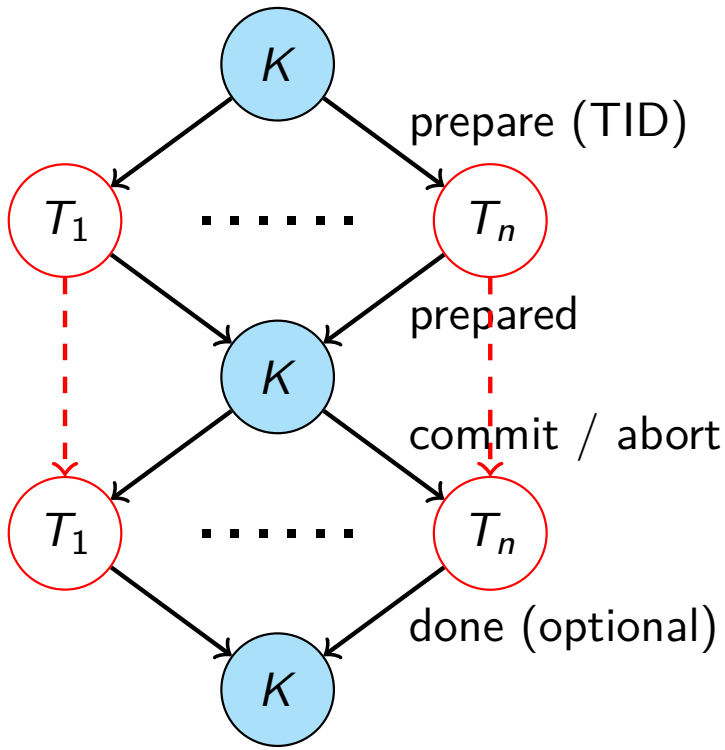


lokales Vorbereiten

Warten auf Entscheidung des Koordinators

Aktionen entsprechend Entscheidung ausführen

Ende mitteilen

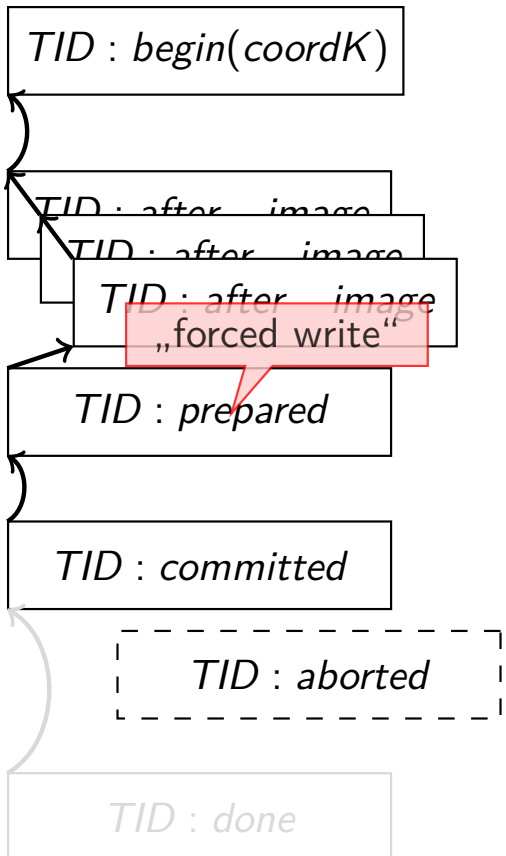


entsprechend bei einseitigem Abort

2-P-C-Protokoll (5)

