

## Was ist NoSQL

- Klare Definition schwierig
- Alles, was nicht dem relationalen Datenmodell entspricht
- Not only SQL
- Verteilung
- Keine vollständige Transaktionsunterstützung (ACID)

## Warum NoSQL

- Unterstützung großer Datenmengen verwaltet durch Computer-Cluster
- Horizontale Skalierbarkeit
- Flexibler Umgang mit Schemata
- Bessere Anpassung an Datenstrukturen in Anwendungen
- Polyglot Persistence

## NoSQL vs SQL

- Relationale Systeme integrieren immer mehr nicht-relationale Strukturen
- Realisieren ebenfalls Verteilung
- NoSQL-Systeme unterstützen vermehrt SQL und Transaktionen
- Die Grenze verschwimmt immer weiter

## Schlüssel-Wert-Systeme

- Einfaches Datenmodell
- Schlüssel üblicherweise Zeichenketten
- Werte z.B. Listen, Mengen, Hashes, Blobs

## Spaltenfamilien-Systeme

- Nicht zu verwechseln mit spaltenorientierten Datenbanksystemen, tatsächlich findet eine zeilenweise Speicherung statt
- Jede Zeile besteht aus einer Menge von Schlüssel-Wert-Paaren
- Die Zeilen werden partitioniert gespeichert
- Zeilen haben kein festes Schema - verschiedene Zeilen können unterschiedliche Mengen von Schlüssel-Wert-Paaren enthalten

## Dokumentendatenbanken

- Üblicherweise JSON-Dateien als Dokumente
- Geschachtelte Strukturen

## Graphdatenbanken

- Knoten und Kanten als Datenelemente
- Einfache Navigation im Graphen

- Skalierung
  - Horizontal, shared nothing
  - Nahtlos: Dynamisches Hinzufügen/Entfernen von Knoten
- Knoten-Hardware
  - Standard-Rechner (commodity hardware)
  - Heterogen
- Management
  - Dezentrale Kontrolle
  - Uniforme Einsetzbarkeit von Knoten
  - Last-Balanzierung

- Zugriffstransparenz
- Ortstransparenz
- Replikationstransparenz
- Partitionierungsstransparenz
- Migrationstransparenz
- Nebenläufigkeitstransparenz
- Fehlertransparenz



## Konsistenz (Consistency)

- Konsistenter Zustand im verteilten System
- Änderung auf einem Knoten
- Lesezugriff auf replizierten Knoten liefert geänderten Wert

## Verfügbarkeit (Availability)

- System bietet akzeptable Reaktionszeit
- Auch bei Ausfall von Knoten und Netzwerkverbindungen

## Partitionierungstoleranz (Partition Tolerance)

- Bei Netzwerk-Split: Teilnetze können weiter arbeiten

## Netzwerk-Split im System – Entscheidung zwischen C und A

- Entscheidung für C
  - Operationen auf allen Knoten ablehnen
  - Keine Verfügbarkeit
- Entscheidung für A
  - Operationen auf verfügbaren Knoten zulassen
  - Erzeugt möglicherweise inkonstanten Zustand im Gesamtsystem
  - Knoten können verschiedene Zustände für gleiches Objekt haben
  - Konstistenz später herstellen
  - (Eingeschränkte) Verfügbarkeit
- Keine absolute Entscheidung zwischen C und A
- Verschiedene Strategien möglich

## Basic Available

- (Eingeschränkte) Verfügbarkeit als oberstes Ziel

## Soft State

- Fenster der Inkonsistenz

## Eventually Consistent

- Irgendwann wird ein konsistenter Zustand erreicht



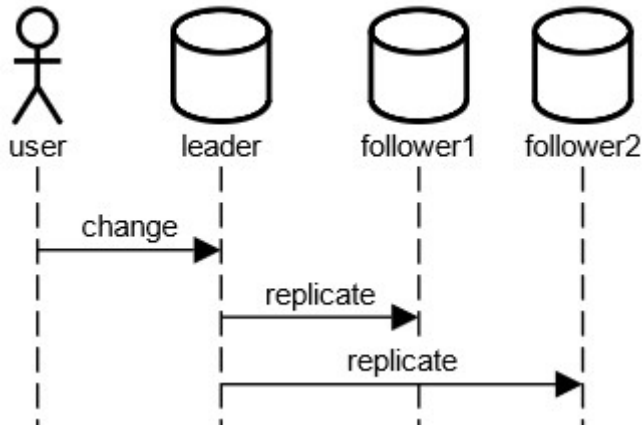
- **Beteiligte**
  - Verteiltes Speichersystem
  - Prozesse A, B schreiben und lesen
- **Starke Konsistenz**
  - Nach Änderung: A und B sehen immer den neuen Wert
  - Alle Replikate müssen aktualisiert sein
- **Schwache Konsistenz**
  - A oder B sehen den alten Wert
  - z.B. beim Lesen eines Replikats
  - Eventual Consistency ist eine Form schwacher Konsistenz

Diese Folien basieren auf dem Buch:

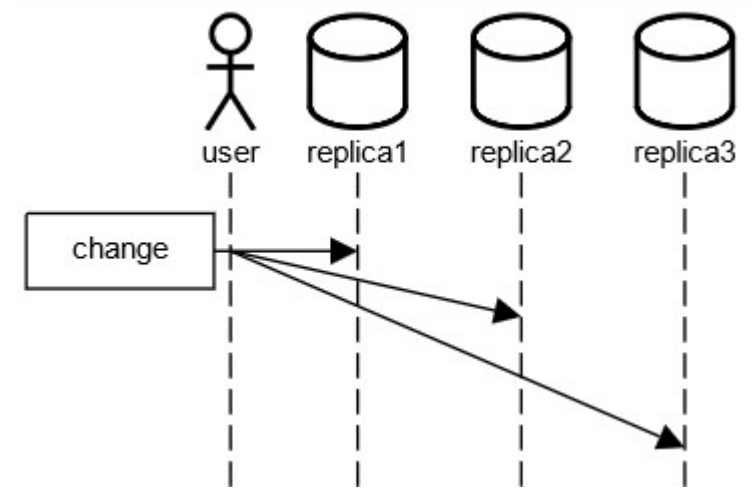
Martin Kleppman: Designing Data-Intensive Applications, O'Reilly

<https://www.oreilly.com/library/view/designing-data-intensive-applications/9781491903063/>

