Conflict Free Data Replication in Actor Systems

Marian Triebe
marian.triebe@haw-hamburg.de

iNET RG, Department of Informatik
Hamburg University of Applied Sciences

16. März 2016

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

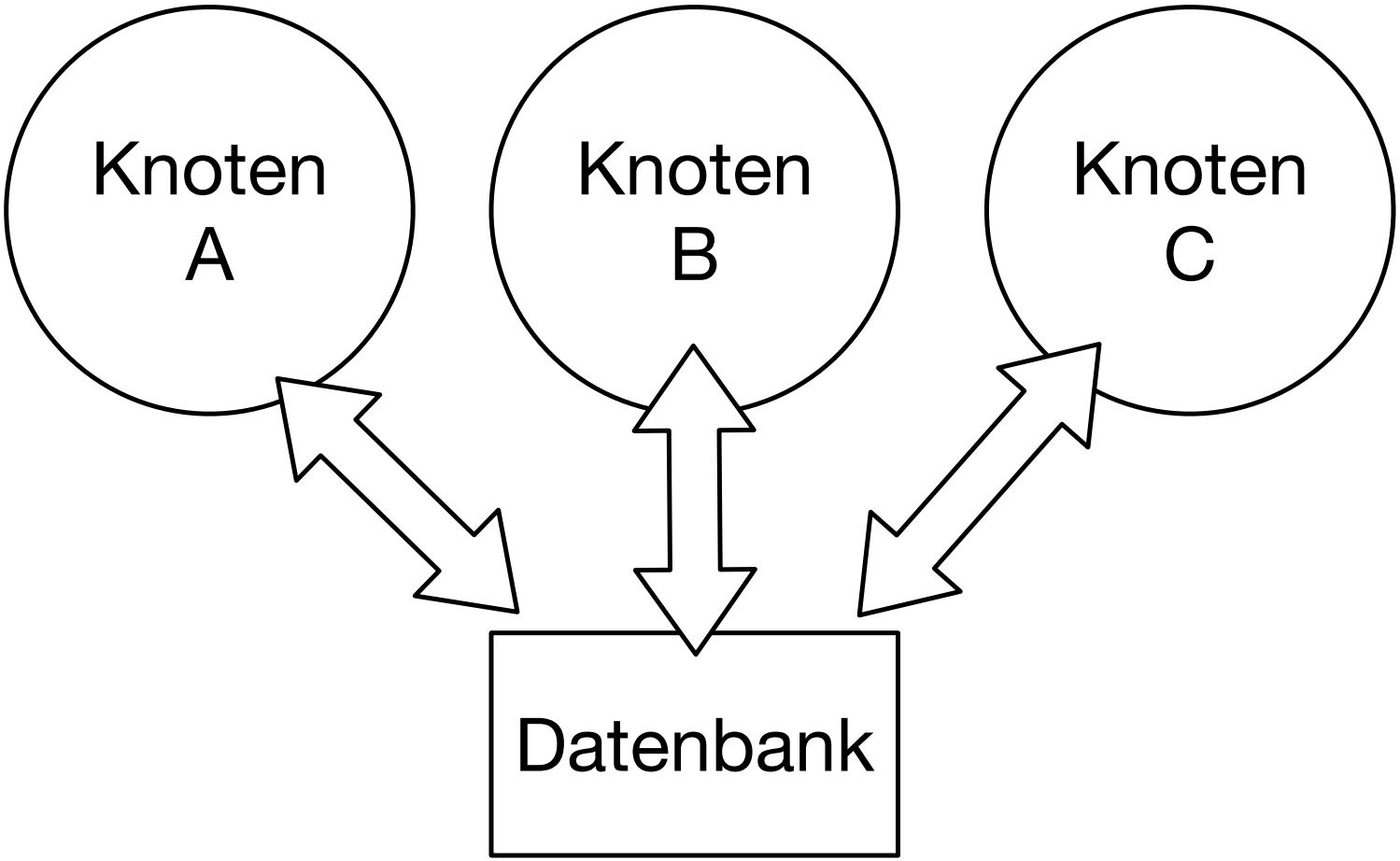
Übersicht

Agenda

Kontext

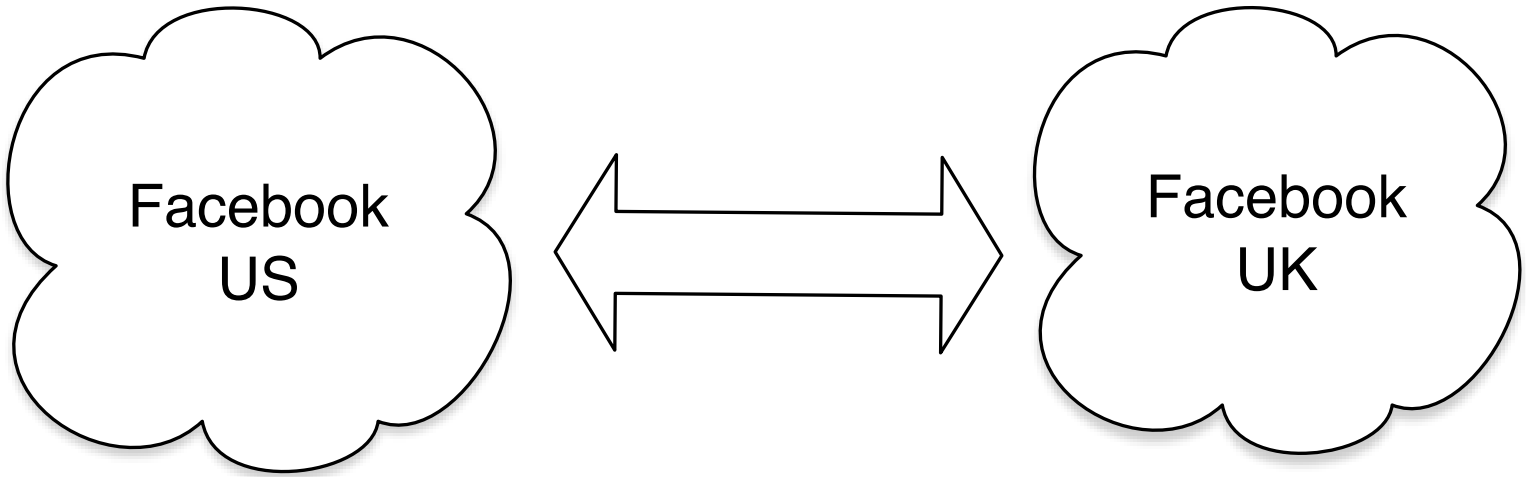
- CAF ist eine Platform für nebenläufige & verteilte Programmierung
- Generelles Problem konsistenter Datenstrukturen in Verteilten Systemen
- Zentrale Datenhaltung
- Vorstellung algorithmischer Lösung

Problem



Use Case

Weit verteilte Anwendung



- Posts müssen nicht sofort für jeden sichtbar sein
- Zentraler Ansatz skaliert schon nicht im Datenzentrum

Lösungsansatz

- Anforderungen:
  1. Knoten müssen reaktiv sein
  2. Konsistente Sicht auf alle Daten
  3. Globale Datenkonsistenz, müssen nicht jederzeit synchronisiert sein
- Probleme:
  1. Komplexität der Konvergenz
  2. Komplexität für Programmierer (Anwender)

Agenda

CRDT

- "Conflict Free Replicated Data Type" (CRDT)
- Kein zentraler Ansatz
- "Strong eventual consistency" (SEC)
- Zwei Konvergenzstrategien
  - State-Updates
  - δ-Updates

Eventual consistency

- Eventual consistency (EC)
  - Knoten dürfen eigene Sicht auf Daten haben
  - Stellt nur sicher, dass Daten konsistent bleiben
  - Wenn lange nichts passiert, haben alle irgendwann den gleichen State
- Strong eventual consistency (SEC)
  - Sicherheitsgarantie

Sicherheitsgarantie

Zwei Knoten, die die selben Updates erhalten haben (in beliebiger Reihenfolge), haben den selben Zustand.