Hochschule RheinMain

Logging der Teilhaber

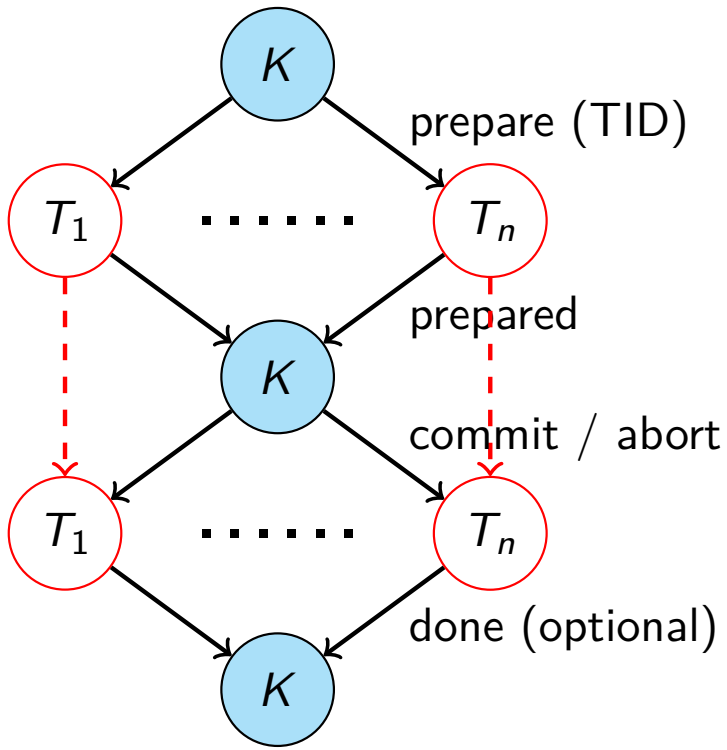


lokales Vorbereiten

Warten auf Entscheidung des Koordinators

Aktionen entsprechend Entscheidung ausführen

Ende mitteilen

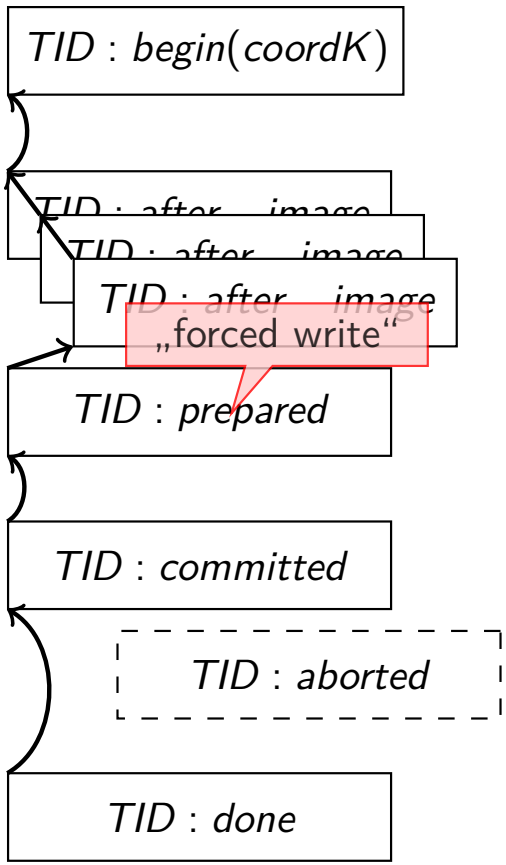


entsprechend bei einseitigem Abort

2-P-C-Protokoll (5)