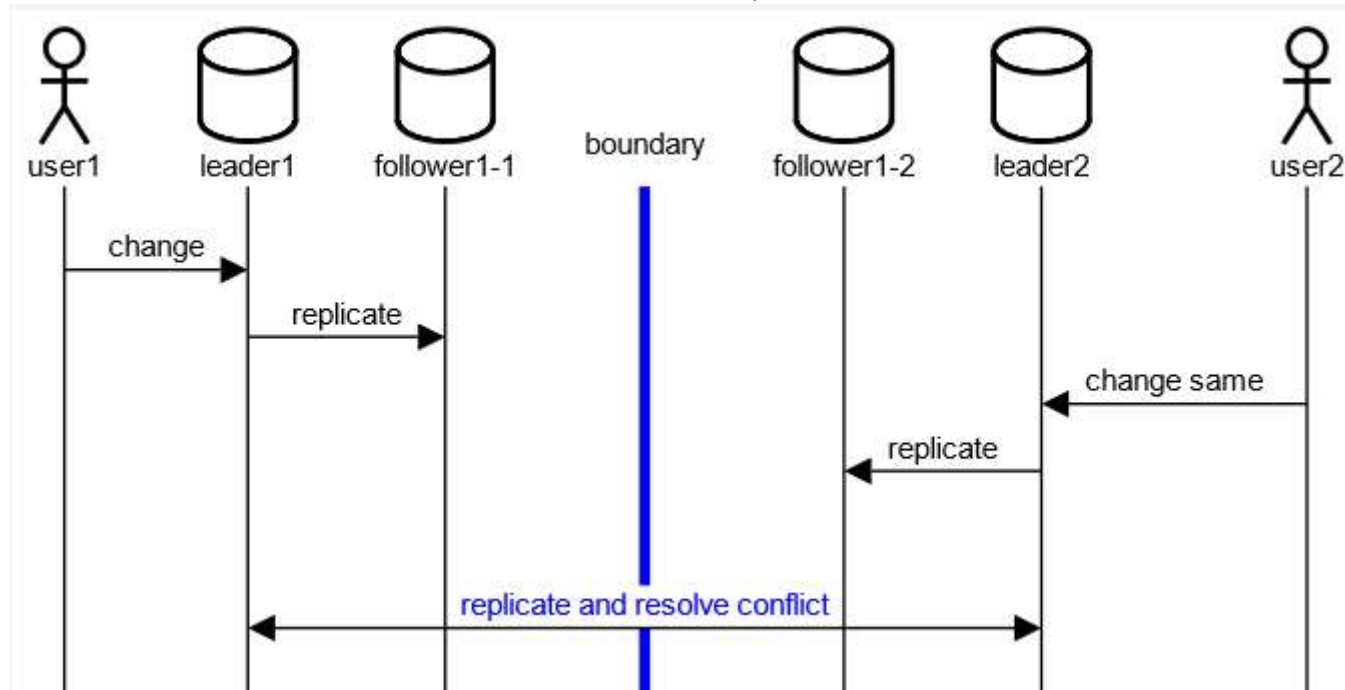
## Leader-based Replication

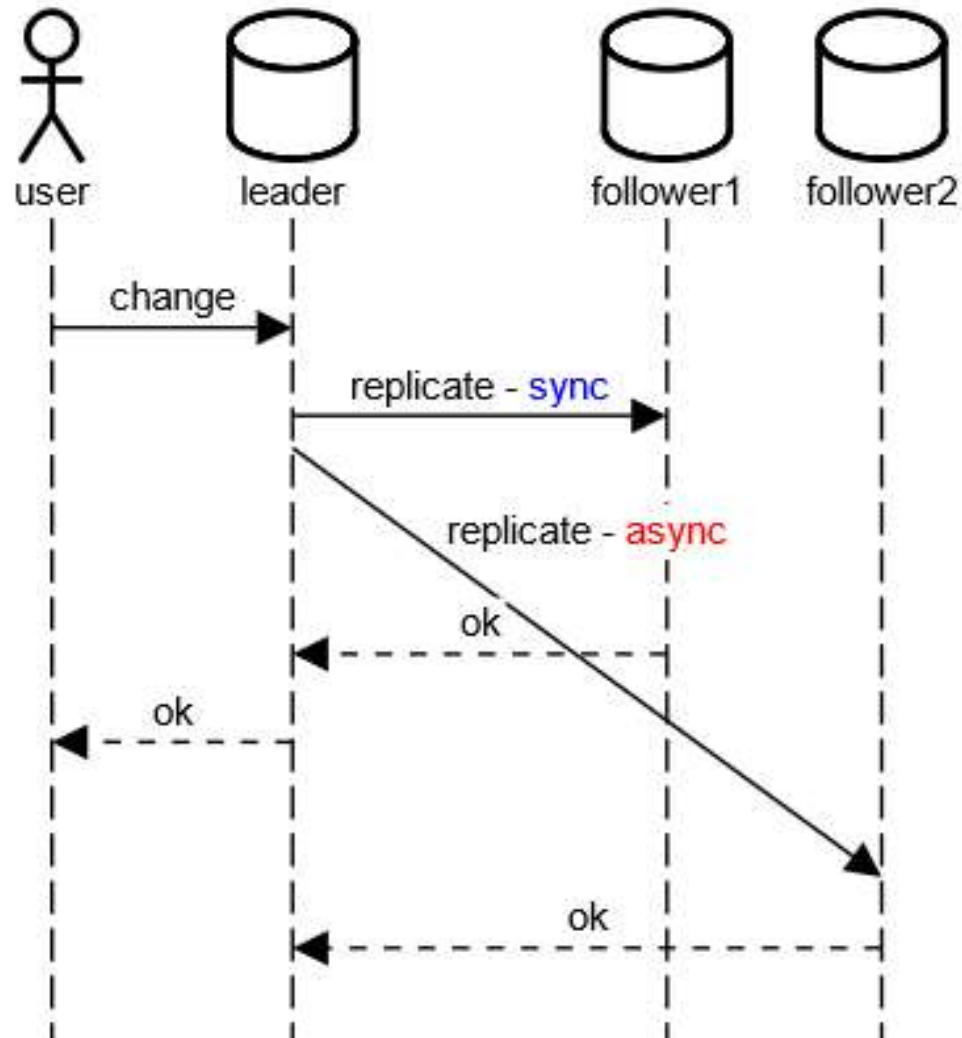


## Leaderless Replication



## Multi-Leader Replication





ok an User:

- erst nachdem Follower1 bestätigt hat (synchron)
- aber schon bevor Follower2 bestätigt hat (asynchron)

1. Konsistenter Schnappschuss der Datenbank zum Zeitpunkt  $t$
2. Kopieren des Schnappschusses auf den neuen Follower
3. Follower verbindet sich mit Leader, um Änderungen zu aktualisieren
4. Follower ist arbeitsfähig

## Follower

- Hält auf der lokalen Disk ein Log vor, das die vom Leader erhaltenen Änderungen enthält
- Absturz des Knotens
  - Neustart und Wiederherstellung
  - Aktualisierung der Änderungen (Rückfrage beim Leader)
- Netzwerkausfall
  - Aktualisierung der Änderungen (Rückfrage beim Leader)
- Follower ist wieder arbeitsfähig

## Leader

- Feststellung Leader-Ausfall
- Auswahl neuer Leader
- Rekonfiguration System auf neuen Leader
- Mögliche Probleme
  - Welcher Timeout für Feststellung Leaderausfall
  - Alter Leader wird wieder aktiv, Split Brain - zwei Knoten denken, sie sind Leader
  - Neuer Leader ist nicht auf neustem Stand der Daten (bei asynchronen Replikation)
  - Alter Leader hat neueren Stand an externe Systeme bereits übertragen

## Übertragung von Anweisungen

- Nichtdeterministische Funktionen - z.B. now()
- Reihenfolge der Anweisungsausführung
- Seiteneffekte - Trigger, Stored Procedures

## Übertragung Transaktionsprotokoll (Write-Ahead-Log, WAL)

- Anwendung des WAL auf dem Follower-Knoten
- Direkte Kopplung an die Speichermaschine
- Kompatibilitätsprobleme bei unterschiedlichen Software-Versionen auf den Knoten

## Übertragung von Datensätzen

- Logische Übertragung der Datensatzdaten, nicht physisch wie bei WAL
- Eine Datenbankanweisung kann zu mehreren logischen Datensatzprotokolleinträgen führen

## Übertragung auf Anwendungsebene

- Große Flexibilität
- Change Data Capture (CDC)
- Z.B. über Trigger - Eintragung in Protokolltabellen - Auslesen dieser Tabellen