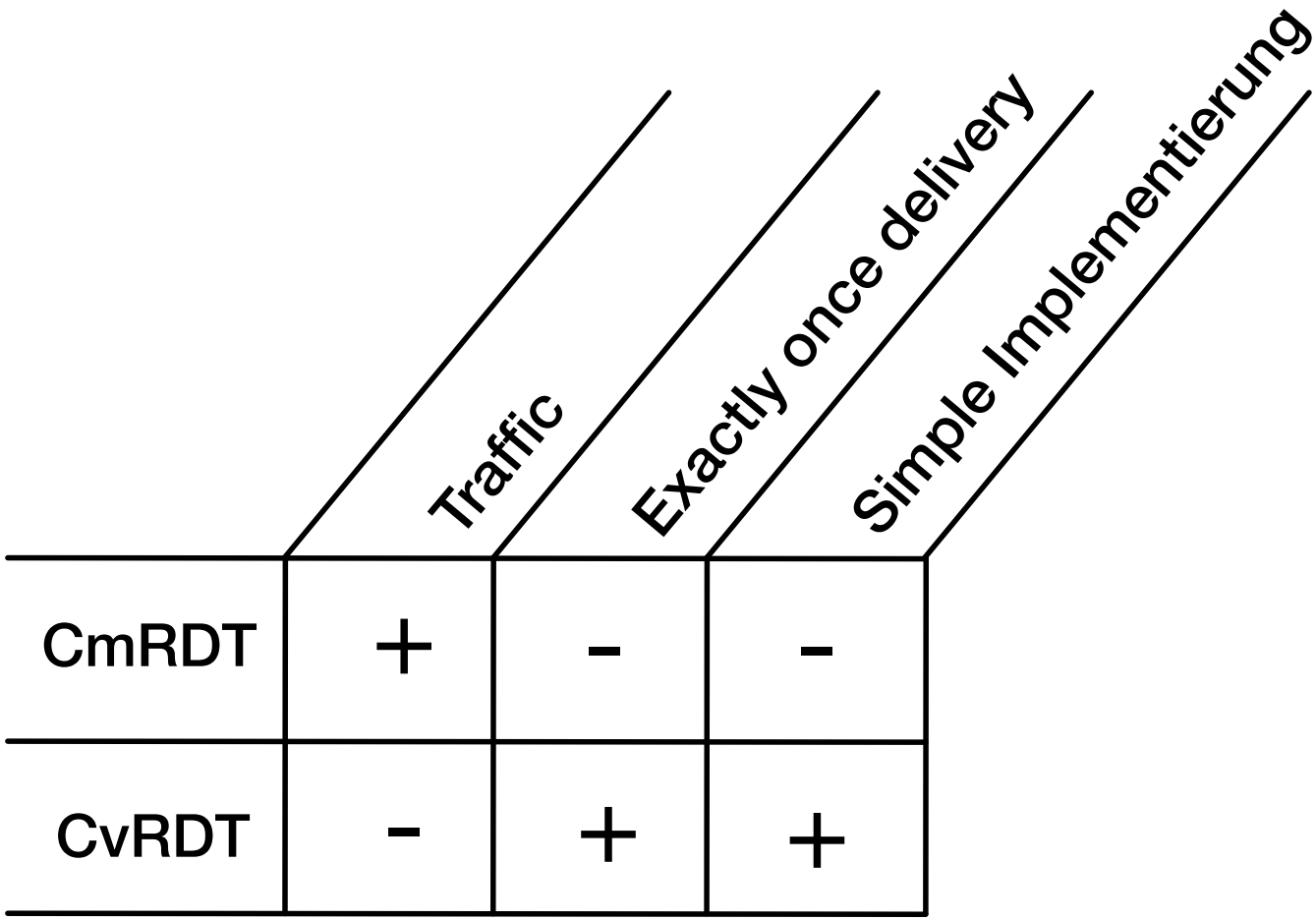
CmRDT vs. CvRDT

- δ-Updates
  - "Commutative replicated data types" (CmRDT)
  - Operation-Based
  - Updates sind kommutativ
- State-Updates
  - "Convergent replicated data types" (CvRDT)
  - State-Based
  - Merge-/Join-Funktion

- Exactly once delivery
  - δ-*Updates*: muss garantiert sein (CmRDT)
  - *State-Updates*: muss **nicht** garantiert sein (CvRDT)

Vergleich von

δ-updates & state-updates



CRDT Eigenschaften

- Addition & Vereinigung können natürlich abgebildet werden
- CmRDT:
  - Updates müssen kommutativ sein
- CvRDT:
  - Updates müssen assoziativ, kommutativ & idempotent sein

Arten von CRDTs

- Register
- Zähler
- Mengen
- Graphen & Bäume

Arten von CRDTs

- Register
  - LWW-Register $\rightarrow$ *Last-Writer-Wins-Register*
  - MV-Register $\rightarrow$ *Multi-Value-Register*
- Zähler
- Mengen
- Graphen & Bäume

Arten von CRDTs

- Register
- Zähler
  - GCounter → *Grow-Only-Counter*
  - PN-Counter → *Positive-Negative-Counter*
- Mengen
- Graphen & Bäume

Arten von CRDTs

- Register
- Zähler
- Mengen
  - GSet $\rightarrow$ *Grow-Only-Set*
  - 2P-Set $\rightarrow$ *Two-Phase-Set*
  - LWW-Set $\rightarrow$ *Last-Writer-Wins-Set*
  - PN-Set $\rightarrow$ *Positive-Negative-Set*
  - OR-Set $\rightarrow$ *Observed-Remove-Set*
- Graphen & Bäume

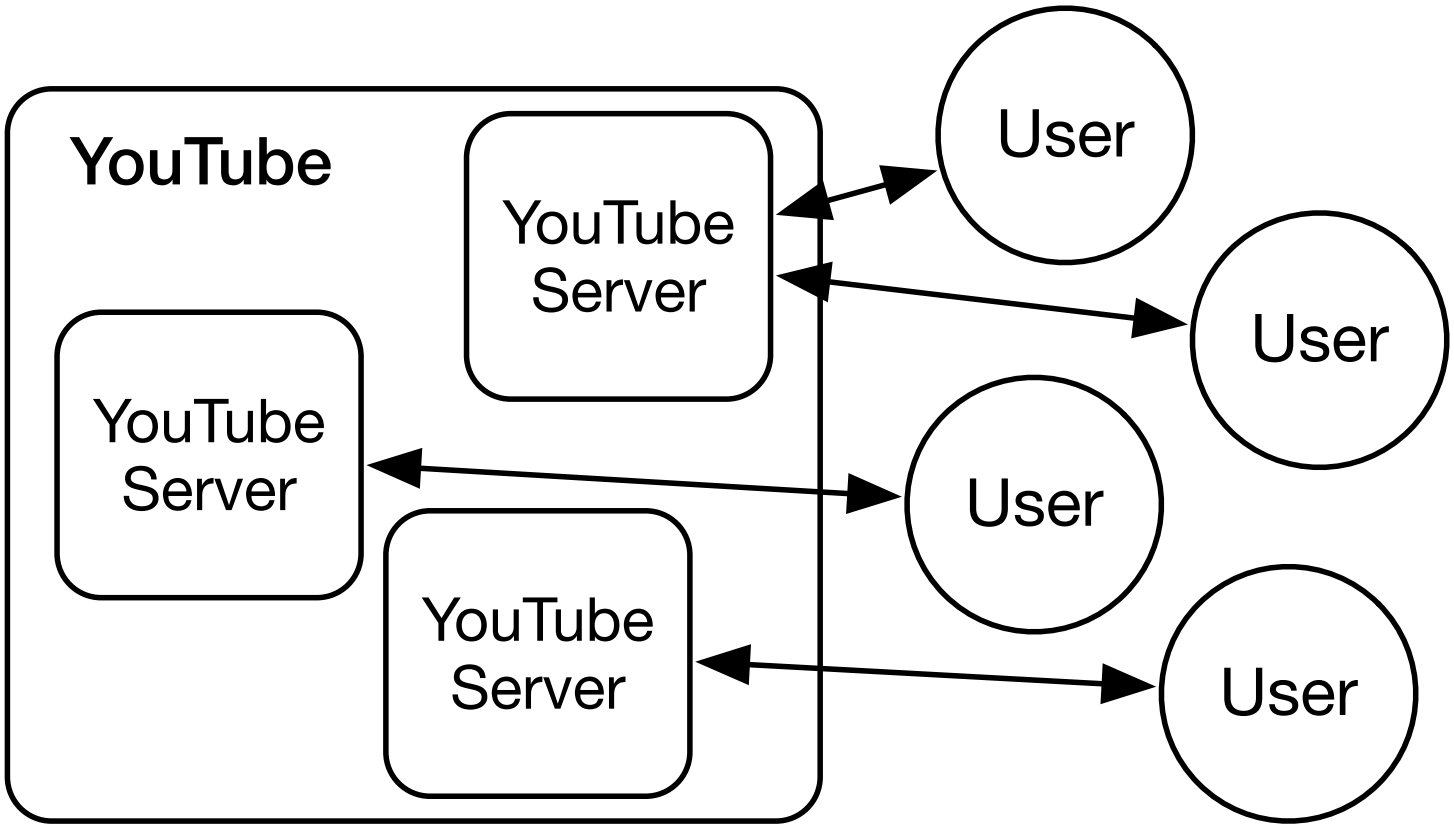
Arten von CRDTs

- Register
- Zähler
- Mengen
- Graphen & Bäume
  - 2P2P-Graph $\rightarrow$ *Two-Phase-Two-Phase-Graph*
  - DAC $\rightarrow$ *Add-only-monotonic-DAC*