Hochschule RheinMain

Logging der Teilhaber

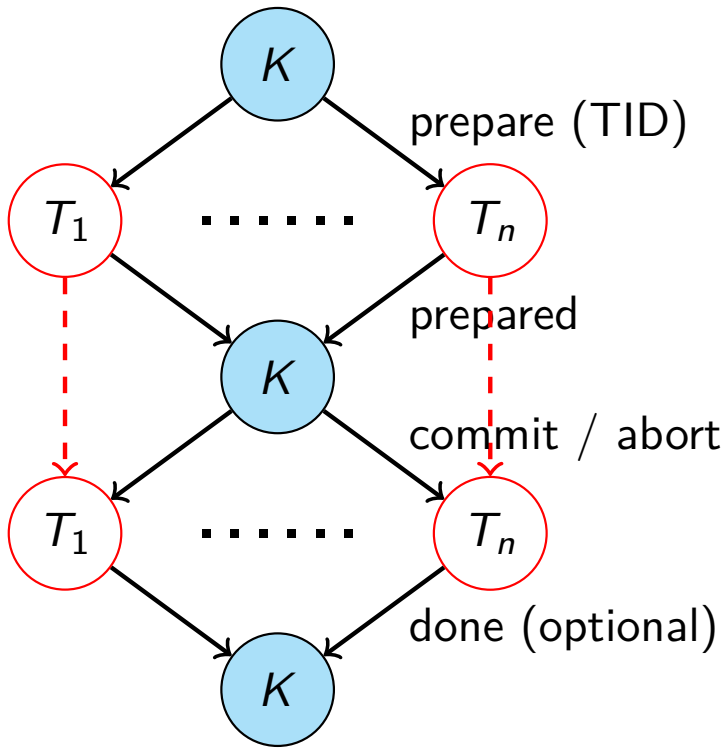


lokales Vorbereiten

Warten auf Entscheidung des Koordinators

Aktionen entsprechend Entscheidung ausführen

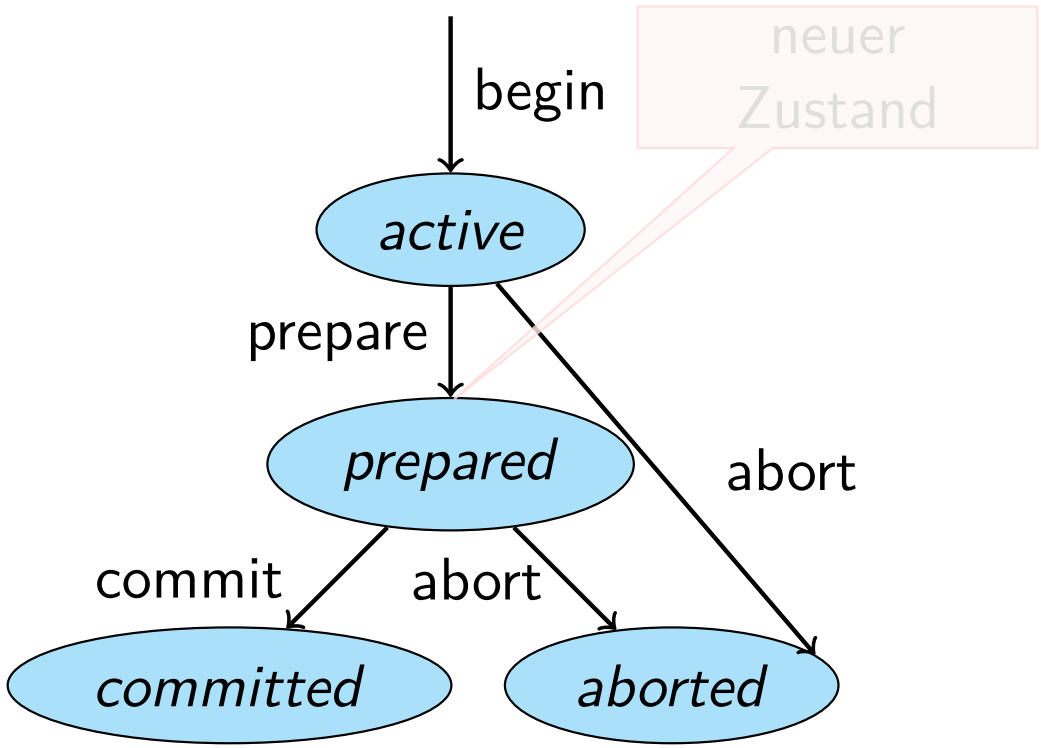
Ende mitteilen



entsprechend bei einseitigem Abort

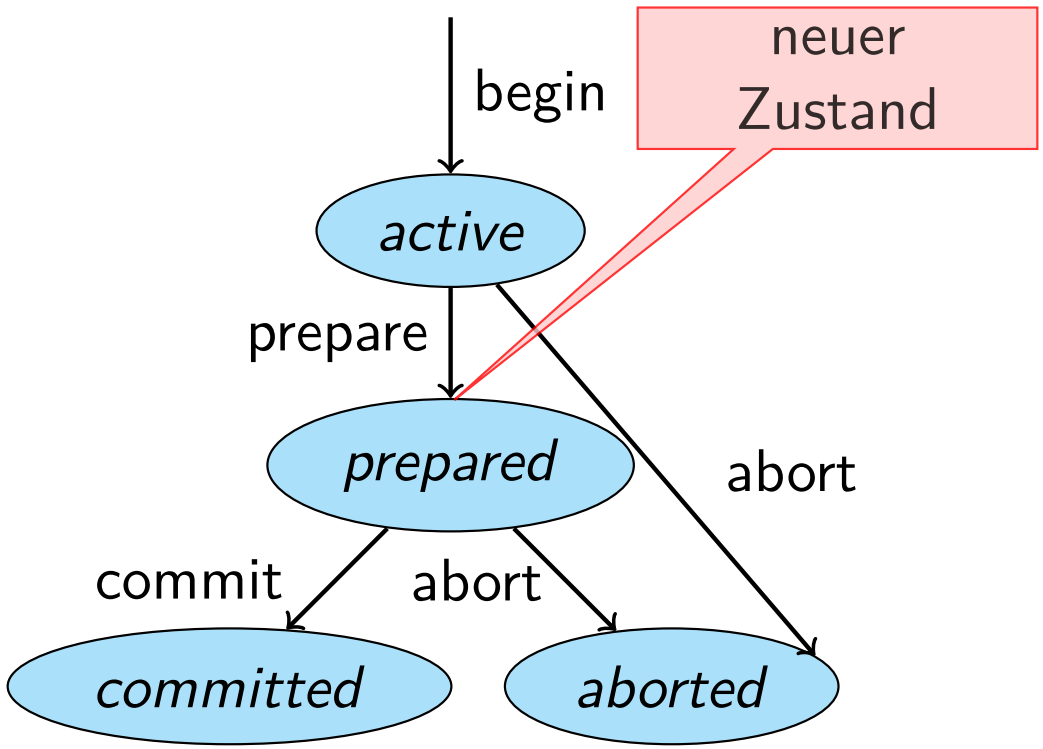
2-P-C-Protokoll (6)

zusammenfassendes Zustandsdiagramm für verteilte Transaktionen



2-P-C-Protokoll (6)

zusammenfassendes Zustandsdiagramm für verteilte Transaktionen



2-P-C-Protokoll (7)

Behandlung von Transaktionsfehlern

- Übergang von aktiven Transaktionen in den Zustand aborted
- unilaterale Abort-Entscheidung von Koordinator und jedem Teilhaber möglich
  - Koordinator sendet später abort, auch wenn alle Teilhaber mit prepared geantwortet haben
  - Teilhaber antworten auf prepare-Request des Koordinators mit prepared (failed).

