Big-Data-Technologien

Kapitel 11: NoSQL – Wide Column Stores

Hochschule Trier
Prof. Dr. Christoph Schmitz

Wide Column Stores

- Daten organisiert in **Zeilen** und „**Spalten**“
- jede Zeile kann **unterschiedliche** Spalten haben
- Zugriff über **Schlüssel**

key123	**name**	**age**	**hair**
	alice	23	red

key543	**name**	**address**	**phone**	**email**
	bob	…	12345	…@…

Wide Column Stores vs. RDBMS

Wide Columns

key123	**name**	**age**	**hair**	
	alice	23	red	
key543	**name**	**address**	**phone**	**email**
	bob	…	12345	…@…

key	**name**	**age**	**hair**	**address**	**phone**	**email**
key123	alice	23	red	NULL	NULL	NULL
key543	bob	NULL	NULL	…	12345	…@…

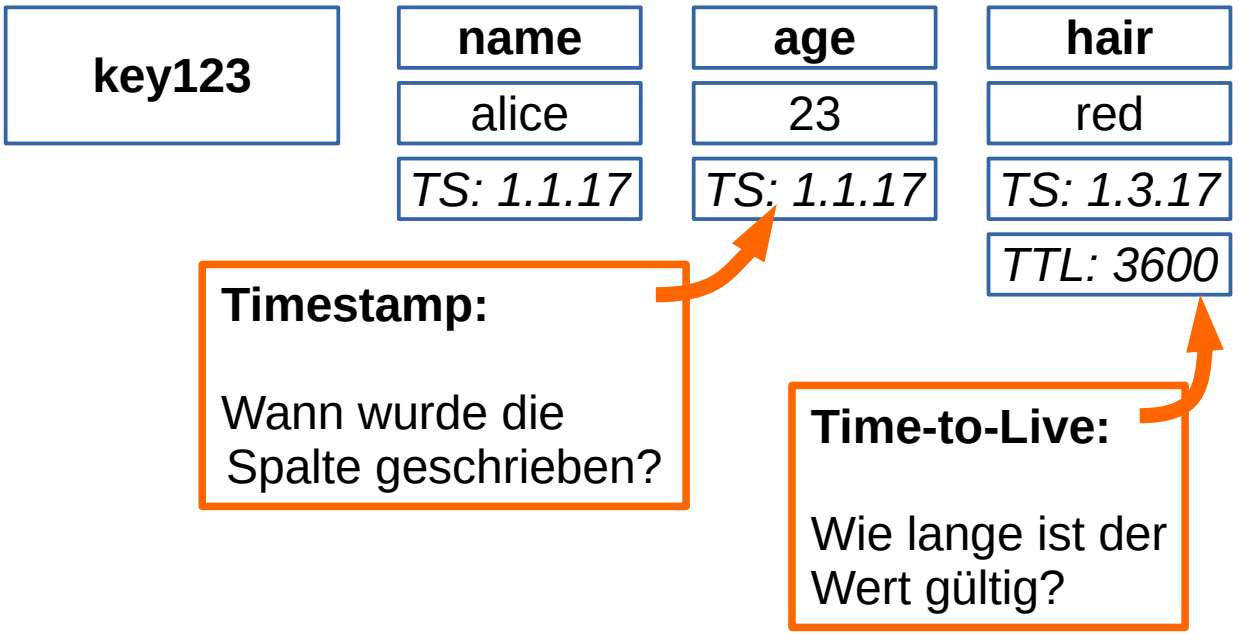
Relational

subj	**pred**	**obj**
key123	name	alice
key123	age	23
key123	hair	red
key543	name	bob
key543	address	…
key543	phone	12345
key543	email	…@…