GCounter

Use Case

- Beispiel: YouTube views
  - Gleiches Video auf verschiedenen Systemen
  - CRDT: Grow-Only-Counter (GCounter)



GCounter

Parameter

Parameter

- P := Container aller Werte (lokal)
- X := Eingehender Container (Update)
- n := Anzahl der Knoten
- id := Eigener Slot im Container

GCounter

Funktion

Initial gilt:

$\forall_i \in [1...n] : P_i = 0$

GCounter

Funktion

Initial gilt:

$\forall_i \in [1...n] : P_i = 0$

Join/Merge:

$\forall_i \in [1...n] : P_i := max(P_i, X_i)$

GCounter

Funktion

Initial gilt:

$\forall_i \in [1...n] : P_i = 0$

Join/Merge:

$\forall_i \in [1...n] : P_i := max(P_i, X_i)$

Totaler Wert:

$\sum_{i=1}^{n} P_i$

GCounter

Funktion

Initial gilt:

$\forall_i \in [1...n] : P_i = 0$

Join/Merge:

$\forall_i \in [1...n] : P_i := max(P_i, X_i)$

Totaler Wert:

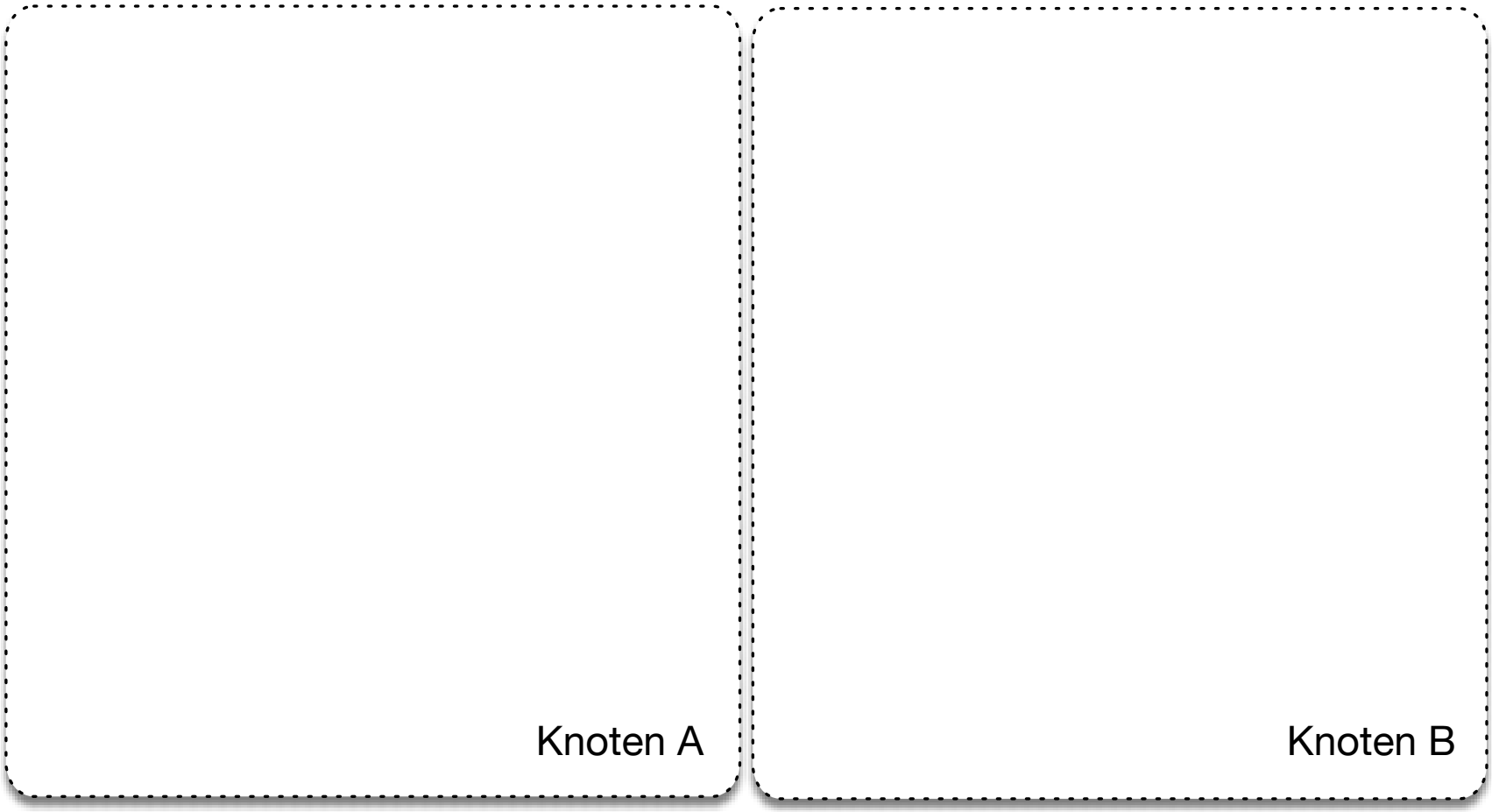
$\sum_{i=1}^{n} P_i$

Lokale Änderung von x:

