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-Syteme 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 (commoditiy hardware)
  - Heterogen
- Management
  - Dezentrale Kontrolle
  - Uniforme Einsetzbarkeit von Knoten
  - Last-Balanzierung

Verteilungstransparenz

Montag, 9. Januar 2023 15:59

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

---- Konsistenz

Prof. Dr. Ingo Claßen

Montag, 9. Januar 2023 16:25

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 inkonstistenten Zustand im Gesamsystem
  - 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

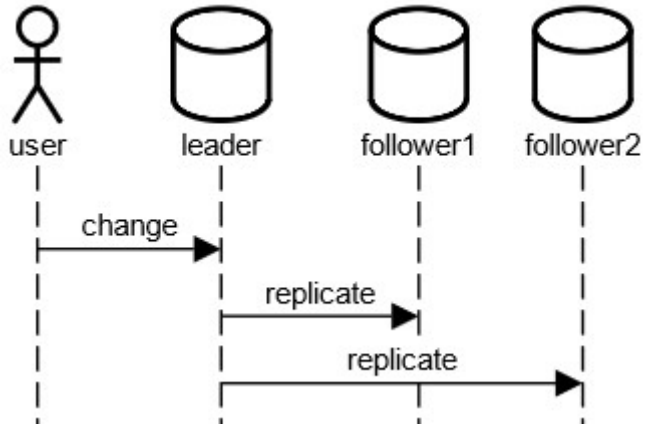
Soft State
- Fenster der Inkonsistenz

Eventually Consistent
- Irgendwann wird ein konsistenter Zustand erreicht

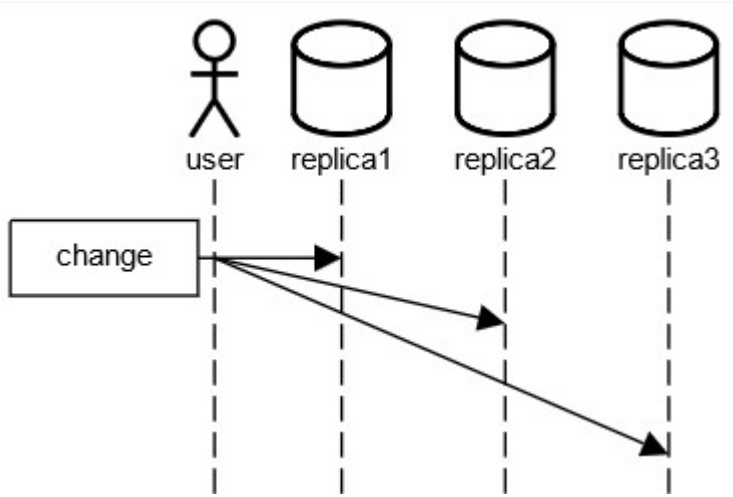
- Beteiligte
  - Verteiltes Speichersystem
  - Prozesse A, B schreiben und lesen
- Starke Konsistenz
  - Nach Änderung: A und B sehen immer den neuen Wert
  - Alle Replikate müssen aktualisiert sein
- Schwache Konsistenz
  - A oder B sehen den alten Wert
  - z.B. beim Lesen eines Replikats
  - Eventual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