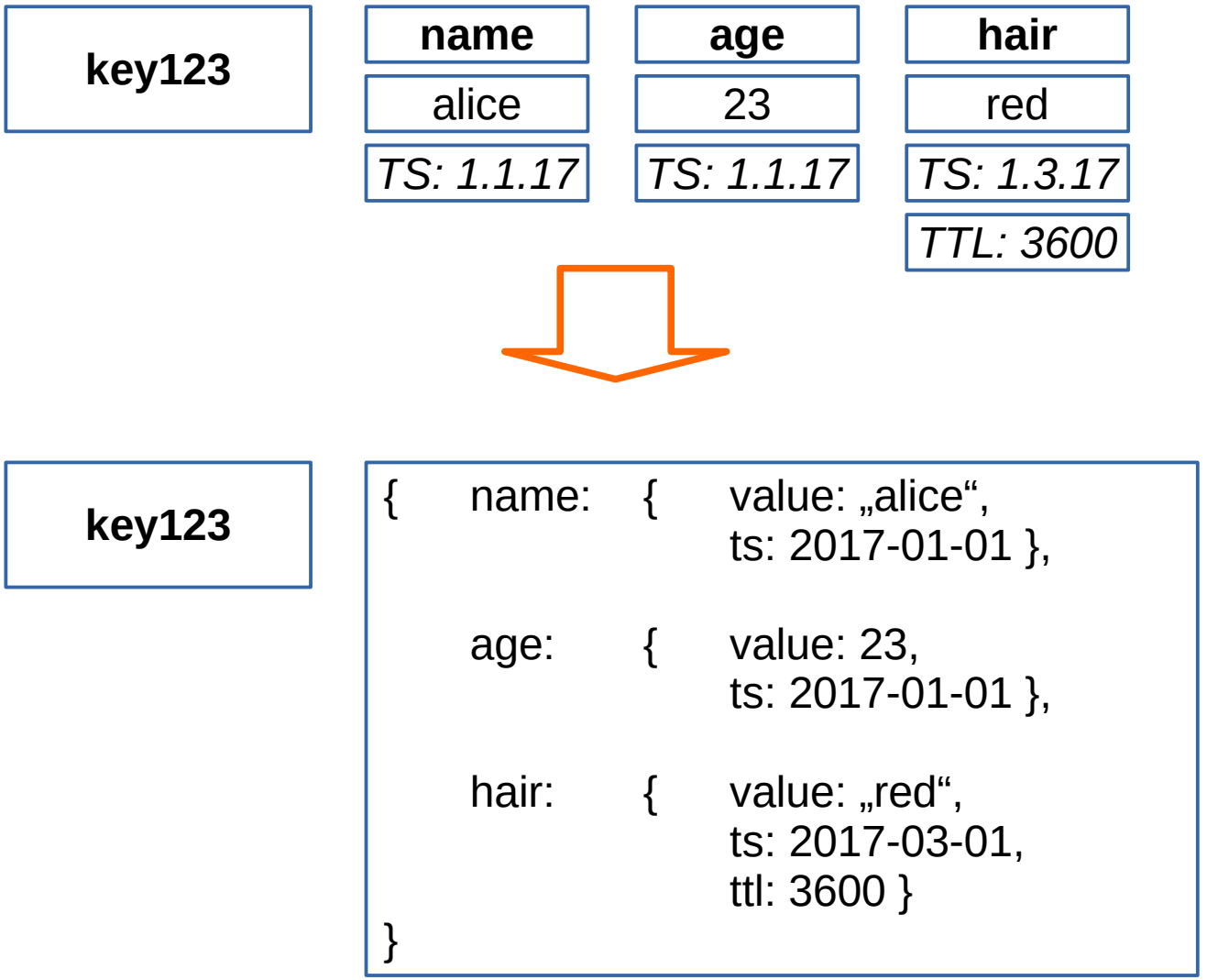
Eigenschaften

- Zugriff nur über **Schlüssel**
- sehr **viele Spalten** möglich
- **Metadaten** (Zeitstempel, TTL) pro Spalte
- (Mehrere **Versionen** der Daten pro Spalte)

Datenmodell detailliert: Metadaten



Andere Sicht auf das Datenmodell



Beispiele

- Webseiten

http://…	title	contents	a:faz.net	a:bild.de	a:…
	foo bar	<html>…	FAZ	Bild	…

- Zeitreihen

weather_st_1	geo_long	geo_lat	2017-03-…	2017-03-…
	7.432	49.543	15°	13°

- Einkaufswagen

user_12	item_123	item_234	item_321
	5	3	1

- Suchmaschine: invertierter Index

affe	doc_1	doc_12	doc_28	doc_…
	3	1	1	2

baum	doc_1	doc_38	doc_112	doc_118	doc_…
	1	2	3	2	1

Google Bigtable (2006)