$P_{id} := P_{id} + x$, für $x >= 0$

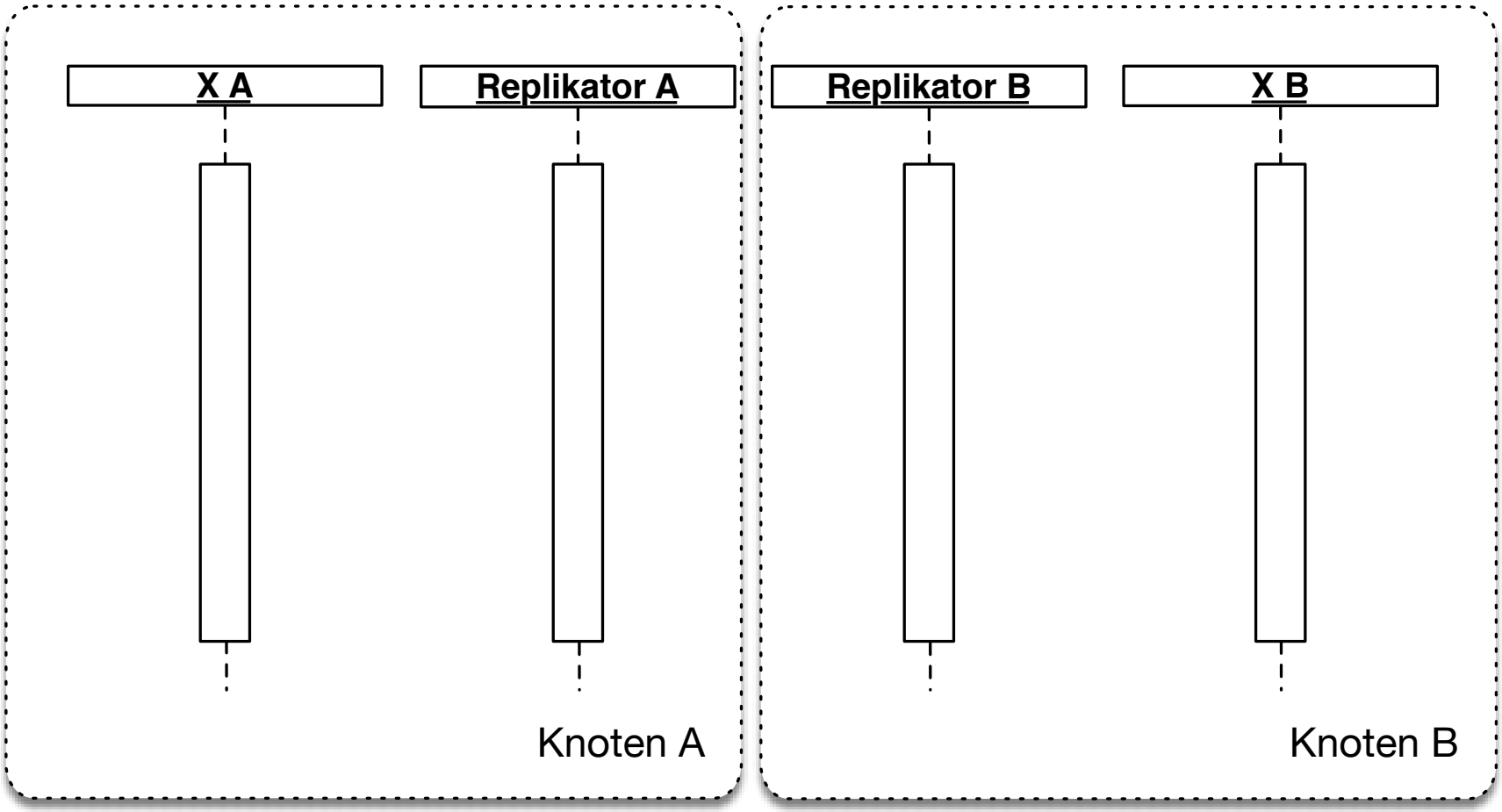
GCounter

Sequenzdiagram



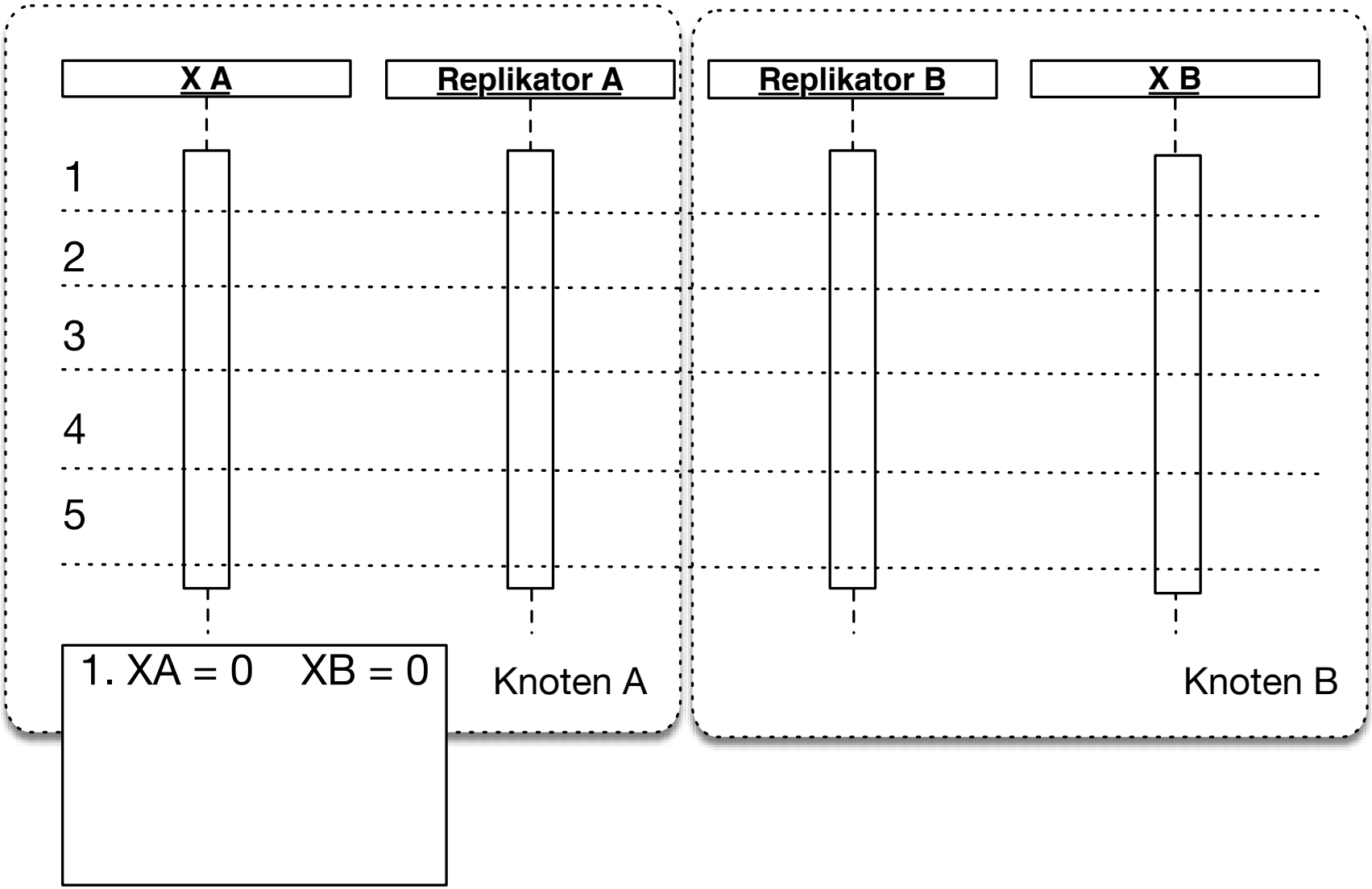
GCounter

Sequenzdiagram



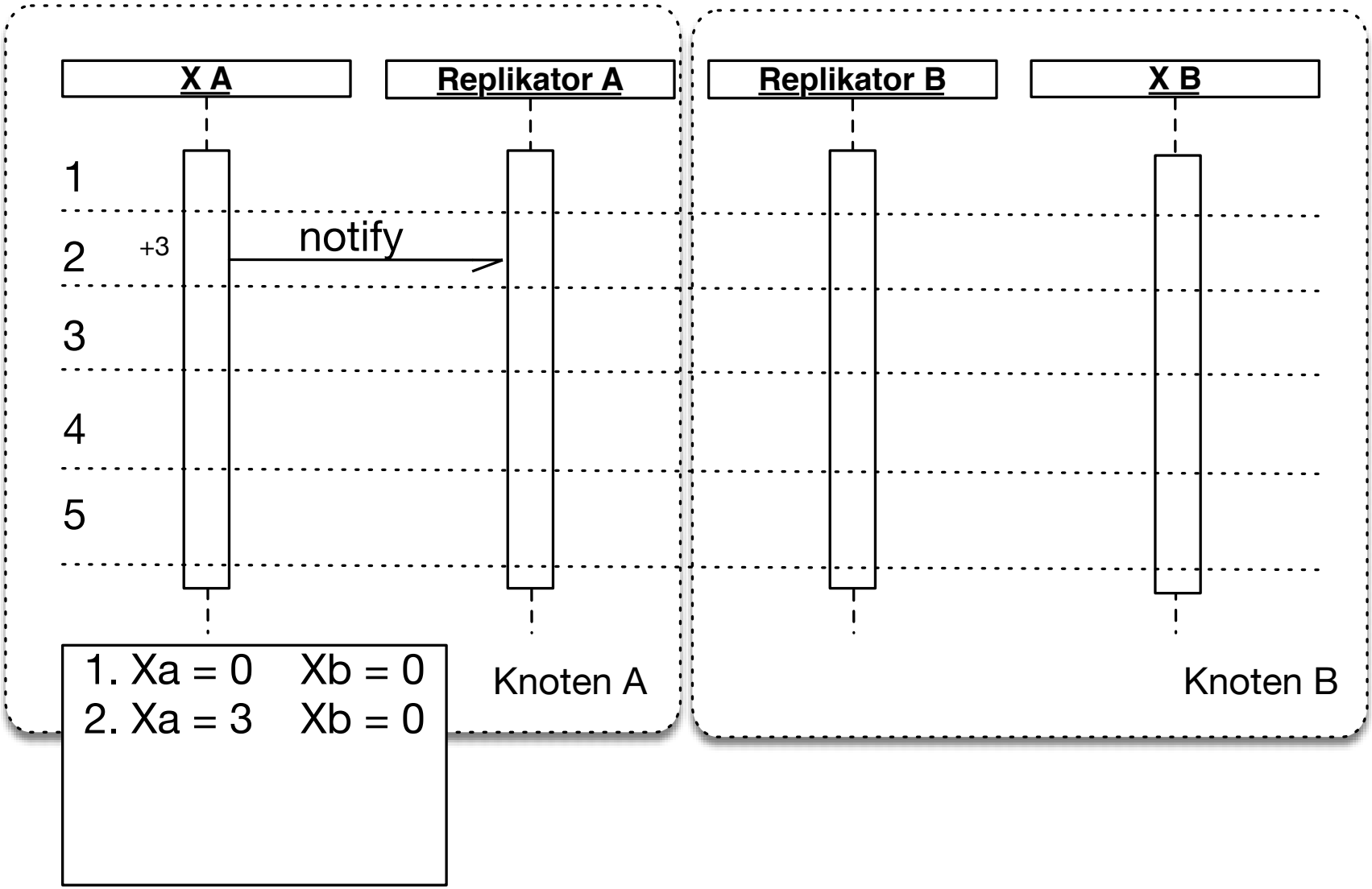
GCounter

Sequenzdiagram



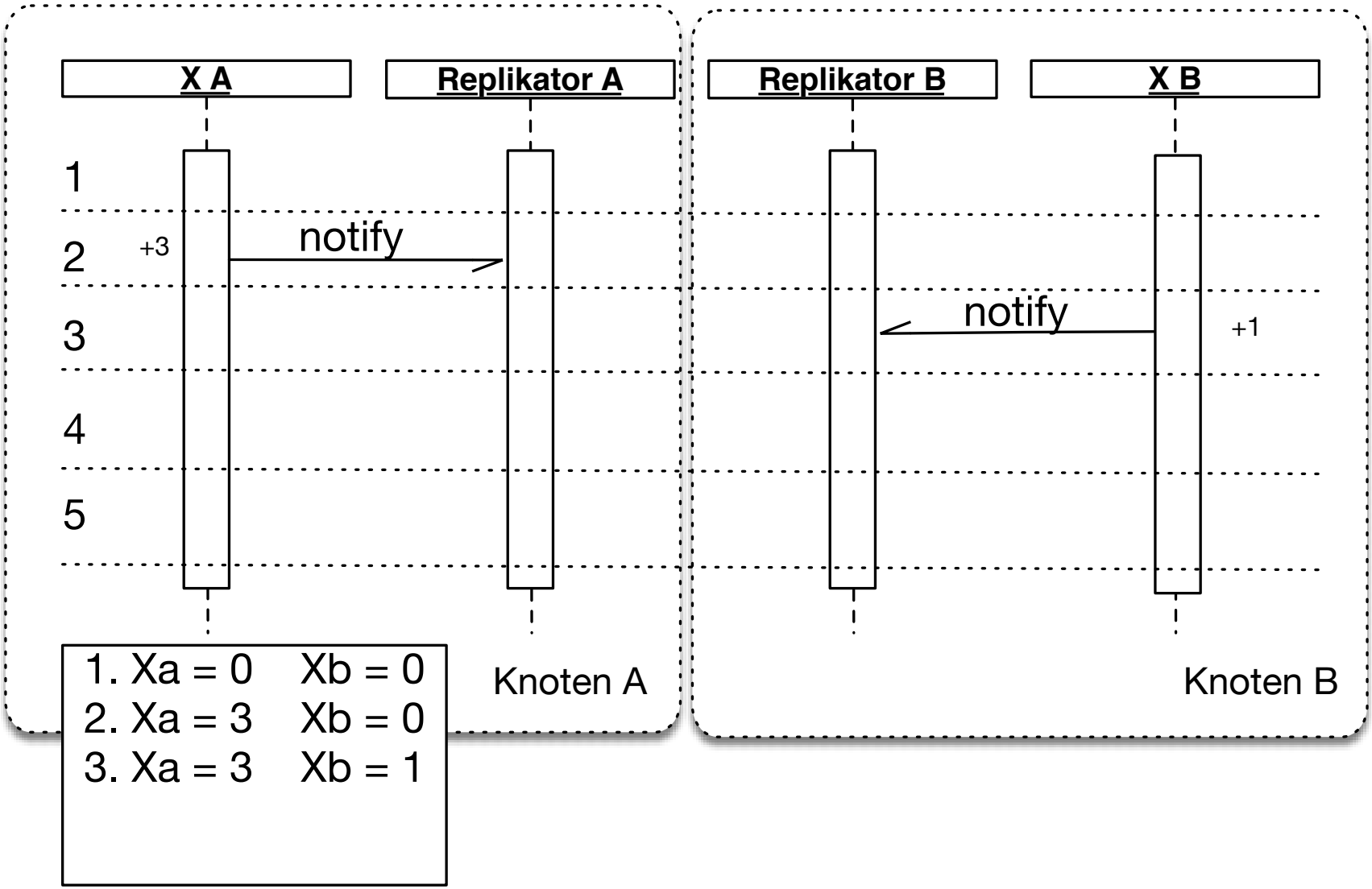
GCounter

Sequenzdiagram



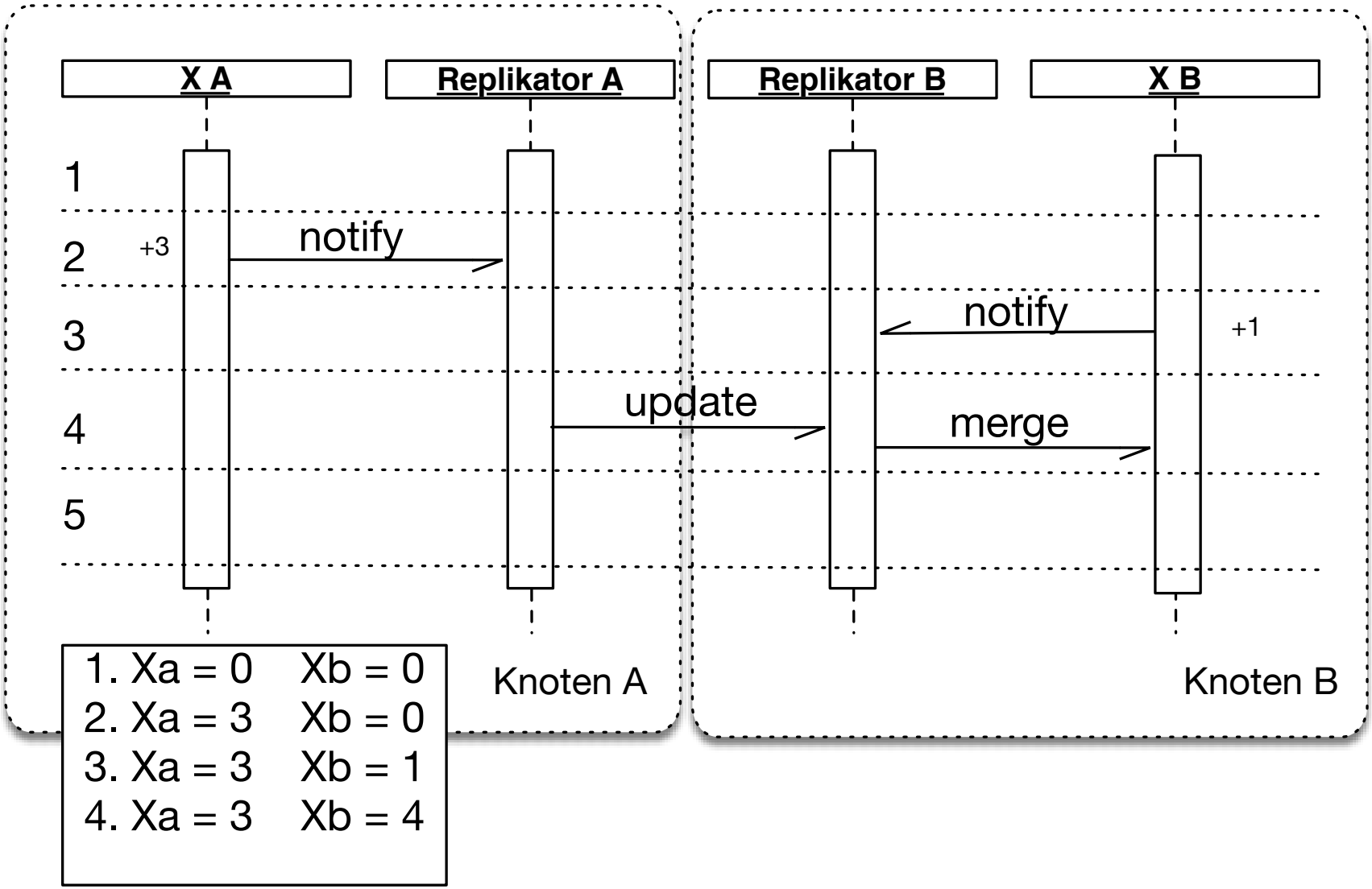
GCounter

Sequenzdiagram



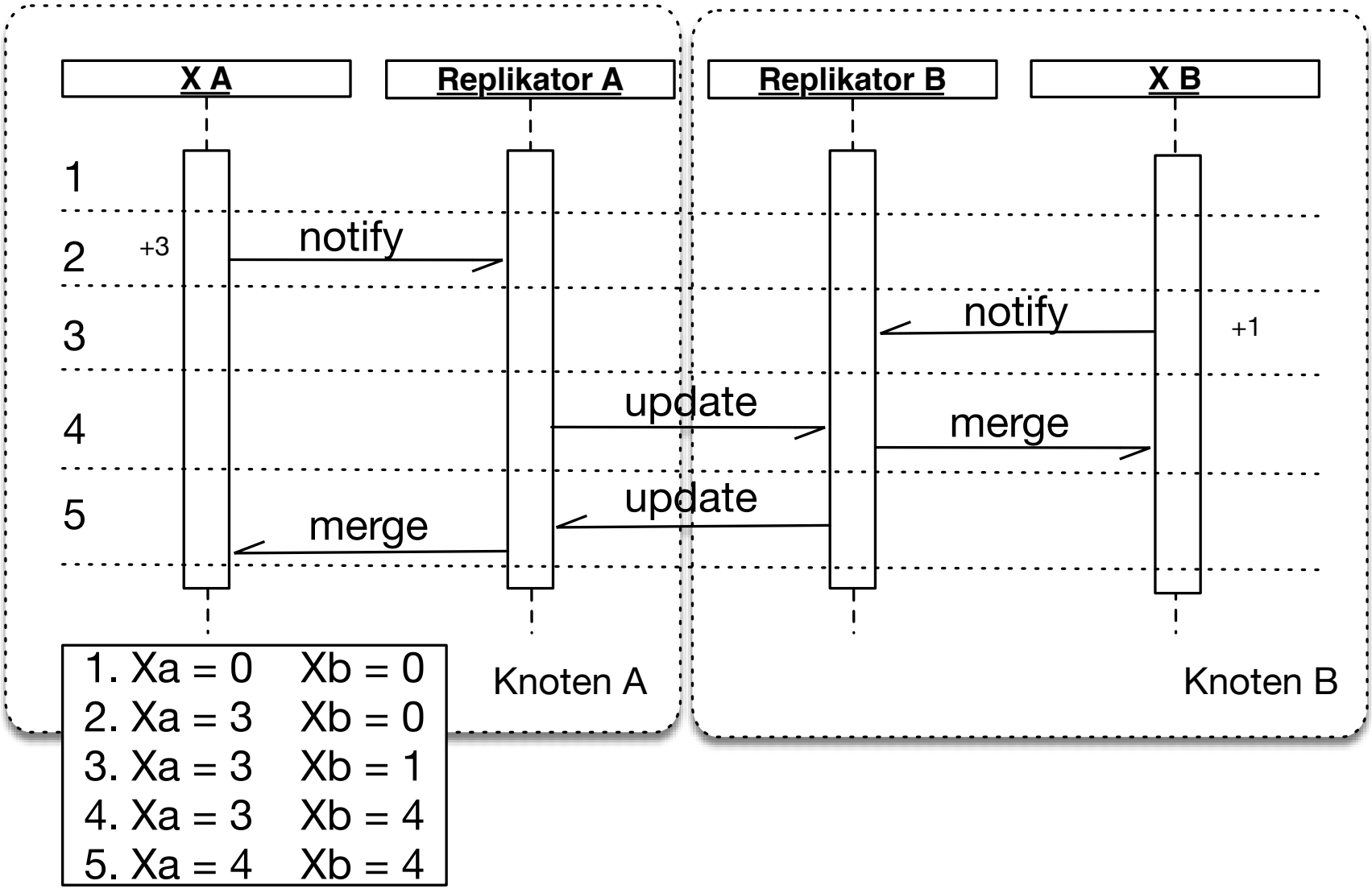
GCounter

Sequenzdiagram



GCounter

Sequenzdiagram



PN-Counter

Use Case

- Beispiel: Stackoverflow
  - Fragen und Antworten können mit +/- bewertet werden
  - Zähler können negativ werden
- Eigenschaften:
  - Kann nicht $>= 0$ sichern

PN-Counter

Parameter

P := Container positiver Werte

N := Container negativer Werte

X_P := Eingehender positiver Container

X_N := Eingehender negativer Container

n := Anzahl der Knoten

id := Eigener Slot im Container $\rightarrow id \in [1...n]$

PN-Counter

Funktion

Initial gilt:

$\forall_i \in [1...n] : P_i = N_i = 0$

PN-Counter

Funktion

