JAVACREAM

Training
Consulting
Projectmanagement

Apache Cassandra

Kurze Vorstellung

- Name, Rolle im Unternehmen
- Themenbezogene Vorkenntnisse (Datenbank-Systeme, Cassandra)
- Aktuelle Problemstellung
- Konkrete individuelle Zielsetzung
- Fragen
  - Funktioniert die Technik?
    - insbesondere GitHub und das digitale Flipchart
  - Nutzen Sie die Integrata-Cegos-Rechner
  -

Umgebung

- Eigener Rechner mit Docker-Installation
- Arbeiten mit Remote-Rechnern der Integrata-Cegos
  - https://integrata-cegos.deskmate.me
  - User
    - tn20.raum01@integrata-cegos.de -> Ebeling
    - tn21.raum01@integrata-cegos.de -> Zerbe
  - Passwort und weiteres Vorgehen im Seminar
- Cassandra-Installation auf h2908727.stratoserver.net
  - Auch hier weitere Informationen während des Seminars

Überblick

Bedarf an Datenbank-Systemen

- Datensenke, die in der Lage ist, Informationen aufzunehmen
- Server-Prozesse, die auf einem Host laufen
- “Quality of Service”
  - Verfügbarkeit
  - Datenvolumen
  - Konsistenz
- Anwendungs-Programme können Daten in einer für sie sinnvollen Struktur ablegen

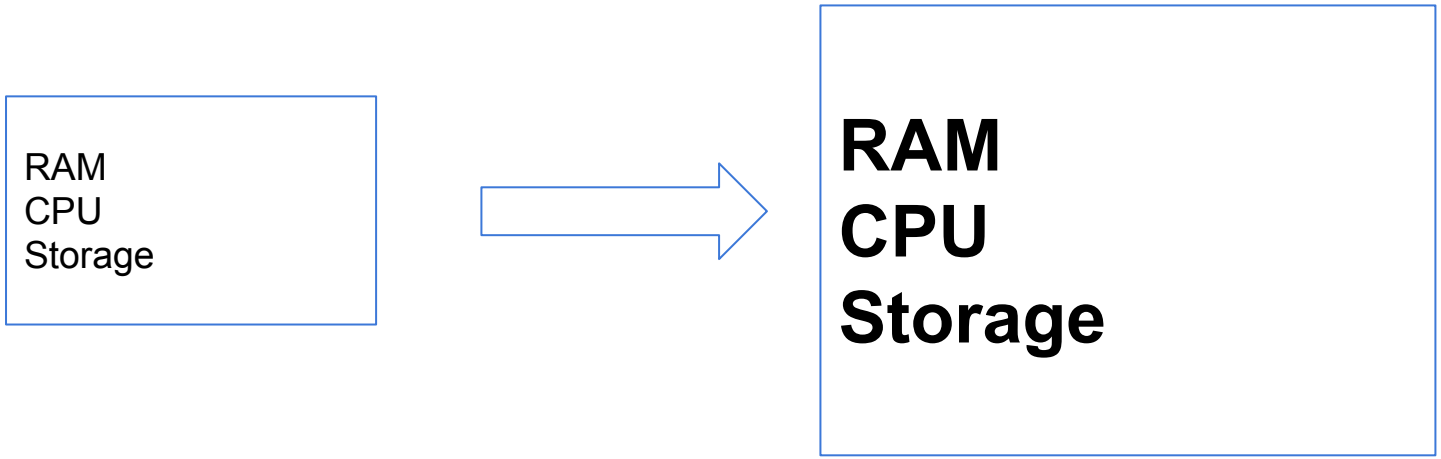
Anforderungen

- Big & Fast Data
  - Menge der zu speichernden Daten immens
    - Daten-Quellen sind nicht nur interaktive Prozesse, sondern auch maschinell erstellte Daten (Log-Dateien) oder auch gekoppelte Prozesse
  - Auswertungen sollen aber immer noch in interaktiven Systemen in Echtzeit erfolgen
- Skalierbarkeit
  - Primär ein “up-scaling”, Bereitstellung zusätzlicher Ressourcen
    - down-scaling ist auch wünschenswert
  - Dynamisch ohne Bedarf für Wartungsfenster = Totzeiten

Problem: Realisierung eines skalierbaren Systems

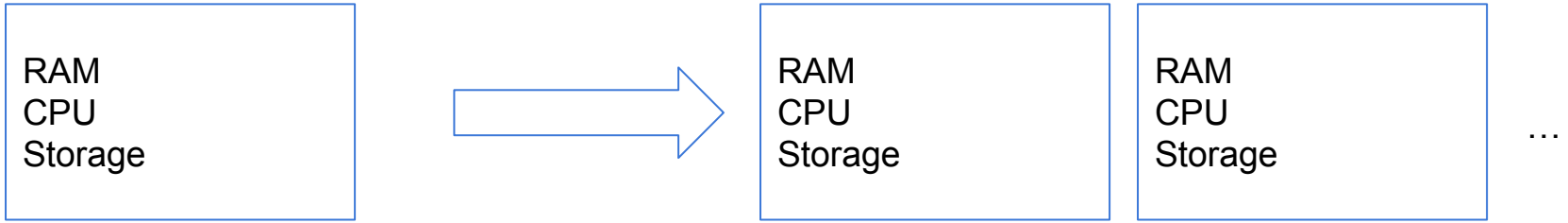
- 2 Kategorien
  - Vertikale Skalierbarkeit
  - Horizontale Skalierbarkeit

Vertikale Skalierung



Das ursprüngliche System wird auf eine neue Umgebung migriert und anschließend neu gestartet

Horizontale Skalierung



Das ursprüngliche System wird durch weitere “Knoten” ergänzt, die sich die Last neu aufteilen (Rebalancing)

Diskussion: Welche Skalierung passt in der heutigen Welt am Besten zu unseren Anforderungen?

- Horizontale Skalierung ist die bessere, modernere Variante
  - Up- und Down-Scaling kann eine automatische Orchestrierung durchführen
  - Totzeiten können vermieden werden
    - Rebalancing kann “im Hintergrund” während des laufenden Betriebs erfolgen
  - Kosten für die Bereitstellung der Host-Rechner
    - Kaufmännische Limits (Mächtige Hardware ist teuer) und technische Limits (es gibt Grenzen für Rechner-Ressourcen)

Skalierbare Datenbank-Systeme

- Oracle, MySql, Mssql
  - Primär vertikal skalierende Systeme
- MongoDB
  - Horizontale Skalierung, allerdings mit einem zentralen Primary-Server (“Master”)
- Apache Cassandra
  - Horizontal skalierendes System ohne Primary
  - Damit ist die Cassandra-Datenbank konzeptuell “unendlich groß”
  - Couchbase, Redis, Memcached, … ebenfalls horizontal skalierbar

Cassandra-Überblick

Distributionen

- Apache Cassandra
  - Open Source (!)
    - Lizenzfrei in der Benutzung
  - Community ohne kommerziellen Support
- Anbieter mit kommerziellem Support
  - DataStax
  - …
- Anbieter mit fertigen Cloud-Lösungen
  - Betrieb der Cassandra erfolgt in einer Cloud des Anbieters
  - z.B. DataStax mit der Astra DB

Umsetzung

- Cassandra ist Java-basiert
  - Ein Cassandra-Knoten läuft in einem Java-Prozess, der Java Virtual Machine
- Damit ist die Überwachung des Cassandra-Prozesses primär eine Überwachung der Java Virtual Machine
- Als Java-Anwendung ist Cassandra portabel
  - Linux-Distributionen
  - Windows

Ein erstes Arbeiten

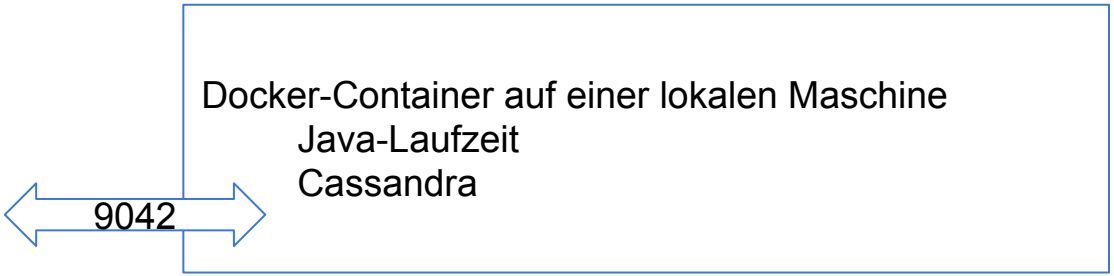
- Download der Cassandra-Distribution
  - https://cassandra.apache.org/_/download.html
- Alternative: Verwendung eines Docker-Containers
  - https://hub.docker.com/_/cassandra
  - In einer Konsole: docker pull cassandra
- ToDo:
  - Auf Ihren Deskmate-Maschinen bitte beides durchführen

Ein Blick in die Cassandra-Distribution

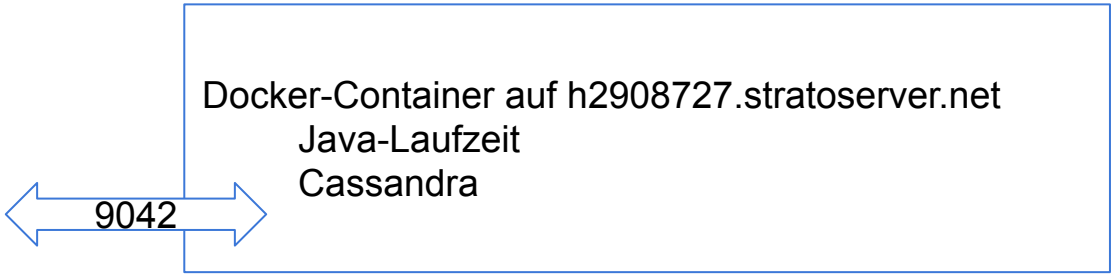
- Größe etwa 50 MByte
  - Vorsicht: Runtime der Java Virtual Machine ist hier nicht enthalten
- Hauptverzeichnisse
  - bin
    - Aufruf-Skripte
    - cassandra zum Aufbauen eines Java-Aufrufs
  - conf
    - cassandra.yaml als zentrale Konfigurationsdatei
    - etwa 50 Parameter
  - lib
    - Große Anzahl von jar-Dateien, die die gesamte Java-Logik der Anwendung enthalten

Start des Datenbank-Servers

- Direkt
  - Notwendig ist eine installierte Java-Umgebung
    - bevorzugt Java 11
  - cassandra -f
- Bevorzugt
  - docker run – rm -p 9042:9042 –name cassandra cassandra

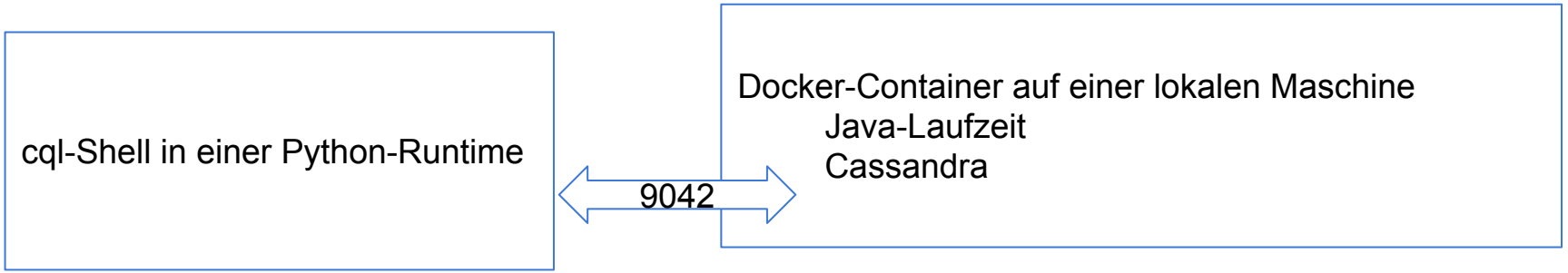


Vereinfachte Umgebung



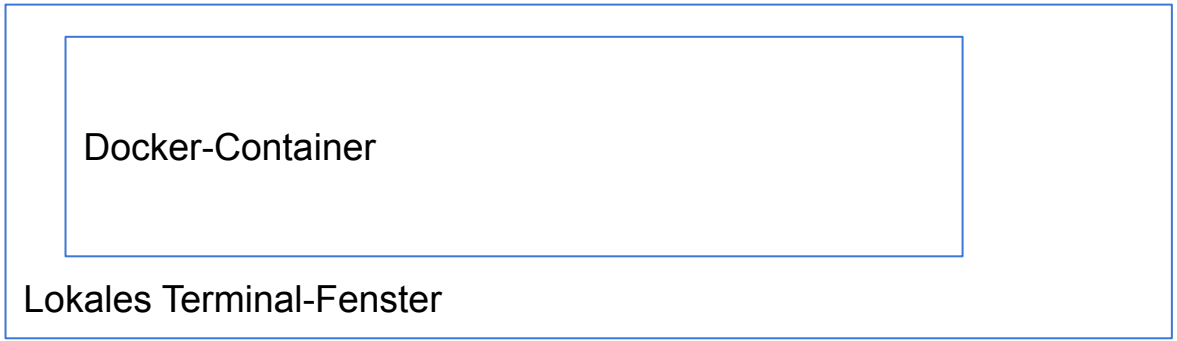
Zugriff auf den Datenbank-Server

- cql-Shell als Bestandteil der Cassandra-Distribution
  - Aufruf cqlsh <host>
- Im Detail etwas aufwändiger
  - Die Shell läuft als Python-Anwendung
    - Auf einer Windows-Maschine nicht ganz trivial

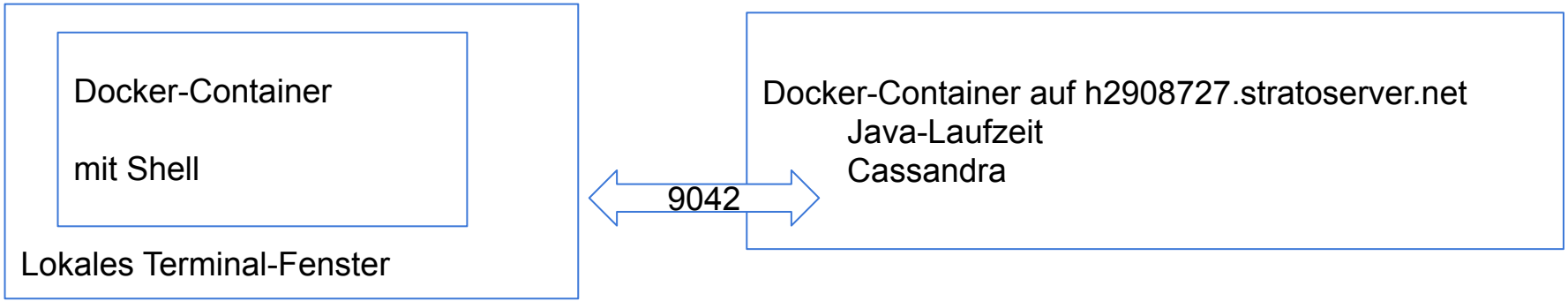


CQL-Shell als Docker-Container

- docker run – rm -it –name cqlshell cassandra cqlsh <host>



Vereinfachte Umgebung



docker run -it --rm cassandra cqlsh h2908727.stratoserver.net

```
rainersawitzki@sophie-HP-Pavilion-Laptop-15-cs3xxx:~$ docker run -it --rm cassan
dra cqlsh h2908727.stratoserver.net
Connected to Test Cluster at h2908727.stratoserver.net:9042
[cqlsh 6.0.0 | Cassandra 4.0.1 | CQL spec 3.4.5 | Native protocol v5]
Use HELP for help.
cqlsh>
```