Bigtable: A Distributed Storage System for Structured Data

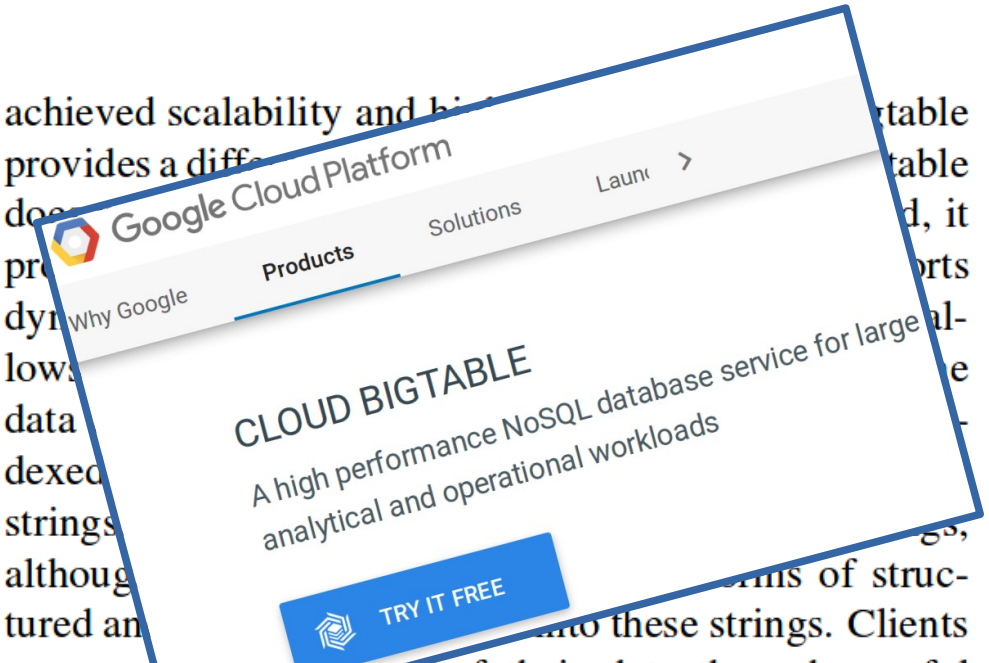
Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber

{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com

Google, Inc.

Abstract

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving). Despite these varied demands, Bigtable has successfully



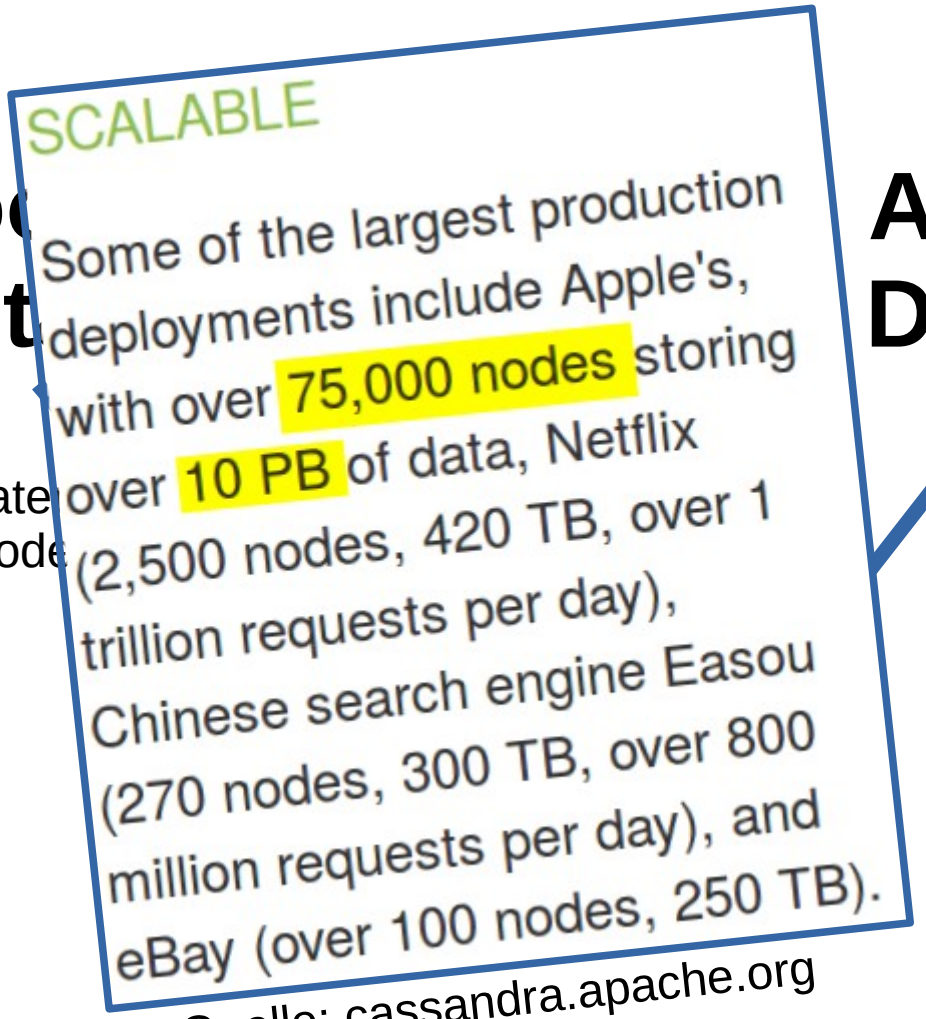
Apache Cassandra

Goo
Bigt

Date
Mode

Amazon
Dynamo

Verteilung
Konsistenz
Garantien



Quelle: cassandra.apache.org

Bild: Wikipedia

Apache Cassandra

- **Datenmodell** wie **Google Bigtable**
  - Schlüssel, Zeilen, variable Spalten
- **Verteilung** wie **Amazon Dynamo**
  - Verteilung über Consistent Hashing
  - Eventual Consistency, Reparaturmechanismen
  - Einstellbare Garantien
- CQL: **SQL-artige Sicht** für Anfragen
- Optimiert für **hohe Schreiblast**