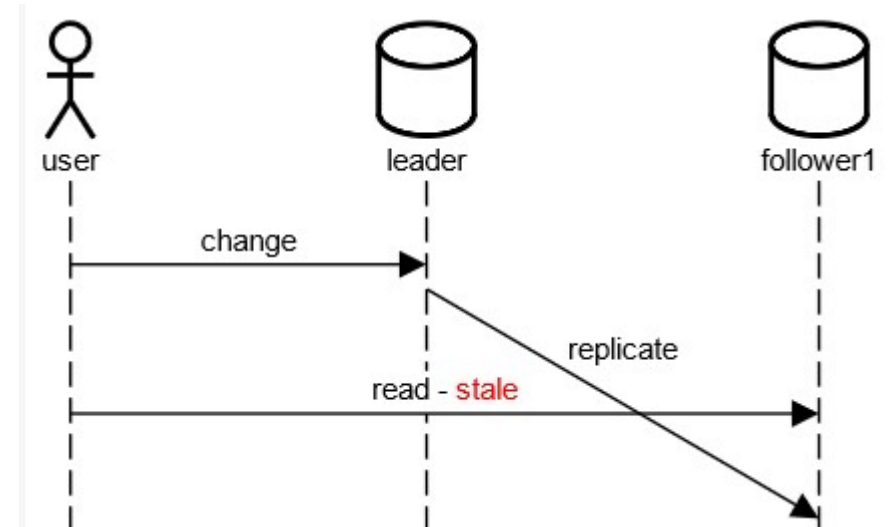


## Problem

- Ändern auf dem Leader
- Lesen von einem Follower - z.B. Neuladen der Seite
- Lesen von unterschiedlichen Geräten

## Lösungsansätze

- Daten, die nur vom Nutzer geändert werden können (z.B. Profil) - Lesen immer vom Leader
- Zeitpunkt der letzten Änderung merken - innerhalb einer Minute immer vom Leader lesen, danach beliebig
- Zeitpunkt der letzten Änderung merken - nur dann von einem Follower lesen, wenn dieser aktueller ist

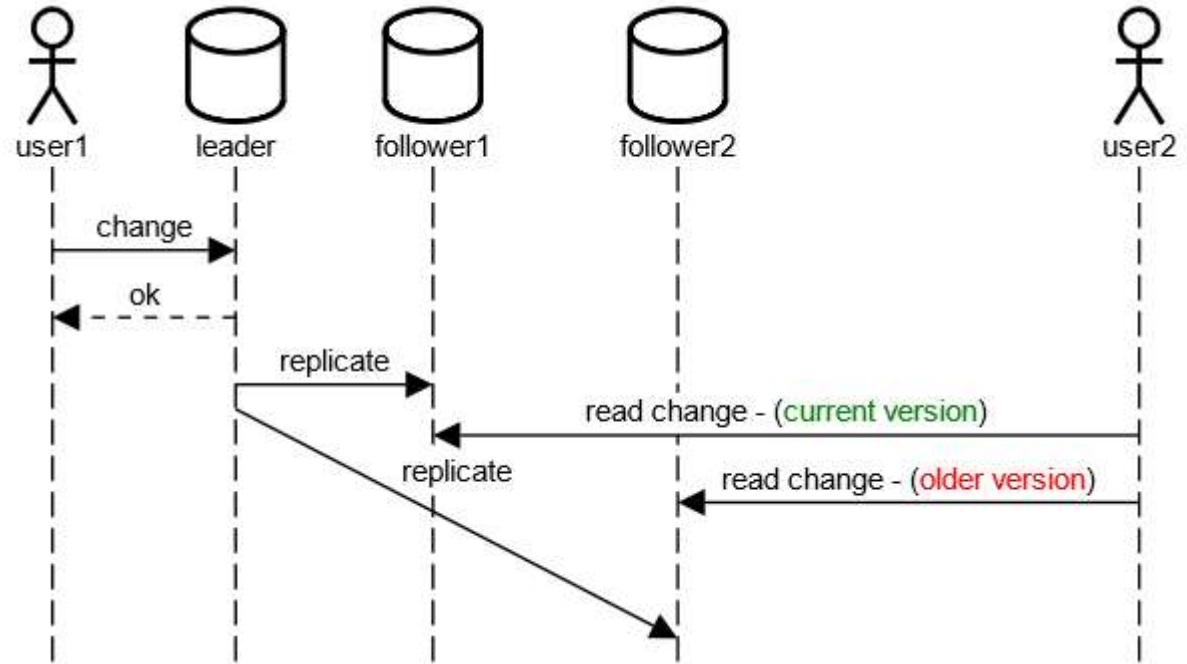


## Problem

- Lesen von unterschiedlichen Follower - z.B. bei Load Balancing
- Erstes Lesen liefert aktuellere Version als zweites Lesen

## Lösungsansatz

- Gleicher Nutzer immer vom gleichen Replikat lesen
- Verschiedene Nutzer von unterschiedlichen Replikaten