Erstes Arbeiten mit der Cassandra Datenbank

Einführung einiger Begriffe

- Cluster
  - Eine Menge von Cassandra-Instanzen, den Cassandra-Nodes
    - Aktuell auf dem Host: Der “Cluster” besteht nur aus einem einzigen Knoten
- Keyspace
  - Ein Cluster kann eine beliebige Menge von Keyspaces verwalten
  - Jeder Keyspace definiert die Daten-Ablage für eine Anwendung
  - Nach dem Hochfahren des Clusters sind ausschließlich interne Keyspaces vorhanden
- Table
  - Konkrete Daten-Ablage mit Struktur

Einführung einiger Begriffe, Teil 2

- Row
  - Vorsicht: Das ist eine Vereinfachung!
  - Das ist ein strukturierter Datensatz
  - Manipulation eines Datensatzes erfolgt “CRUD”
    - Create (Neuanlage)
    - Read (Lesen)
    - Update (Aktualisierung)
    - Delete (Löschen)

Namenskonventionen

- Die Namen in Cassandra werden stets in Kleinbuchstaben umgewandelt
  - create keyspace HUGO;
  - use hugo;
- Konvention:
  - “Snake-Syntax”: bei längeren Namen die Betonung durch Unterstriche umsetzen
    - testsawitzki -> test_sawitzki
      - Camel-Case TestSawitzki -> testsawitzki
- Theoretisch kann Cassandra auch Großbuchstaben
  - create keypace ‘HUGO’
  - use ‘HUGO’

Stand

- Ein laufendes Datenbank-System steht zur Verfügung
- Kommunikation erfolgt über ein Konsolen-basiertes Hilfsprogramm, die CQL-Shell
- Wir kennen die Befehle, um Daten in der Cassandra-Datenbank verwalten zu können

Stand: Wertung

- Ein laufendes Datenbank-System steht zur Verfügung
  - Dieses System ist in keinster Weise repräsentativ
    - Weniger als ein Hello-World!
- Kommunikation erfolgt über ein Konsolen-basiertes Hilfsprogramm, die CQL-Shell
  - Diese Shell wird in der Praxis gerne verwendet
  - Ist zu ergänzen um weitere z.B. grafische Tools
- Wir kennen die Befehle, um Daten in der Cassandra-Datenbank verwalten zu können
  - VORSICHT
    - Jegliche Analogie zu anderen Datenbank-Systemen ist brandgefährlich!
    - Cassandra ist “anders”
    - Begriffe wie Tabellen, Zeilen, auch die Abfrage-Sprache sind nicht aus Systemen wie Oracle zu übernehmen

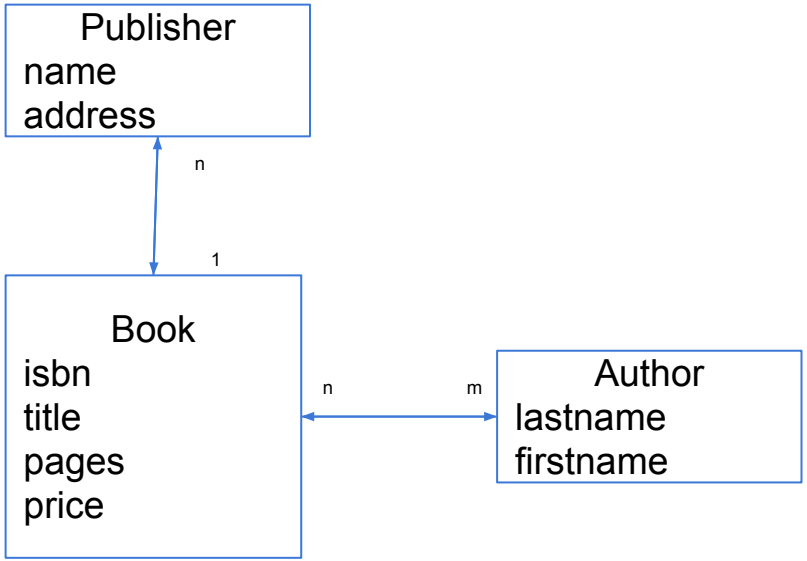
Exkurs: NoSql

Fachliche Modellierung der Daten

Entity-Modell

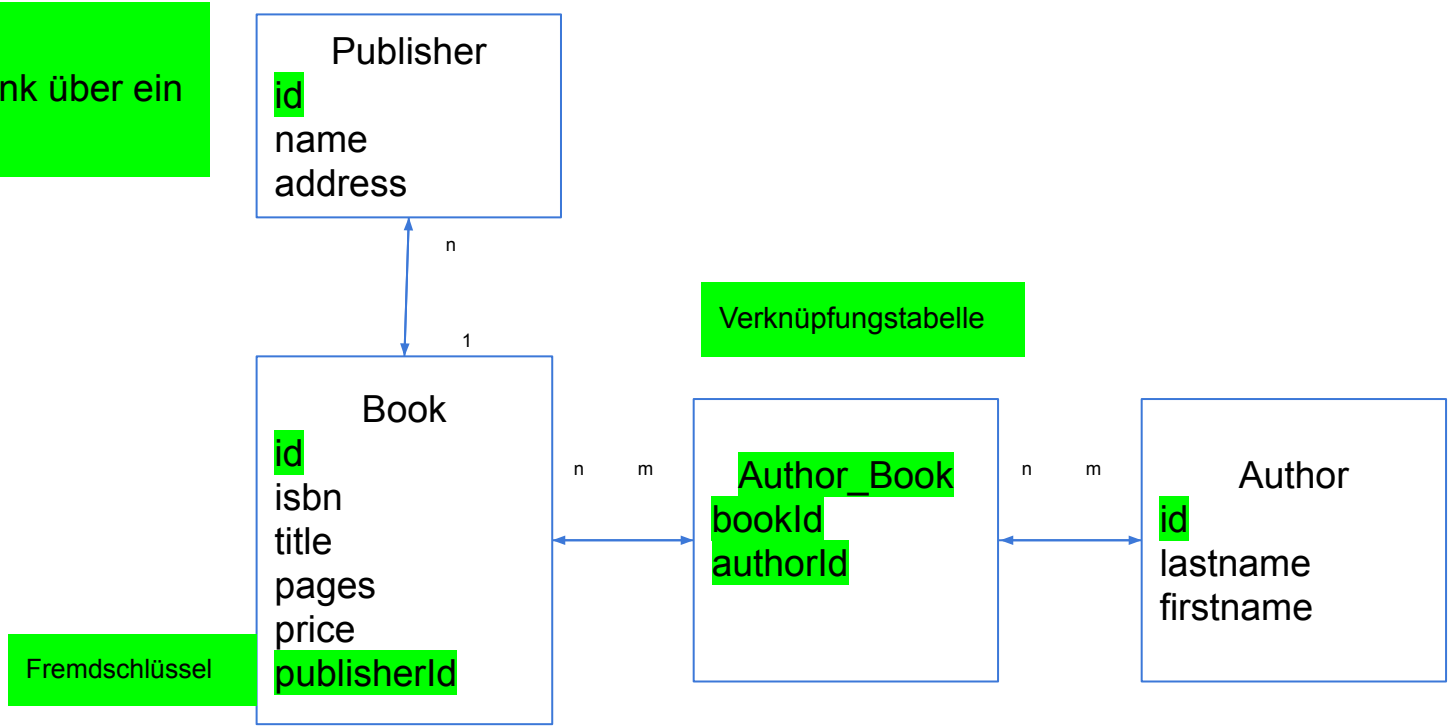
enthält fachliche Informationen

gesetzt für alles weitere



Technisches Modell: Technologie-abhängig

Umsetzung mit einer relationalen Datenbank über ein relationales Modell



Relationale Modelle

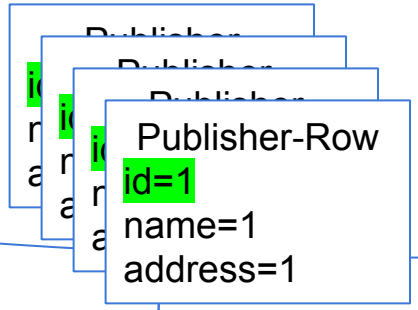
- Navigation auf beliebige Daten ist von allen Einstiegspunkten heraus beliebig möglich
- Joinen
  - select author.lastname from Publisher p where p.name = ‘Springer’ JOIN Book on b.publisherid = p.id JOIN …
- Relationale Datenbanken optimieren ihre interne Datenhaltung auf Joins
  - Selbst komplexe Modelle können mit bis zu 10 Joins effizient und schnell verarbeitet werden

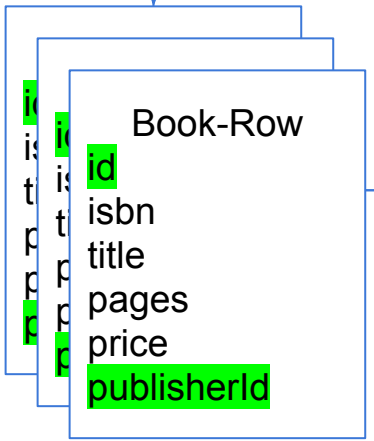
Diskussion

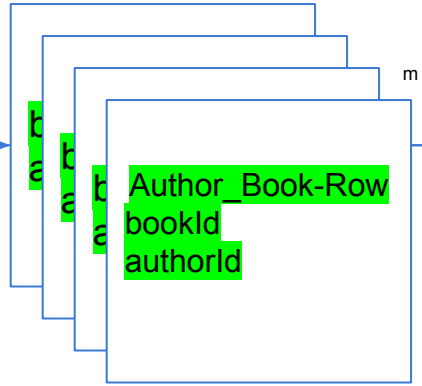
- Können relationale Datenbanken in einem horizontal skalierenden System betrieben werden?
- Neuer Knoten
  - Wie würde das denn aussehen?
    - Ausgangssituation: Ein Knoten, in dem alles verwaltet wird
    - Rebalancing beim Start eines neuen Knotens, um ein Lastproblem zu beheben?
    - Hinweis: Dafür gibt es zumindest 2 Möglichkeiten
- Welche Auswirkungen erwarten Sie bezüglich der Möglichkeiten des Join?

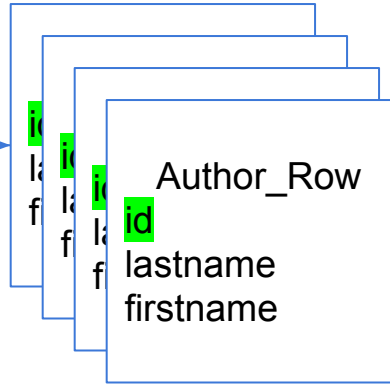
Ausgangssituation

Relationale Daten, ein Knoten

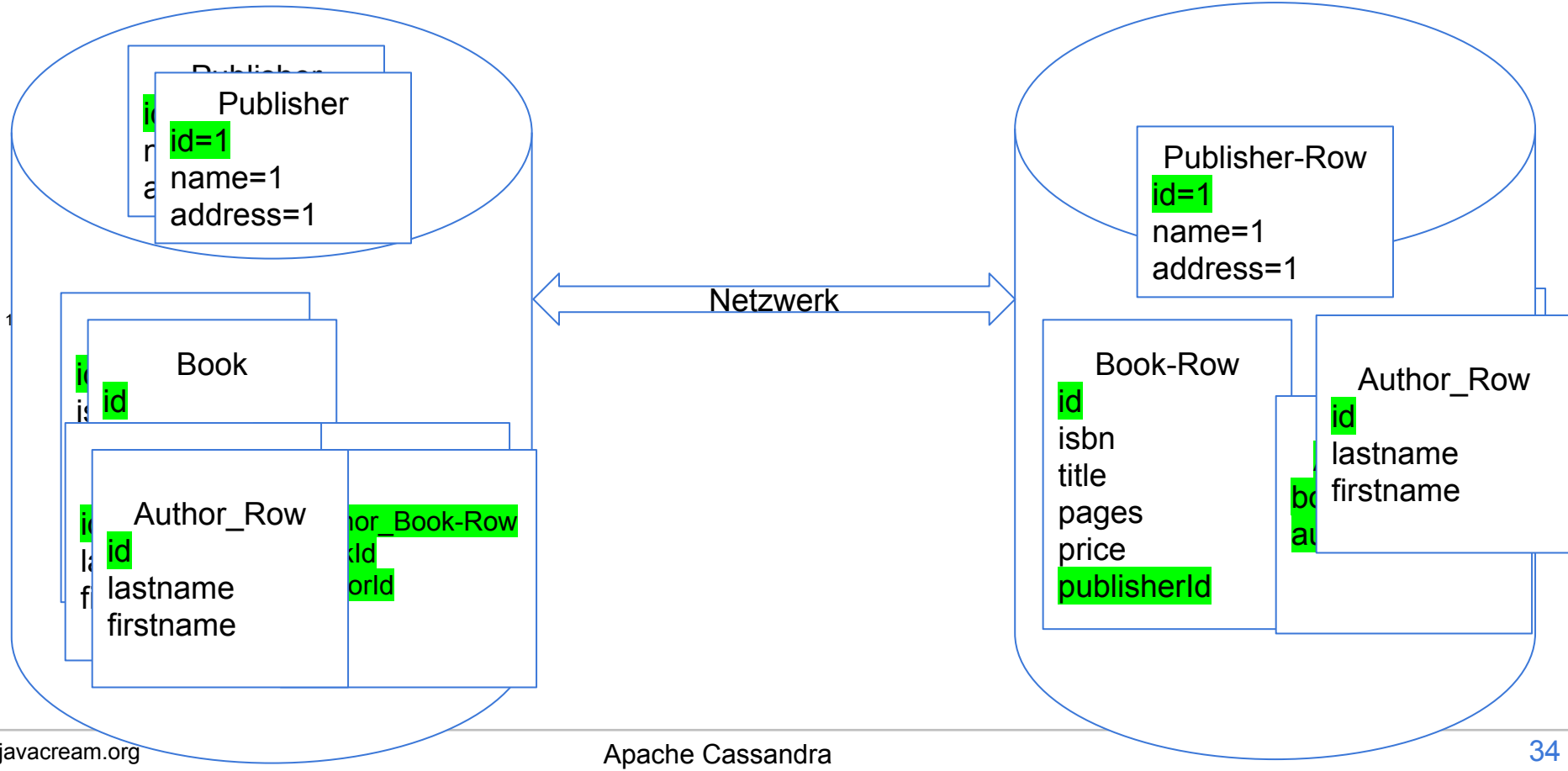




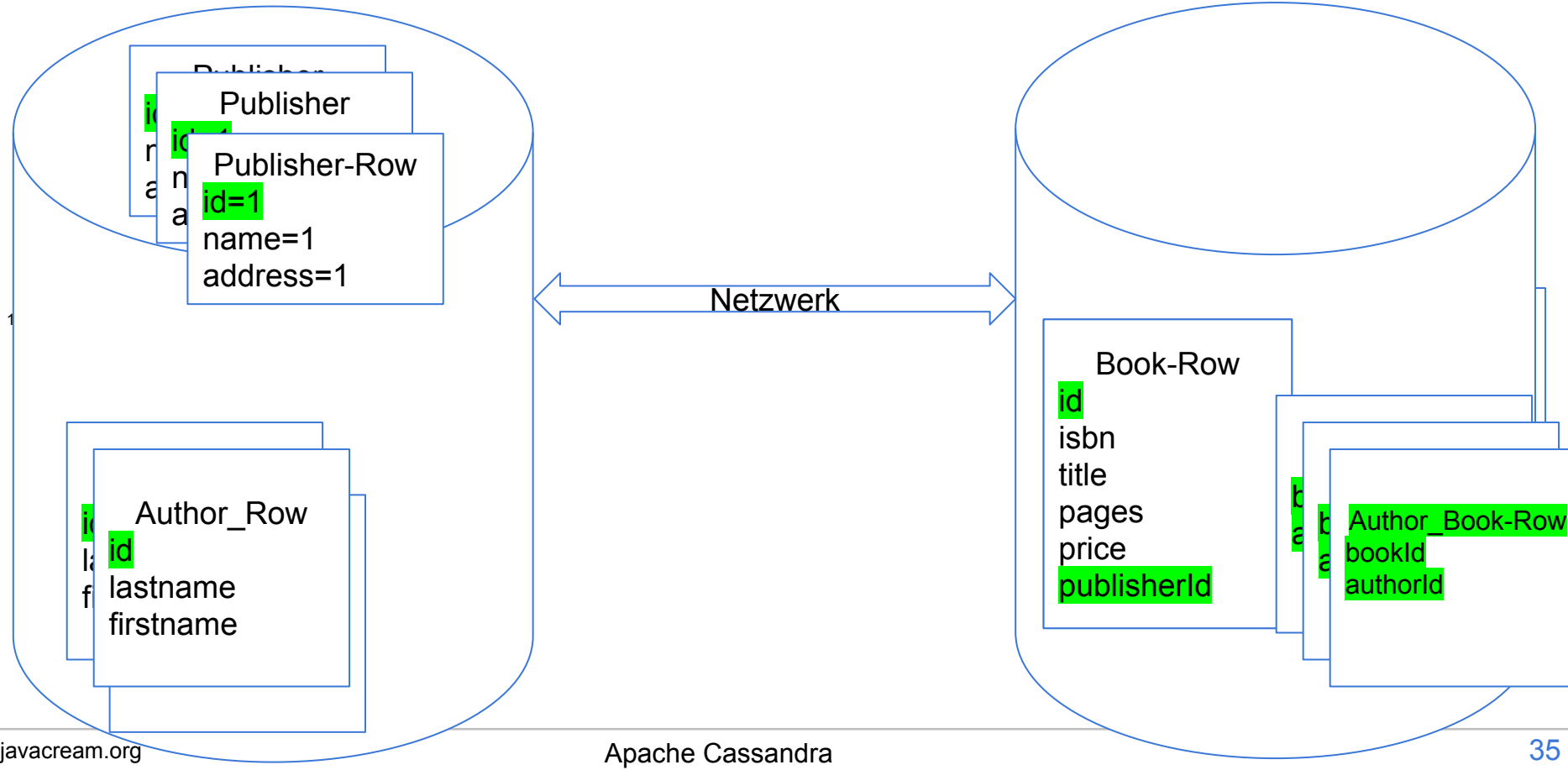




Rebalancing 1: Partitionierung der Daten



Rebalancing 2: Verteilung der Tabellen



Konsequenzen für Join

- Joinen von Datensätzen ist deutlichst langsamer als vorher
  - Verteilung der Tabellen “geht vielleicht gerade so”
  - Blinde Partitionierung “mission impossible”