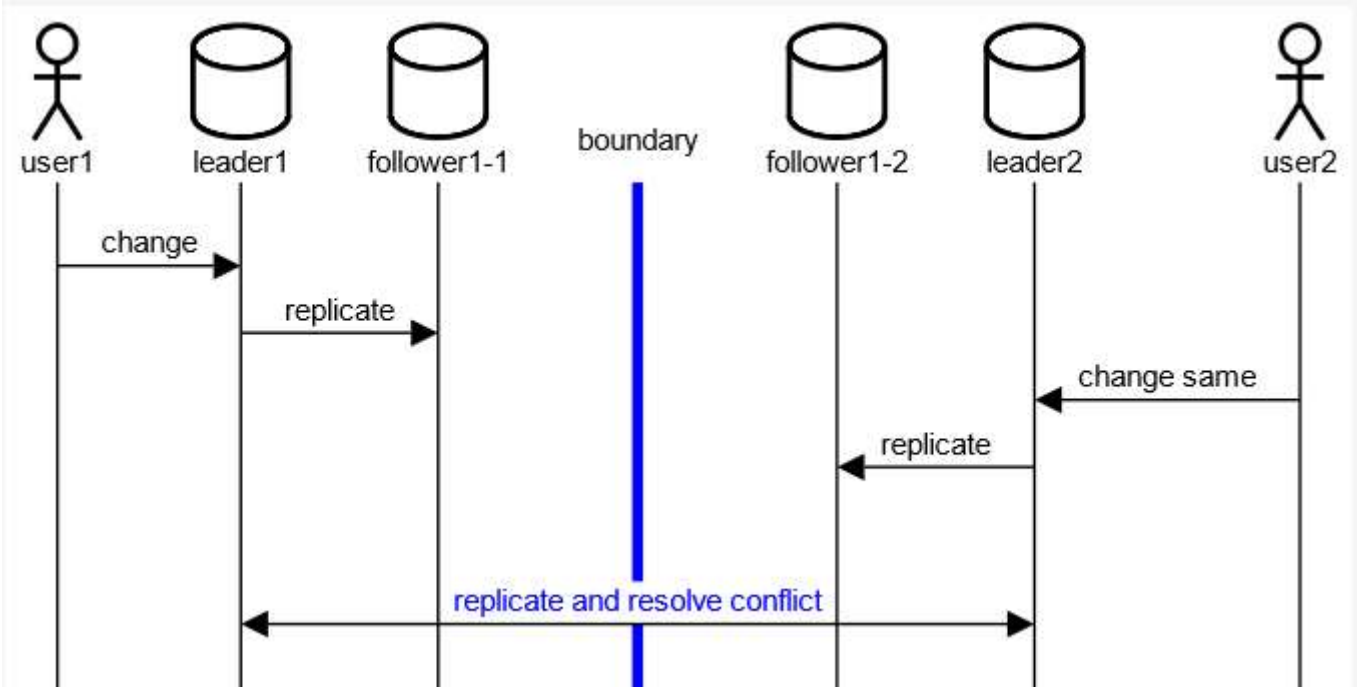
Leader-based Replication

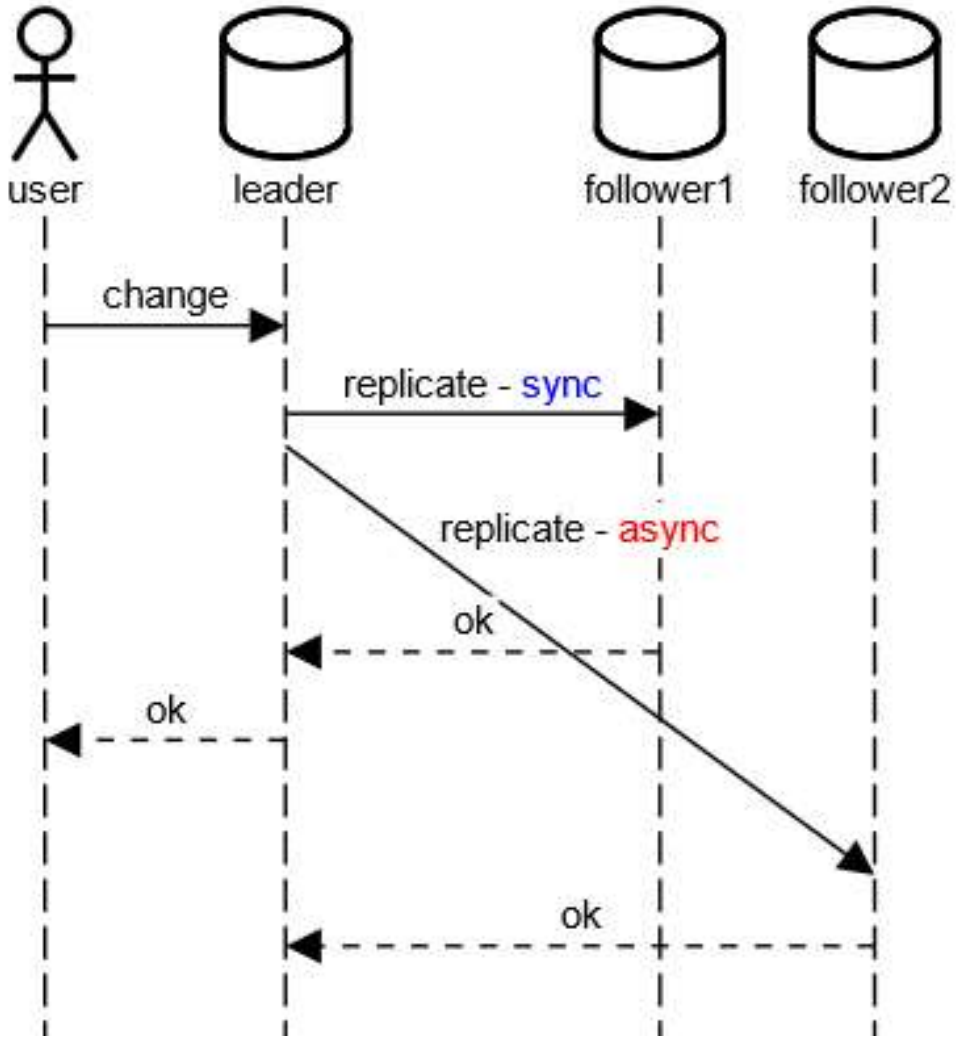


Leaderless Replication



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

Read your own writes

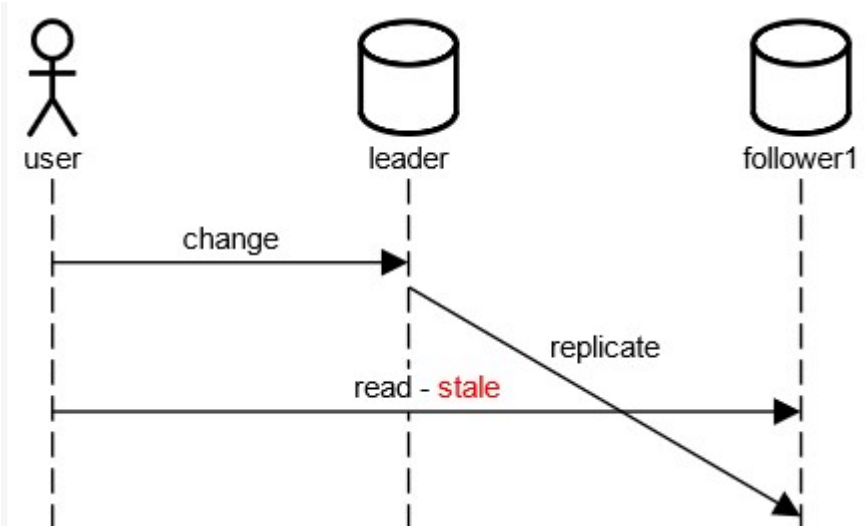
Mittwoch, 28. Dezember 2022 17:40

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

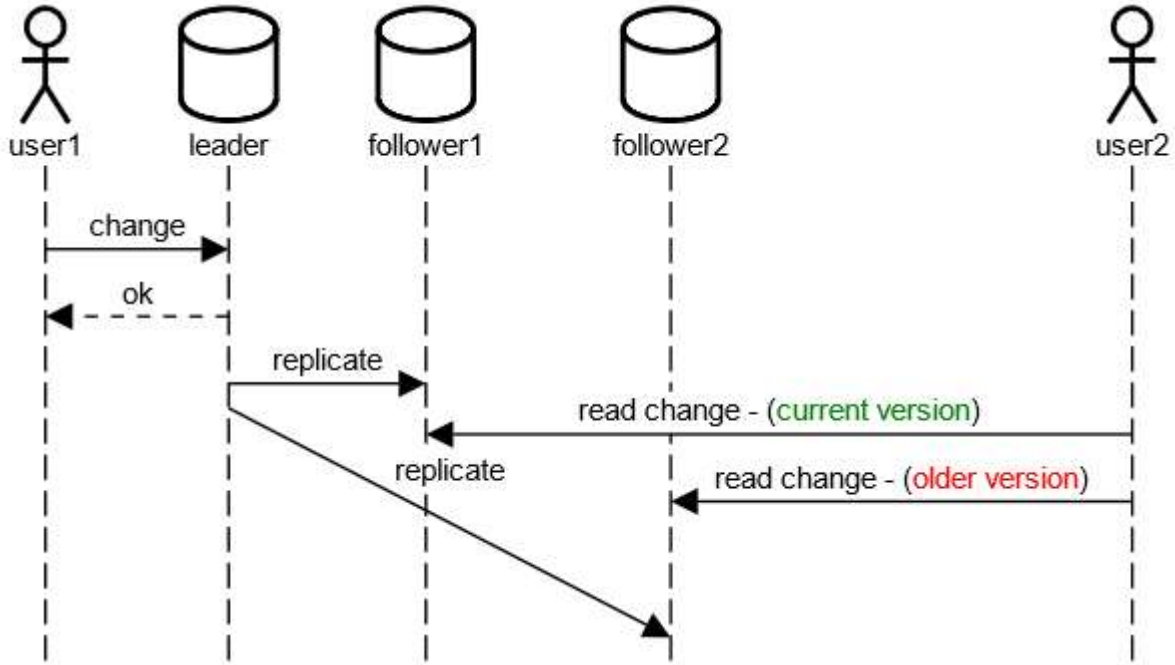


Problem

- 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

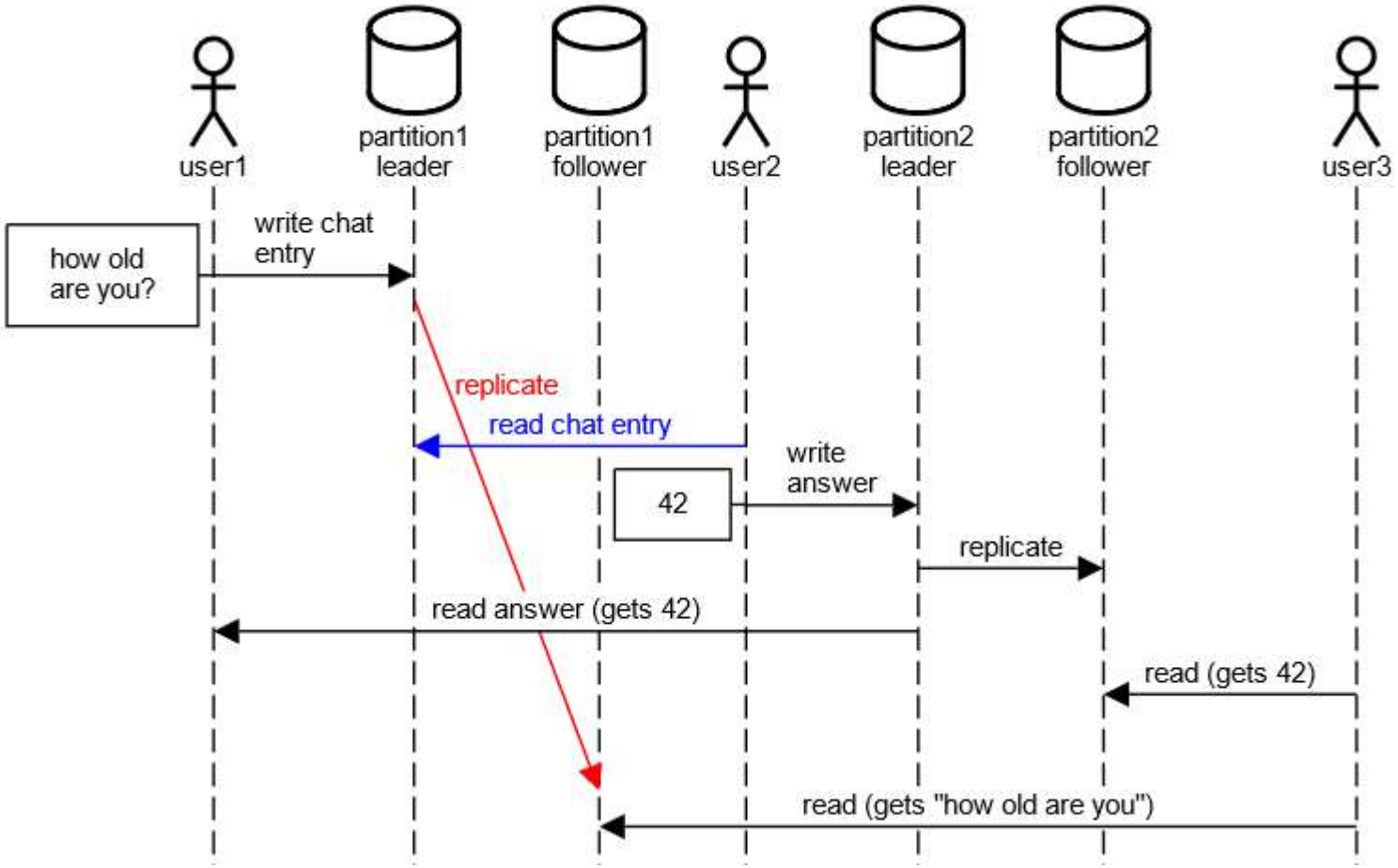


Problem

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

Lösungsansatz

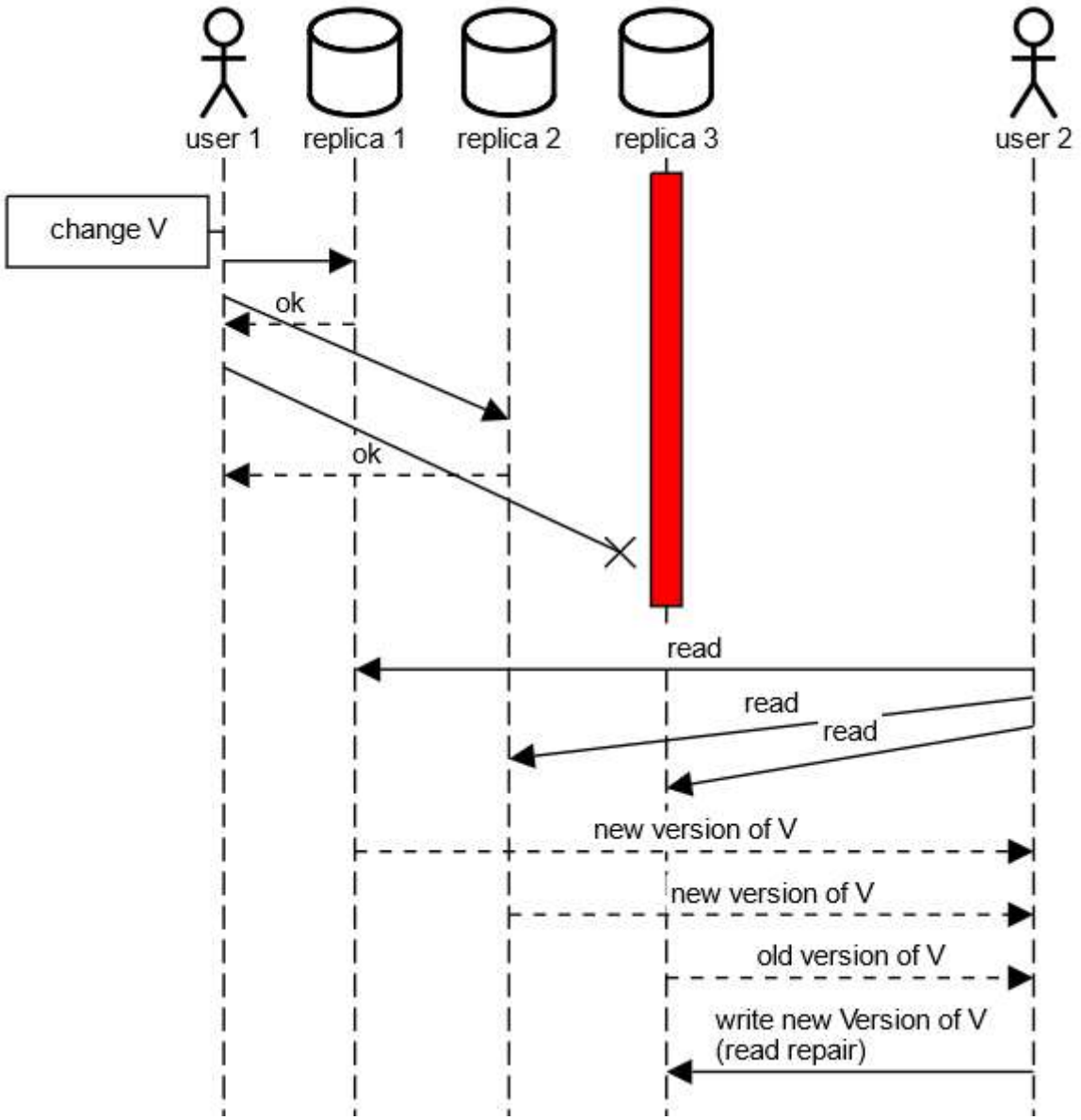
- Kausale Abhängigkeiten erkennen und berücksichtigen
- Im Beispiel liest Nutzer 2 einen von Nutzer 1 geschriebenen Wert
- Das begründet eine kausale Abhängigkeit



Schreiben bei Knotenausfall

Prof. Dr. Ingo Claßen

Montag, 2. Januar 2023 08:18



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 Replikat 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 neuste Version