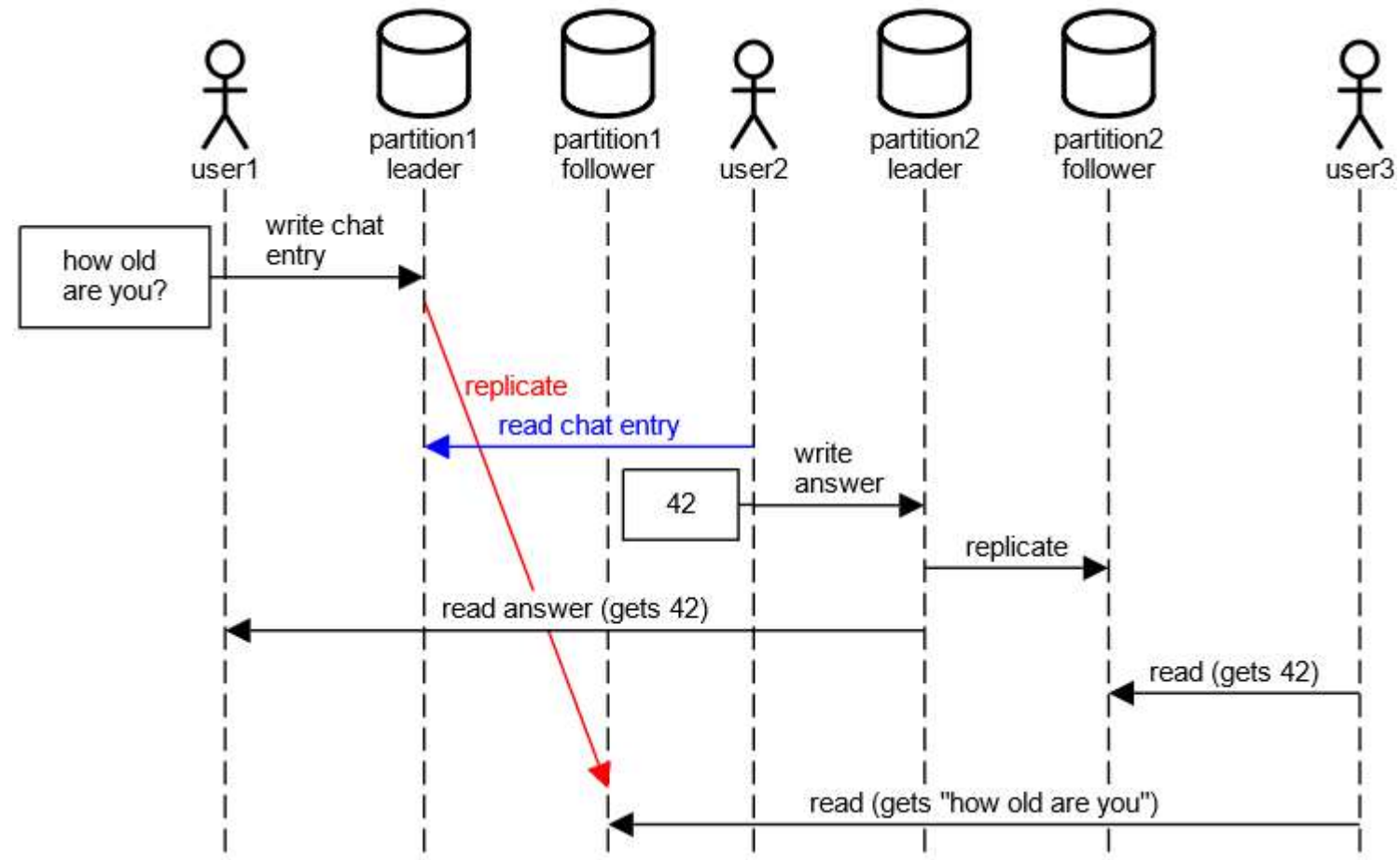


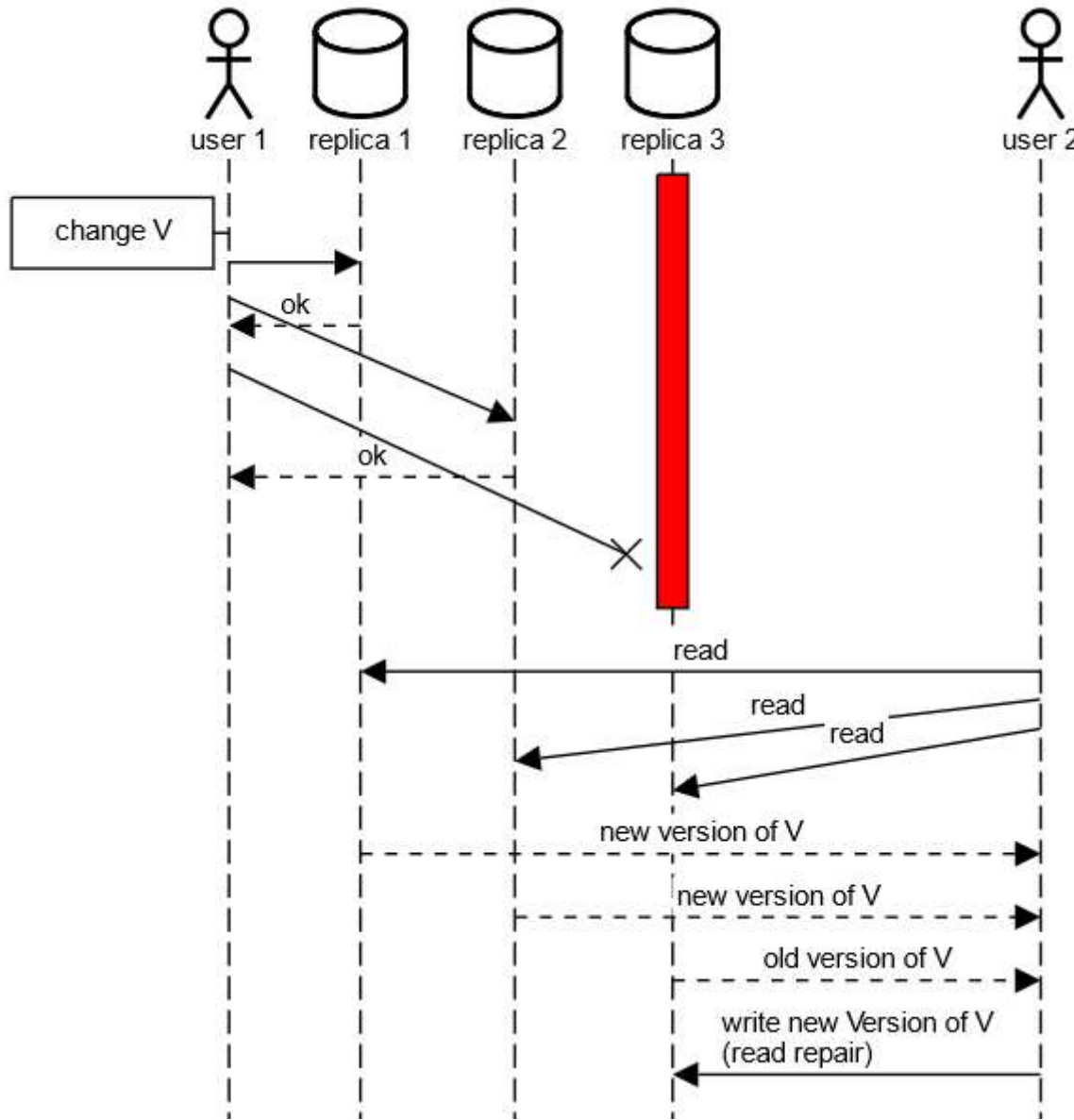
## Problem

- Lesen entgegen der Schreibreihenfolge
- Kann bei Sharding auftreten
- Nutzer 1 und 2 schreiben in unterschiedlichen Shards
- Replike werden entgegen der Schreibreihenfolge aktualisiert
- Shards operieren unabhängig, deshalb keine globale Ordnung auf den Schreibvorgängen

## Lösungsansatz

- Kausale Abhängigkeiten erkennen und berücksichtigen
- Im Beispiel liest Nutzer 2 einen von Nutzer 1 geschriebenen Wert
- Das begründet eine kausale Abhängigkeit





## Quorum beim Schreiben (z.B. 2 von 3)

- Fehler beim Schreiben wird ignoriert, wenn Quorum erreicht

## Quorum beim Lesen (z.B. 2 von 3)

- Lesen ok, wenn Quorum erreicht
- Schreiben des neuen Wertes auf Replik mit altem Wert (Read Repair)

## Anti Entropy

- Kontinuierlicher Prozess, der nach alten Versionen schaut und diese aktualisiert

## Konsistenz

- Replikationsfaktor:  $n$
- Anzahl gelesener Replikate:  $r$
- Anzahl geschriebener Replikate:  $w$
- $r + w > n$  stellt Überlappung von Lesern und Schreibern sicher
- wenigstens ein Leser liefert die neueste Version

## Konsistenz versus Verfügbarkeit

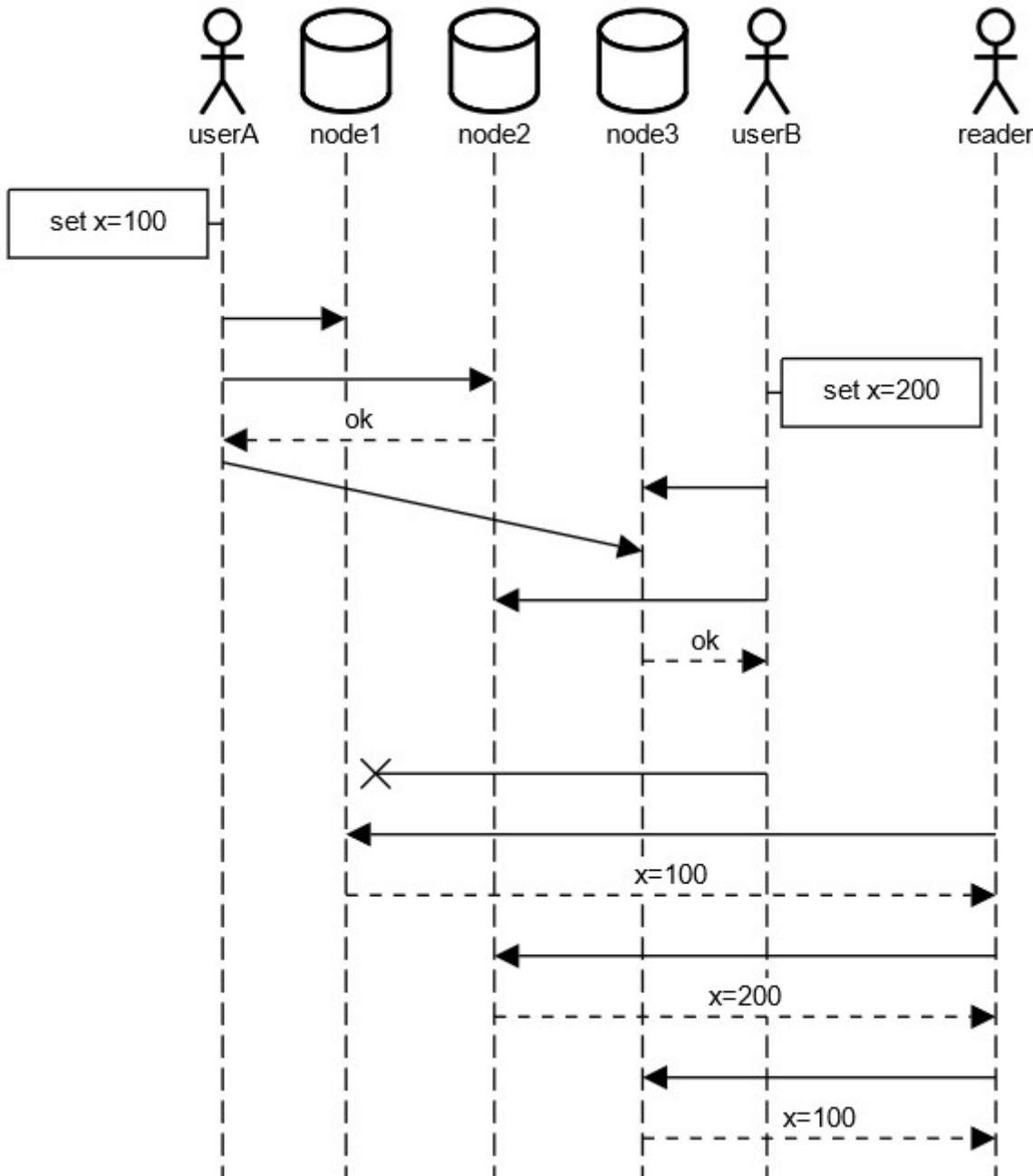
- $r + w < n$  stellt Konsistenz nicht sicher
- liefert aber eine gewisse Wahrscheinlichkeit für das Lesen der aktuellen Version
- erhöht die Verfügbarkeit des Systems, wenn mehrere Knoten ausfallen