Initial gilt:

$\forall_i \in [1...n] : P_i = N_i = 0$

Join/Merge:

$\forall_i \in [1...n] : P_i := max(P_i, X_{Pi})$
$\forall_i \in [1...n] : N_i := max(N_i, X_{Ni})$

PN-Counter

Funktion

Initial gilt:

$\forall_i \in [1...n] : P_i = N_i = 0$

Join/Merge:

$\forall_i \in [1...n] : P_i := max(P_i, X_{Pi})$
$\forall_i \in [1...n] : N_i := max(N_i, X_{Ni})$

Totaler Wert:

$\sum_{i=1}^{n} P_i - \sum_{i=1}^{n} N_i$

PN-Counter

Funktion

Initial gilt:

$\forall_i \in [1...n] : P_i = N_i = 0$

Join/Merge:

$\forall_i \in [1...n] : P_i := max(P_i, X_{Pi})$
$\forall_i \in [1...n] : N_i := max(N_i, X_{Ni})$

Totaler Wert:

$\sum_{i=1}^{n} P_i - \sum_{i=1}^{n} N_i$

Lokale Änderung von x:

$P_{id} := P_{id} + x$, für $x > 0$
$N_{id} := N_{id} + |x|$, für $x < 0$

GSet

Use Case

- Beispiel: Facebook-Likes
  - Menge von Personen, die einen Beitrag jemals "geliked" haben

GSet

Parameter

P := Lokaler Container

X := Eingehender Container

GSet

Funktion

Initial gilt:

$P = \emptyset$

GSet

Funktion

Initial gilt:

$P = \emptyset$

Join/Merge:

$P := P \cup X$

GSet

Funktion

Initial gilt:

$P = \emptyset$

Join/Merge:

$P := P \cup X$

Element x enthalten?

$x \in P$

GSet

Funktion

Initial gilt:

$P = \emptyset$

Join/Merge:

$P := P \cup X$

Element x enthalten?

$x \in P$

Lokales hinzufügen von x

$P := P \cup \{x\}$

2P-Set

Use Case

- Problem:
  - Abgeschlossene und ausstehende Bestellungen

2P-Set

Parameter

A := Container hinzugefügter Werte

R := Container entfernter Werte

X_A := Eingehender Container hinzugefügter Werte

X_R := Eingehender Container entfernter Werte

2P-Set

Funktion

Initial gilt:

$A = R = \emptyset$

2P-Set

Funktion

Initial gilt:

$A = R = \emptyset$

Join/Merge:

$A := A \cup X_A$

$R := R \cup X_R$

2P-Set

Funktion

Initial gilt:

$A = R = \emptyset$

Join/Merge:

$A := A \cup X_A$
$R := R \cup X_R$

Element x enthalten?

$(x \in A) \bigwedge (x \notin R)$

2P-Set

Funktion

Initial gilt:

$A = R = \emptyset$

Join/Merge:

$A := A \cup X_A$
$R := R \cup X_R$

Element x enthalten?