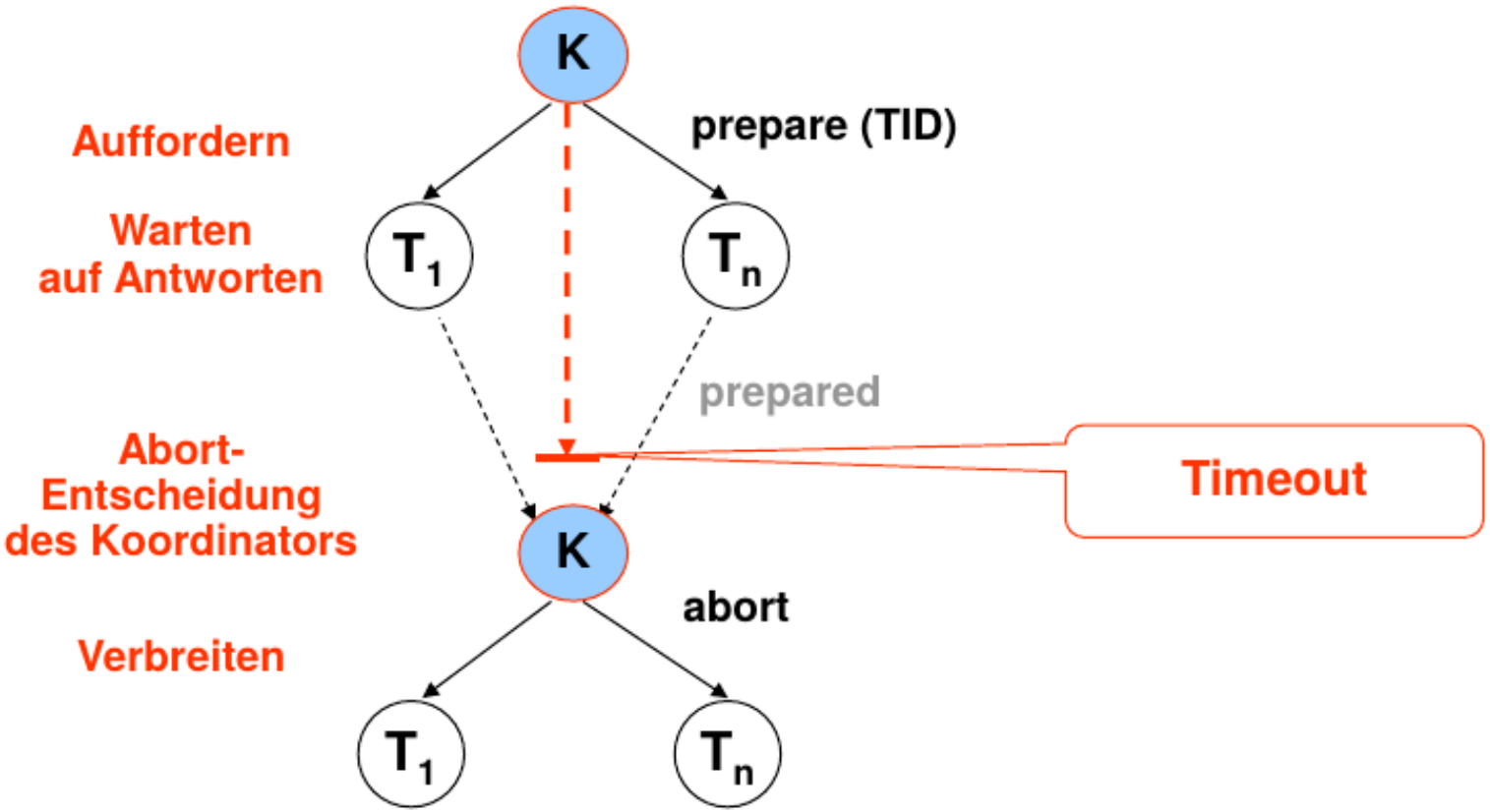
Behandlung von Kommunikationsfehlern bei aktiven Transaktionen

- Erkennung (wie auch anderer Vorkommnisse) durch Timeouts
- Überführung in Transaktionsfehler mit Behandlung wie oben

2-P-C-Protokoll (8)

Hochschule RheinMain

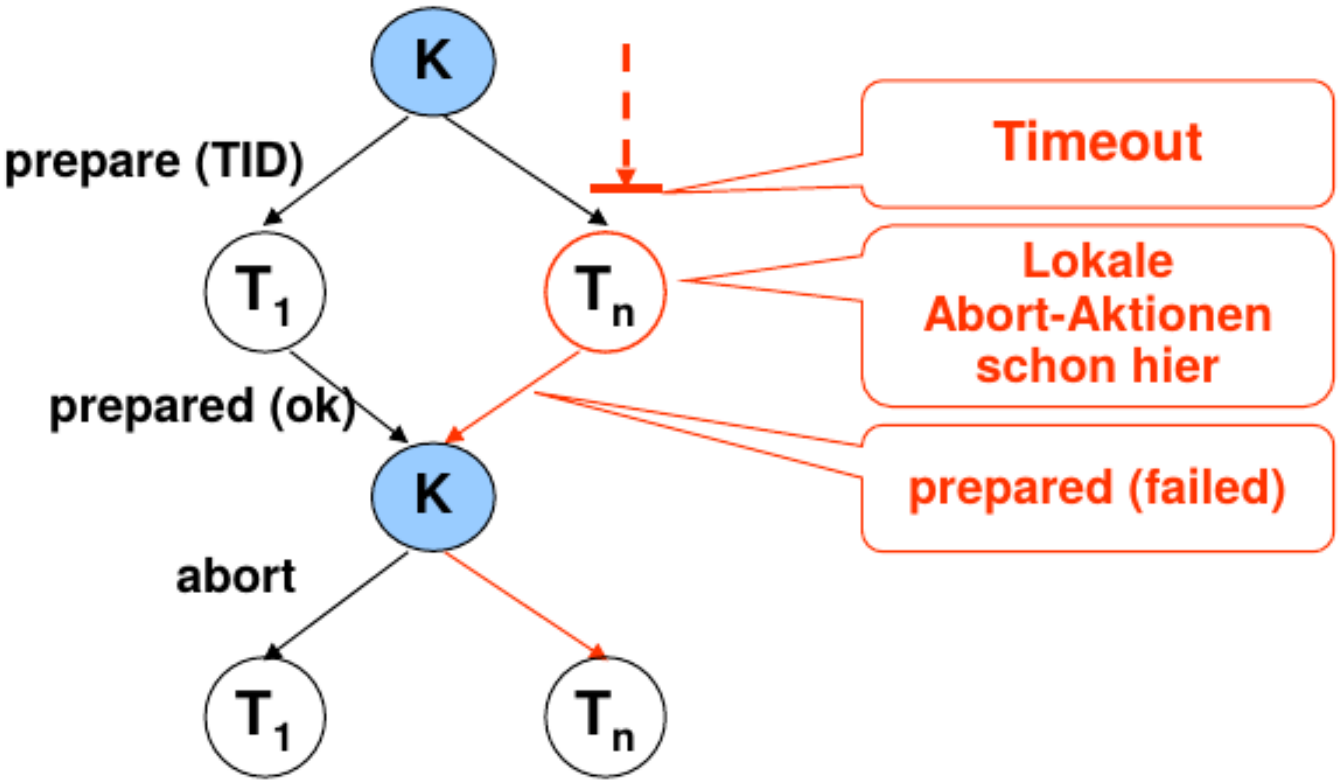
Beispiel für Abort-Entscheidung des Koordinators