Konsistenz versus Verfügbarkeit

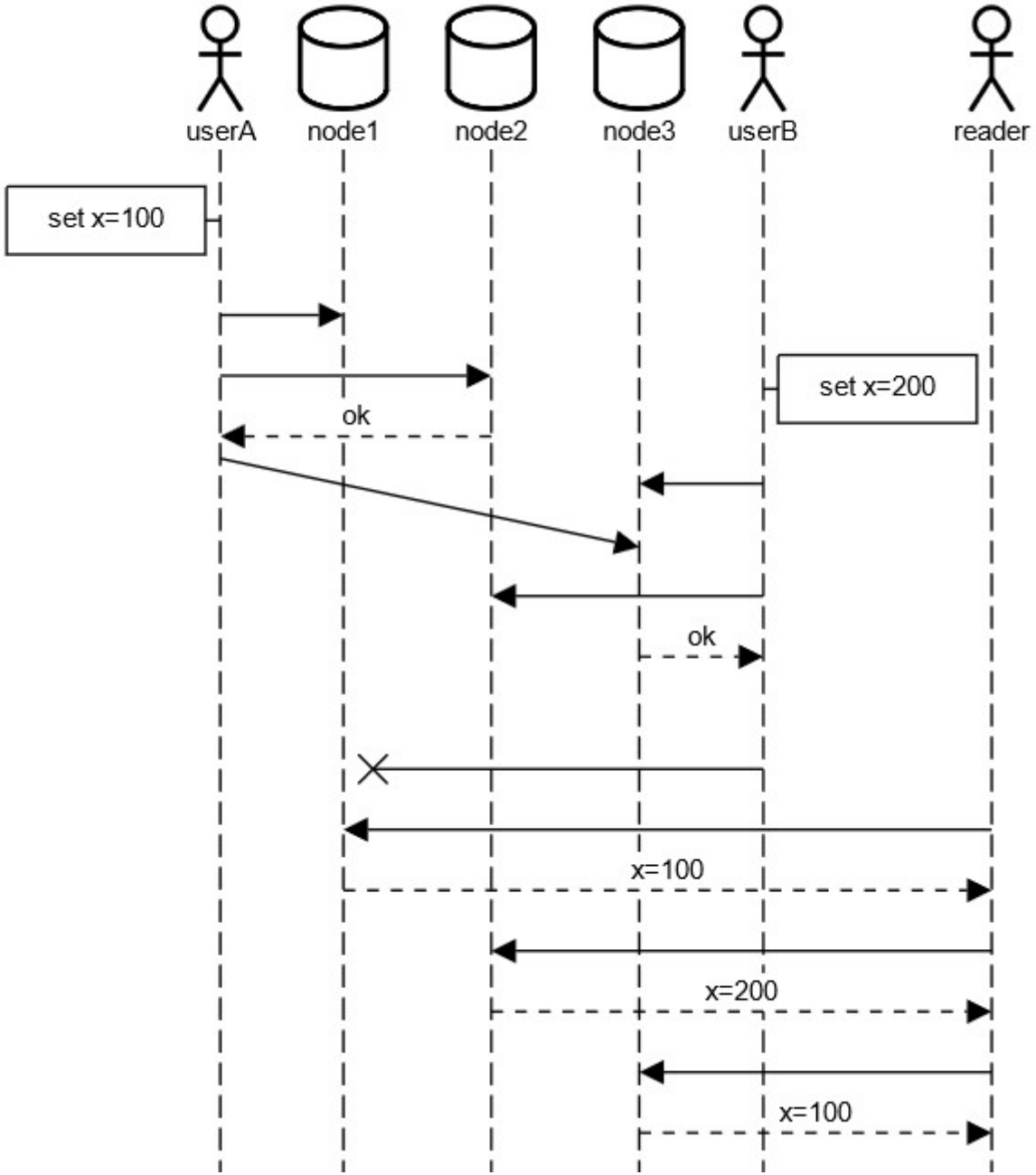
- r + w < n stellt Konsistenz nicht sicher
- liefert aber eine gewisse Wahrscheinlichkeit für das Lesen der aktuellen Version
- erhöht die Verfügbarkeit des Systems, wenn mehrere Knoten ausfallen

Probleme aus praktischer Sicht bei r + w > n

- Paralles 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

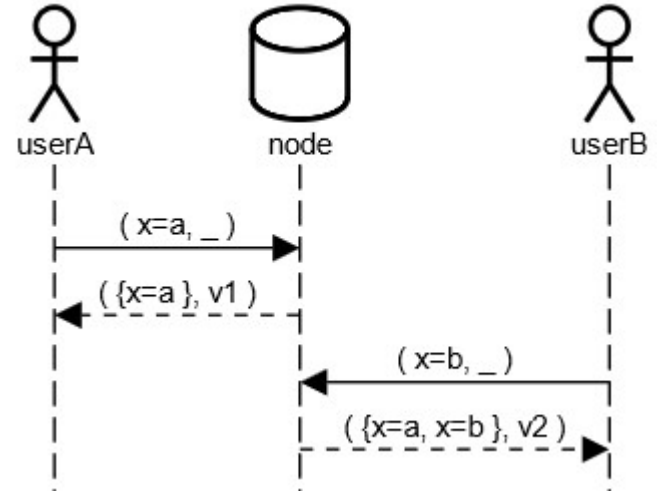
- Große Cluster: echte Anzahl Knoten N > n (Replikationsfaktor)
- Explizite (feste) Knotenzuordnung für Replikate
- Beispiel:
  - Datensatz d auf Knoten k3, k10, k15
  - Ausfall von k10 und k15, danach Änderung von d
  - Kein echtes Quorum erreichbar, aber Schreiben auf k3, k8 und k17 möglich
- Denkbares Vorgehen
  - Schreiben auf k3, k8 und k17 (Sloppy Quorum)
  - k8 und k17 vermerken "echte" Replikatsknoten (Hinted Handoff)
  - Übertragung auf k10 und k15 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