Datenmodell und Anfragesprache

CQL (Cassandra Query Language)

- SQL-artige Sicht auf Wide-Column-Modell

```
bigdata200@infbdt05:~$ cqlsh infbdt05 -u bigdata200
Password: *********

Connected to Test Cluster at infbdt05:9042.
[cqlsh 5.0.1 | Cassandra 2.1.15 | CQL spec 3.2.1 | Native protocol v3]

> USE bigdata200;

> CREATE TABLE person (name TEXT PRIMARY KEY, age INT, hair TEXT);

> INSERT INTO person (name, age, hair) VALUES ('alice', 23, 'red');
> INSERT INTO person (name, age, hair) VALUES ('bob', 22, 'blonde');

> SELECT * FROM person;

 name  | age | hair
-------+-----+--------
   bob |  22 | blonde
 alice |  23 |    red
```

CQL ist nicht SQL…

```
> SELECT * FROM person ORDER BY age;

InvalidRequest: code=2200 [Invalid query] message="ORDER BY is only
supported when the partition key is restricted by an EQ or an IN."

> SELECT * FROM person WHERE name IN ('alice', 'bob') ORDER BY AGE;

InvalidRequest: code=2200 [Invalid query] message="Order by is
currently only supported on the clustered columns of the
PRIMARY KEY, got name"

> SELECT * FROM person GROUP BY hair;

SyntaxException: <ErrorMessage code=2000 [Syntax error in CQL query]
message="line 1:21 missing EOF at 'group' (select * from person
[group] by...)">

> SELECT * FROM person JOIN person;

SyntaxException: <ErrorMessage code=2000 [Syntax error in CQL query]
message="line 1:21 missing EOF at 'join' (select * from person
[join] person...)">
```

Wide Columns in CQL

http://faz.net	title	contents	a:abc.net	a:bcd.de	a:…
	FAZ	<html>…	ABC	BCD	…

```
> CREATE TABLE webpage (url TEXT PRIMARY KEY,
    contents TEXT,
    links MAP<TEXT, TEXT>,
    title TEXT
)
```

Wide Columns!

```
> INSERT INTO webpage(url, title, contents, links) VALUES ('http://faz.net', 'FAZ',
'<html>...', {'a:abc.net': 'ABC'});
> SELECT * FROM webpage;

 url            | contents  | links                  | title
----------------+-----------+------------------------+-------
 http://faz.net | <html>... | {'a:abc.net': 'ABC'} |   FAZ

> UPDATE webpage SET links = links + {'a:bcd.de': 'BCD'} WHERE url = 'http://faz.net';
> SELECT * FROM webpage;

 url            | contents  | links                                          | title
----------------+-----------+------------------------------------------------+-------
 http://faz.net | <html>... | {'a:abc.net': 'ABC', 'a:bcd.de': 'BCD'} |   FAZ
```

Wide Columns in CQL

```
> CREATE TABLE person (name TEXT PRIMARY KEY, age INT,
  friends LIST<TEXT>);

> INSERT INTO PERSON(name, age, friends) VALUES ('alice', 23,
[ 'bob', 'charlie', 'eve' ]);

> SELECT * FROM person;

 name  | age | friends
-------+-----+---------------------------
 alice |  23 | ['bob', 'charlie', 'eve']
```

Wide Columns!

Time-To-Live in CQL

```
> UPDATE person USING TTL 10
  SET friends = friends + ['mallory'] WHERE name = 'alice';

> SELECT * FROM person;

 name  | age | friends
-------+-----+-----------------------------------
 alice |  23 | ['bob', 'charlie', 'eve', 'mallory']
```

… 10 Sekunden später…

```
> SELECT * FROM person ;

 name  | age | friends
-------+-----+-----------------------------
 alice |  23 | ['bob', 'charlie', 'eve']
```

Zeitstempel in CQL

```
> SELECT name, age, friends, WRITETIME(age) FROM PERSON;

 name  | age | friends                          | writetime(age)
-------+-----+----------------------------------+------------------
 alice |  23 | ['bob', 'charlie', 'eve'] | 1489568646008075
```