Konsequenzen aus den Konsequenzen des Join?

- Partitionierung an Hand eines fachlich bestimmbaren “Partition Keys”
  - Rebalancing 1 an Hand eines zusätzlichen Merkmals, die zusammengehörende Datensätze auf einem Knoten beläßt
  - Falls so etwas möglich ist, spricht man von einem so genannten “Sharding”, “Sharding Key”
- Rebalancing 2 führt zu einer verteilten Datenbank
  - Verzicht auf Datenbank-Joins
  - Isolierte Datenbank-Tabellen
- Ein relationales Modell wird allgemein nur in einem vertikal skalierenden System betrieben werden können

Moderne Server-Landschaften wollen horizontal skalierende Systeme

- siehe Seite 10
- Daraus folgt, dass eine Umsetzung eines Entity-Modells in ein technisches Modell nicht notwendigerweise relational sein wird

Einführung NoSql

- Was bedeutet “NoSql”?
  - Prägnanter Begriff, der sich gut aussprechen läßt
  - Sql ist eine standardisierte Abfragesprache für Daten
- Gemeint ist statt NoSql -> NoRelational
  - Relational ist eine technische Modellierung zur Umsetzung eines Entity-Modells
- No ist nicht eine strikte Ablehnung, sondern “Not only”
- Damit bedeutet NoSql eigentlich, dass ein Modell nicht notwendigerweise immer relational umgesetzt werden soll

Kategorien von NoSql-Datenbanken

- Ursprünglich gab es 5 klar getrennte Kategorien
  - Datenbank-Systeme sind heute kaum noch eindeutig exakt einer Kategorie zuzuordnen
- Liste der ursprünglichen Kategorien
  - Key-Value-Store
  - Graphen-orientierte Datenbanken
  - Dokumenten-orientierte Datenbanken
  - Spalten-orientierte Datenbanken
  - Relationale Datenbanken (!)

WICHTIG

- NoSql trifft die technische Modellierung!!!

Grobe Zuordnung von Produkten zu den Kategorien

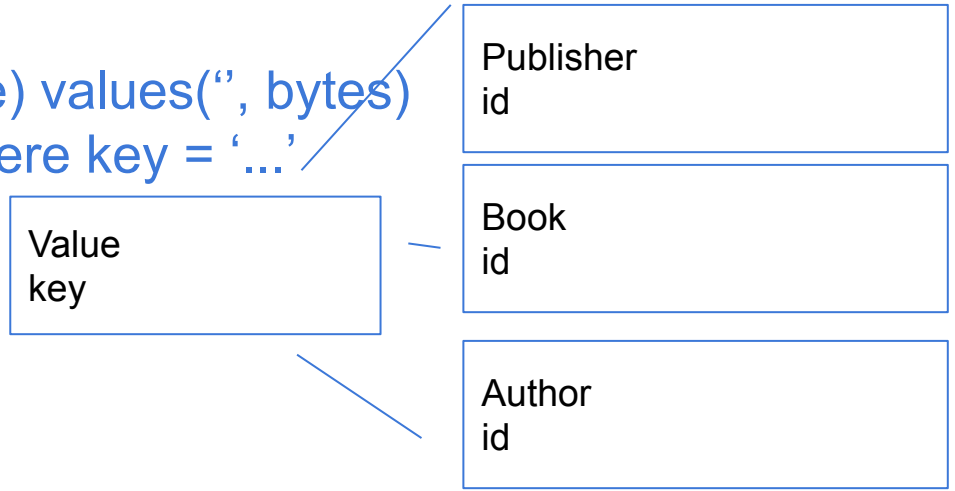
- Key-Value-Store
  - Redis, Memcached, …
- Graphen-orientierte Datenbanken
  - Neo4J
- Dokumenten-orientierte Datenbanken
  - MongoDB, Couchbase
- Spalten-orientierte Datenbanken
  - Apache HBase, Amazon Dynamo, Google Big Table
- Relationale Datenbanken (!)
  - Oracle, MySql, MsSql, …

Was ist mit der Apache Cassandra?

- Eine Mischung, genannt eine “Wide Column”-Datenbank
  - Einstiegspunkt ist Key-Value-Store
  - ab dann: Spalten-orientiert

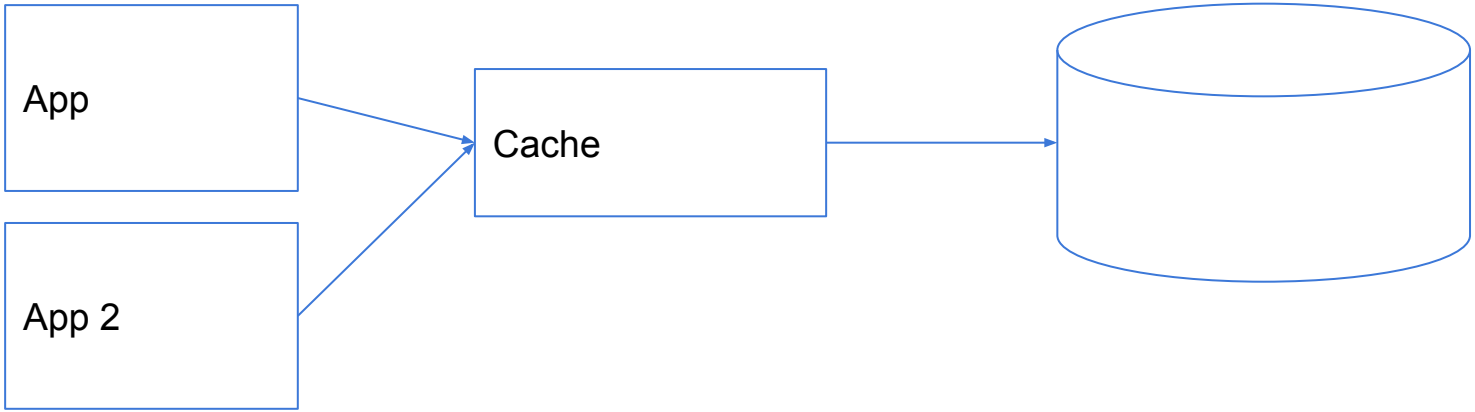
Key-Value-Store

- create store Publisher
- insert into Publisher (key, value) values(‘’, bytes)
- select value from Publisher where key = ‘...’
-

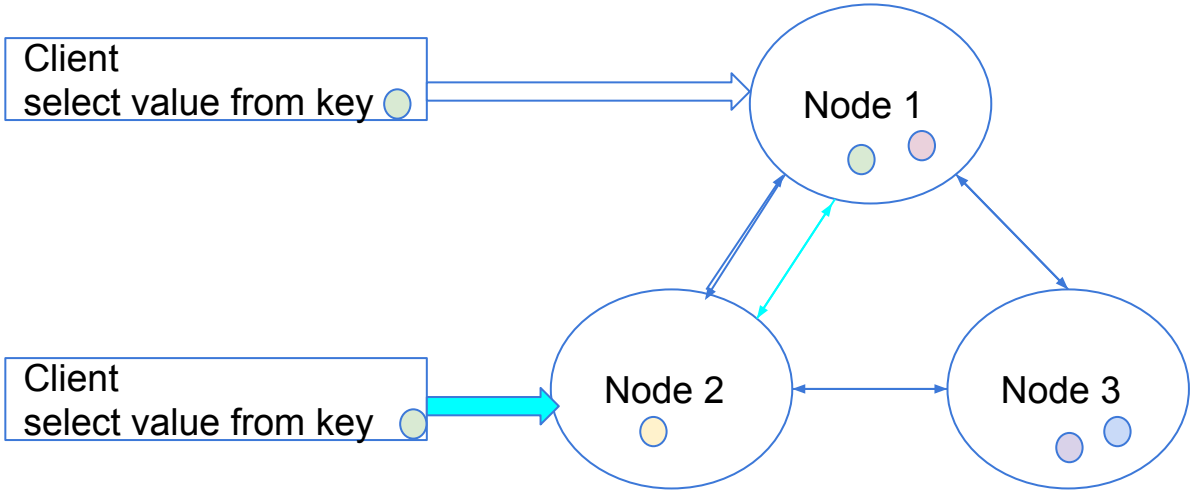


Wertung Key-Value-Store

- Modell ist natürlich extrem eingeschränkt
- Verbreitung ist ebenfalls extrem
  - “Ss gibt kaum System-Architektur, in der kein Key-Value-Store im Einsatz ist”
  - Jeder Cache ist ein Key-Value-Store



Key-Value-Stores im Cluster




```
key1=value1
key2=value2
key3=value3
key4=value4
key5=value5
…
```

Rebalancing für den Key-Value-Store

- Partitionierung erfolgt auf Basis der keys
- Jeder Knoten im Cluster hält einen Bereich von key-Werten
- Rebalancing führt zu einer Neu-Verteilung der keys über den gewachsenen Cluster

Organisation des Clusters

- Verlangt keinen Einsatz eines Primaries!
  - “master-less”
- Orchestrierung zum Starten/Stoppen von Knoten verlangt keine Administration
  - “admin-less”
  - Ein Knoten, der in einem Netzwerk gestartet wird, findet automatisch die bereits laufenden Knoten
- Das ist ein so genannter “Ring Cluster”

Bedarf für einen Load-Balancer

- Eigentlich nicht
  - dadurch, dass sich die Knoten untereinander kennen, kann jeder Knoten jeden Request entgegen nehmen und Antwort liefern
- Die Verfügbarkeit des Clusters ist sehr gut, ein Client bekommt immer Antwort, sobald er mit irgendeinem Knoten kommunizieren kann

Graphenorientierte Datenbanken

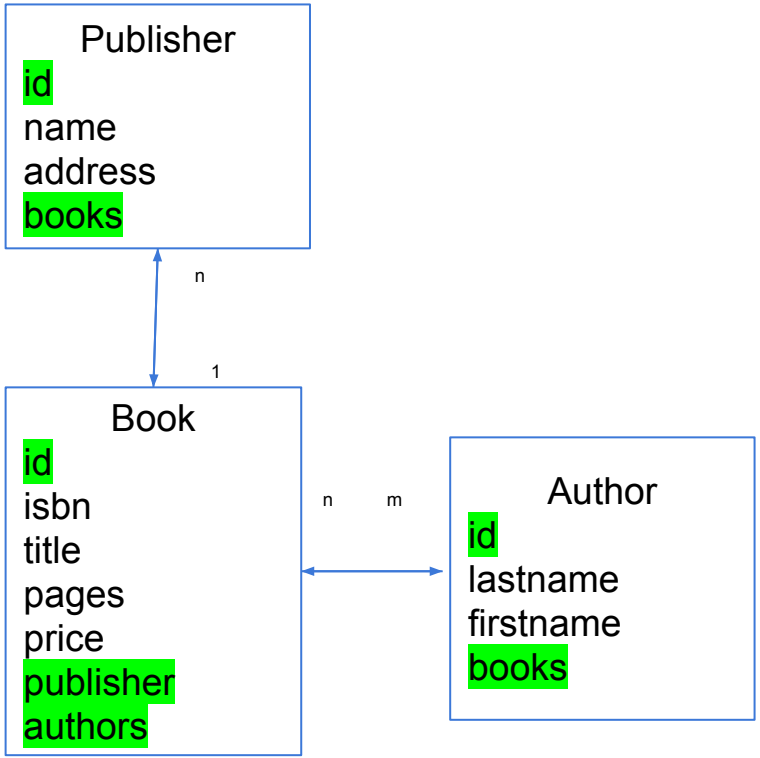
- Modell ist identisch zu den konkreten Daten
  - “whiteboard friendly”
- Einsatzbereich fokussiert auf Daten, die sehr dynamisch in Relation werden sollen
  - Relationen werden hier zu vollwertigen Daten aufgewertet
  - Social Media, Fahrpläne, …

Dokumenten-orientierte Datenbanken

- Das Modell einer Dokumenten-orientierten Datenbank ist fast identisch zum relationalen Modell
  - Beziehungen werden über Verlinkungen realisiert
- Horizontaler Cluster kann sehr einfach aufgebaut werden
  - Durch die Verlinkung ist es Aufgabe des Clients, sich notwendige Daten nachzuladen
  - Partitionierung ist über die Dokumenten-ID ebenfalls einfach möglich