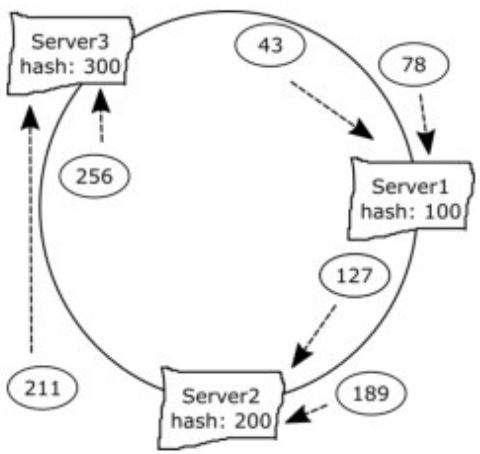
- reader liest x per Quorum
- Quorum liefert x=100
- das Schreiben von x=200 geht verloren, obwohl für user2 der Wert x=200 per Quorum bestätigt wurde

Schreiben mit Versionen

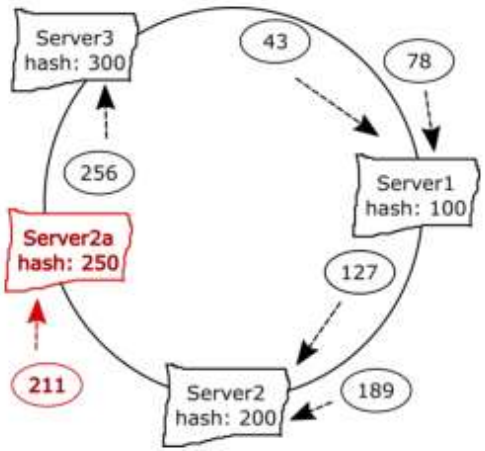


- Daten auf verschiedenen Knoten verteilen (Sharding = Partitionierung)
- Horizontale Skalierung
- Allokation Daten: Consistent Hashing