## Probleme aus praktischer Sicht bei $r + w > n$

- Paralleles Schreiben und Überschreiben von Werten (z.B bei "first writer wins)
- Paralleles Lesen und Schreiben, welche Version wird geliefert
- Wiederherstellung von Knoten durch möglicherweise alte Versionen
- Sloppy Quorums

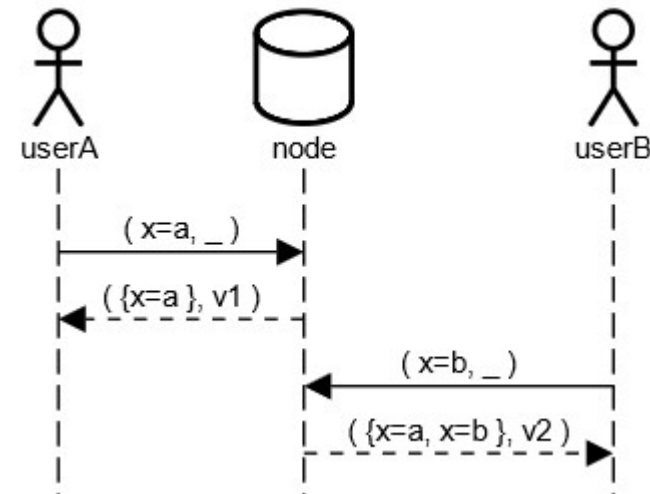
## Sloppy Quorums und Hinted Handoff

- Große Cluster: echte Anzahl Knoten  $N > n$  (Replikationsfaktor)
- Explizite (feste) Knotenzuordnung für Replikate
- Beispiel:
  - Datensatz  $d$  auf Knoten  $k_3, k_{10}, k_{15}$
  - Ausfall von  $k_{10}$  und  $k_{15}$ , danach Änderung von  $d$
  - Kein echtes Quorum erreichbar, aber Schreiben auf  $k_3, k_8$  und  $k_{17}$  möglich
- Denkbares Vorgehen
  - Schreiben auf  $k_3, k_8$  und  $k_{17}$  (Sloppy Quorum)
  - $k_8$  und  $k_{17}$  vermerken "echte" Replikatsknoten (Hinted Handoff)
  - Übertragung auf  $k_{10}$  und  $k_{15}$  bei Verfügbarkeit

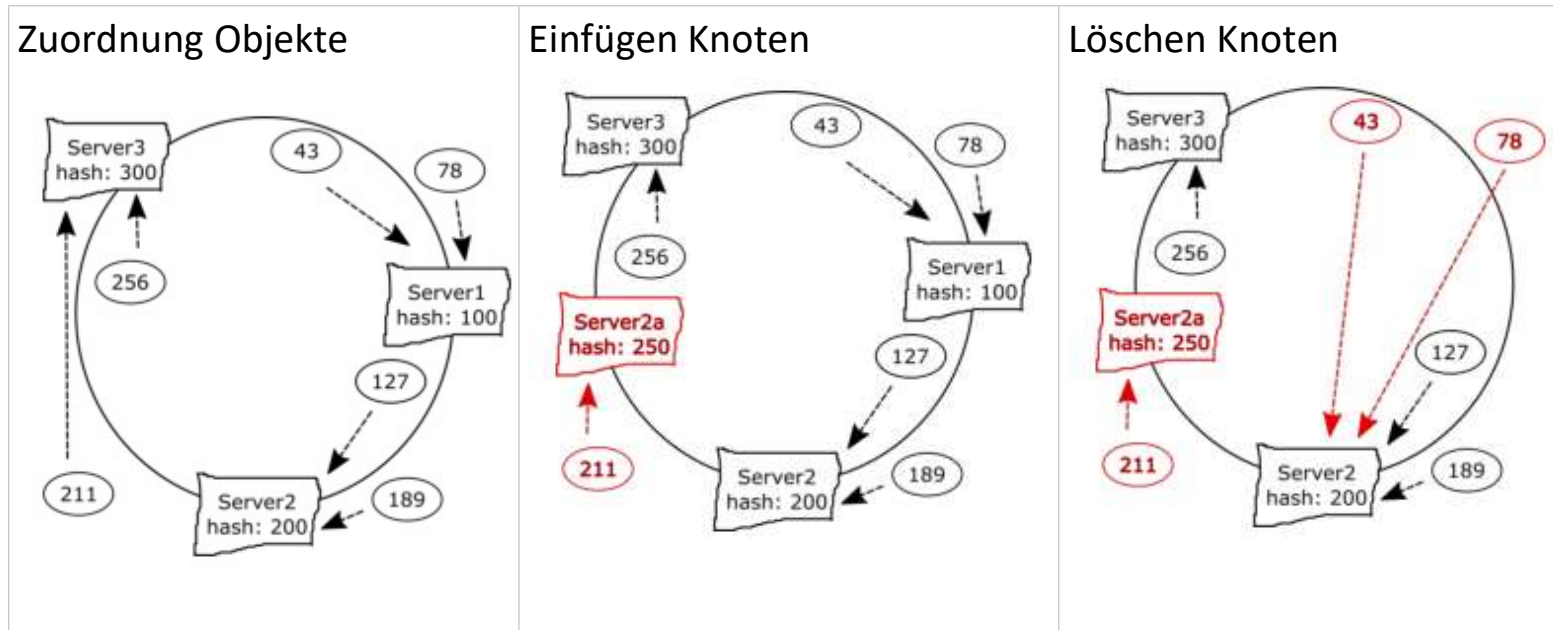


- userA erhält Rückmeldung, dass x=100 geschrieben wurde (Quorum erfüllt)
- userB erhält Rückmeldung, dass x=200 geschrieben wurde (Quorum erfüllt)
- node3: x=200 wird durch x=100 überschrieben
- node2: x=100 wird durch x=200 überschrieben
- reader liest x per Quorum
- Quorum liefert x=100
- das Schreiben von x=200 geht verloren, obwohl für user2 der Wert x=200 per Quorum bestätigt wurde