Technisches Modell: Technologie-abhängig

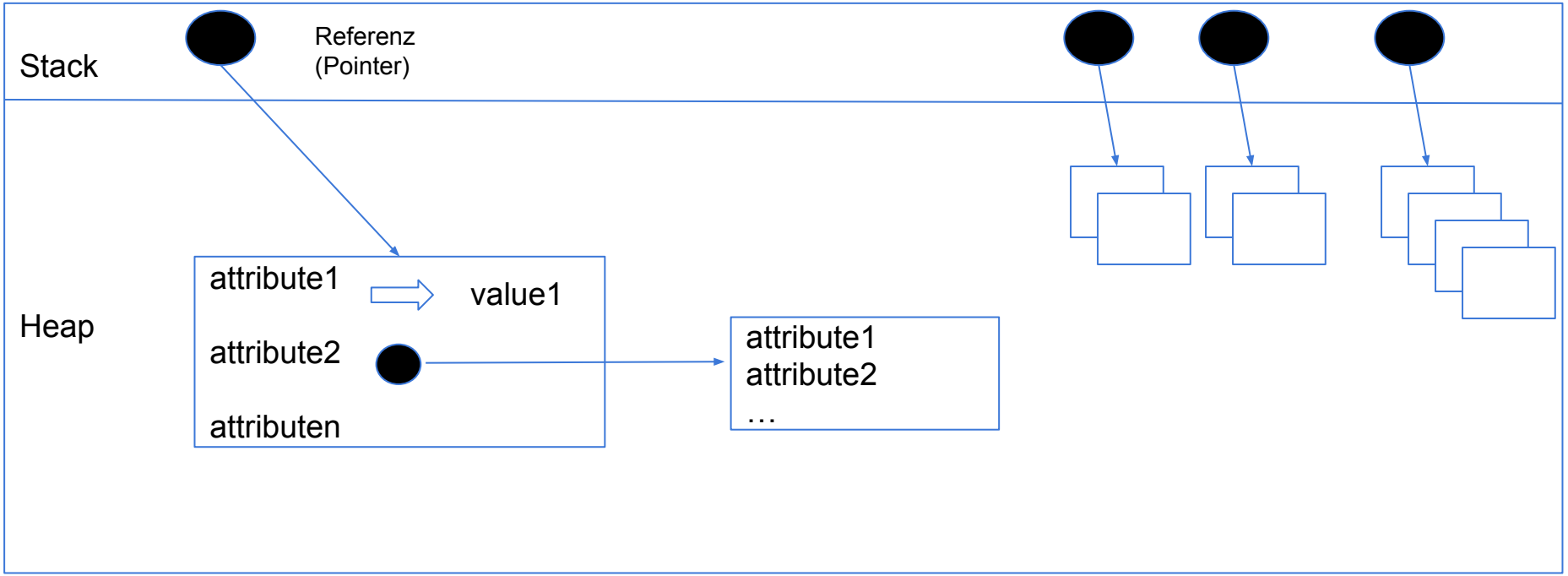
Umsetzung mit einer Dokumenten-orientiert



Column-orientierte Datenbanken

- Die Daten-Organisation einer solchen Datenbank unterscheidet sich fulminant von einem relationalen Modell
- Analogie zur Objekt-orientierten Programmierung ist vorhanden
  - Die Prinzipien der OOP sind relativ “menschlich” nachvollziehbar

OOP-Speichermodell



Begriffe der Spalten-Orientierung

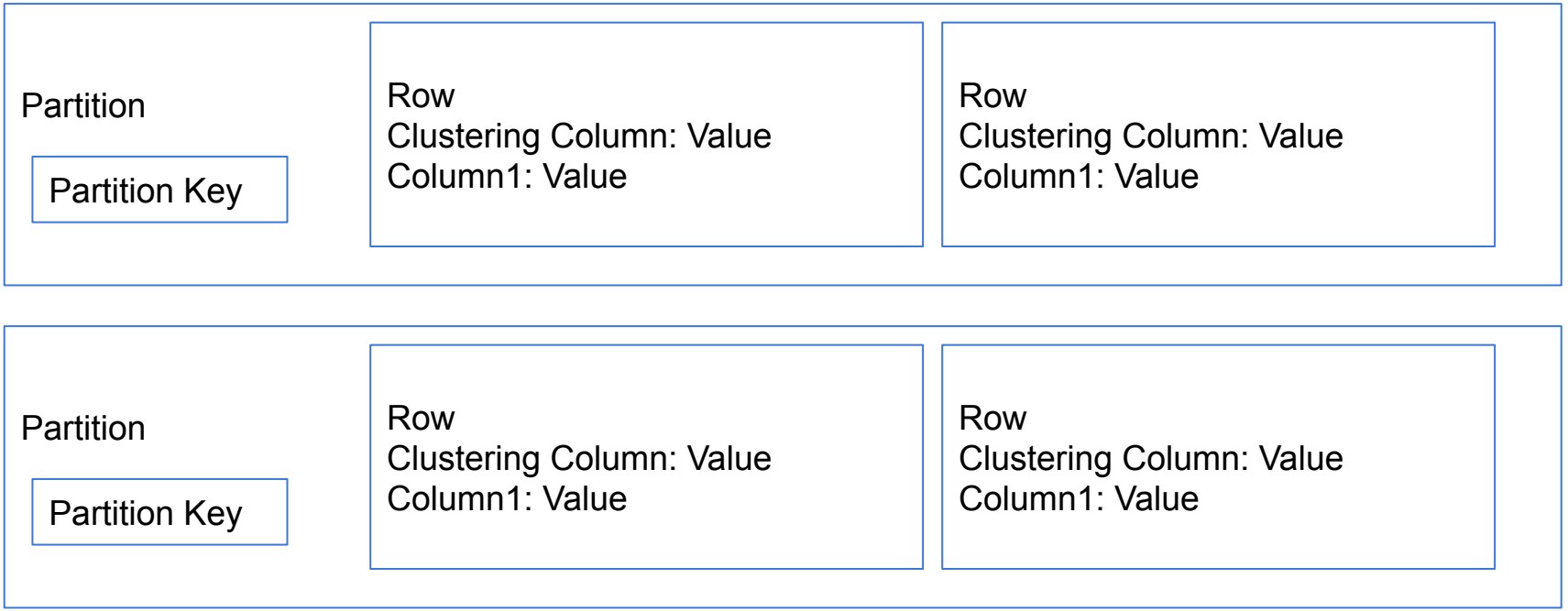
- Referenz
  - Row Key
- Attribute des referenzierten Objekts
  - Column Families (Google Big Data)
  - “Map”, “Dictionary”
    - Eine beliebig große Menge von Key-Value-Elementen

Konsequenzen

- Jede zusammengehörige Datenmenge wird über den gemeinsamen Row-Key identifiziert
- Jede Column-Family, jede Map ist für sich selber ein eigener Daten-Container
  - Daten-Abfragen können innerhalb einer Column-Family separat definiert und ausgeführt werden
  - Eine Verteilung einer Row auf verschiedene (!) Knoten ist jederzeit möglich

Apache Cassandra

Organisation der Daten: Level 1



Die Bedeutung des Partition Key

- Der Partition Key identifiziert wie der Key eines Key-Value-Stores den Knoten, auf dem die Daten abgelegt sind
- Eine Abfrage, die diesen Knoten nicht eindeutig identifizierbar macht ist
  - technisch möglich
  - allerdings für den Cluster sehr aufwändig
    - alle Knoten müssen abgefragt werden
    - Naive Abfrage resultiert in einer Fehlermeldung
- Eine gute Cassandra-Abfrage selektiert als erstes den Partition Key
  - ab da ist der Knoten identifiziert
  - und damit sind weitere Abfragen/Selektionen effizient möglich

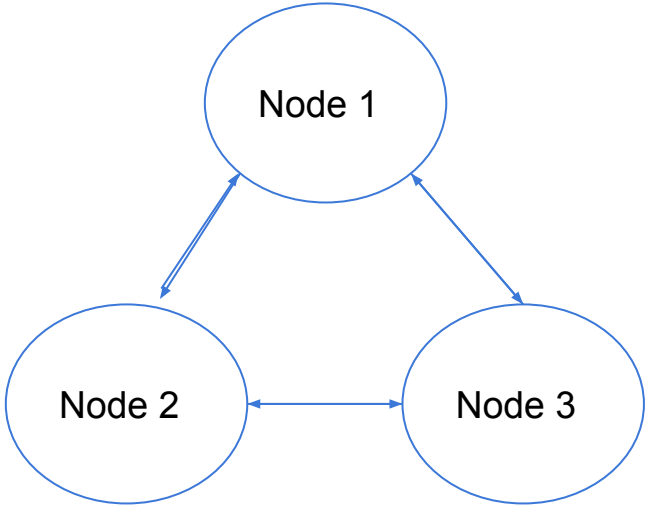
Bedeutung des Primary Keys

- Primary Key (lastname, firstname)
- Das ist **nicht** ein zusammengesetzter Schlüssel (!)
- Der erste (und eventuell einzige Teil) ist der Partition Key
- Der (optionale) zweite Teil definiert eine Cluster Column

Ring Cluster, Tabellen, Partitionen, Primärschlüssel

- create keyspace training_sawitzki with replication {...}
  - use keyspace training_sawitzki;
- create table user (lastname text, firstname text, description text, primary key (lastname, firstname));
- insert into user (lastname, firstname, description) values (‘Sawitzki’, ‘Rainer’, ‘a trainer’);
- insert into user (lastname, firstname, description) values (‘Sawitzki’, ‘Klaus’, ‘a brother’);
- insert into user (lastname, firstname, description) values (‘Mustermann’, ‘Hanna’, ‘a participant’);
- insert into user (lastname, firstname, description) values (‘Metzger’, Georg’, ‘another participant’);

Ausgangssituation



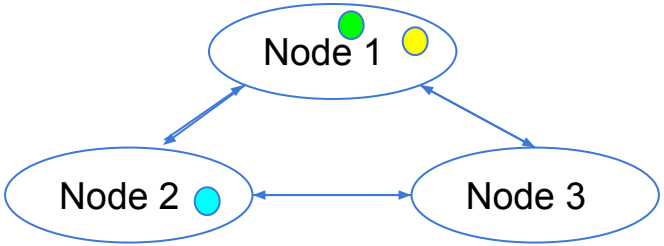
Partition

Partition Key

Row
Clustering Column: Value
Column1: Value

Row
Clustering Column: Value
Column1: Value

Lösung



Hinweis:
Die Verteilung der Partitionen auf die einzelnen Knoten erfolgt intern mit einem Verfahren, das eine gleichmäßige Auslastung garantiert

Partition
Sawitzki
Row
firstname: Rainer <pk>
description: a trainer
Row
firstname: Klaus <pk>
description: a brother

Partition
Mustermann
Row
firstname: Hanna <pk>
description: a participant

Partition
Metzger
Row
firstname: Georg <pk>
description: another participant

Ein paar Select-Ideen

- select * from user;
- select description from user;
- select * from user where lastname=’Sawitzki’;
- select lastname from user where description=’a trainer’;
- select * from user where firstname=’Rainer’
- …
- ToDo:
  - Machen Sie sich klar, welche selects sind sinnvoll / technisch möglich?
  - Wie viele Knoten müssen bei diesen selects jeweils angesprochen werden?
  - was steuert dieses ALLOW FILTERING
    - Lösung: “Ich übernehme die Verantwortung dafür, dass die Abfrage über alle Knoten des Ring-Clusters hinweg ausgeführt werden muss”

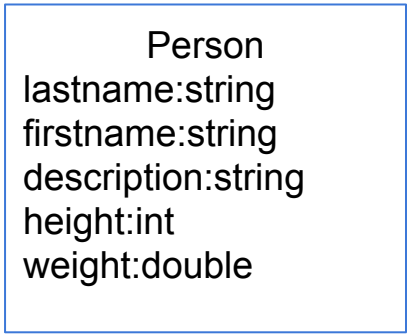
Ein paar Details

- Tabellen
  - drop table user;
  - describe table user;
  - alter table user add | drop column-definition | column-name
- Columns
  - Nicht-Primärschlüssel Spalten sind “nullable”
    - Nicht gesetzte Spalten sind in der Datenstruktur einfach nicht vorhanden
  - INSERT INTO ist bei bereits vorhandenem Primärschlüssel ein Update (!), ein ‘upsert’

Ein paar weitere Details

- Implizit ist wird für jede Spalte ein Zeitstempel geschrieben
  - Auslesen mit der Funktion writetime(column)
  - Beim Aktualisieren einer Spalte kann stets auch USING TIMESTAMP angegeben werden
- Time To Live (TTL)
  - ttl(column)
  - USING TTL
    - Aktualisierung einer Spalte
    - INSERT INTO

Ein Entity-Modell



Technisches Modell für Cassandra

<<table>>
Person
lastname:text
firstname:text
description:text
height:smallint
weight:float

Offener Punkt: Primärschlüssel?

Cassandra Datentypen

text (Zeichenketten)
    varchar, ascii
int (Ganzzahlen)
    tinyint (8bit), smallint (16), int (32), bigint (64)
double (Komma-Zahlen, 64bit)
    float (32 bit)
boolean
timestamp, date, time
blob
uuid
    erzeugt über die uuid()-Funktion

Zur Bestimmung des Primärschlüssels

- Einführung eines technischen Schlüssels
  - Einführen einer Spalte id: uuid, primary key(id)
    - Jede Person bekommt damit eine eigene Partition
    - Suche nach Personen sind ohne ALLOW FILTERING praktisch nicht möglich
- Der notwendige Partition Key ist immer Bestandteil des Entity-Modells
  - Ein fachlich sinnvoller Einstieg in jegliche weitere Suche
  - Partition Keys müssen eine genügende Variabilität aufweisen
    - NORD, SÜD, WEST, OST
      - Ein Ring-Cluster kann damit sinnvoll maximal 4 Knoten haben (!)

Technisches Modell für Cassandra

<<table>>
Person
lastname:text <<partition>>
id:uuid <<primary key>>
firstname:text
description:text
height:smallint
weight:float

Personen-Identität ist aus den Feldern der Entität so nicht eindeutig bestimmbar

Damit führen wir eine weitere Spalte als uuid ein

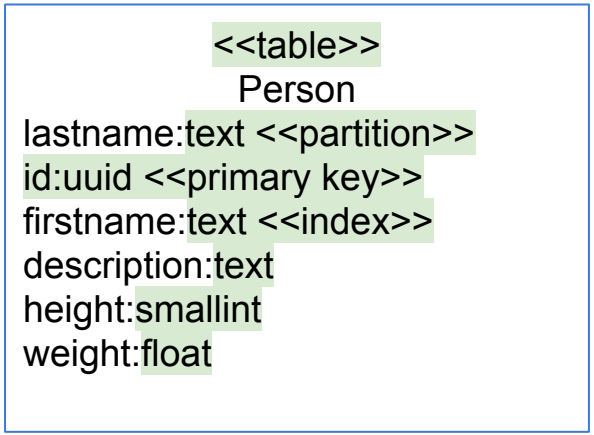
ToDo

- Einführung einer Person-Tabelle in der angegebenen Struktur
- Fügen Sie ein paar Test-Daten ein!
- Versuchen Sie, repräsentative Abfrage zu formulieren
  - Lösung: Selbst einfache Suchen wie “nach id” = fachlicher Primärschlüssel oder nach nachname und vorname funktionieren nur mit ALLOW FILTERING

Potenzielle Lösung

- “Ich führe noch weitere Hilfs-Tabellen ein, z.B. ein Mapping Vorname -> id
  - create table firstname_to_if (firstname text, id uuid)
    - Jeder insert einer Person muss begleitet werden von einem Insert in die firstname_to_id-Tabelle
  - Suche nach lastname und firstname ist ebenfalls zweistufig: Über die Vornamen die ids und anschließend über die lastname + id-Liste
- Jetzt: Einführung eines Indexes
  - Diese Hilfstabelle entspricht quasi diesem Index
  - Aktualisierung des Index erfolgt automatisch
  - create index firstname on person (firstname);
    - drop index firstname;

Technisches Modell für Cassandra



Exkurs zu Indizes in Cassandra

- Indizes wurden erst relativ spät eingeführt
  - Verwaltung der Indizes ist durchaus aufwändig
- Heute sind Indizes in Cassandra durchaus gebräuchlich, werden allerdings immer noch kritisch diskutiert
  - https://docs.datastax.com/en/cql-oss/3.3/cql/cql_using/useWhenIndex.html#useWhenIndex__when-no-index
  -

Technisches Modell für Cassandra: counter

Datentyp ‘counter’

Eine Spalte vom Typ counter kann nicht direkt gesetzt werden, sondern nur im Rahmen einer Aktualisierung hochgesetzt werden

Einsatzbereich ist typischerweise eine fortlaufende Versionssnummer

ein update sollte in seiner where-Bedingung immer den aktuell bekannten counter berücksichtigen

VORSICHT: Eine counter-Spalte muss die einzige nicht-Primary-Key-Spalte sein!