Mikrosekunden seit 1970-01-01T00:00:00Z

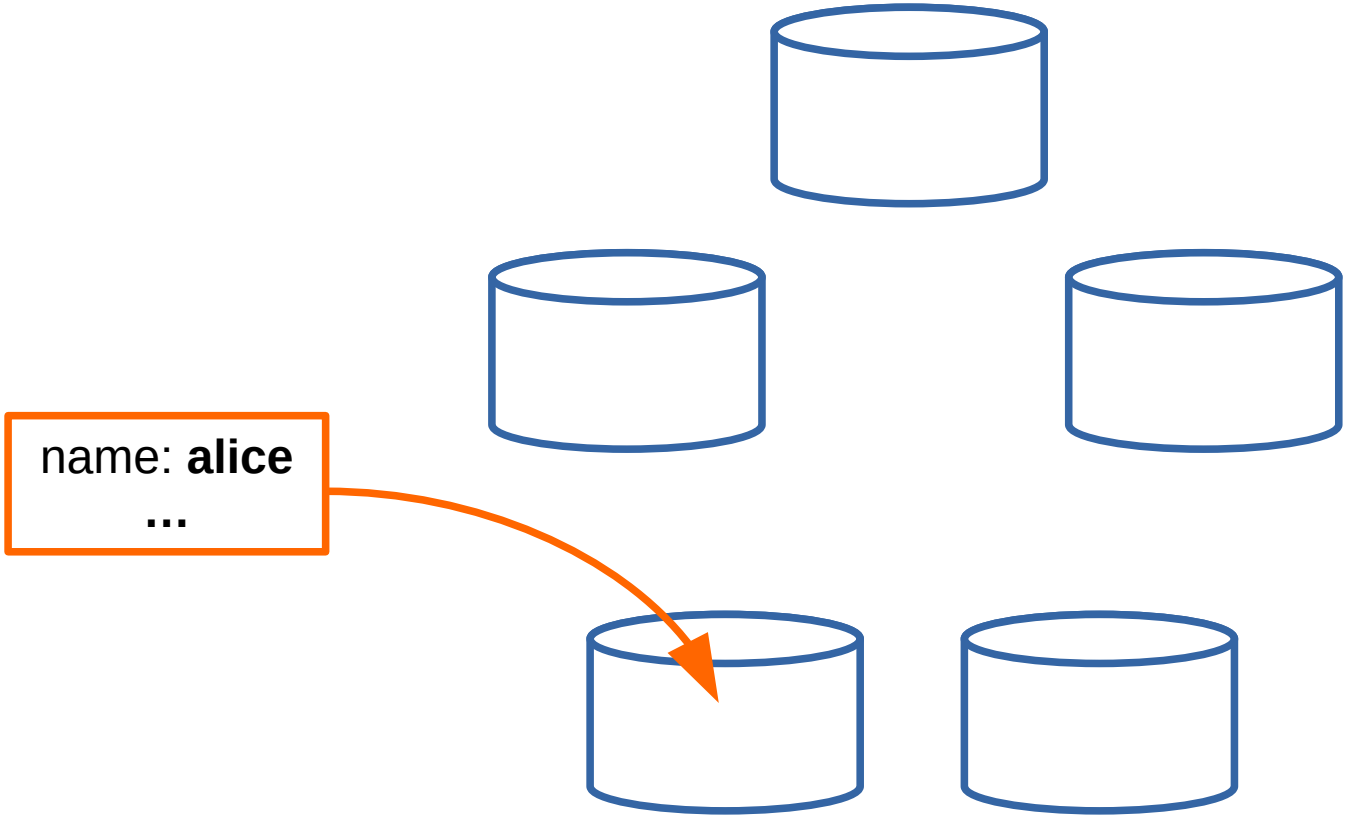
Zwischenfazit: CQL

- Anfragesprache ähnlich SQL
- Wide Columns abgebildet auf Listen, Maps, Mengen
- Zeitstempel und TTL pro Spalte
- Viele SQL-Operationen nicht möglich