2-P-C-Protokoll (9)

Hochschule RheinMain

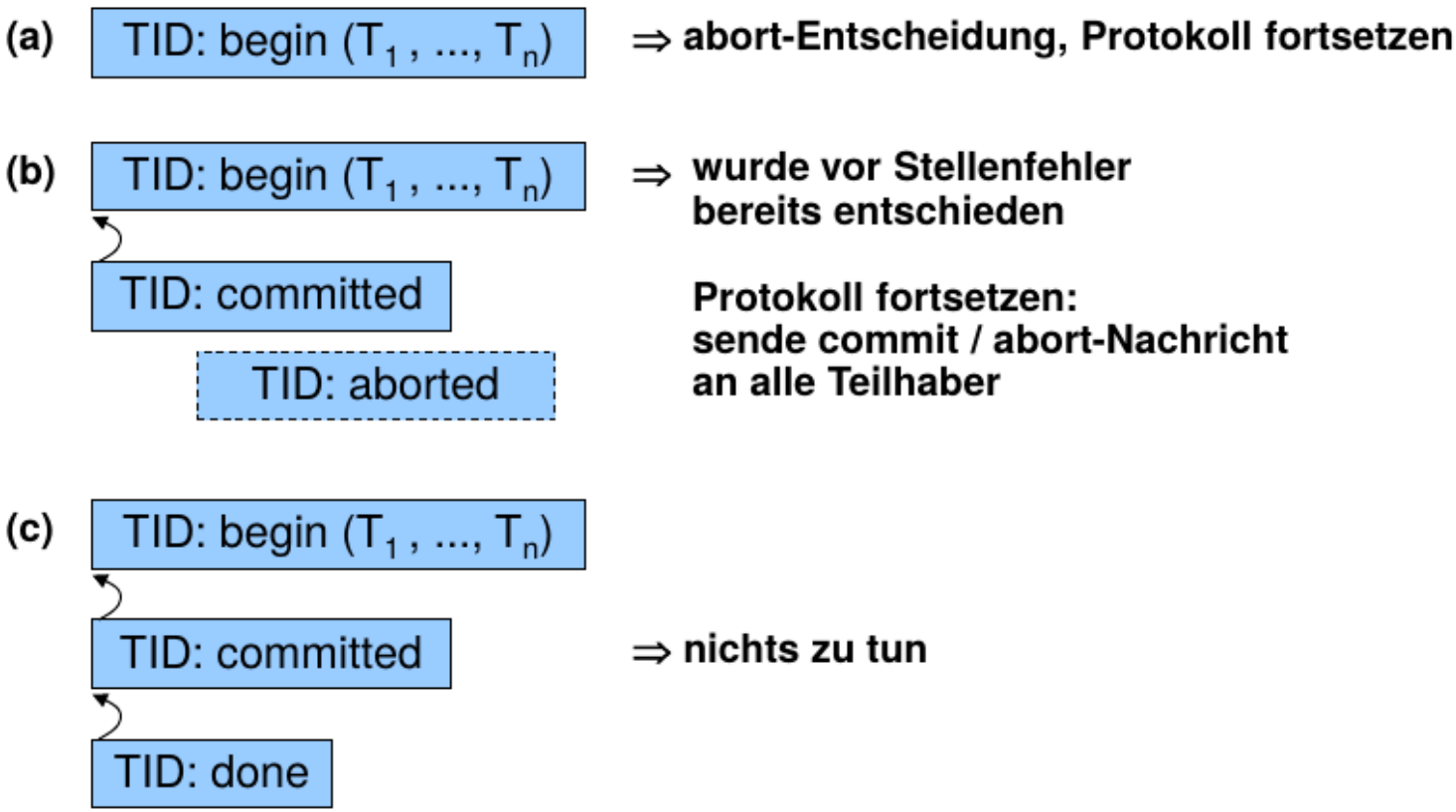
Beispiel für Abort-Entscheidung eines Teilhabers



2-P-C-Protokoll (10)