Collection-Typen

- 3 Kategorien
  - set
    - Menge von Elementen, die keine Duplikate enthalten kann
    - set hat keine innere Ordnung, “gib mir Element 3” geht nicht
  - list
    - Menge von geordneten Elementen, Zugriff ist möglich über einen Index-Wert
  - map
    - Menge von Elementen, die über einen key-Wert identifiziert werden

Technisches Modell für Cassandra

<<table>>
Person
lastname:text <<partition>>
id:uuid <<primary key>>
firstname: text <<index>>
firstnames: set<text>
description:text
height:smallint
weight:float

set<text>, gesprochen “set of text” bedeutet hier, dass die weiteren Vornamen in einem Set gespeichert werden

Setzen eines Set:

{‘Rainer’, Ulrich}

Zugriff auf einzelne Set-Elemente ist nicht möglich

SET firstnames = firstnames + ‘Eduardo’
SET firstnames = firstnames - ‘Eduardo’
SET firstnames = {}
drop firstnames …

Selektion auf set-Elemente ist möglich, wenn ein Index verwendet

ToDo

- Erweitern Sie die Personen-Tabelle um das set der firstnames!
- emails als eine Liste von Zeichenketten
- Für beide Erweiterungen ein paar Test-Daten anlegen und selektieren, Elemente der Liste/dem Set hinzufügen, wegnehmen, …
- Hinweis:
  - In einem relationalen Modell würden solche Felder Relationen zu anderen Tabellen bedeuten

Technisches Modell für Cassandra

<<table>>
Person
lastname:text <<partition>>
id:uuid <<primary key>>
firstname: text <<index>>
firstnames: set<text>
description:text
height:smallint
weight:float
phone_numbers: map<text, text>

{'home': '12345678', 'work': '8765432'}

ToDo

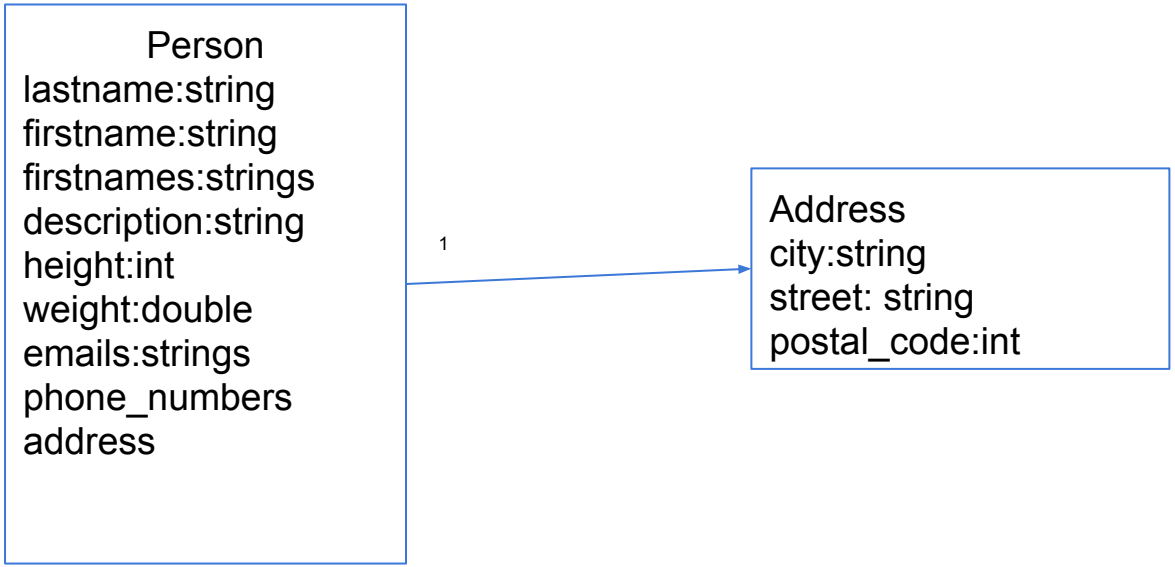
Cassandra Tuples

- alter table add address tuple<text, text, smallint>
- … set address = (‘Marienplatz’, ‘München’, 81371) …

User defined Types

- Erzeugung
  - create type address (street text, city text, postal_code smallint);
- Zuweisung
  - {street: ‘Marienplatz’, city: ‘München’, postal_code: 81373}

Das Entity-Modell



Technisches Modell für Cassandra

<<table>>
Person
lastname:text <<partition>>
id:uuid <<primary key>>
firstname: text <<index>>
firstnames: set<text>
description:text
height:smallint
weight:float
email
phone_numbers: map<text, text>
address: address

UDT address

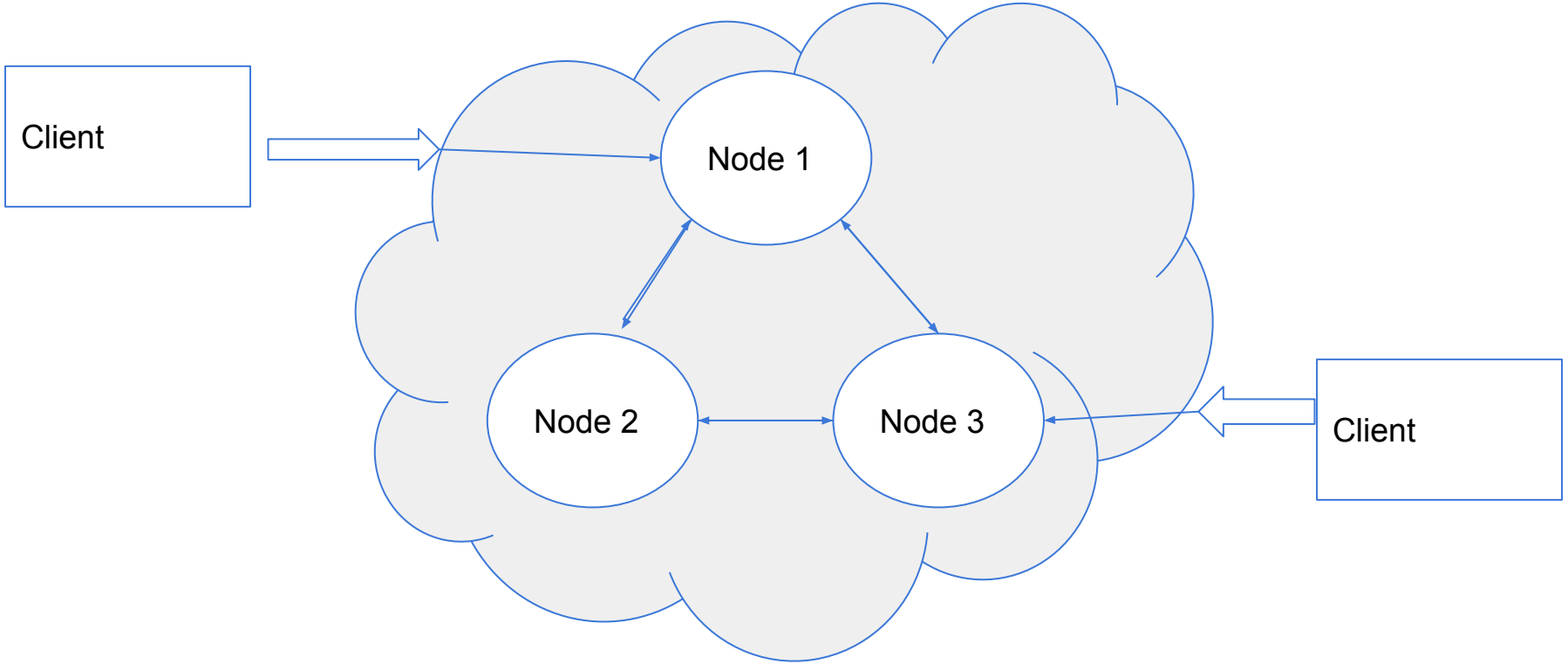
ToDo

Exkurs:
Collections in Collections geht ohne Vorbereitung nicht
Fehler: “... is not frozen…”

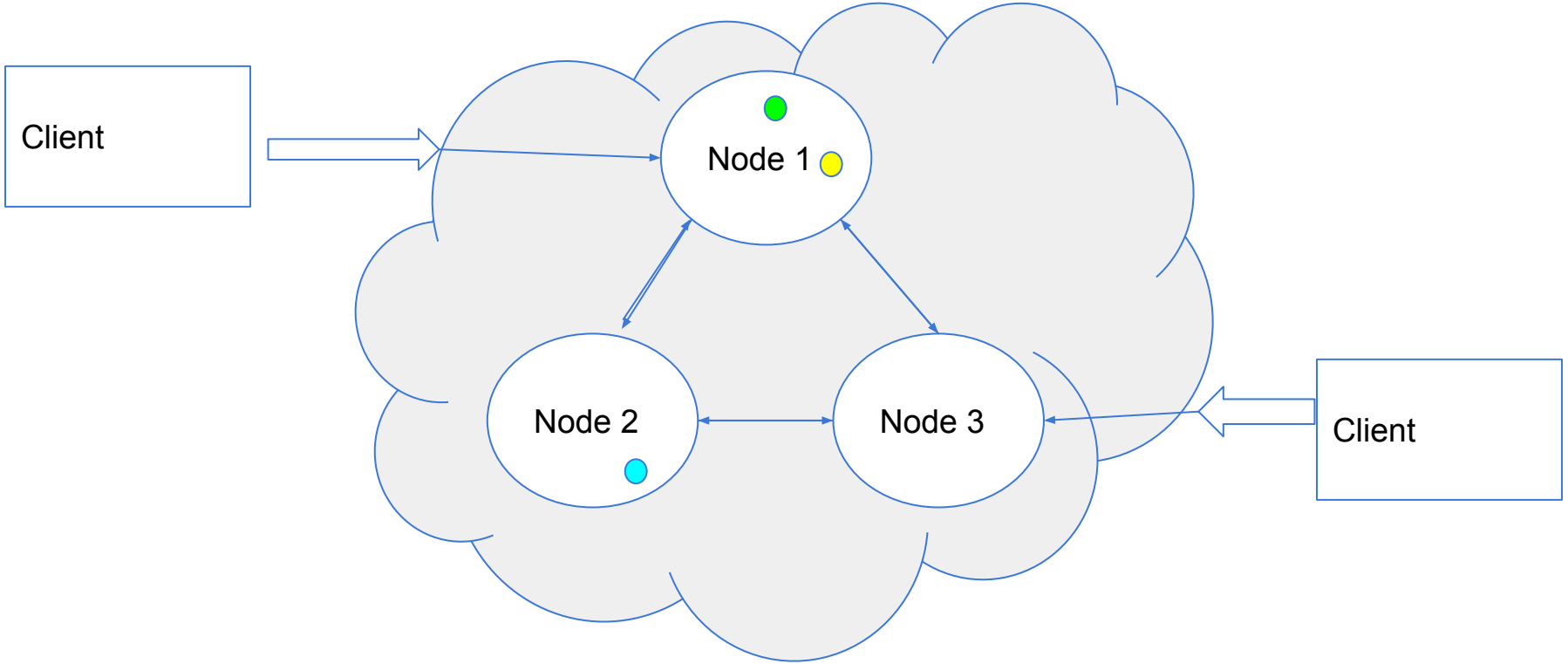
frozen(collection)

Replikation

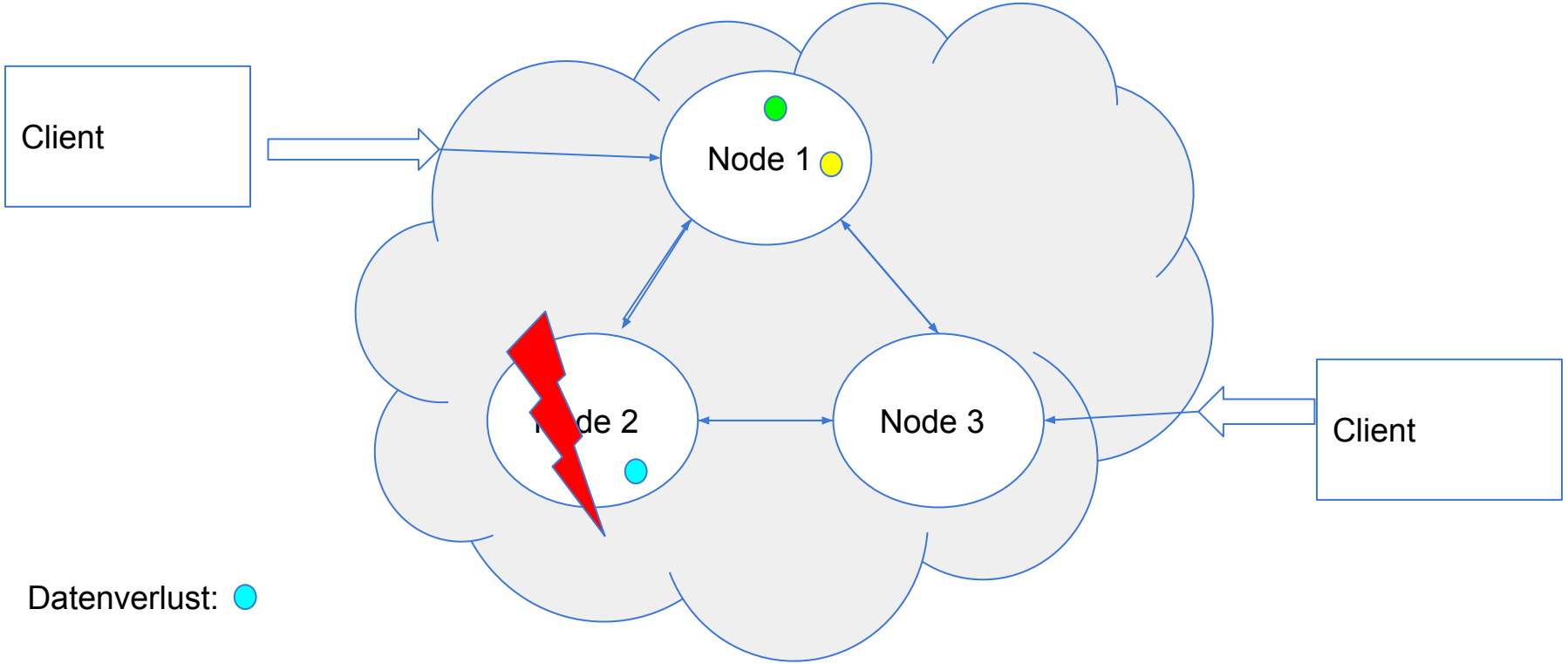
Cassandra Ring Cluster



Cassandra Ring Cluster und Daten

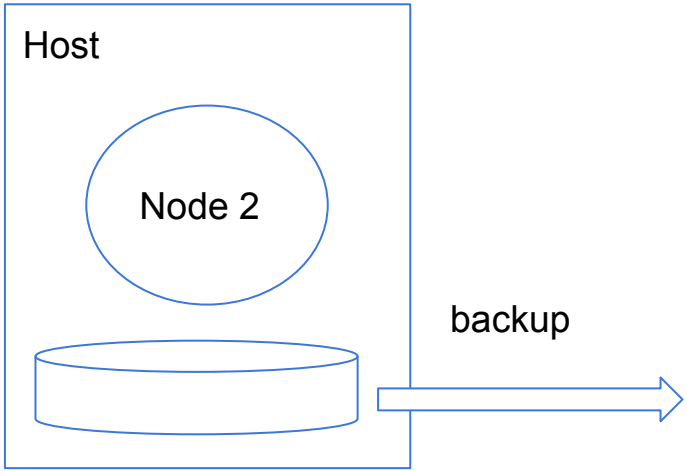


Cassandra Ring Cluster, Daten und Ausfall-Situation



Strategien zur Ausfallsicherheit

- Klassisch
  - Gespeicherter = Persistenter Zustand wird über klassische Verfahren gesichert