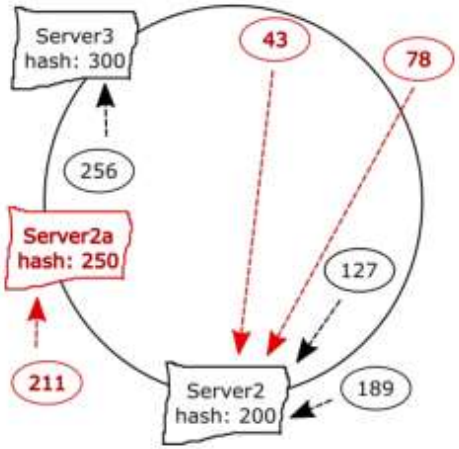
Zuordnung Objekte



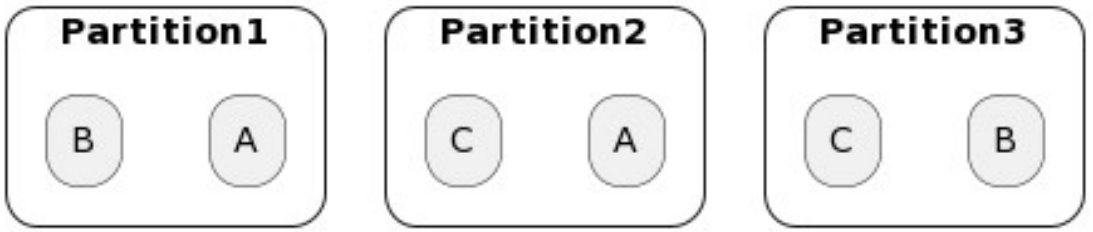
Einfügen Knoten



Löschen Knoten



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

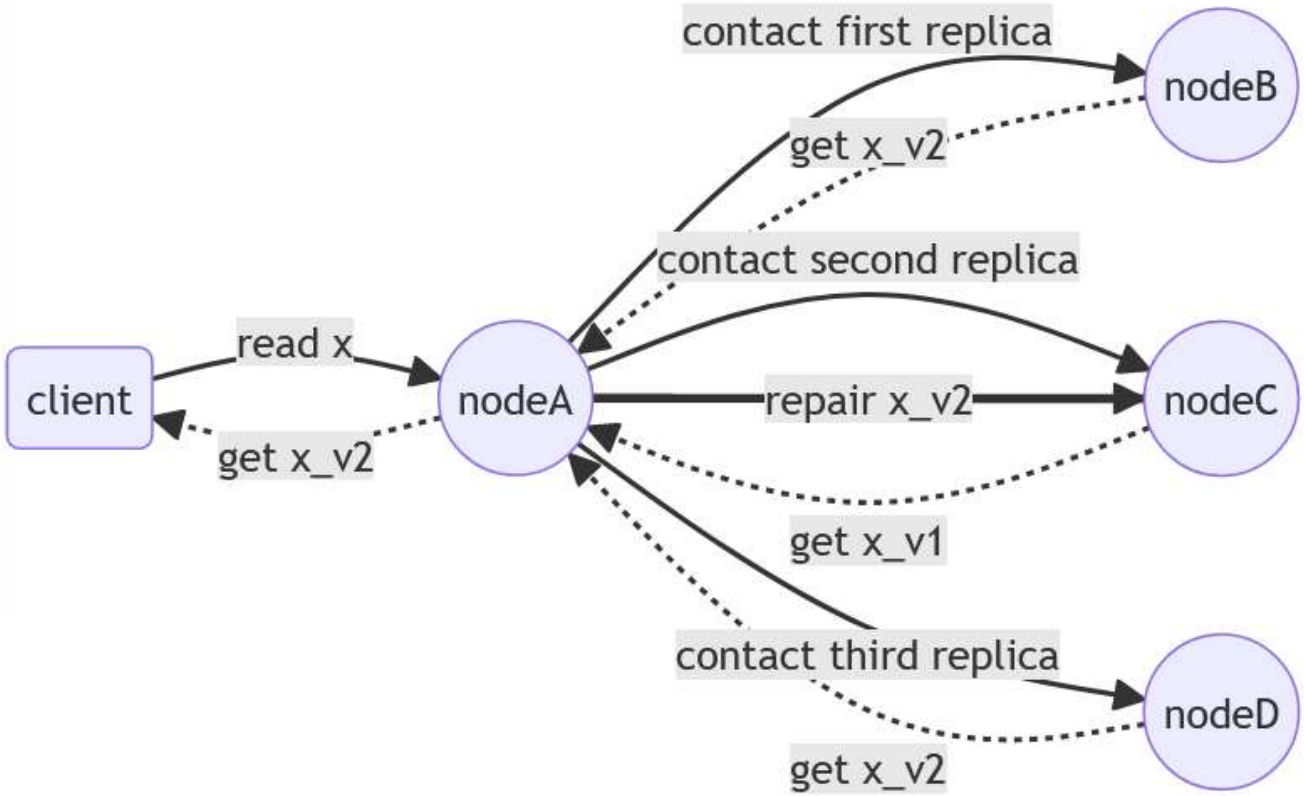


---- Mechanismen

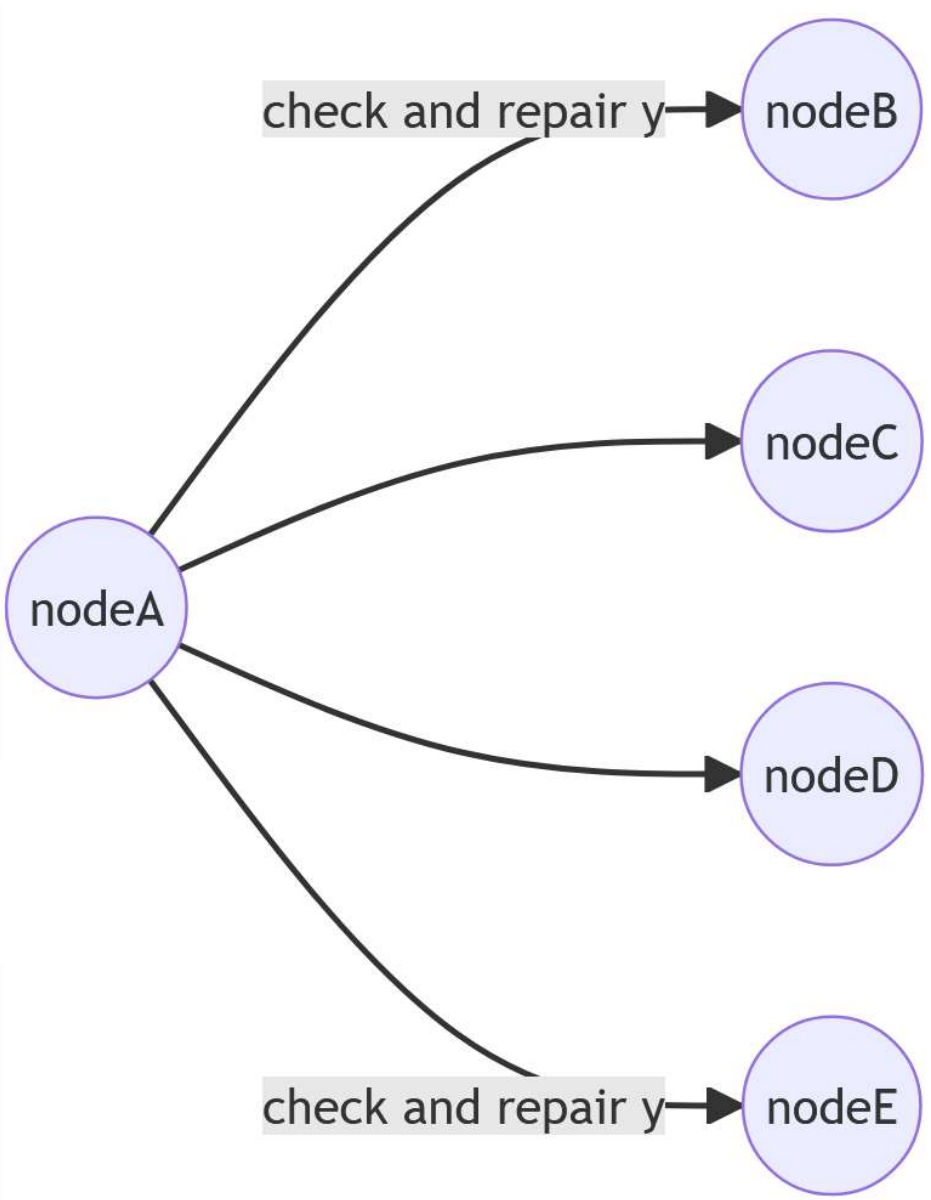
Prof. Dr. Ingo Claßen

Mittwoch, 8. März 2023 16:20

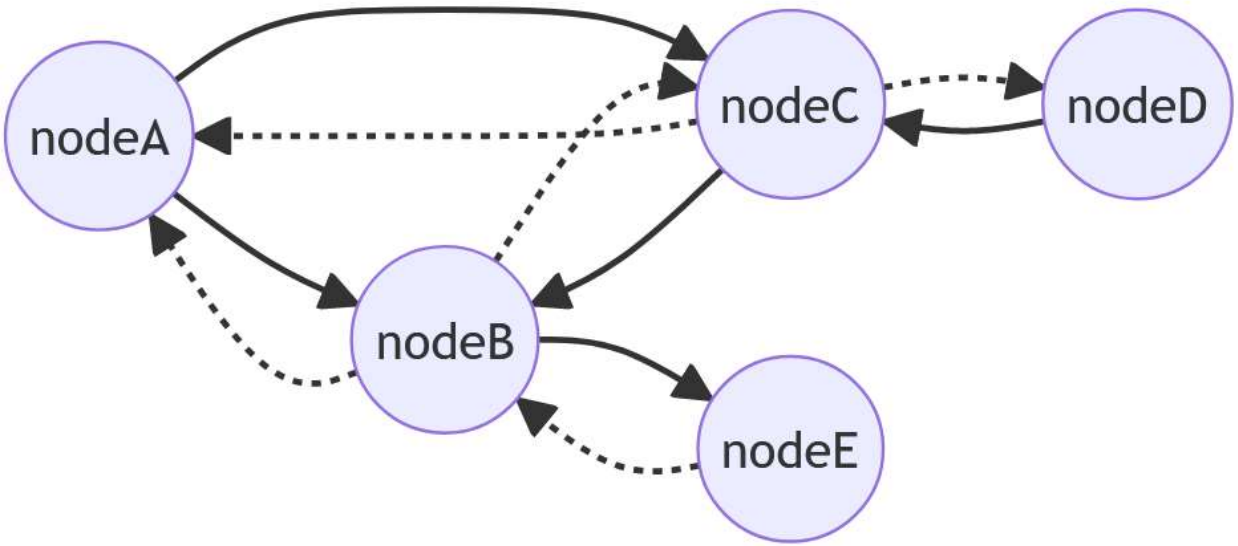
- Client kontaktiert irgendeinen Knoten um x zu lesen, in diesem Fall nodeA
- NodeA enthält kein replikat und kontaktiert Knoten mit Replikaten entsprechend Quorum
- NodeB und nodeD liefern Version v2 von x, nodeC alte Version v1
- Client erhält x in Version v2
- NodeA aktualisiert nodeC (repair)



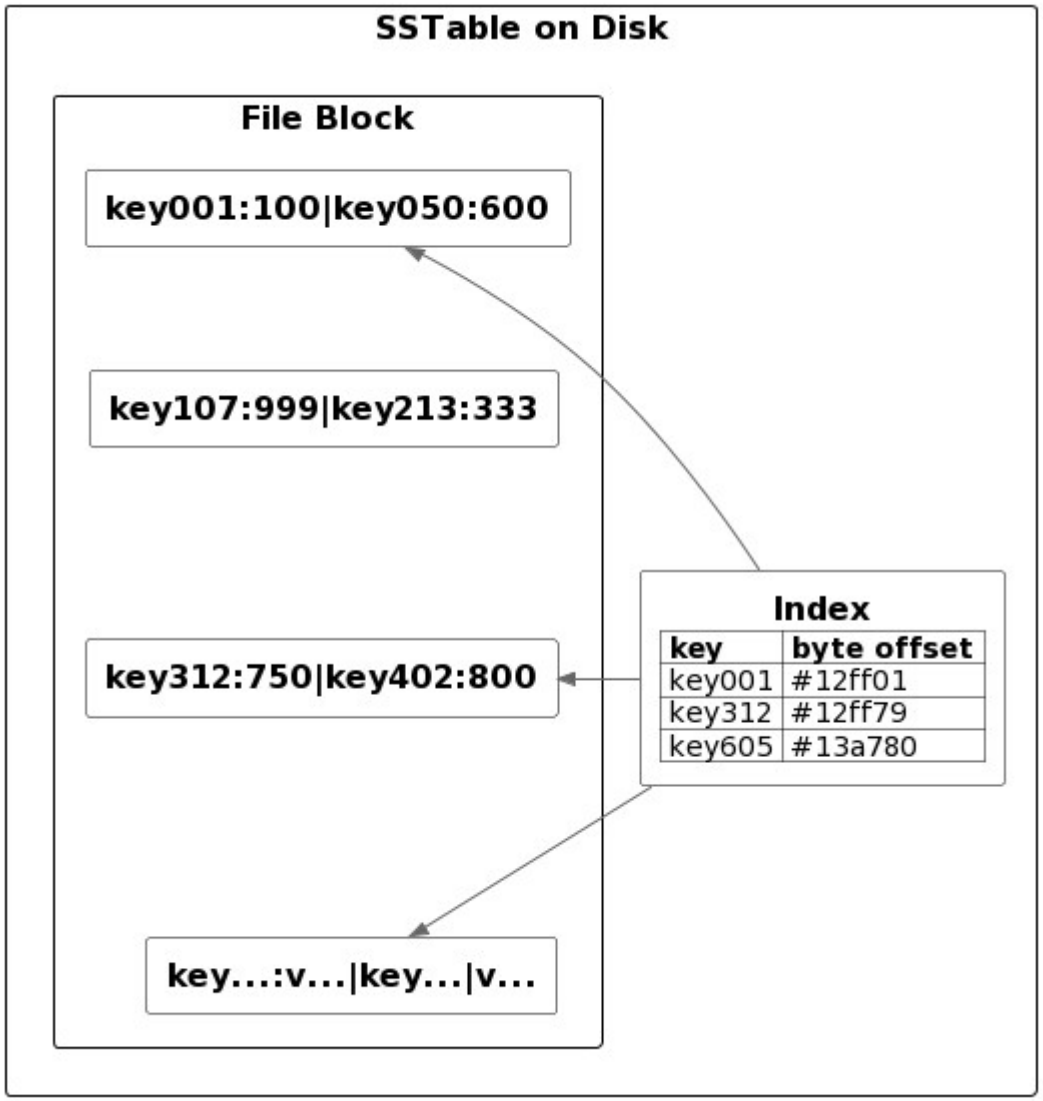
- Zufällig Auswahl eines Knoten mit aktueller Version eines Wertes, in diesem Fall y
- Zufällige Auswahl von Replikatsknoten
- Überprüfung ob neuste Version von y vorhanden
  - Wenn nichr, Ersetzung mit der neuesten Version von y (repair)