Behandlung eines Stellenfehlers des Koordinators

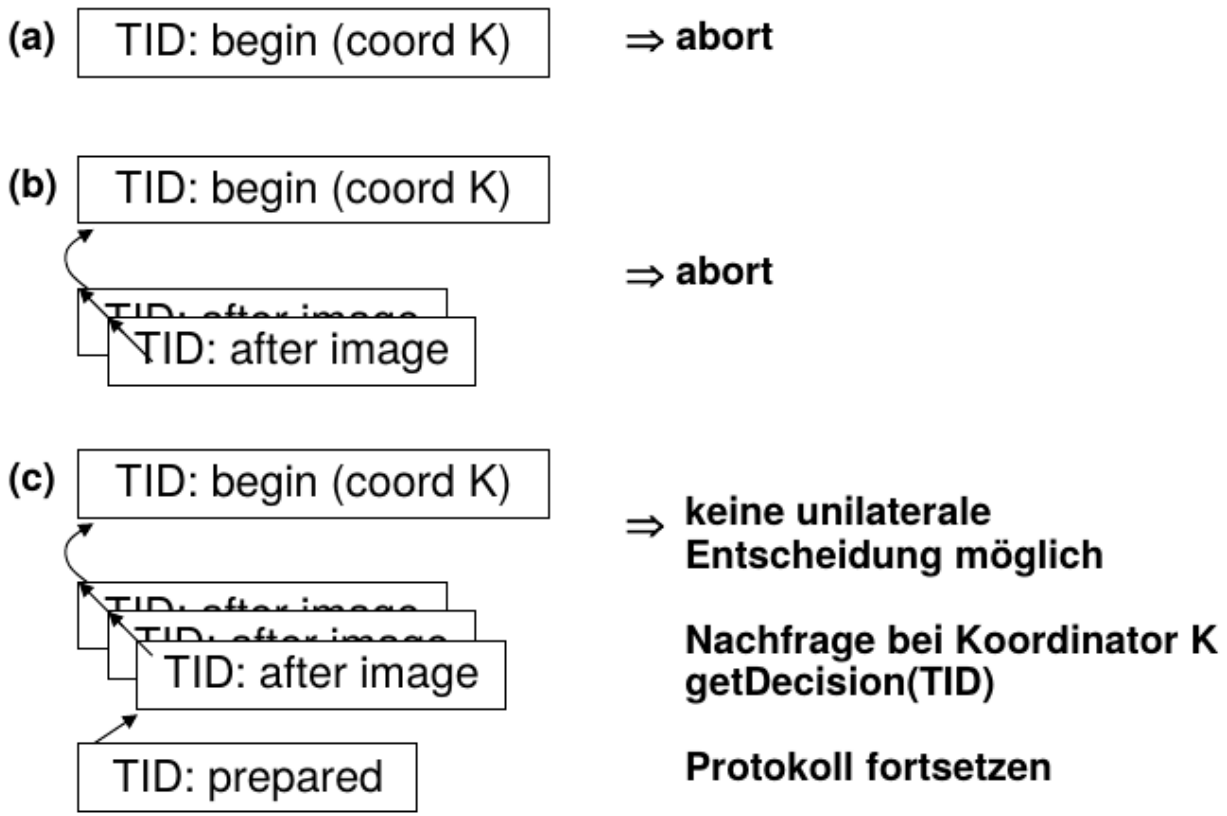
- Vorgehensweise abhängig von Information im Log



2-P-C-Protokoll (11)

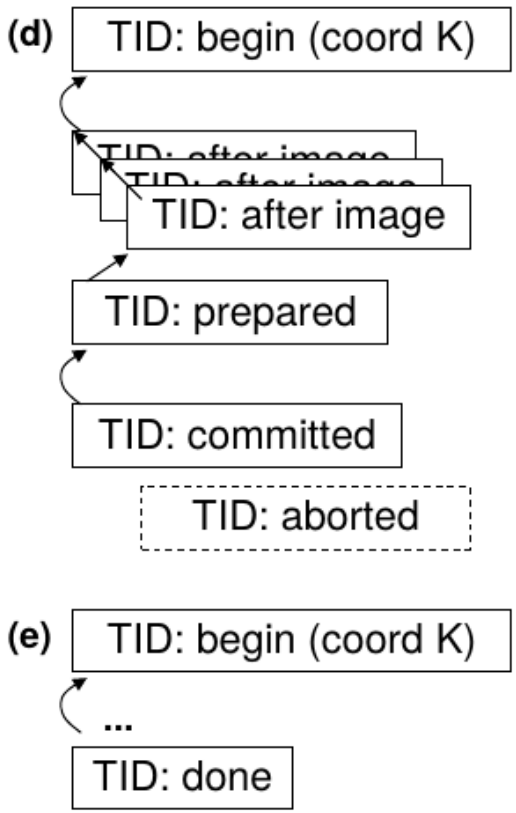
Behandlung eines Stellenfehlers eines Teilhabers

- Vorgehensweise ebenfalls abhängig von Information im Log



2-P-C-Protokoll (12)

Behandlung eines Stellenfehlers eines Teilhabers (2)



⇒ **unabhängiges Recovery möglich**

lokale commit / abort-Aktionen wiederholen

Protokoll fortsetzen

⇒ **nichts zu tun**

Erweiterungen

Coordinator Migration

- Die Koordinator-Rolle wird einem hochzuverlässigen Rechner übertragen
- Damit wird Blockierung der Teilhaber bei Ausfall des Koordinators unwahrscheinlicher

Group Commitment

- Gemeinsames Commitment mehrerer Transaktionen
- Weniger forced write-Operationen
- Durchsatzsteigerung bei geringfügig erhöhter Ausführungszeit des Protokolls

Kooperative Terminierung

- Teilhaber kennen sich gegenseitig
- Nach Stellenfehler kann Nachfrage bei anderen Teilhabern die Blockierung vermindern, falls jetzt Koordinator ausgefallen ist