$(x \in A) \bigwedge (x \notin R)$

Lokales entfernen von x

Vorbedingung: $x \in A$
$R := R \cup \{x\}$

Agenda

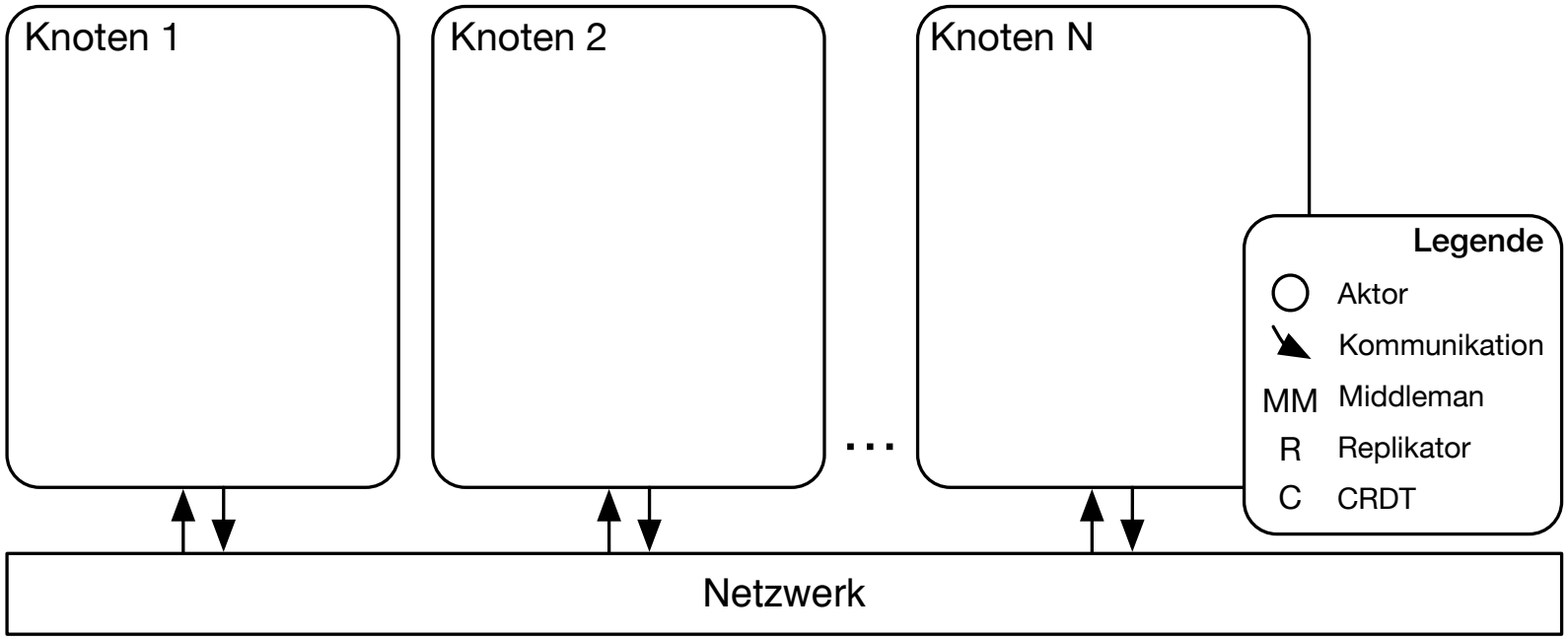
Implementierung (CAF)

Design

- Ziele:
  - CRDTs als Aktoren
  - Klassisches OOP problematisch, da State zwischen Aktoren
  - Aktoren dürfen nicht blockieren
  - Asynchrone Kommunikation in CAF
  - Namenbasierter Zugriff auf CRDTs

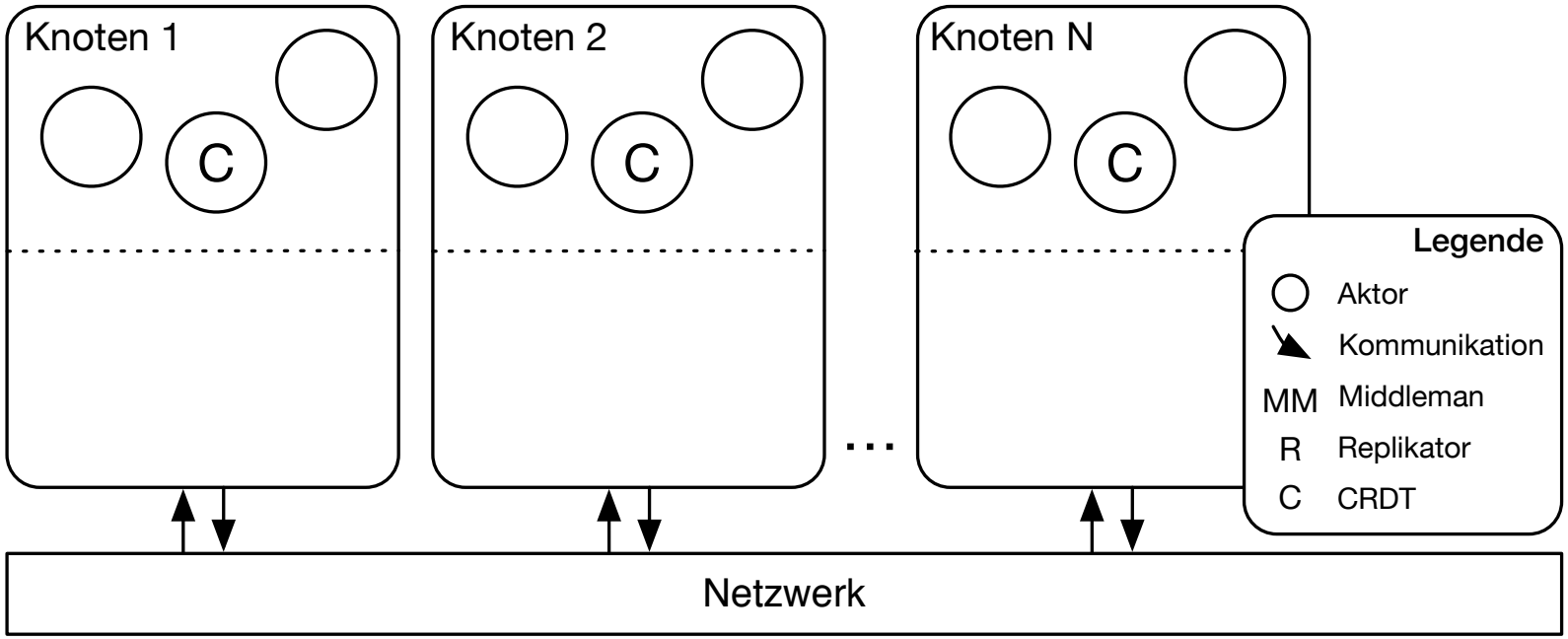
Implementierung (CAF)

Actor-System



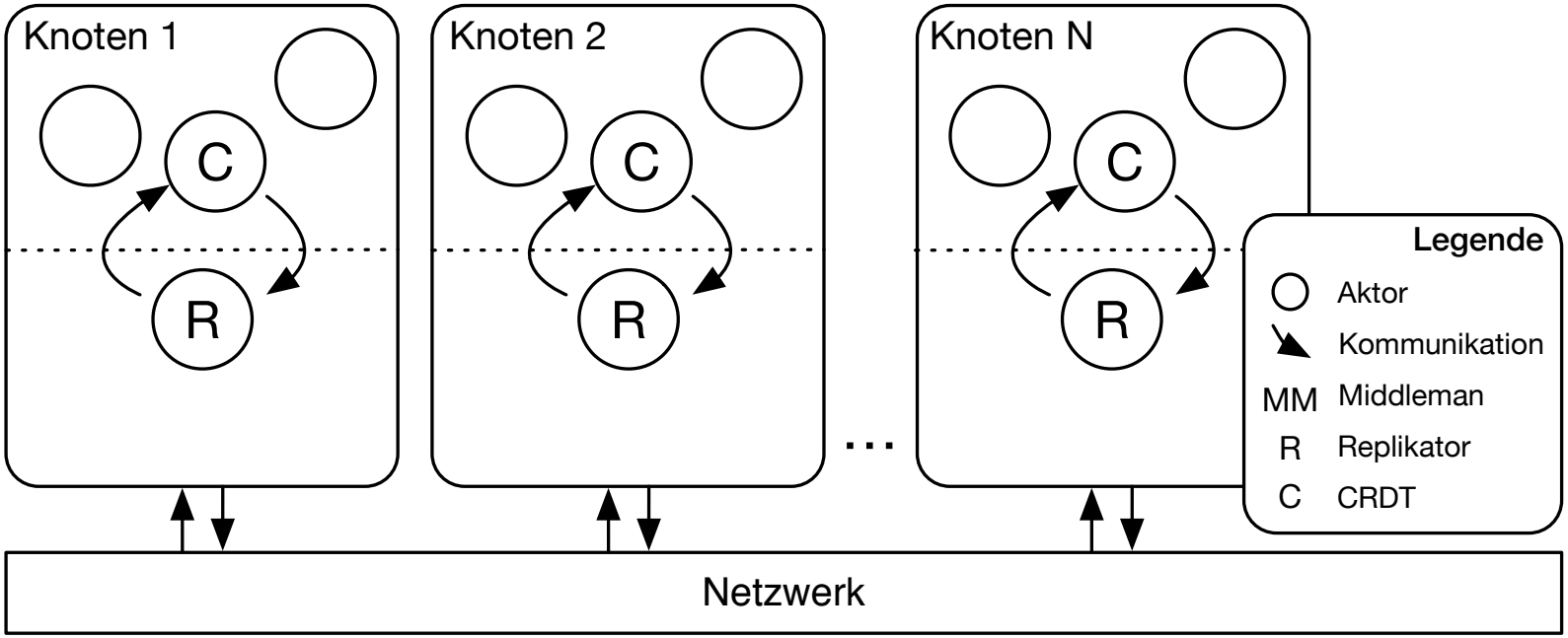
Implementierung (CAF)