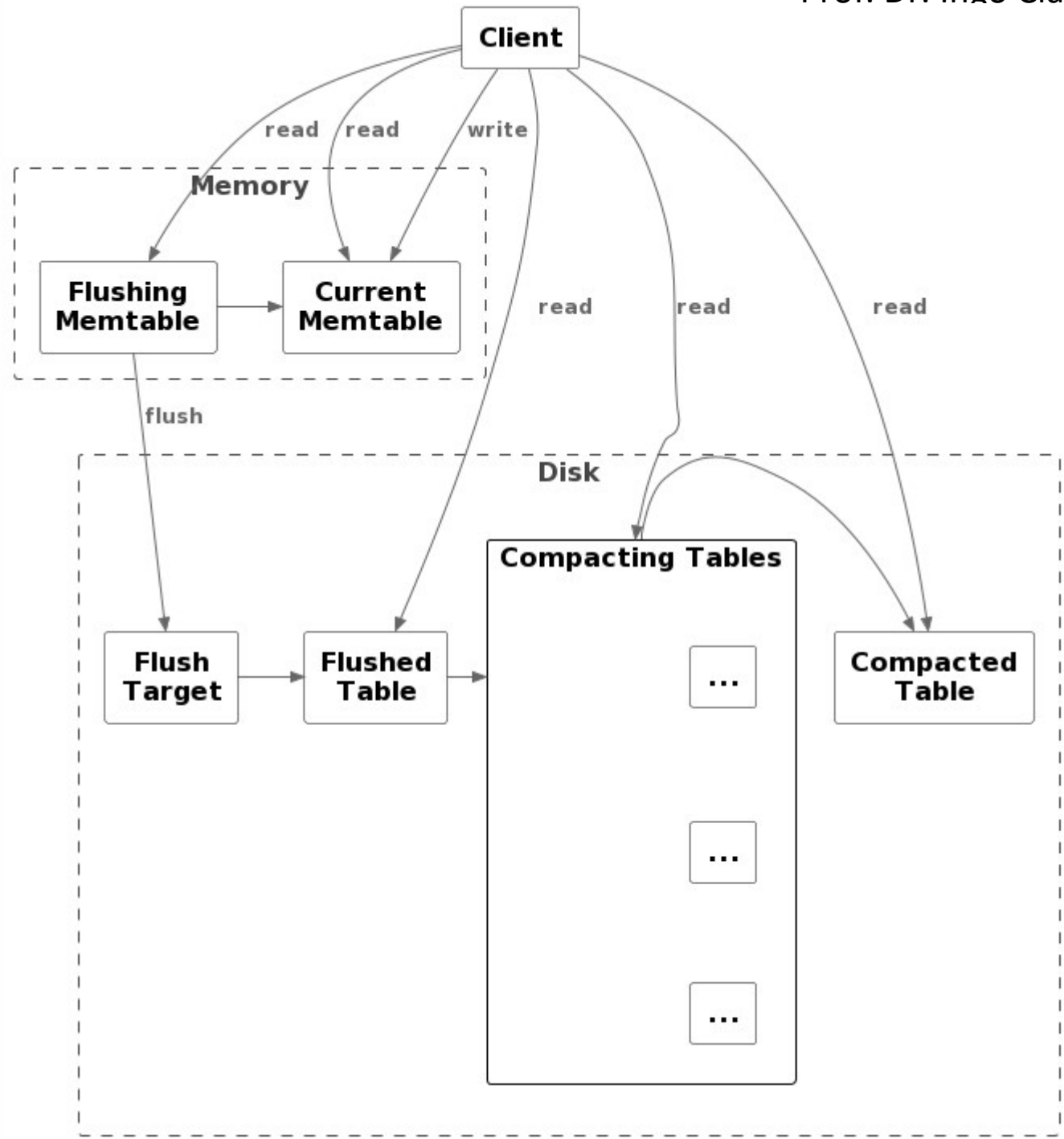


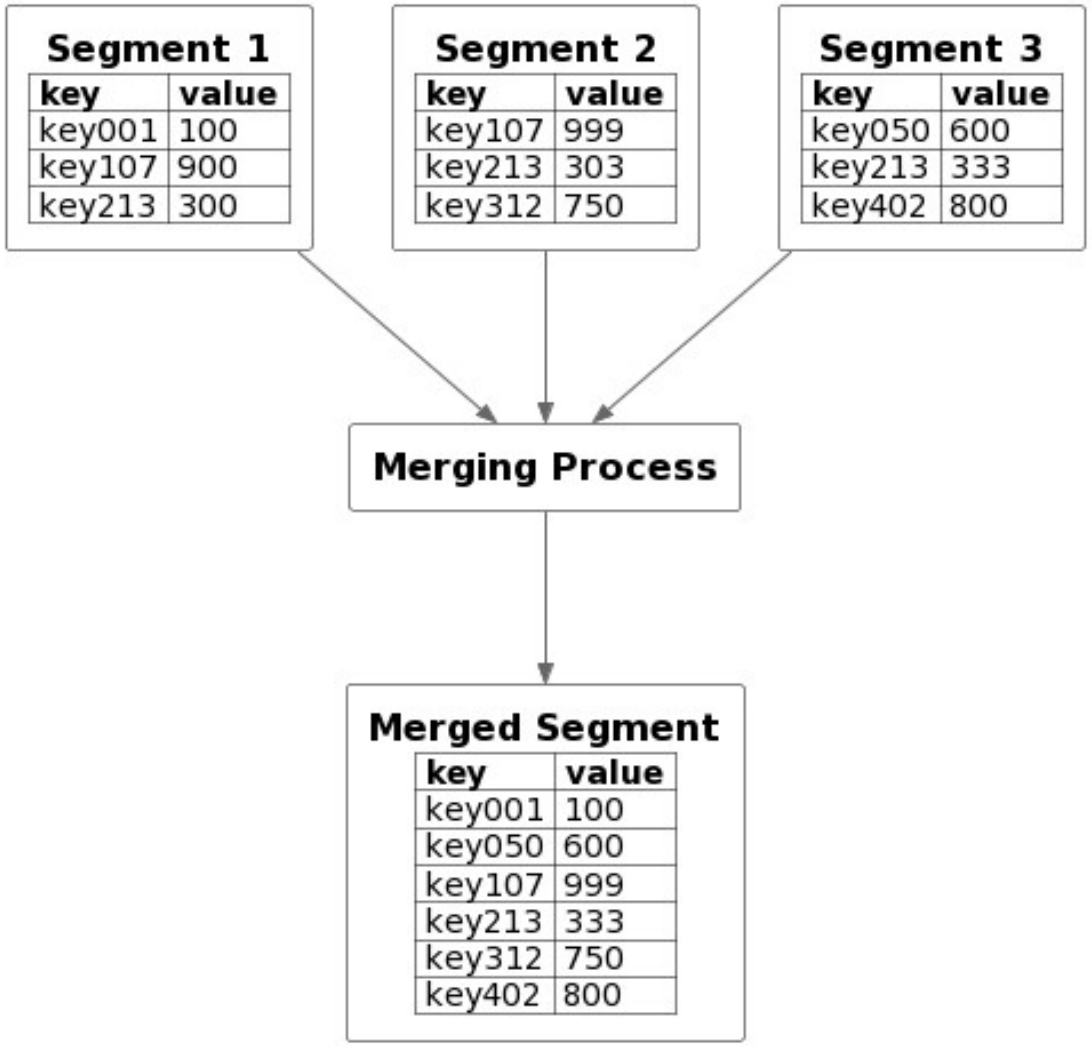
- Knoten kontaktieren zufällig andere Knoten



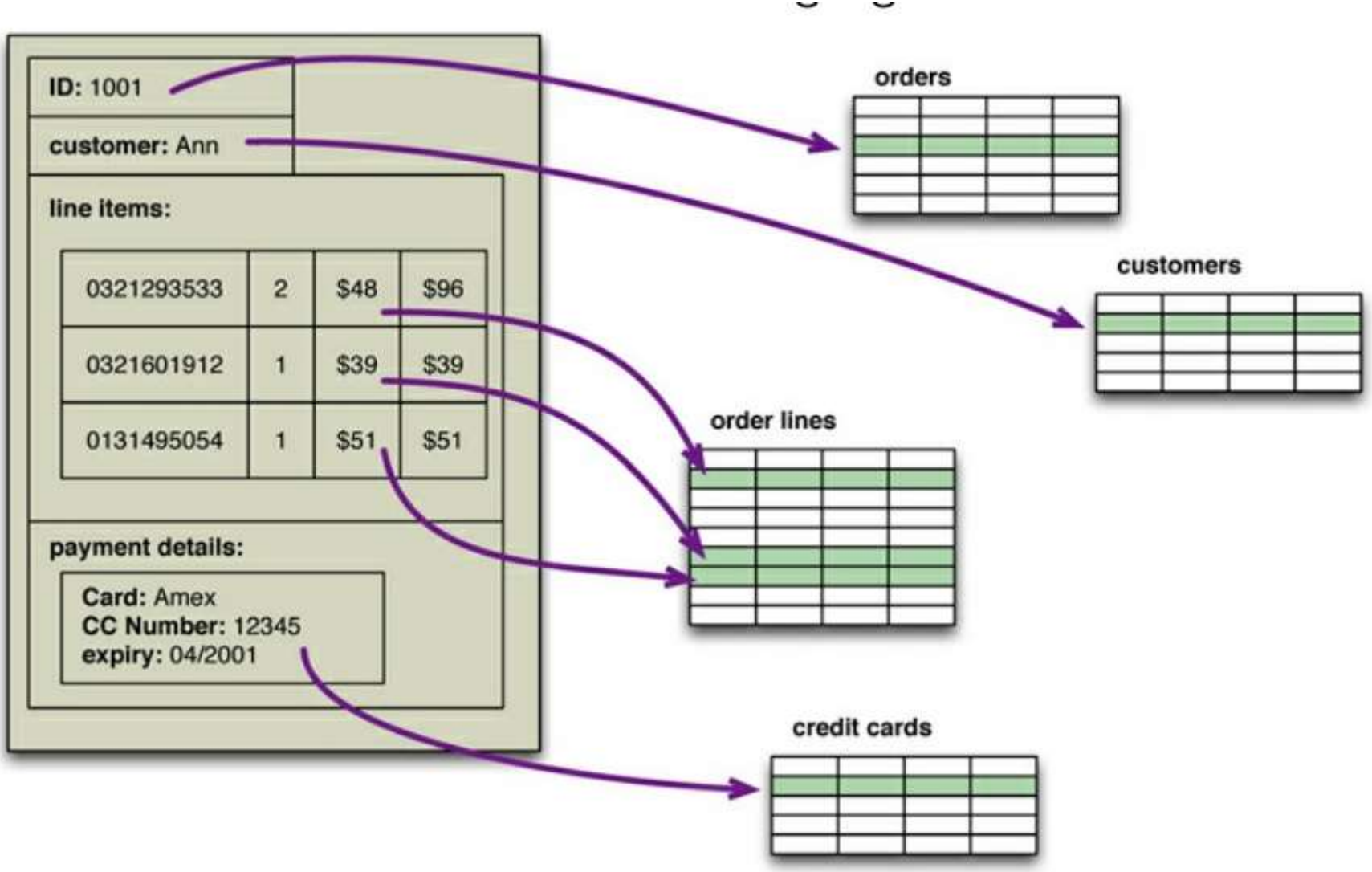
- Schreib-optimierte Speicherstrukturen, die häufig in NoSql-Systemen verwendet werden
- Typischer Einsatz bei schreibintensiver Arbeitslast,
- Gespeichert in Sorted Strings Tables (SSTable), unveränderlich
- Enthalten Schlüssel-Wert-Paare
- Memtable im Haupspeicher