Erweiterungen (2)

Presumed Abort / Presumed Commit

- Default-Wert für Ausgang der Transaktion angenommen, wenn kein spezifischer Outcome Record im Log vorhanden ist
- Vereinfachungen bei Log-Verkürzung möglich

Dezentrales 2-Phasen-Commit-Protokoll

- Kein zentraler Koordinator
- Kommunikation der Teilhaber untereinander, z.B. vorteilhaft über Broadcast-Protokoll
- Verringert Zeitkomplexität

Anmerkung: von Erweiterungen ist nur Coordinator Migration prüfungsrelevant

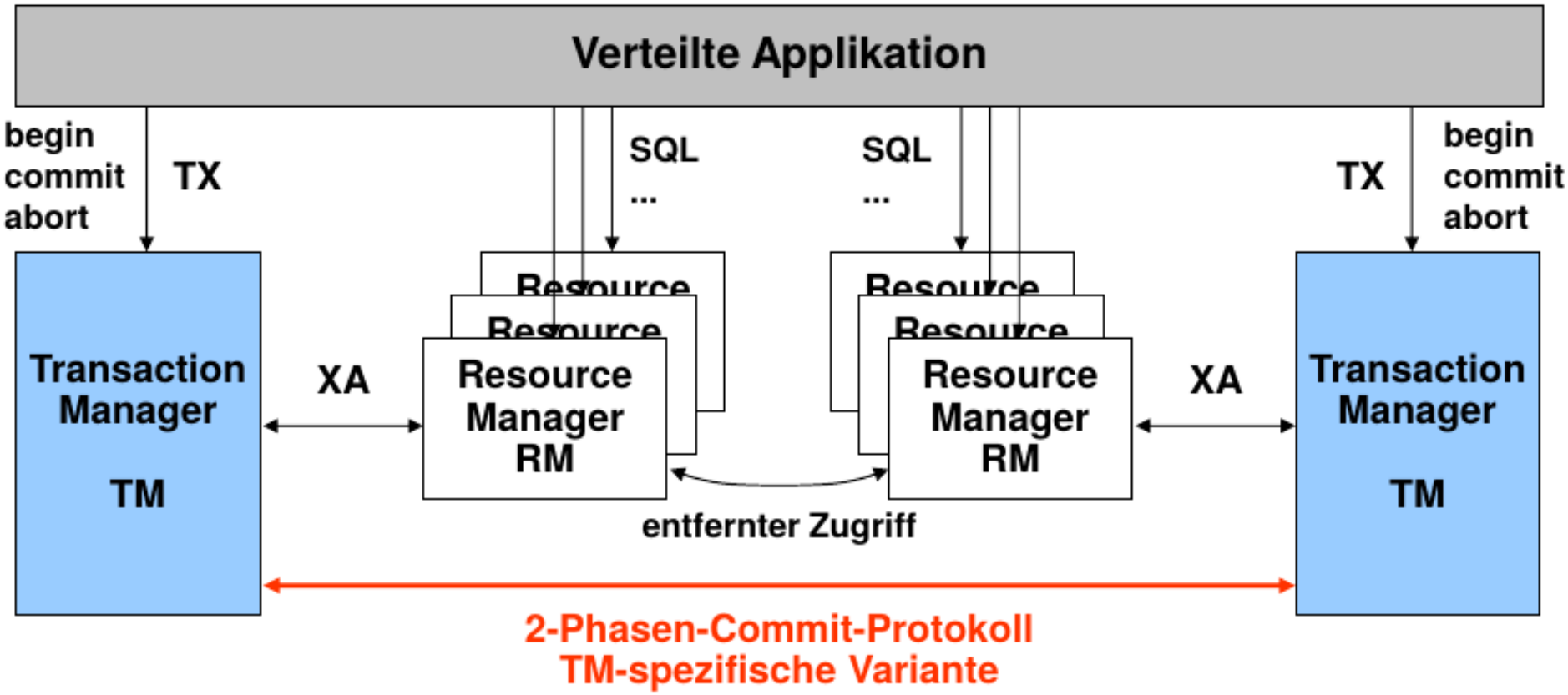
X/Open XA

- X/Open
  - Historisches Industriekonsortium zur Förderung offener Systeme
  - Bildet 1996 zusammen mit Open Software Foundation (OSF) die Open Group
- Modell für Transaktionsverarbeitung in offenen verteilten Umgebungen
- Überwindung von Hersteller-spezifischen Lösungen der Transaktionssteuerung
- Anwendung des 2-Phasen-Commit-Protokolls
- Kommerziell ausgerichtet auf RDBMSe
- Problem klassischer lokaler Datenbanksysteme
  - enge innere Verquickung der verschiedenen Funktionsbereiche
  - Unterstützung für verteilte Transaktionen erfordert Auftrennung zwischen Prepare-Aktivitäten und Commit-Entscheidung
  - Commit-Entscheidung muss auch extern getroffen werden können