Architektur

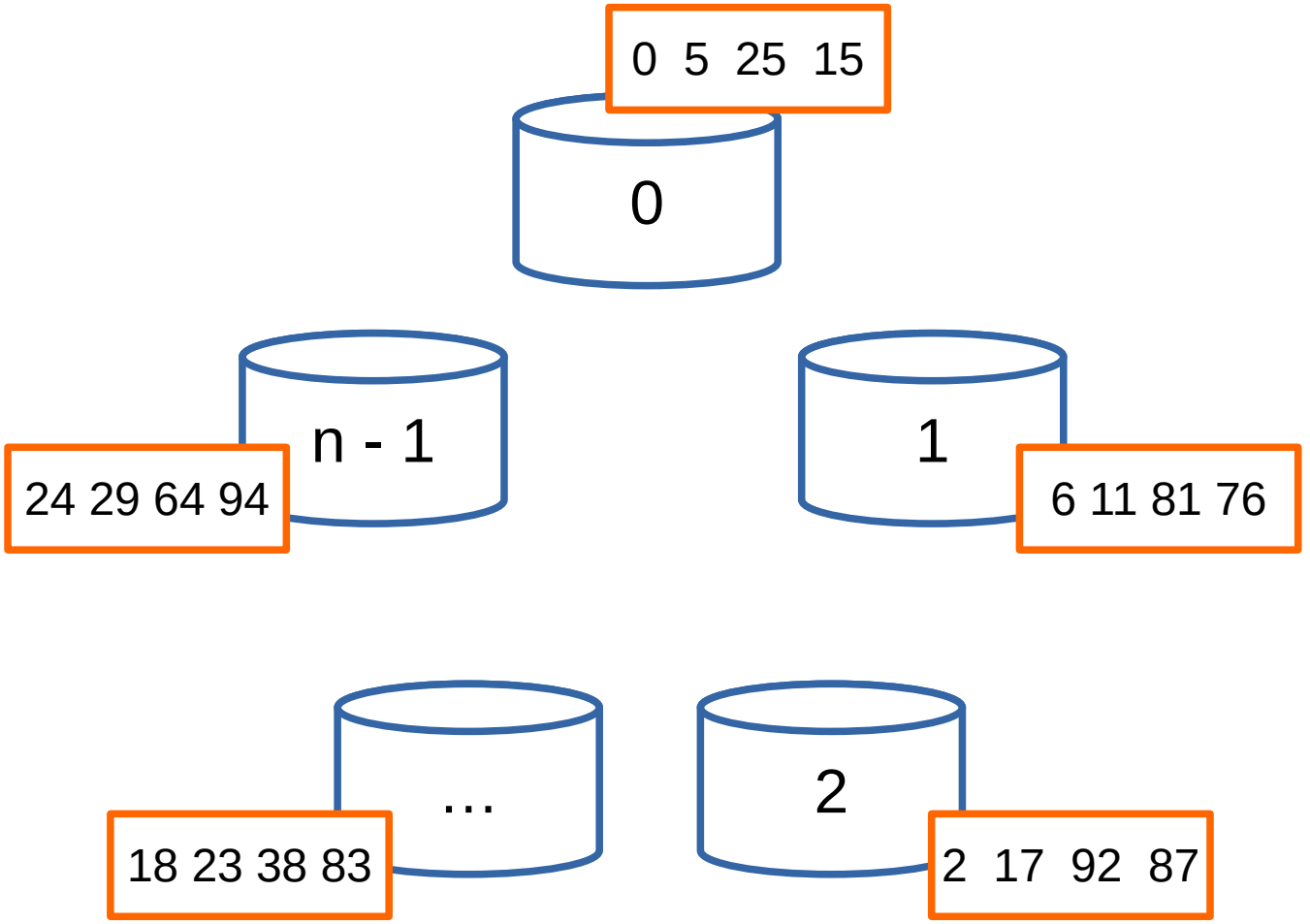
Peer-to-Peer

- Verteilung über gleichartige Knoten
- Verteilung nach Schlüssel



Einfache Idee: Eine Partition pro Knoten

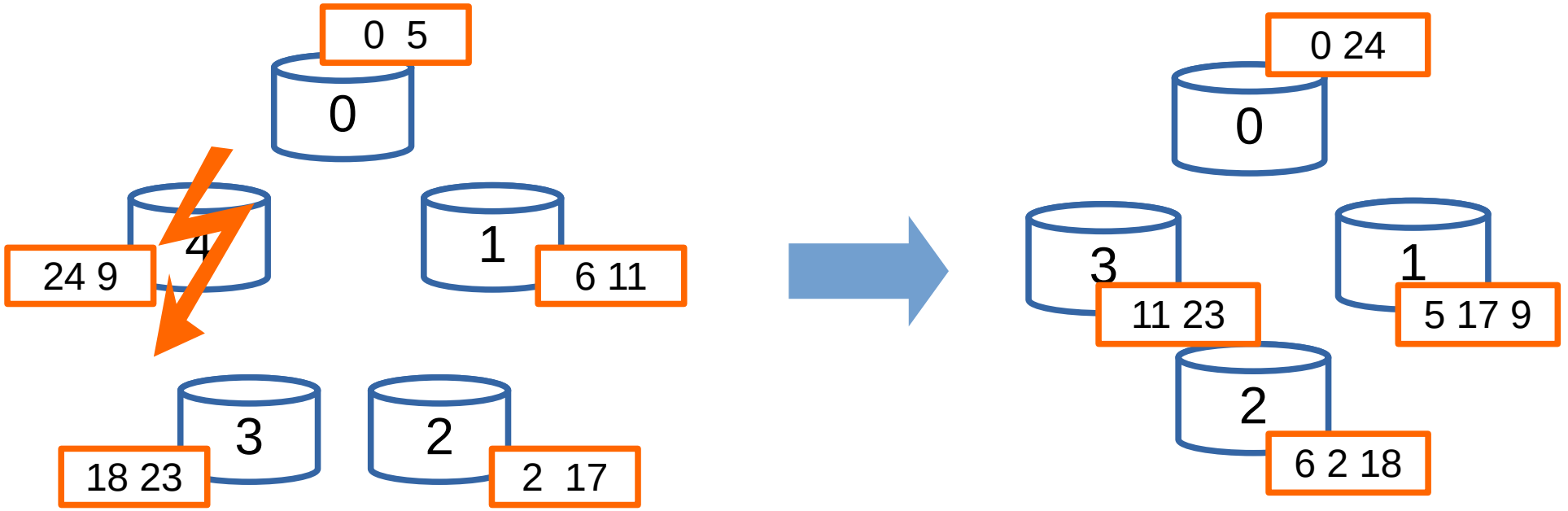
- node = hash(data.key) % n_nodes