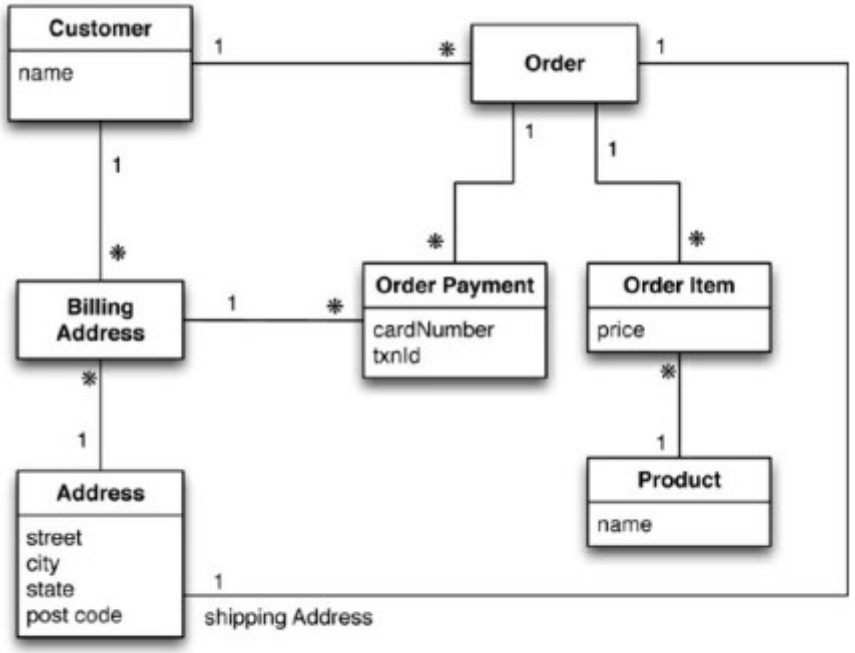

SSTable on Disk
File Block
key001:100|key050:600
key107:999|key213:333
key312:750|key402:800
key...:v...|key...|v...
Index
key
byte offset
key001
#12ff01
key312
#12ff79
key605
#13a780


Client
read
read
write
Memory
Flushing Memtable
Current Memtable
read
read
read
flush
Disk
Compacting Tables
Flush Target
Flushed Table
...
...
...
Compacted Table


Segment 1
key value
key001 100
key107 900
key213 300
Segment 2
key value
key107 999
key213 303
key312 750
Segment 3
key value
key050 600
key213 333
key402 800
Merging Process
Merged Segment
key value
key001 100
key050 600
key107 999
key213 333
key312 750
key402 800

- Anwendung arbeitet mit komplexen Strukturen
- Relationales Datmodell erfordert Zerlegung in flache Tabellen





Customer

Id	Name
1	Martin

Orders

Id	CustomerId	ShippingAddressId
99	1	77

Product

Id	Name
27	NoSQL Distilled

BillingAddress

Id	CustomerId	AddressId
55	1	77

OrderItem

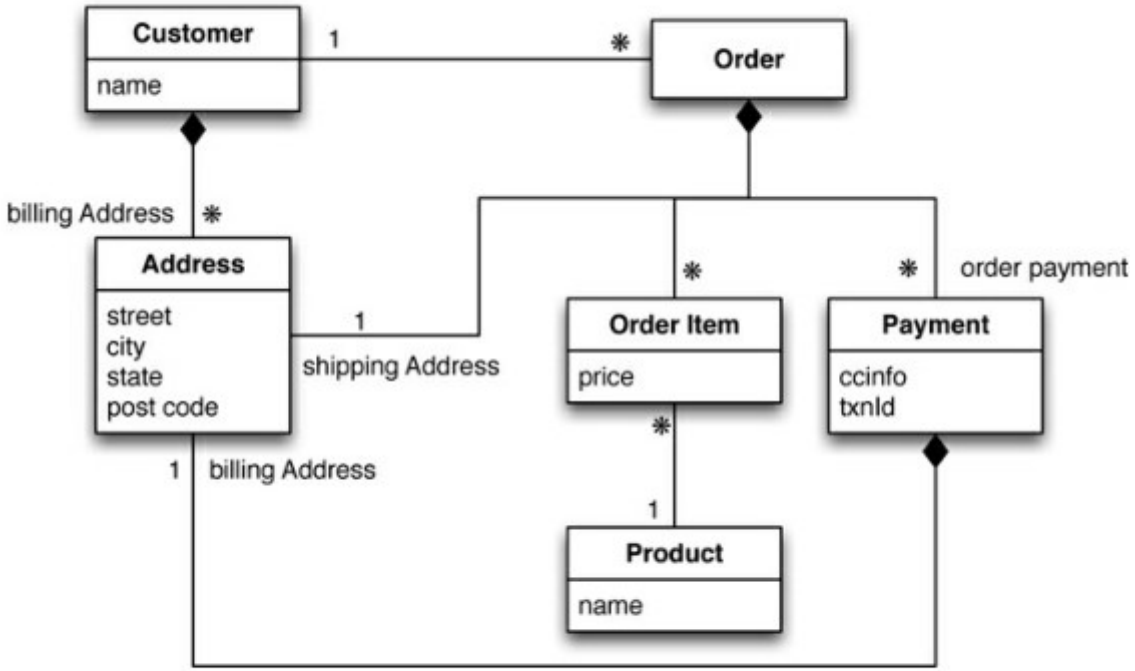
Id	OrderId	ProductId	Price
100	99	27	32.45