Architekturüberblick

Hochschule RheinMain

Homogener Fall

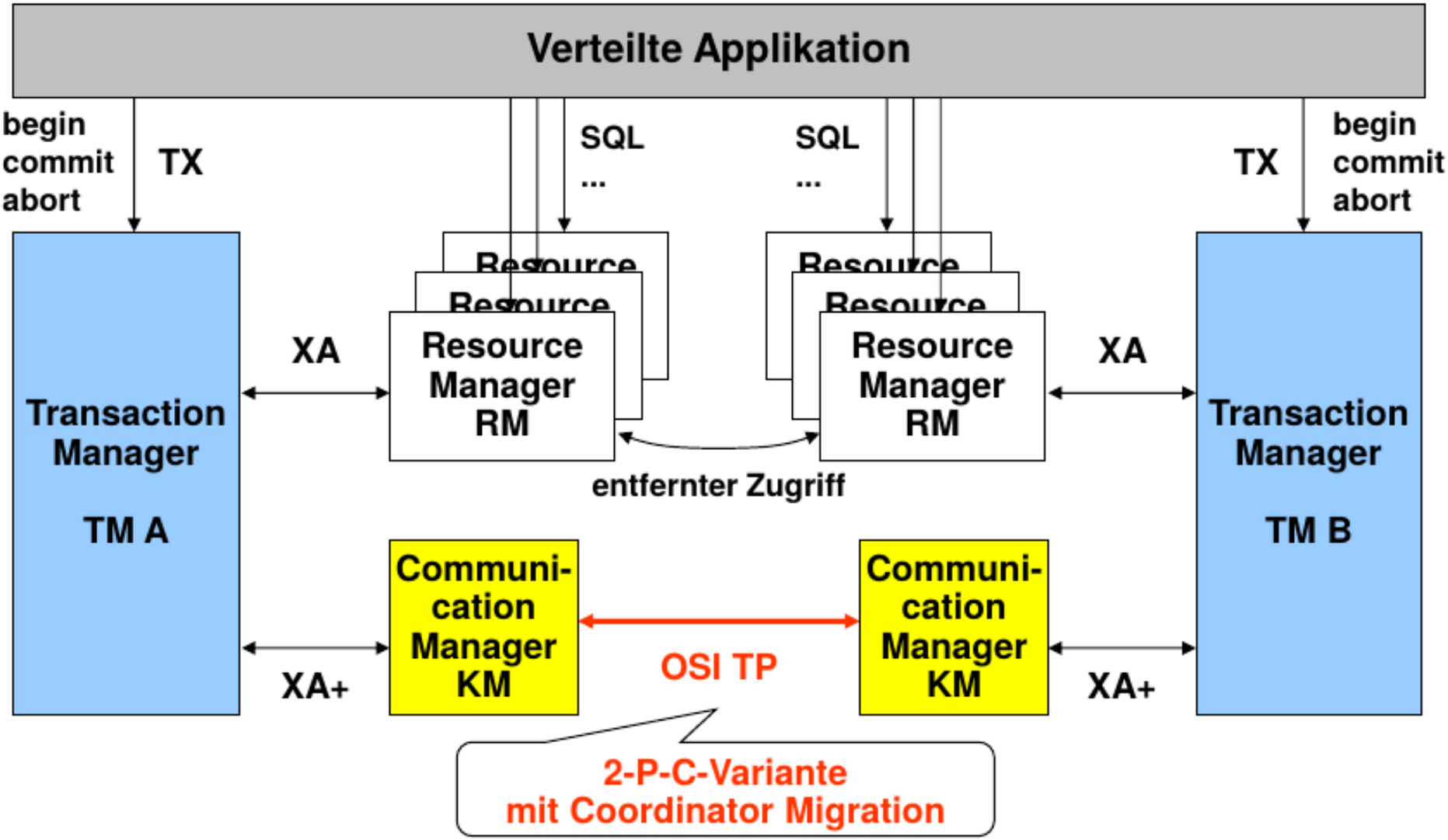


- RM kapseln log. Betriebsmittel (DB-Tupel, Dateien, Messages, Objekte, ...)
- TM kapselt Transaktionssteuerung
- für identische TMs, unterschiedliche RMs

Architekturüberblick (2)

Hochschule RheinMain

Heterogener Fall



Beispiele

X/Open DTP-konforme Transaction Manager

- Tuxedo
  - historischer De-Facto-Standard
  - von USL, jetzt BEA, jetzt Oracle
  - nur flache Transaktionen
- Encina
  - unterstützt auch genestete Transaktionen
  - nutzt OSF DCE
  - von Transarc (Ausgründung CMU), jetzt eingegliedert in IBM
- CICS/6000
  - CICS: IBM Host-Standard
  - Encina für AIX (RS 6000)
- NCR Top End

Beispiele

Transaction Manager in CORBA- und J2EE-Produkten

- BEA Weblogic: Tuxedo
- IONA Orbix: Encina
- Heute i.d.R. in Applikationsserver mit integriert

X/Open DTP-konforme Resource Manager (XA-Schnittstelle)

- Relationale DBMSe
  - historisch erstes System mit XA-Schnittstelle: Informix
  - heute von allen groSSen DB-Systemen unterstützt
  - selbst bei Microsoft (MS DTC)
- Message Queueing Systeme
  - MQ Series
  - Swift MQ
- Dateisysteme
  - ISAM XA (Gresham Computing)

Zusammenfassung

Hochschule RheinMain

- Transaktionen dienen der reduzierten Komplexität verteilter Anwendungen.
- Das zugehörige Fehlermodell muss zwischen Transaktions-, Stellen-, Medien- und Kommunikationsfehlern unterscheiden.
- Das 2-Phasen-Commit-Protokoll ist das am meisten eingesetzte Commit-Protokoll.