Einfache Idee reicht nicht aus

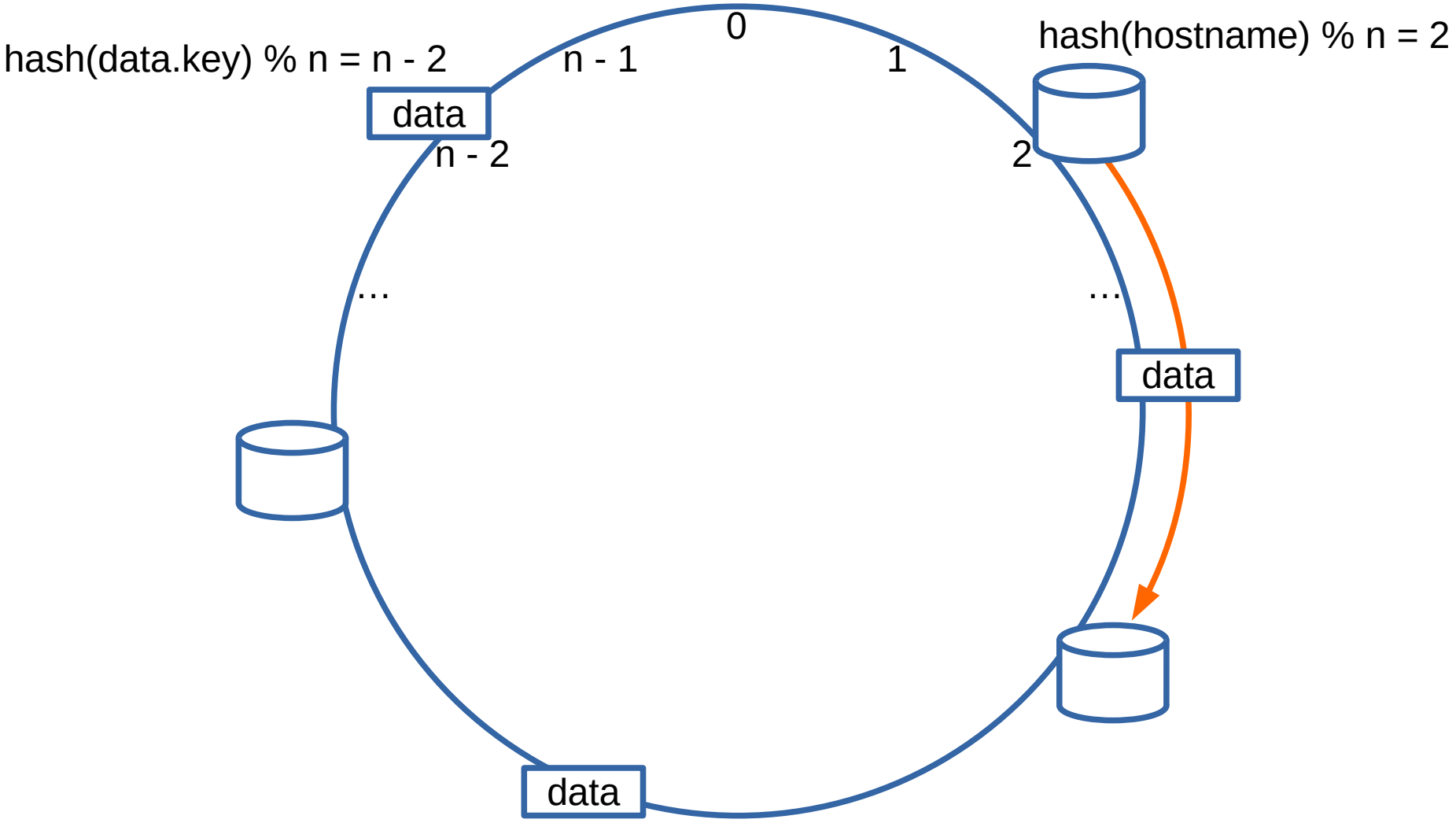
- **Problem 1:** Was tun, wenn Knoten…
  - … hinzukommen?
  - … entfernt werden?
  - … ausfallen?
- **Problem 2:** Was tun bei schiefer Verteilung?