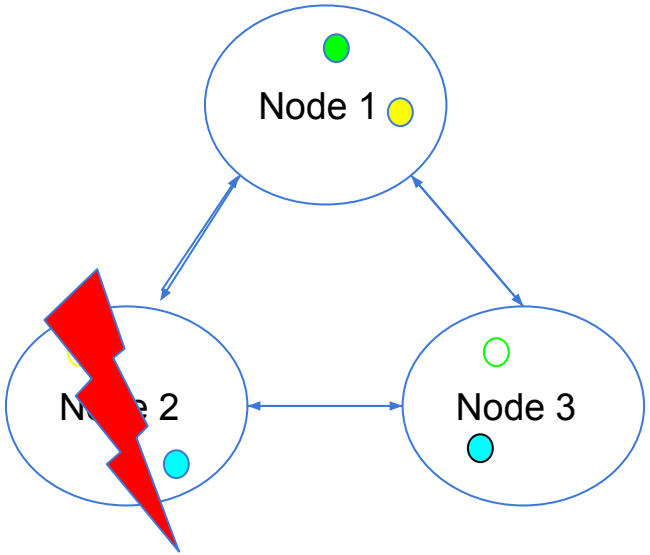
Strategien zur Ausfallsicherheit

- Replikation
  - Partitionen werden auf Backup-Servern im Ring gehalten
    - Jeder Knoten im Ring kann Backups anderer Knoten halten

Strategien zur Ausfallsicherheit

Kein Datenverlust

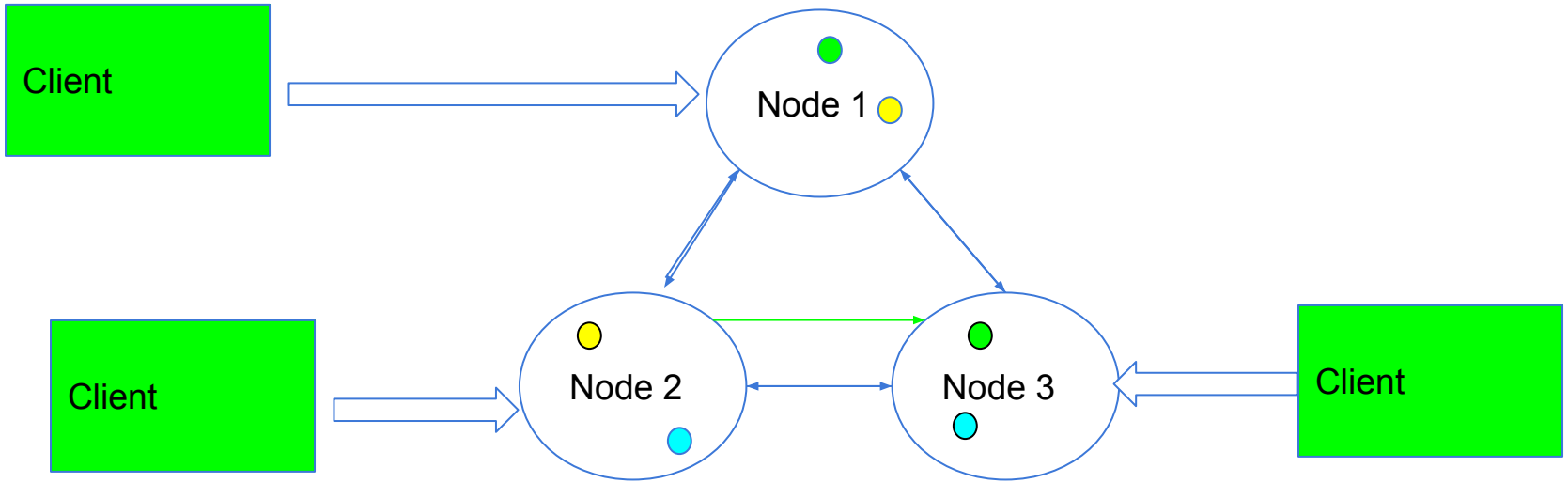
es kann ein weiterer Knoten ausfallen…



Keyspace reviewed

- create keyspace with replication {‘class’: ‘SimpleStrategy’, ‘replication_factor’: 3}

Replikate in Cassandra: reviewed

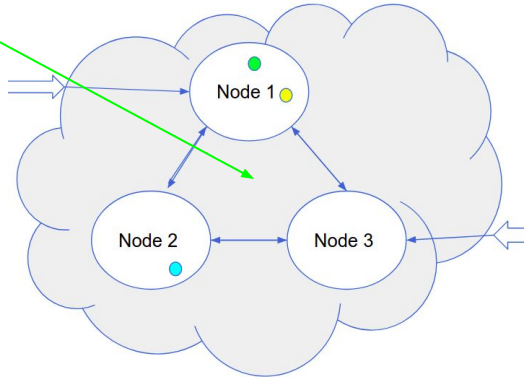


Keyspace reviewed 2

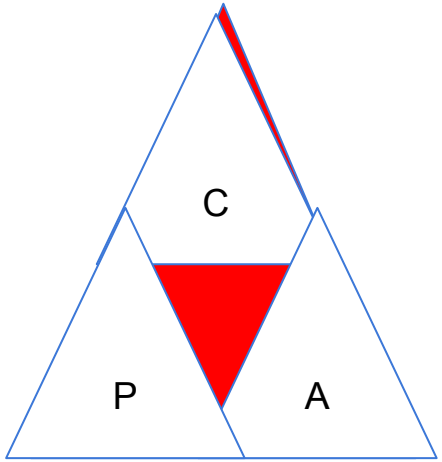
- create keyspace with replication {‘class’: ‘SimpleStrategy’, ‘replication_factor’: 3}
- ‘class’ ist eine Implementierung, die einem Knoten Informationen gibt, wie die Replikation durchgeführt wird
  - SimpleStrategy sucht sich irgendeinen Satz von Knoten
  - NetworkTopologyStrategy berücksichtigt das Netzwerk
    - Server im näheren Netzwerk werden bevorzugt
    - Server im entfernten Netzwerk werden ebenfalls benutzt

Was sind eigentlich die Qualitätsansprüche an den Cluster?

- Fachliche Sicht
  - Daten sind konsistent (Consistency)
  - Daten sind immer verfügbar (Availability)
- Betriebliche Sicht
  - Es ist stets eine genügende Anzahl von Knoten verfügbar
    - Das ist die Orchestrierung des Rings
  - Netzwerk-Ausfälle zwischen den Knoten können kompensiert werden (Partition Tolerance)



CAP-Theorem

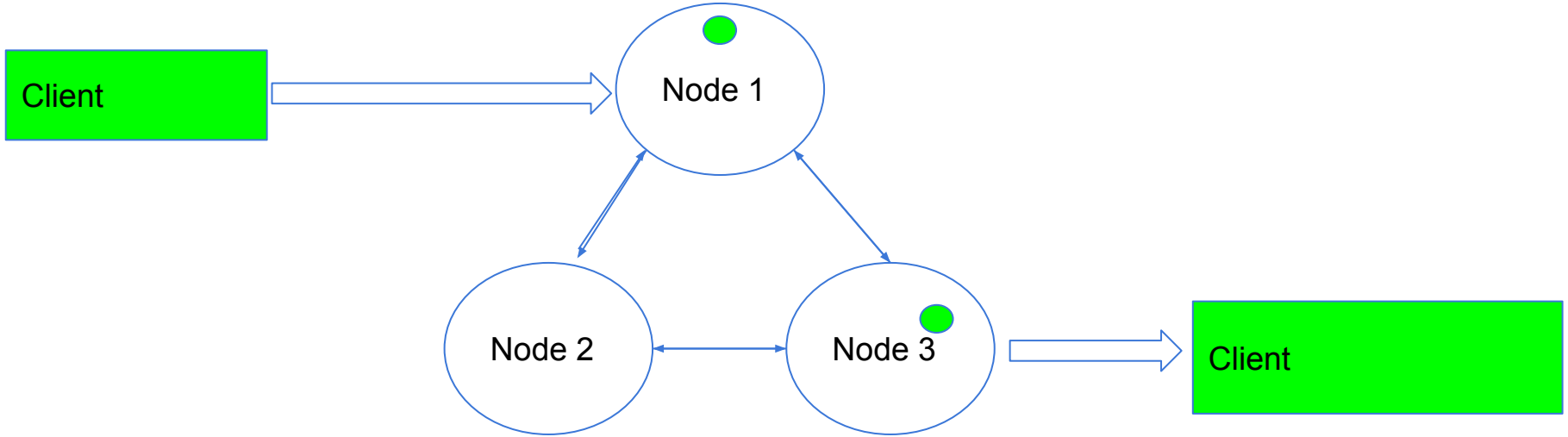


Es gibt keine gemeinsame Schnittmenge

Es geht nur “2 aus 3”

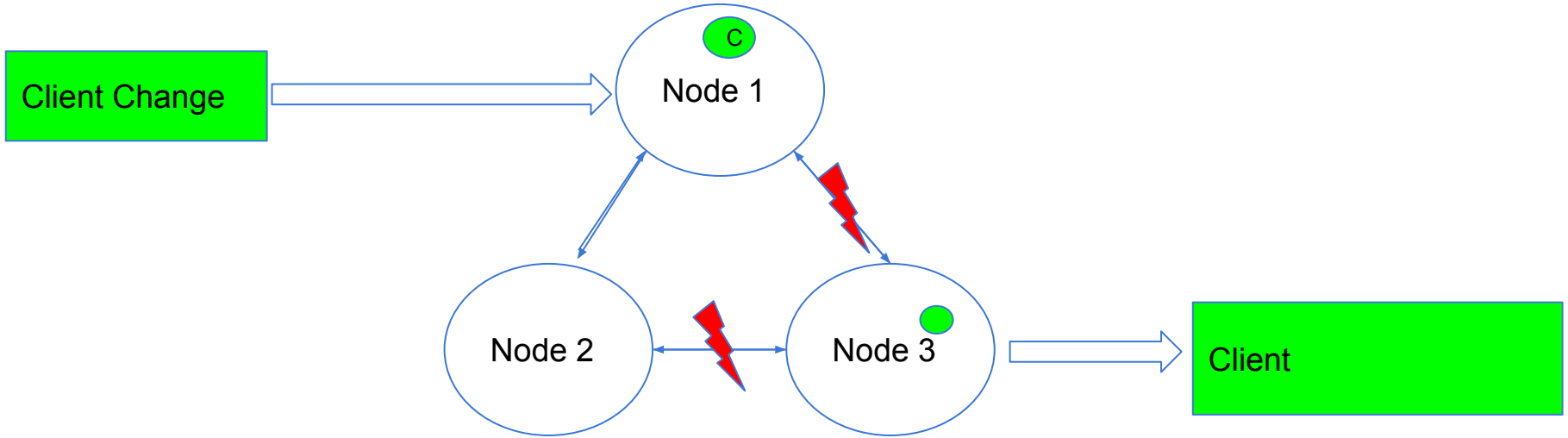
AC
AP
CP

“Split Brain” in einer AP-Architektur



Alles OK, ein Client schreibt Daten, die an anderer konsistent lesen kann

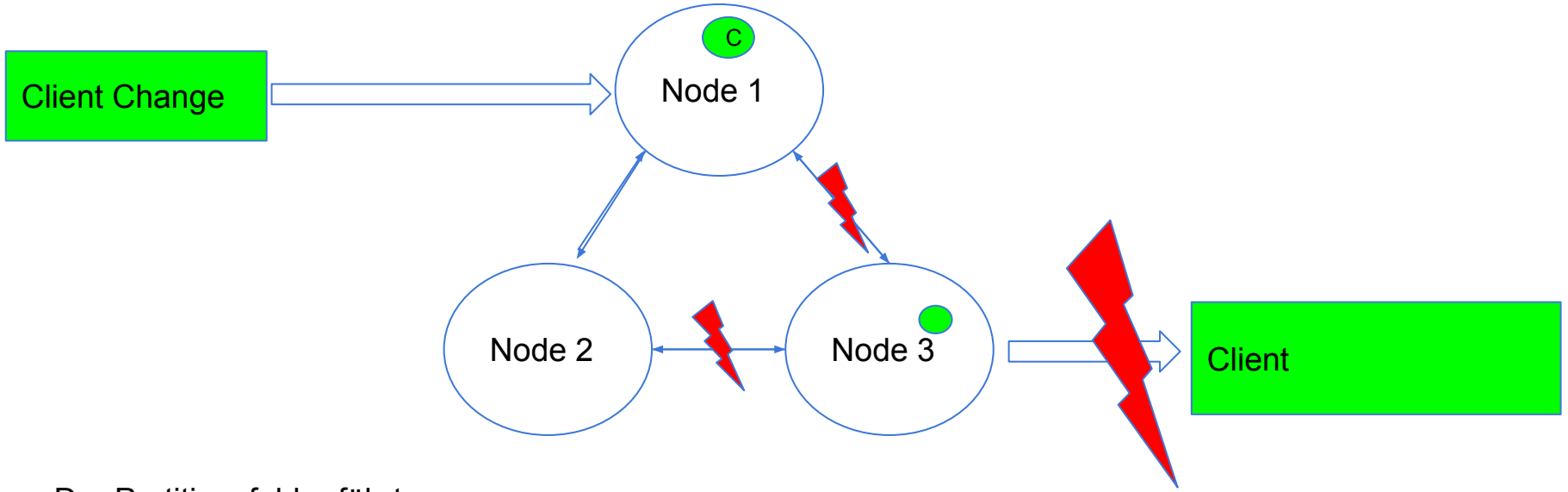
“Split Brain” in einer AP-Architektur



Der Partitionsfehler führt nun dazu, dass der lesende Client veraltete Daten liest

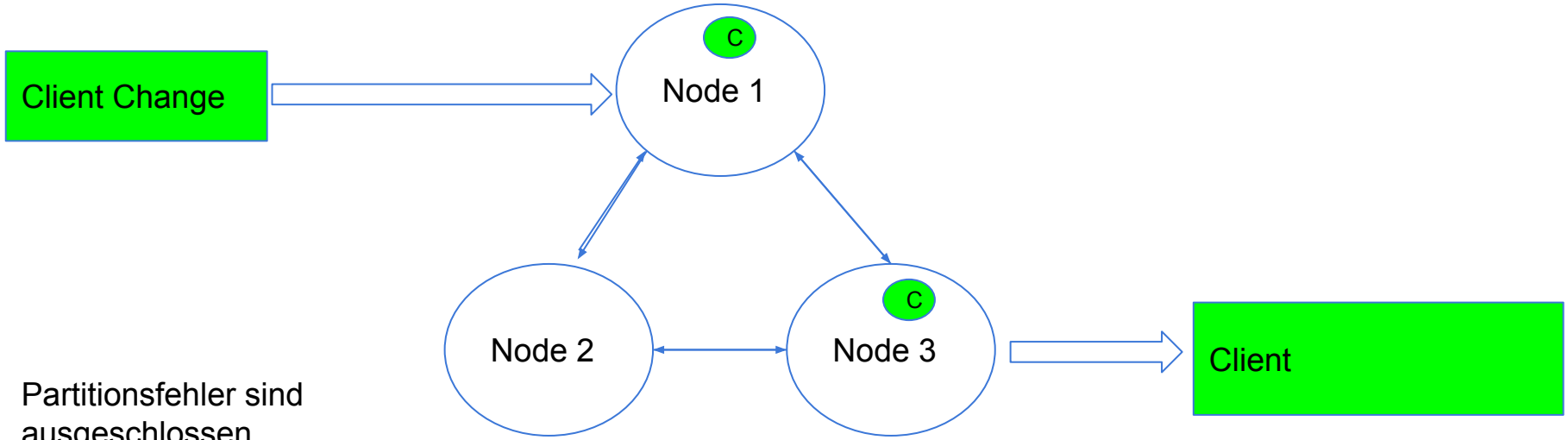
Damit ist keine Konsistenz gewährleistet!

“Split Brain” in einer CP-Architektur



Der Partitionsfehler führt nun dazu, dass der lesende Client eine Fehlermeldung bekommt
Damit ist keine Availibility gewährleistet!