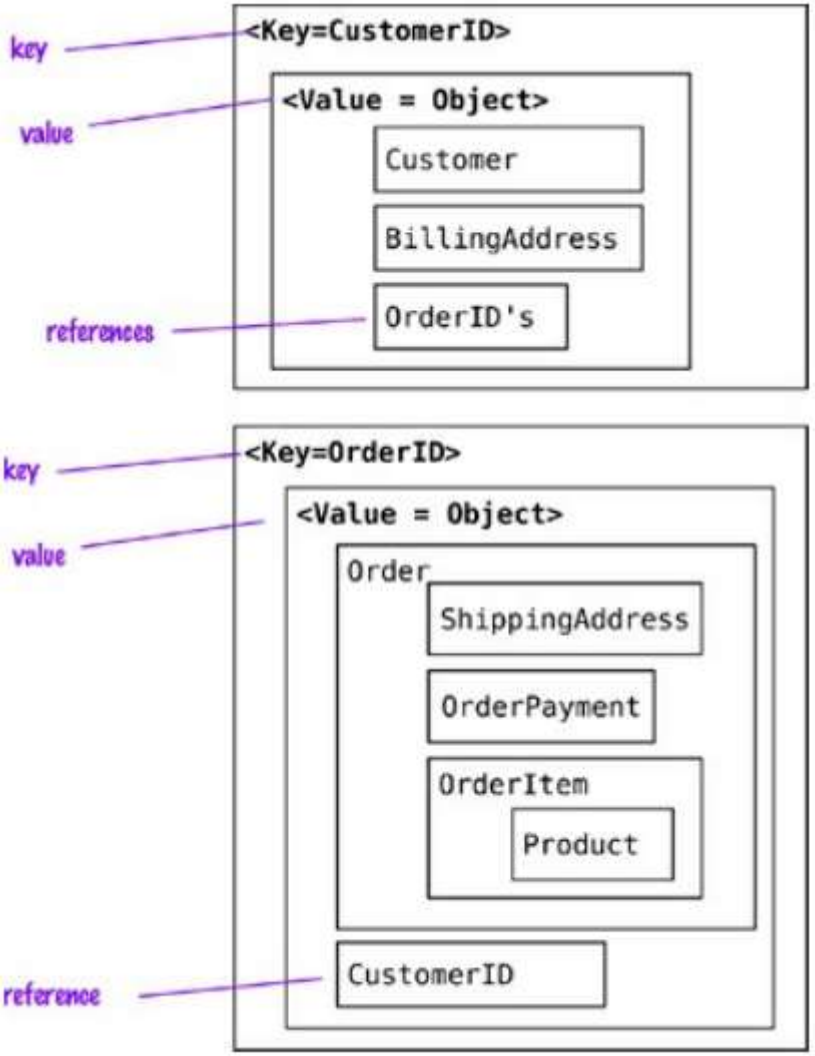
Address

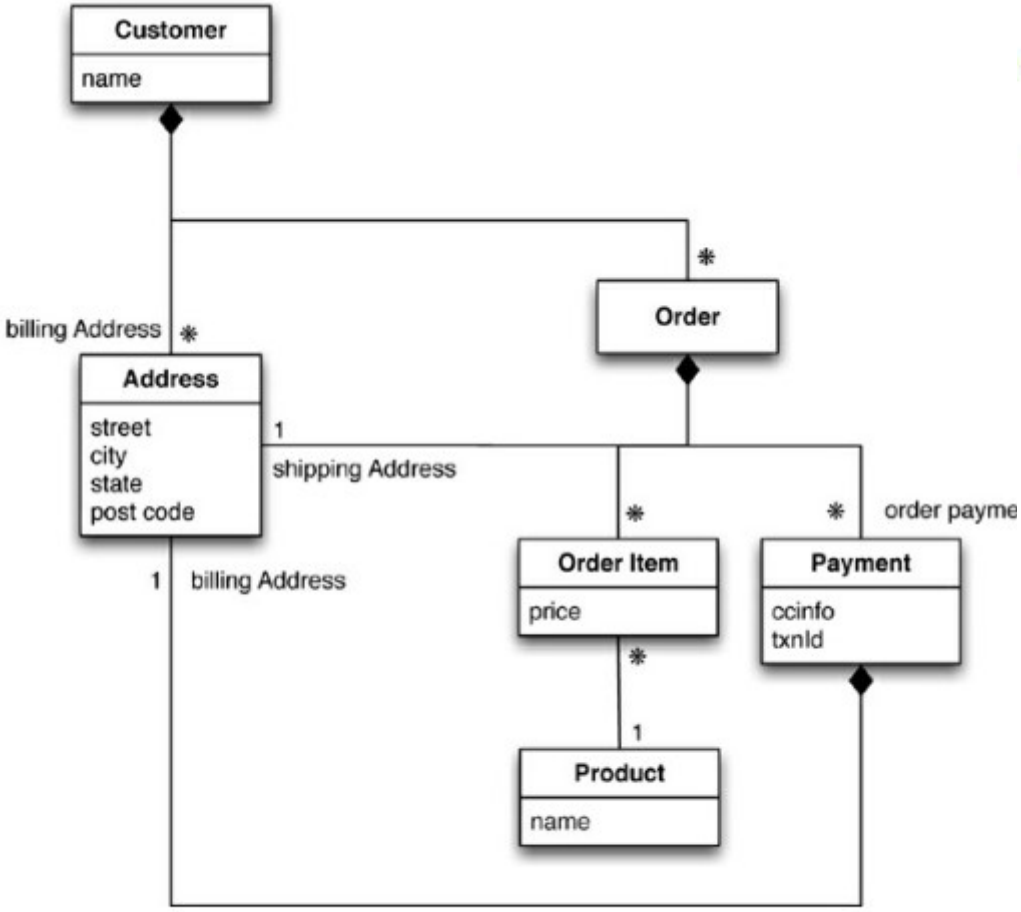
Id	City
77	Chicago

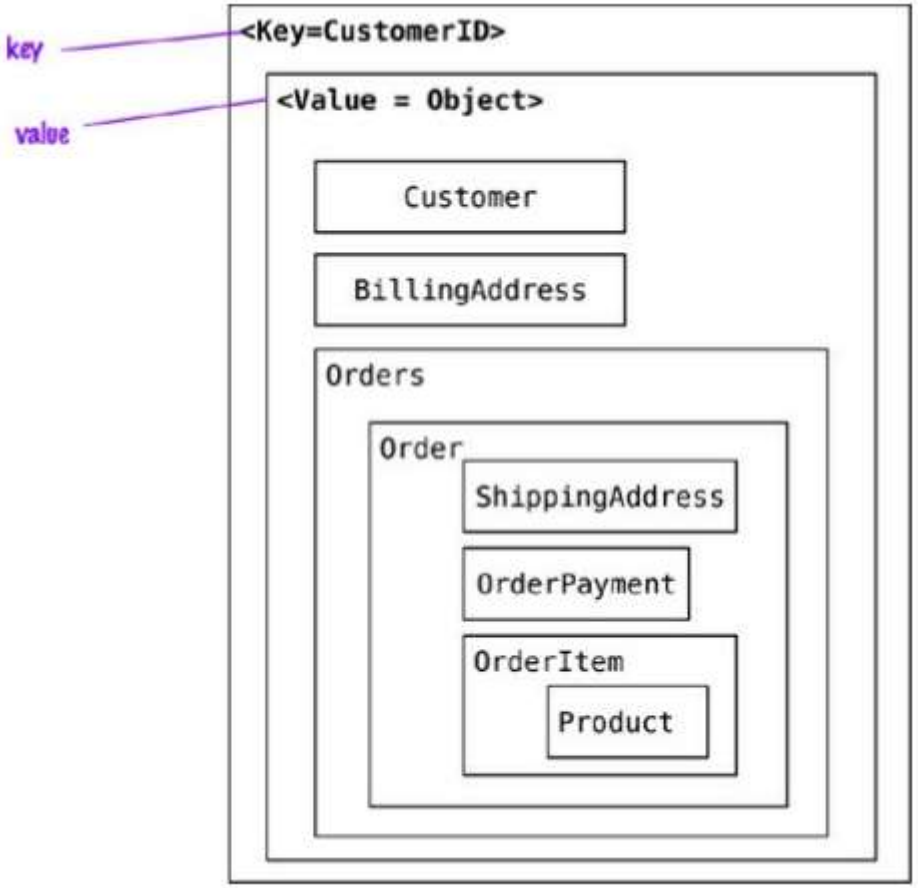
OrderPayment

Id	OrderId	CardNumber	BillingAddressId	txnId
33	99	1000-1000	55	abelif879rft


Customer
name
1
*
Order
billing Address
*
Address
street
city
state
post code
1
shipping Address
*
Order Item
price
*
Payment
ccinfo
txnId
order payment
1
billing Address
1
Product
name


key
<Key=CustomerID>
value
<Value = Object>
Customer
BillingAddress
references
OrderID's
key
<Key=OrderID>
value
<Value = Object>
Order
ShippingAddress
OrderPayment
OrderItem
Product
reference
CustomerID


Customer
name
billing Address
Address
street
city
state
post code
1
billing Address
Order
shipping Address
order payme
Order Item
price
Payment
ccinfo
txnId
Product
name


key
value
<Key=CustomerID>
<Value = Object>
Customer
BillingAddress
Orders
Order
ShippingAddress
OrderPayment
OrderItem
Product

- Selbstbeschreibende baumartige Strukturen
- Aggregate, z.B in JSON:

```
{
  "text":
  "Das Web ist heutzutage eine wichtige Quelle der
  Datenanalyse. Wikipedia, News-Portale, Soziale
  Medien usw. werden zu Informationslieferanten.
  ",
  "author": {
    "name": "Classen",
    "location": "Berlin, Germany",
  },
  "tags": ["Searching", "Classifiction", "Clustering"]
}
```

- Datenbank = Menge von Kollektionen
- Kollektion
  - Menge von Dokumenten
  - entspricht Tabelle
  - flexibles Schema
- Dokument
  - entspricht einer Zeile in einer Tabelle
  - ähnliche Dokumente in einer Kollektion

Anwendungsbereiche dokumentenorientierter Datenbanksysteme

Prof. Dr. Ingo Claßen

Montag, 16. Januar 2023 08:03

- Anwendungen mit reichhaltigen Domänenmodellen
- Web-Anwendungen mit großer Nutzeranzahl
- Anwendungen mit starker horizontaler Skalierung
- CRM-Systeme
- Katalogsysteme
- Portale, News, Wiki, Foren, Blogs

- Martin Kleppman: Designing Data-Intensive Applications, O'Reilly, 2017
- NoSQL - Einstieg in die Welt nichtrelationaler Web 2.0Datenbanken, 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