Actor-System



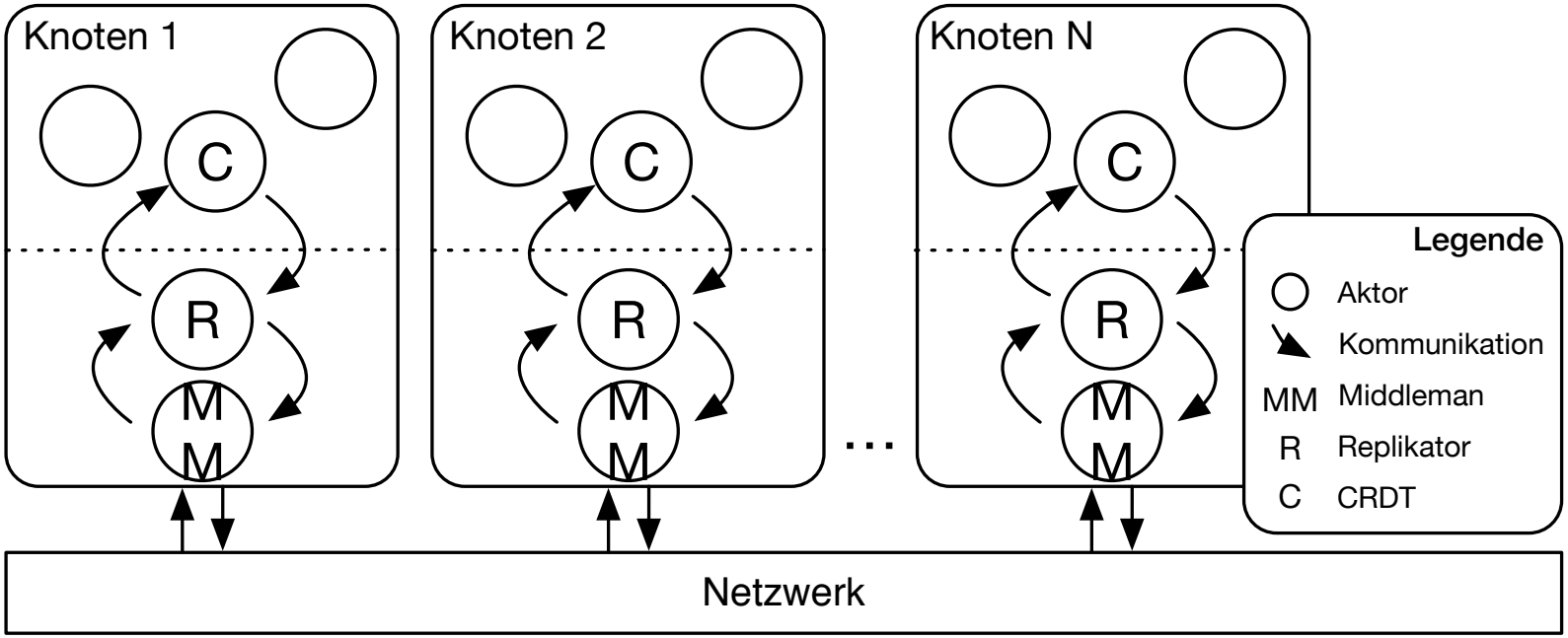
Implementierung (CAF)

Actor-System



Implementierung (CAF)

Actor-System



Agenda

Zusammenfassung

- Problem: Verteilte Datenhaltung
- Algorithmischer Ansatz
- Transparenter Zugriff auf Daten
- CAF erweitern, um CRDT-Datentypen bereitzustellen

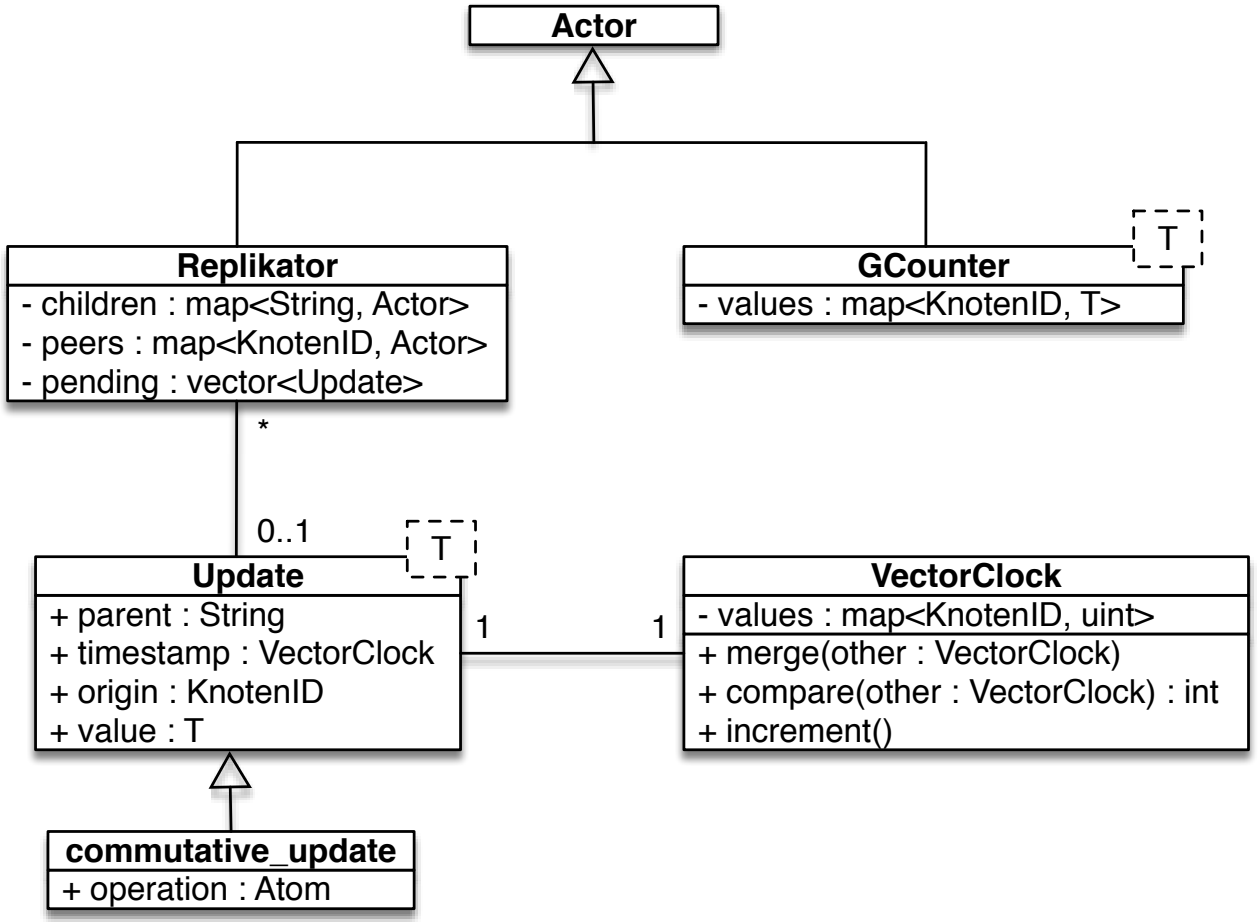
Ausblick

- Implementierung als Modul in CAF
- Datentypen vermessen & optimieren

Vielen Dank für Ihre Aufmerksamkeit!
Gibt es Fragen?

Backup Slides

Klassendiagramm



Backup Slides

Namensbasierter Zugriff

- CRDT Aktoren haben einen ID im System
- Mangling mit Typinformationen
- Replikator dient als Factory für CRDTs

```
// ...
actor_system system;
// ...
system.replicator().create(gset<int>, "ID");
// ...
```

Backup Slide

Middleman

- Gateway zur Außenwelt für Aktoren
- Ein Middleman pro Laufzeit

Backup Slide

Inter-Knoten Kommunikation

- Mögliche Implementierungen:
  - CAF Remote-Group von Replikatoren
  - CAF Instanzen kennen Replikatoren und verteilen Updates "per Hand"


Sebastian Burckhardt, Alexey Gotsman, Hongseok Yang, and Marek Zawirski, *Replicated data types: specification, verification, optimality*, ACM SIGPLAN Notices, vol. 49, ACM, 2014, pp. 271–284.


Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski, *A comprehensive study of convergent and commutative replicated data types*, Ph.D. thesis, Inria–Centre Paris-Rocquencourt, 2011.


———, *Conflict-free replicated data types*, Stabilization, Safety, and Security of Distributed Systems, Springer, 2011, pp. 386–400.


———, *Convergent and commutative replicated data types*, Bulletin of the European Association for Theoretical Computer Science (BEATCS) (2011), no. 104, 67–88.