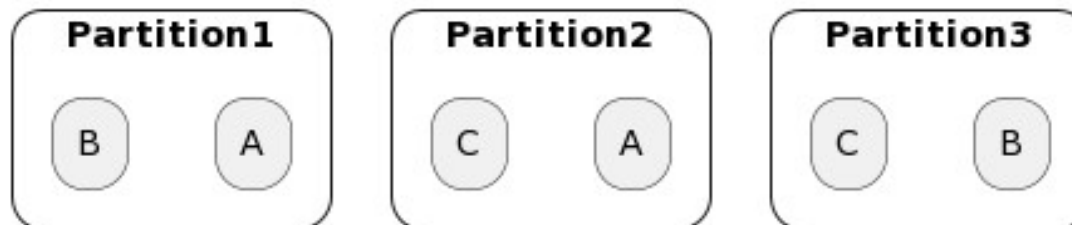
## Schreiben mit Versionen



- Daten auf verschiedenen Knoten verteilen (Sharding = Partitionierung)
- Horizontale Skalierung
- Allokation Daten: Consistent Hashing



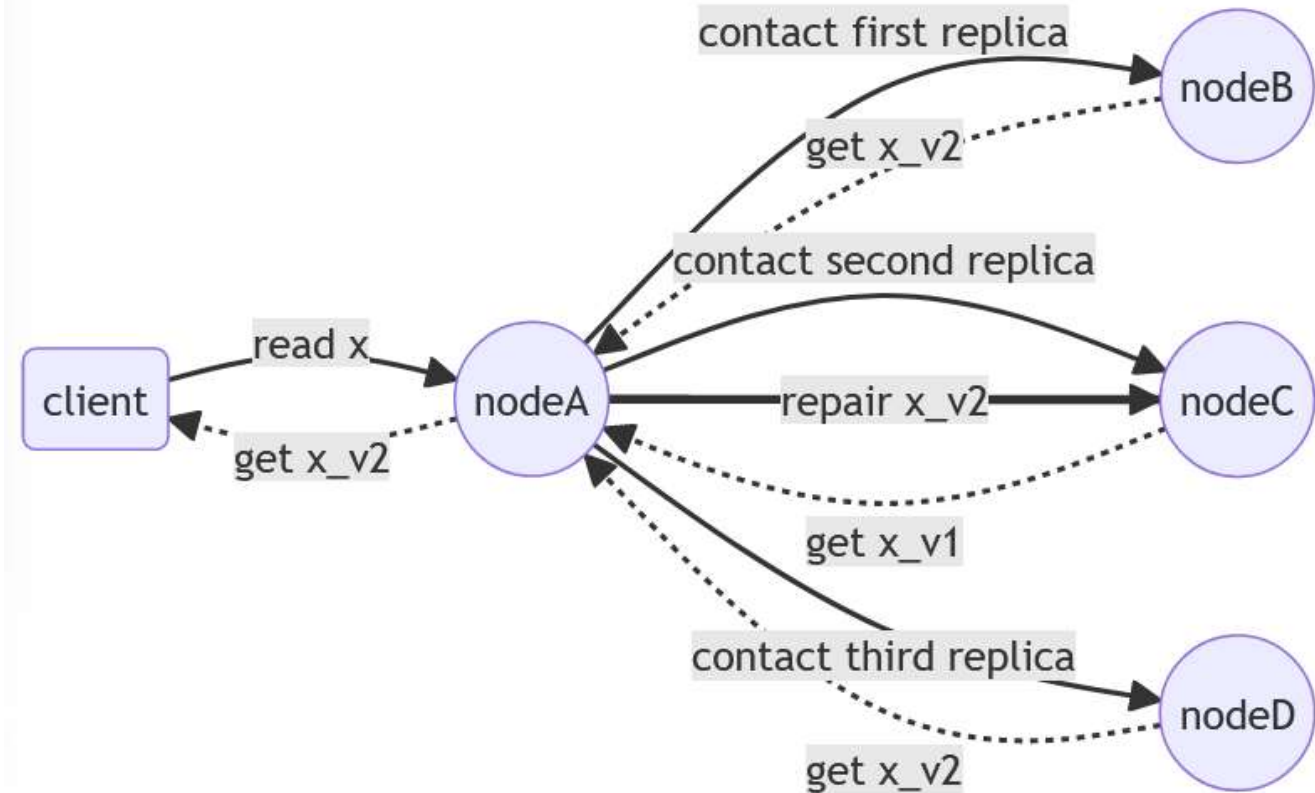
- Sharding + Replikation: Skalierung + Ausfalltolernanz
- Verschiedene Master für verschiedene Datenelemente möglich