“Split Brain” in einer CA-Architektur

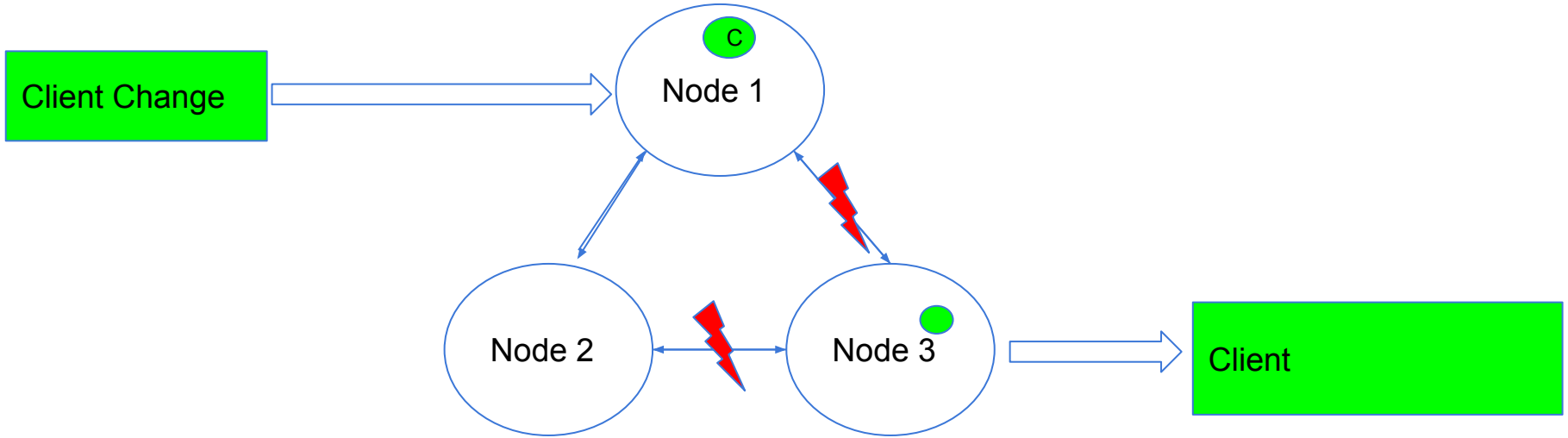


Partitionsfehler sind ausgeschlossen
So etwas geht eigentlich nicht in einem verteilten Cluster, sondern nur innerhalb eines Prozesses.
Bestenfalls vertikale Skalierung möglich

Was ist “die beste” Architektur

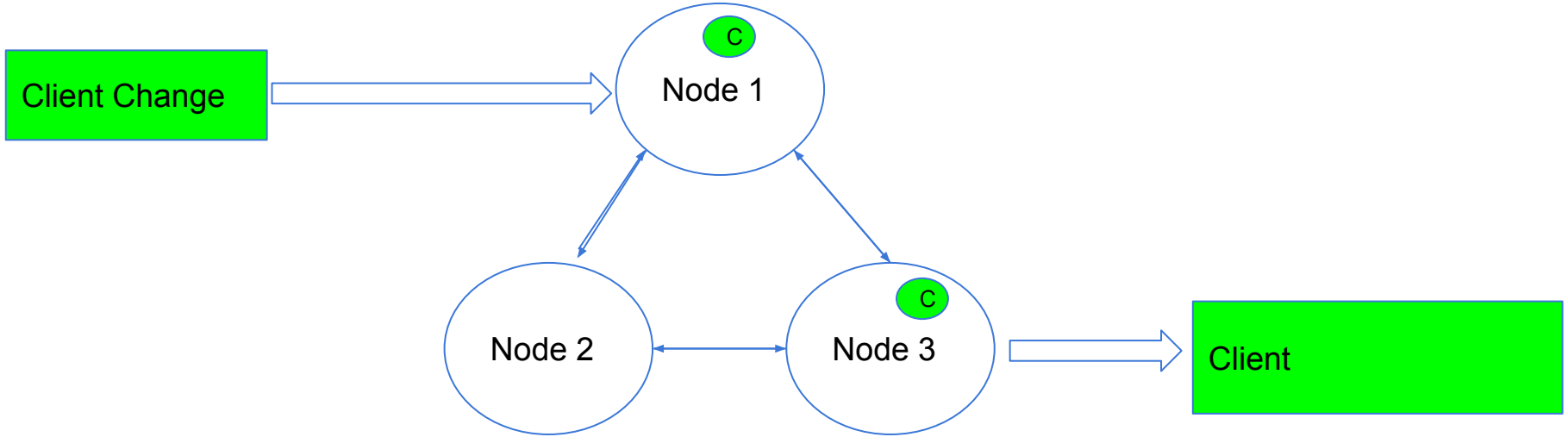
- CA
- CP
- **AP**
  - Die Aussage “keine konsistente Datenhaltung möglich” ist zwar technisch korrekt, aber im Gespräch verheerend
  - BASE-Architektur
    - Basically Availability
    - Soft State (eine Persistierung der Daten ist nicht notwendig)
    - Eventual Consistency
      - “eventual” ist auf deutsch NICHT “eventuell”, sondern “im Endeffekt”
      - Es gibt Zeitfenster für Inkonsistenzen, aber “im Endeffekt” wird der Cluster eine konsistente Datenhaltung garantieren

“Split Brain” mit BASE



Der Partitionsfehler führt nun dazu, dass der lesende Client veraltete Daten liest
Damit ist zu **diesem Zeitpunkt** keine Konsistenz gewährleistet!

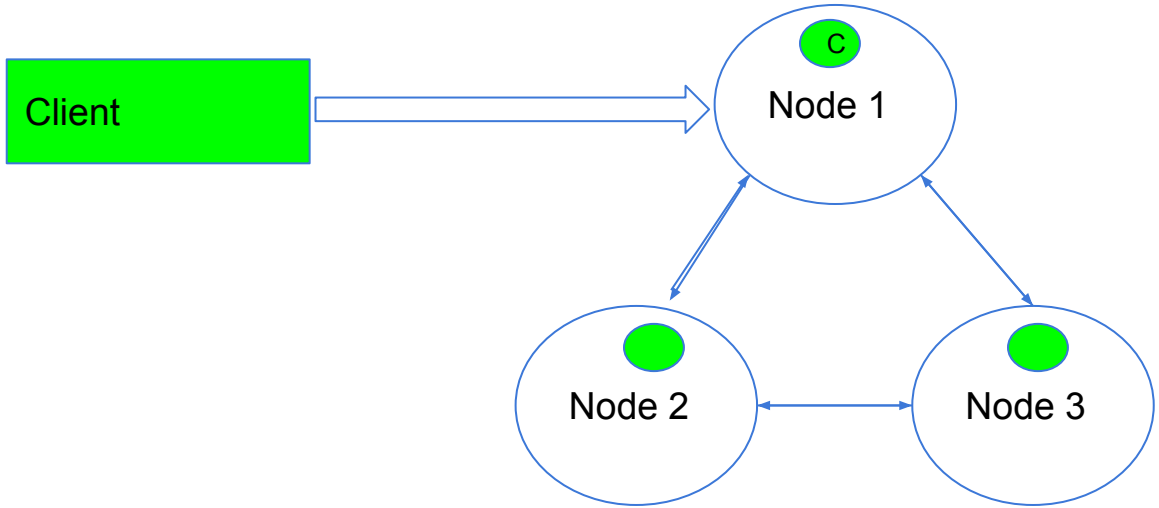
“Split Brain” mit BASE



nach Beheben des Partitionsfehlers werden sich die Knoten synchronisieren
Damit ab **diesem Zeitpunkt** Konsistenz wieder gewährleistet!

- Ring Cluster ist ausgelegt auf eine BASE-Architektur
  - Eventual Consistency ist durch verschiedene Implementierungen / Strategien realisiert
- Zusätzlich: “Tunable Consistency”
  - Utopisches Ziel: “Ich möchte Konsistenz haben und Availability und Partition Tolerance”
  - Mögliches Ziel: “Ich möchte Konsistenz haben und eingeschränkte Availability und eingeschränkte Partition Tolerance”

Tunable Consistency: WRITE

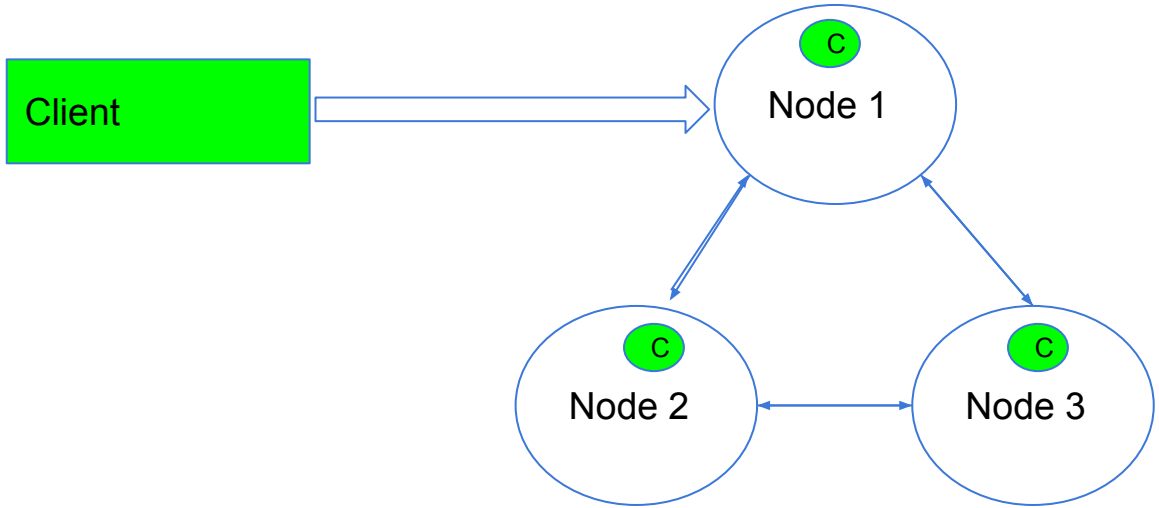


Der Client schreibt den Change in einen Knoten, die Änderung ist noch nicht repliziert

Dies verlangt vom Schreibevorgang einen CONSISTENCY = ONE

Ausgangssituation:
Replikationsfaktor = 3

Tunable Consistency: WRITE

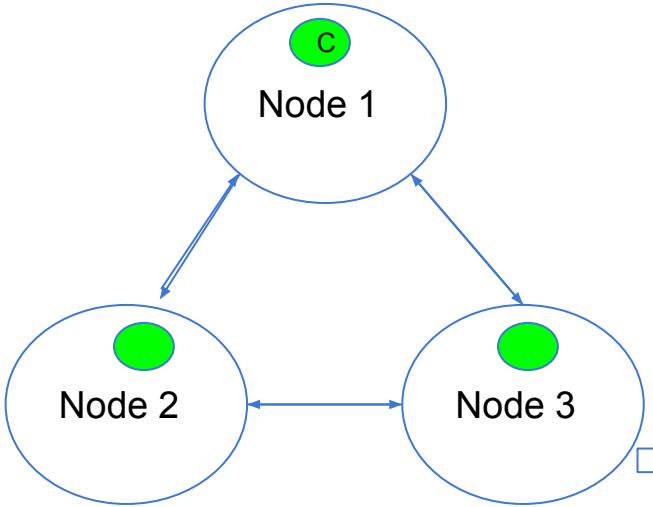


Der Client schreibt den Change in einen Knoten, die Änderung ist auf allen Replikationsservern angekommen

Dies verlangt vom Schreibvorgang einen CONSISTENCY = ALL

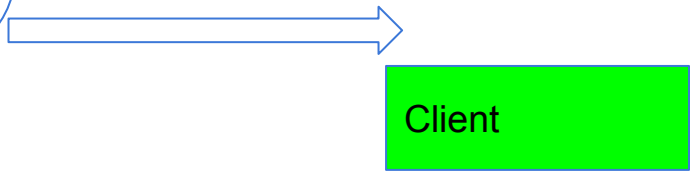
Ausgangssituation:
Replikationsfaktor = 3

Tunable Consistency: READ



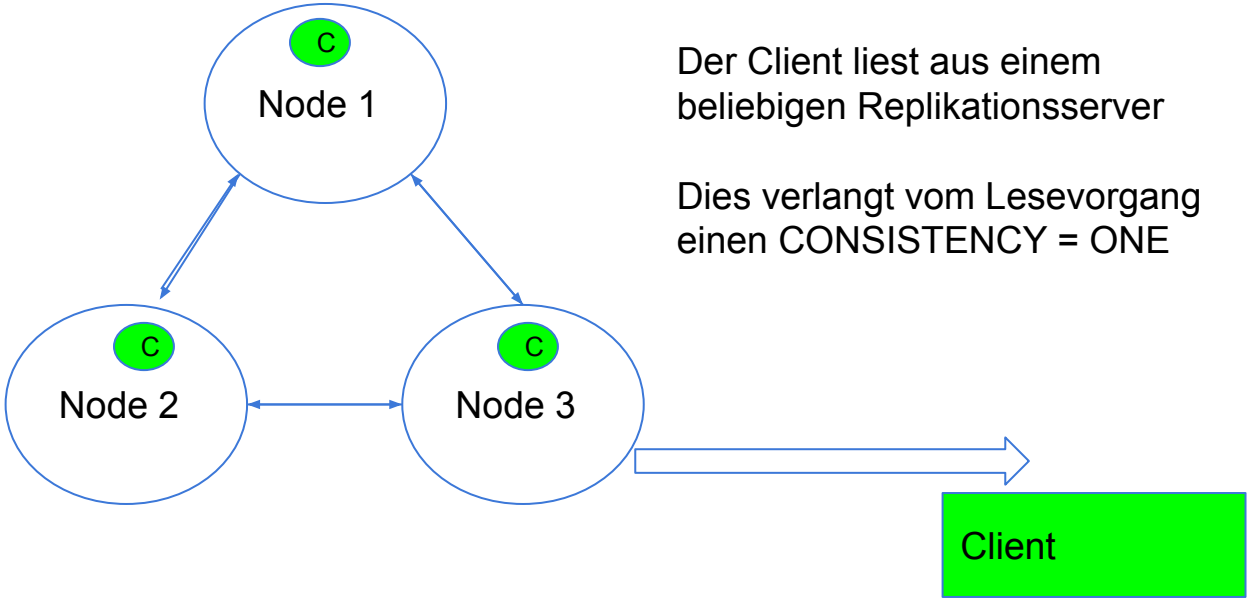
Damit der Client den konsistenten Zustand lesen kann, muss der Lesevorgang über alle Replikationsserver hinweg durchgeführt werden

Dies verlangt vom Lesevorgang einen CONSISTENCY = ALL



Ausgangssituation:
Replikationsfaktor = 3

Tunable Consistency: READ



Der Client liest aus einem beliebigen Replikationsserver

Dies verlangt vom Lesevorgang einen CONSISTENCY = ONE

Ausgangssituation:
Replikationsfaktor = 3

Zu CONSISTENCY ONE/ALL

- Bei beiden Varianten ist ein Partitionsfehler nicht zulässig
- Es gibt noch einen Ausweg:
  - Es ist möglich, Schreib- und Lesevorgänge zuzulassen, obwohl nicht alle Replikationsserver erreichbar sind
  - Wie kann Konsistenz erreicht werden, ohne CONSISTENCY ALL?
  - Replikationsfaktor: 3
  - WRITE: CONSISTENCY 2, READ: CONSISTENCY 2
  - Replikationsfaktor: 5
  - WRITE: CONSISTENCY 2, READ: CONSISTENCY 4
  - Allgemein: QUORUM = (RF + 1)/2
  - WRITE: CONSISTENCY QUORUM, READ: CONSISTENCY QUORUM

Der Betrieb eines Cassandra Clusters

Cassandra-Knoten

- Jeder Knoten sind konzeptuell gleichwertig
  - Es gibt damit nur einen Cassandra-Prozess, eine Cassandra-Implementierung
  - Ausnahme: Seed-Server
    - Sind notwendig, um andere Knoten in den Cluster aufnehmen zu können
    - Mit Hilfe der Seeds kann insbesondere die Verteilung der Partitionen erfolgen
    - Definition erfolgt beim Start: Falls im Aufruf keine Option “Seed-Server” angegeben ist, dann ist der Knoten ein Seed-Server
-

Ressourcen für jeden Knoten

- Netzwerk
  - Konfiguration erfolgt in der cassandra.yaml
  - Standard-Konfiguration
  - Intern: Port 7000
  - Extern: Port 9042
- CPU
  - Relativ unkritisch, CPU-Leistung wird nur für die Speicherverwaltung benötigt
- Haupt-Speicher, “Heap” muss für die Knoten großzügig dimensioniert werden (8G - 64G)
- Storage
  - Kritisch, muss schnell und groß sein
  - Jeder Schreibvorgang wird in ein ein Append-Only-Log eingetragen
    - SSTables pro Cassandra-Tabelle enthalten sämtliche Änderungen

Best Practice: 2 Netzwerke

Cassandra ist ein Java-Prozess

- Als Java-Prozess können eine Reihe von allgemeinen Eigenschaften als gegeben betrachtet werden
  - Ausführung in der so genannten Java Virtual Machine
  - Speicherverwaltung wird von der JVM übernommen, nicht vom Betriebssystem direkt
    - -Xmx: Maximalgröße des Hauptspeichers
    - Garbage Collection ist ein interner Prozess in der JVM, die eine Speicherbereinigung ausführt
      - “Stop-the-World”-Effekt: Während eines GC existieren Zeitfenster, in denen der Java-Prozess exklusiv nur die GC ausführen kann
        - Länge ist abhängig vom Java-Speicher und der CPU-Leistung
        - Unterdimensionierte CPUs können zu Stop-The-World-Effekten im Sekundenbereich führen
        - Richtig dimensionierte Systeme bleiben im unteren zweistelligen Millisekunden-Bereich
- Das cassandra-Start-Skript definiert den Java-Prozesse

Cassandra ist Container-fähig

- Heutzutage wird Cassandra in den meisten Fällen innerhalb eines Containers bereitgestellt
  - z.B. Docker-Container orchestriert in einer Cloud-Lösung wie beispielsweise Kubernetes
- Apache Cassandra Community stellt hierfür ein Official Docker Image bereit
- Starten des Containers
  - docker run --name some-cassandra -d -p 9042:9042 cassandra
    - Vorsicht: Das ist ein Cluster nur mit einem einzigen Knoten
      - damit kann nur ein Replikationsfaktor 1 genutzt werden

Cluster mit Containern

- Nichts anderes als eine Topologie von Cassandra-Containern
- Definition
  - In Kubernetes über einen POD
  - docker-compose und einer docker-compose.yml
    - Beispiel
      - https://github.com/Javacream/org.javacream.training.cassandra/blob/integrata_17.1.2022/docker/docker-compose.simple_cluster.yml
    - Aufruf:
      - docker-compose -f docker-compose.simple_cluster.yml up -d

Das nodetool

- Optionen
  - describecluster
  - status
  - viele viele weitere Aufrufoptionen
- Werkzeug für die Nutzung durch einen “Hardcore-Administrator”, der ein Terminal benutzt

Java-Überwachung mit JMX

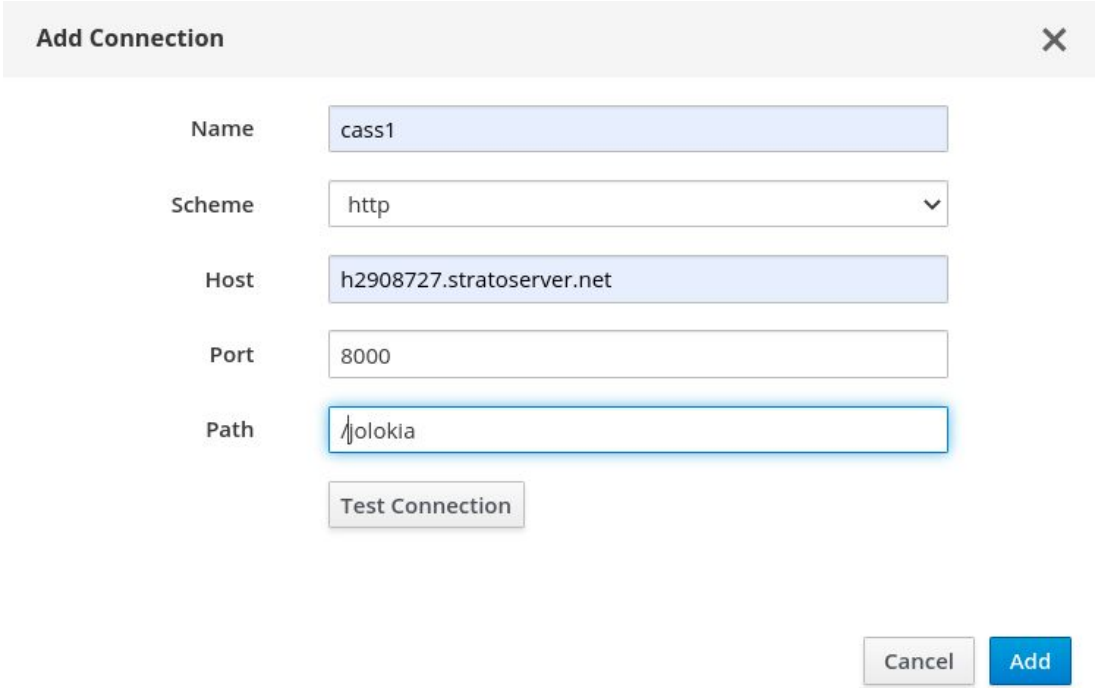
- Java Management Extension
  - integriert in die normale Java Virtual Machine
- Informationen können über ein Standard-Verfahren aus der JVM ausgelesen werden
  - Über einen ObjectName werden zusammengehörende Informationen identifiziert
    - Hierarchischer Name
      - java.lang:type=Memory
  - Jedes JMX-Object hat einen Satz von Attributen und Operationen
    - HeapMemoryUsage
    - gc()

JMX-Clients

- Zugriff auf die JMX-Komponente der JVM ist nicht wirklich gut standardisiert
- Zusatzwerkzeug: “jolokia”
  - Mit Jolokia ist der Zugriff auf JMX über http-Protokoll (über REST) möglich
    - http://host:port/jolokia/read/java.lang:type=Memory/HeapMemoryUsage
  - Ab jetzt kann der Cassandra-Cluster in jegliche existierende Überwachung eingehängt werden
    - Nagios
    - Prometheus
    - …

Hawt als JMX-Client

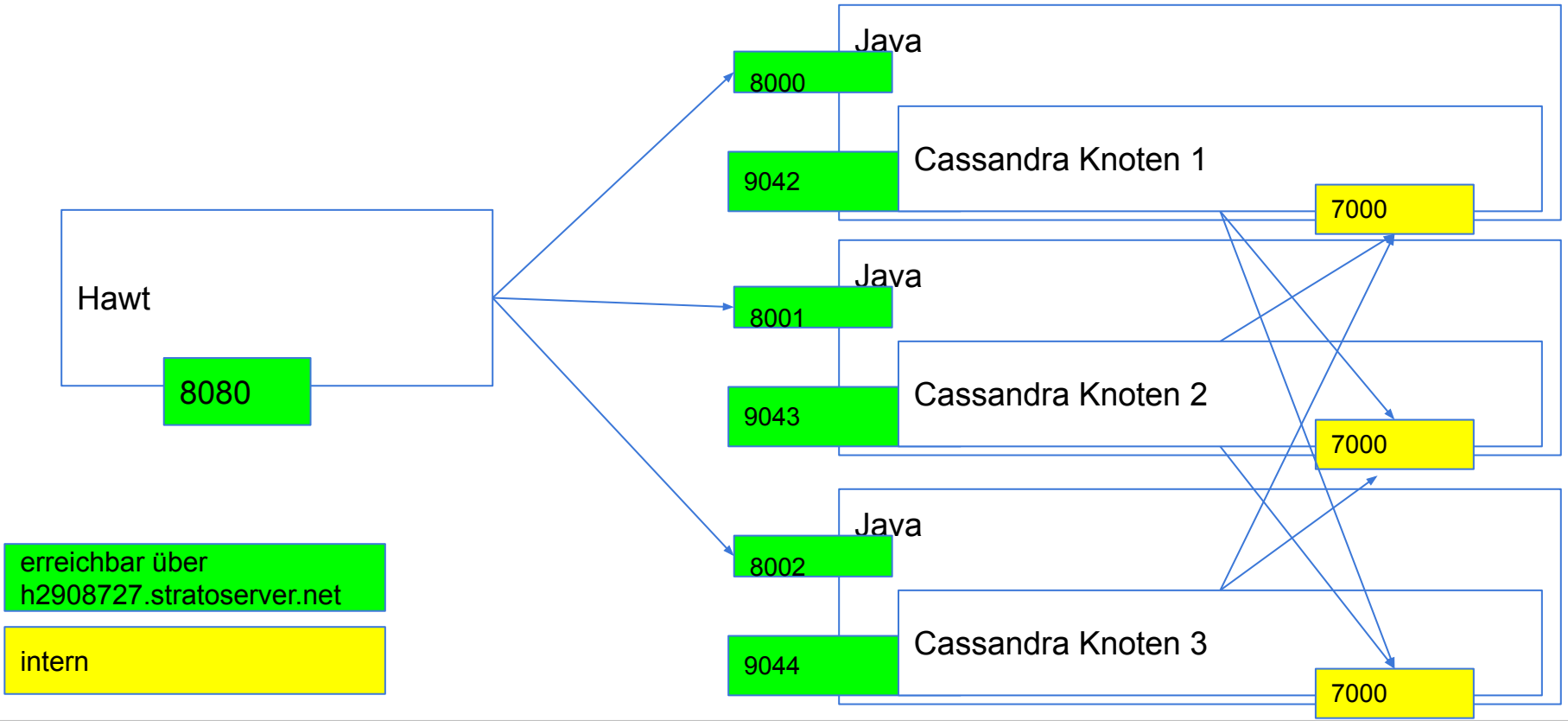
- http://h2908727.stratoserver.net:8080/hawtio/



ToDo

- Verbinden Sie sich auf das Hawt-System-des Cassandra Clusters!
- Verbinden Sie sich mit einem Cassandra-Knoten
- Navigieren Sie z.B. auf java.lang:type=Memory
- Zeichnen Sie sich bitte ein Bild, in dem die auf dem Strato-Server vorhandenen Prozesse eingezeichnet sind!
  - Welche Ports, welche Prozesse?

Die Umgebung auf h2908727.stratoserver.net



Überwachung des Cassandra-Knotens

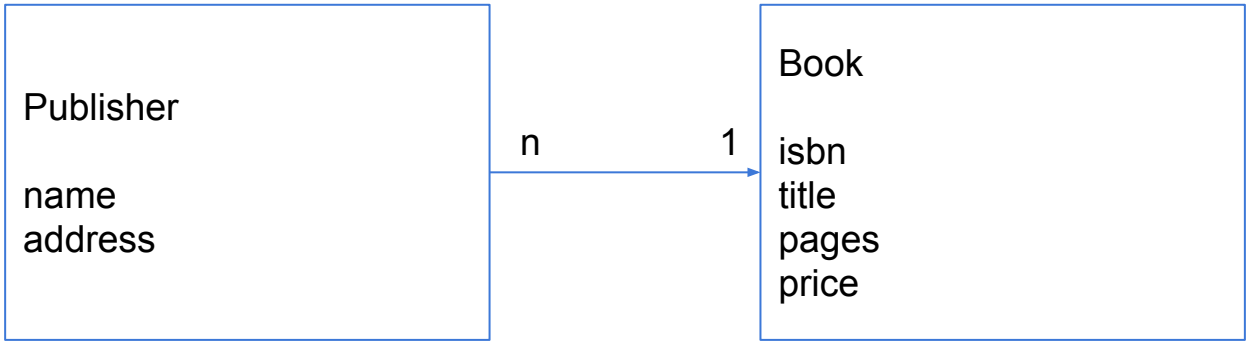
- Gesamte Überwachung von Cassandra basiert ausschließlich auf JMX
  - auch nodetool zieht seine Informationen aus JMX
- Übersicht relevanter ObjectNames
  - org.apache.cassandra.db:type=StorageService
  - …
- https://github.com/soccerties/cassandra-monitoring

Anwendungs-Architekturen mit Apache Cassandra

- Die Cassandra-Community stellt für die gängigen Programmiersprachen Treiber zur Verfügung
- z.B. Pure Java
  - https://github.com/Javacream/org.javacream.training.cassandra/tree/integrata_17.1.2022/org.javacream.training.apache.cassandra.basic
- Nochmals viel komfortabler ist das Spring-Framework mit Spring Data Cassandra
  - https://github.com/Javacream/org.javacream.training.cassandra/tree/integrata_17.1.2022/org.javacream.training.apache.cassandra.springboot
- Notwendig zum Nachvollziehen ist eine Entwicklungsumgebung (z.B. Eclipse)

Anwendungs-Architektur

- Basiert im Endeffekt auf dem Entity-Modell der Fachanwendung



Query “First”

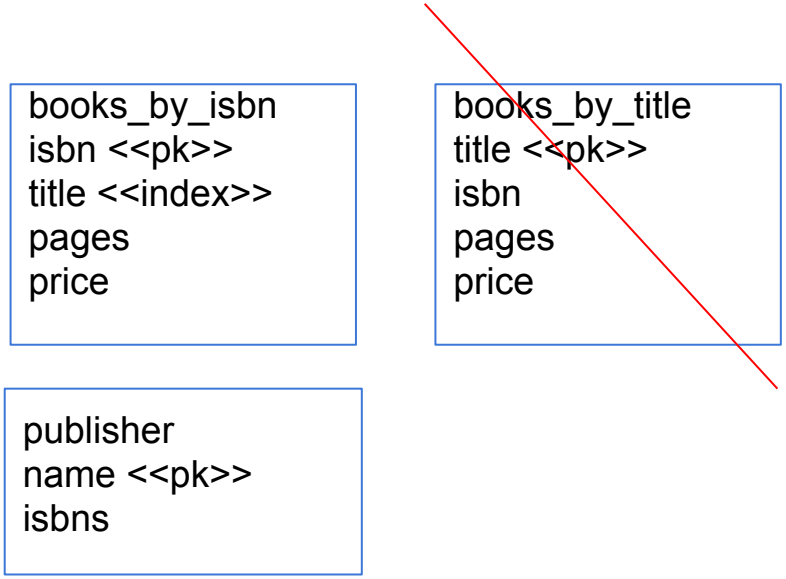
- “finde Buch-Informationen (title, pages, price) an Hand der isbn”
- “finde Buch-Informationen (isbn, pages, price)an Hand des title”
- “finde alle Bücher (isbns) eines Publishers an Hand des Publisher-Namens”

books_by_isbn
isbn <<pk>>
title
pages
price

books_by_title
title <<pk>>
isbn
pages
price

publisher
name <<pk>>
isbns

Review



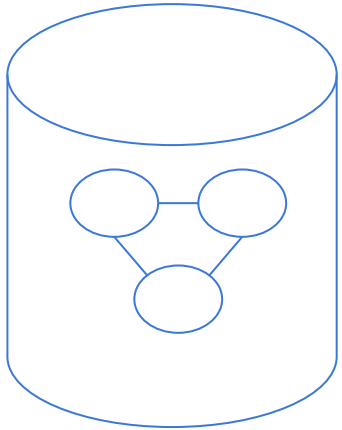
Weitere Operationen

- insert
  - insert into publisher (name, books) values (‘Springer’, {})
  - analog für ein Buch
- Anlegen eines Buches unter Angabe des Verlegers
  - insert into books (...) values (...)
  - update publisher set books = books + {isbn} where ….
- delete
  - Löschen eines Buches muss auch in der Bücher-Liste des Verlegers erfolgen

Eine auf Cassandra basierender Service

Service

Operationen
z.B. deleteBook{
delete from books…
update publisher set
}



Service-Operationen

- Orchestrieren die Aktualisierung der notwendigen Tabellen
  - Aus der Arbeitsweise von Cassandra resultiert zwangsläufig eine redundante Datenhaltung
- Cassandra ist in der Lage, mehrere Aktionen (Aktualisierungen) in einer Transaktion zu gruppieren