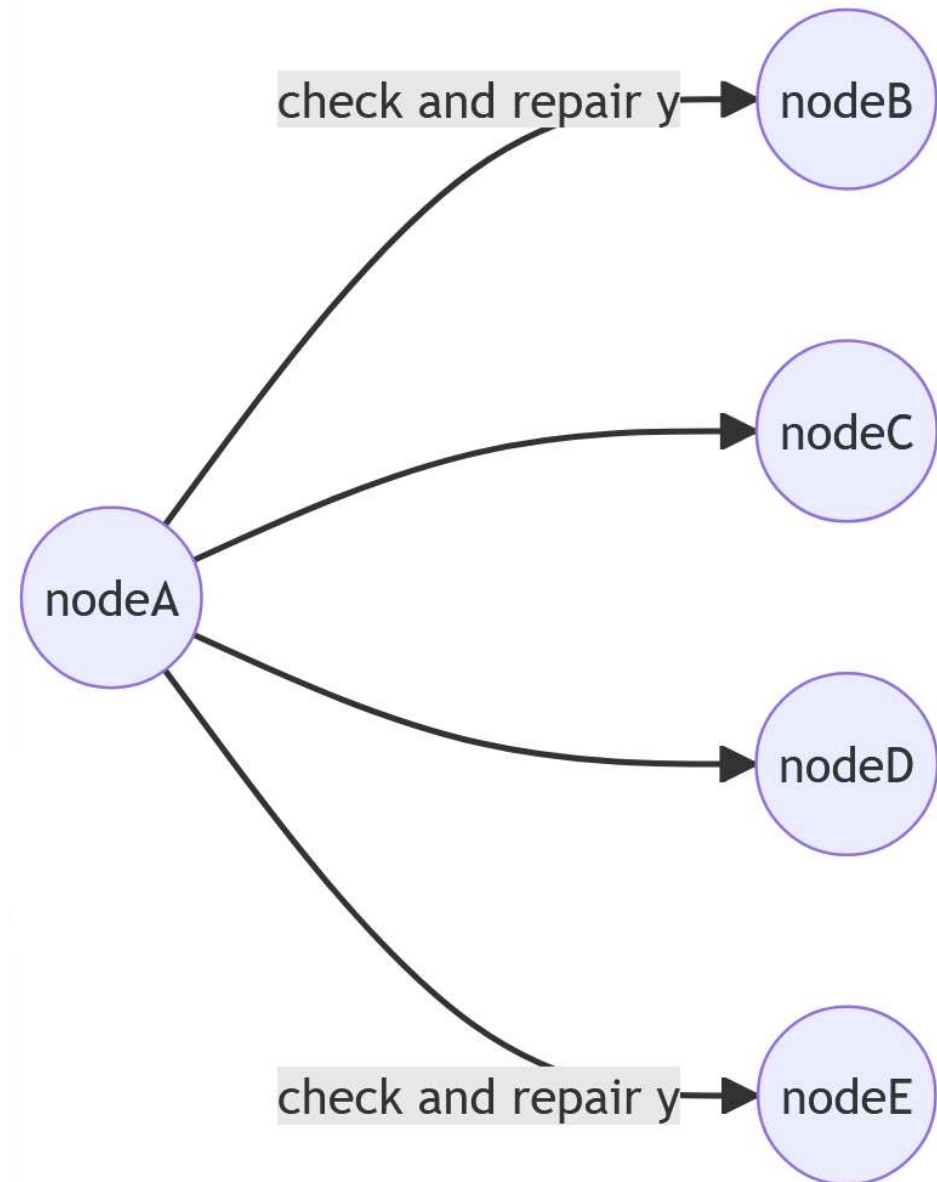




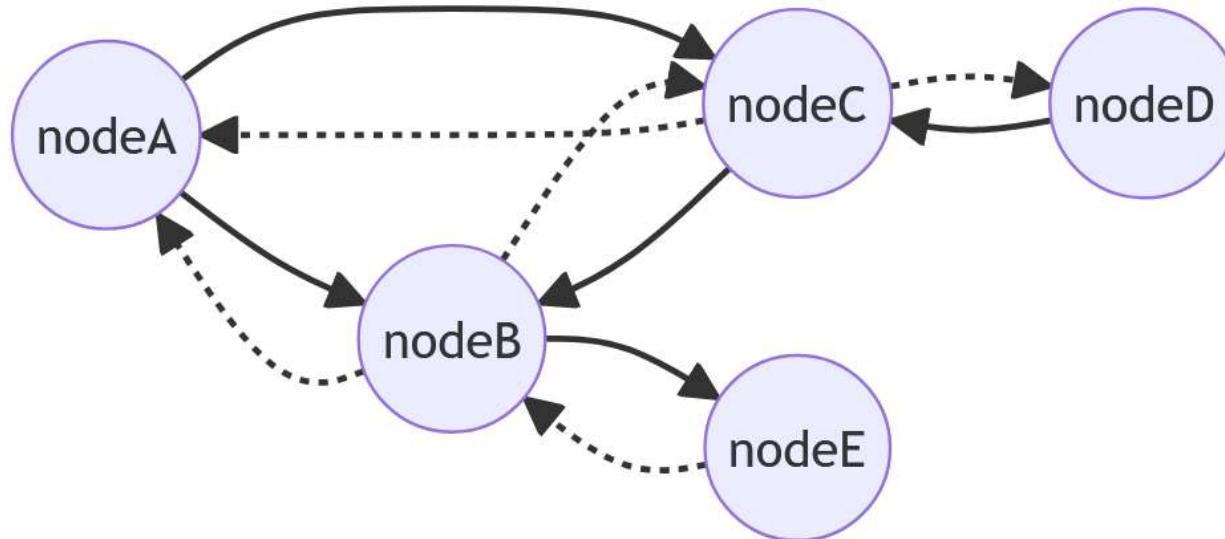
- Client kontaktiert irgendeinen Knoten um x zu lesen, in diesem Fall nodeA
- NodeA enthält kein replikat und kontaktiert Knoten mit Replikaten entsprechend Quorum
- NodeB und nodeD liefern Version v2 von x, nodeC alte Version v1
- Client erhält x in Version v2
- NodeA aktualisiert nodeC (repair)



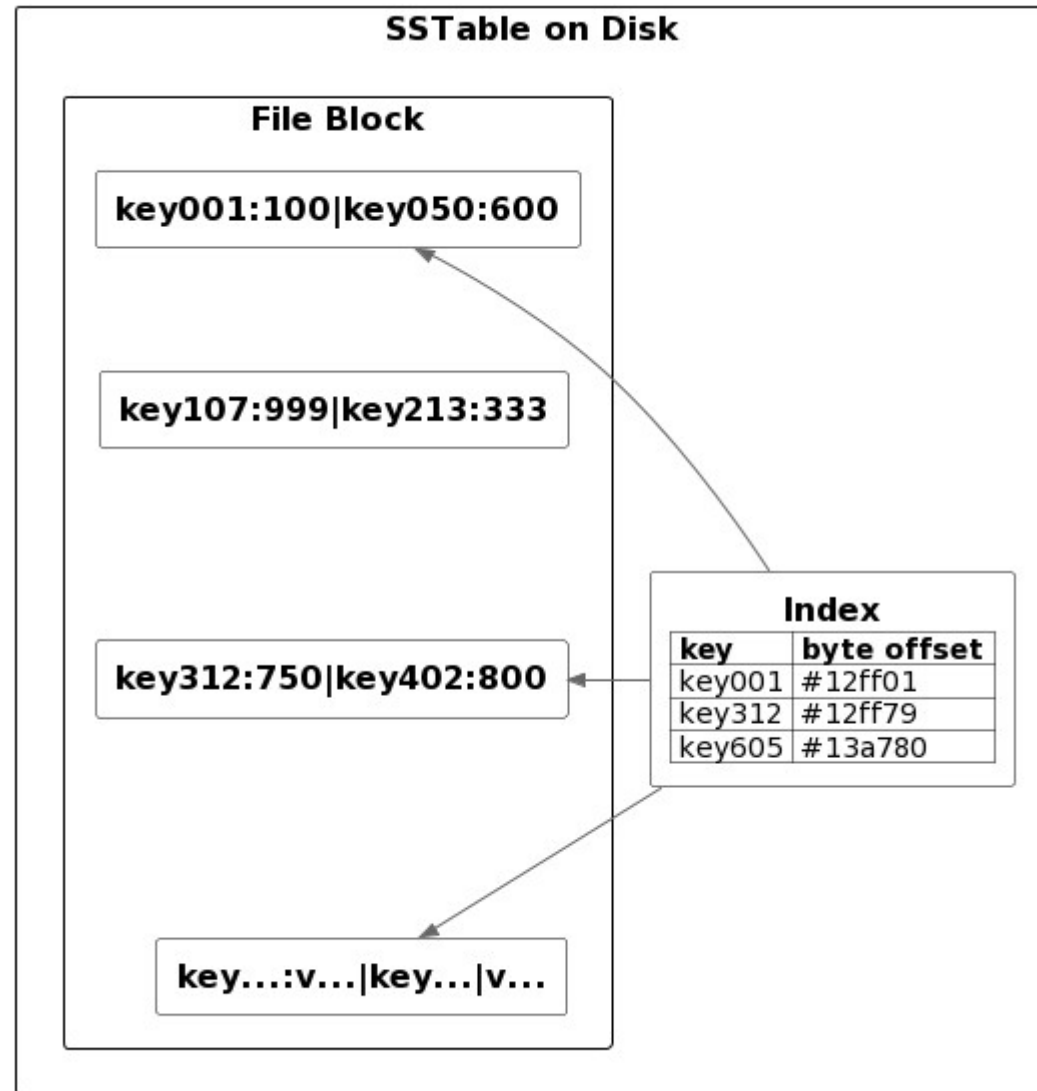
- Zufällig Auswahl eines Knoten mit aktueller Version eines Wertes, in diesem Fall y
- Zufällige Auswahl von Replikatsknoten
- Überprüfung ob neuste Version von y vorhanden
  - Wenn nicht, Ersetzung mit der neuesten Version von y (repair)

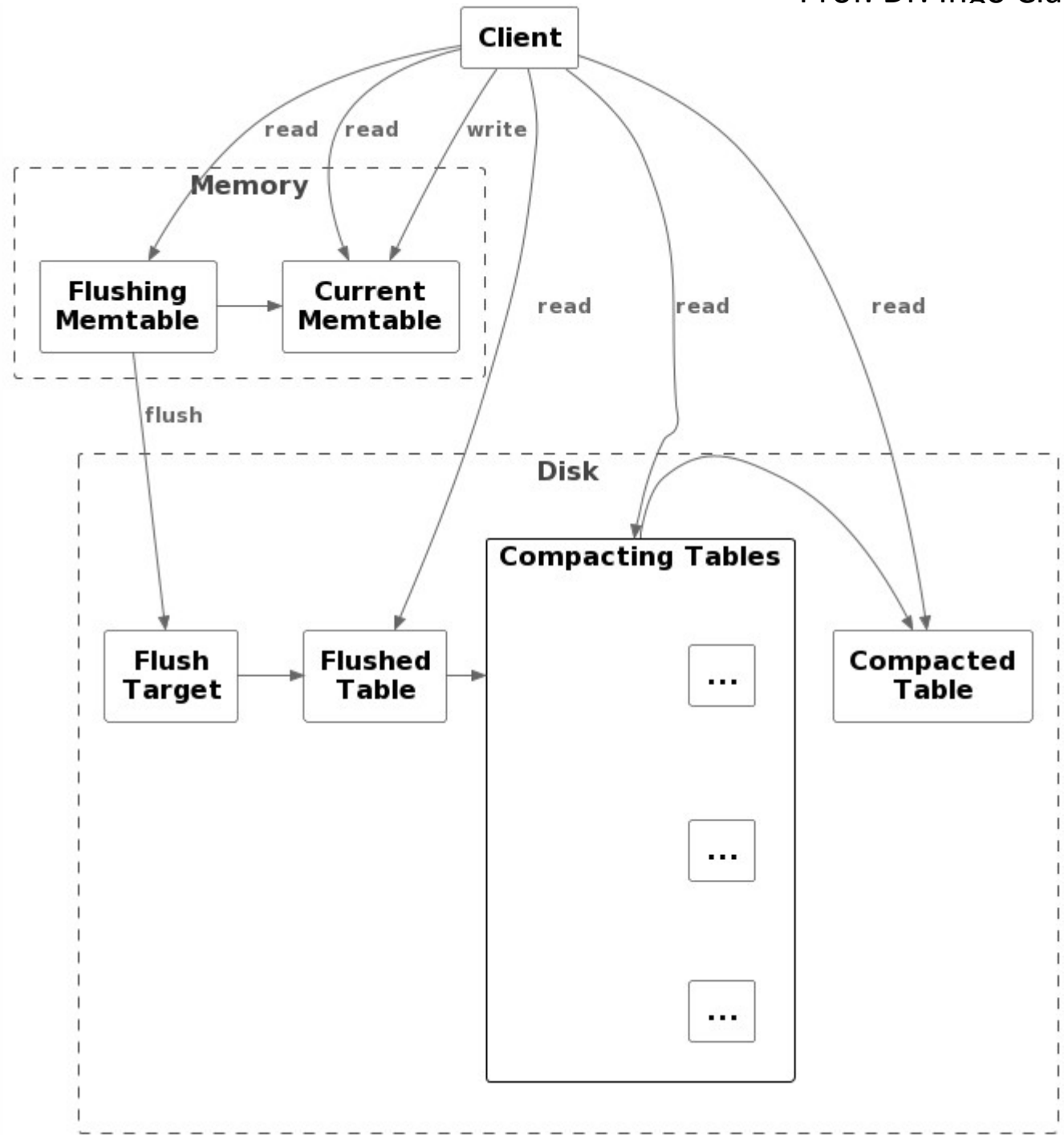


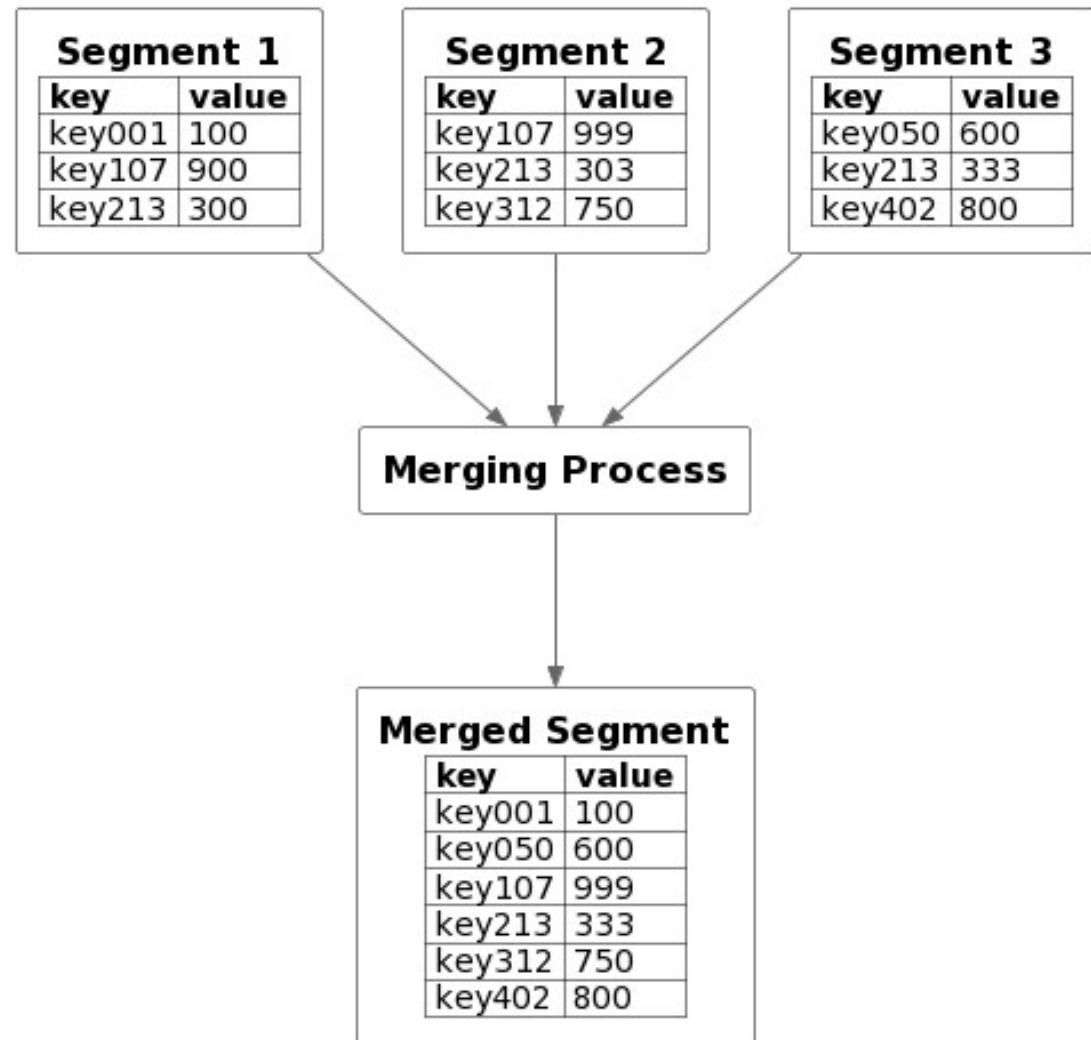
- Knoten kontaktieren zufällig andere Knoten



- Schreib-optimierte Speicherstrukturen, die häufig in NoSql-Systemen verwendet werden
- Typischer Einsatz bei schreibintensiver Arbeitslast,
- Gespeichert in Sorted Strings Tables (SSTable), unveränderlich
- Enthalten Schlüssel-Wert-Paare
- Memtable im Hauptspeicher







- Martin Kleppman: Designing Data-Intensive Applications, O'Reilly, 2017
- NoSQL - Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken, Edlich, Friedland, Hampe, Brauer, Brückner, Hanser, 2011
- Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement, Redmont, O'Reilly UK Ltd, 2012
- NoSQL Distilled, Saldage, Fowler, Addison Wesley, 2012
- Advanced Data Management, Wiese, De Gruyter, 2015
- Eventually Consistent, Vogel, Acm Queue, 2008
- CAP Twelve years later, Brewer, IEEE, 2012