Umpartitionieren bei Ausfall

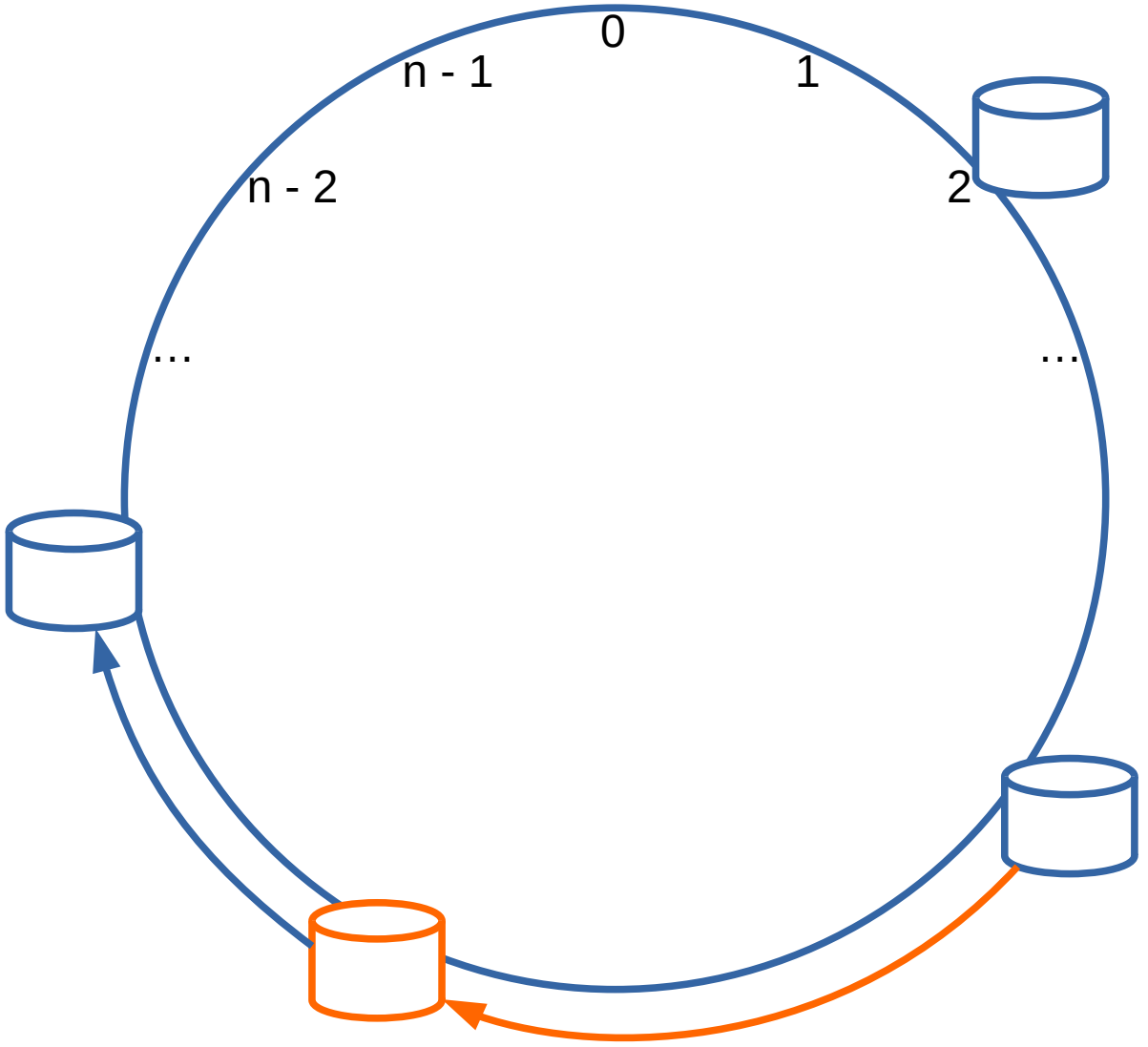


- alle Knoten betroffen
- alle Daten betroffen

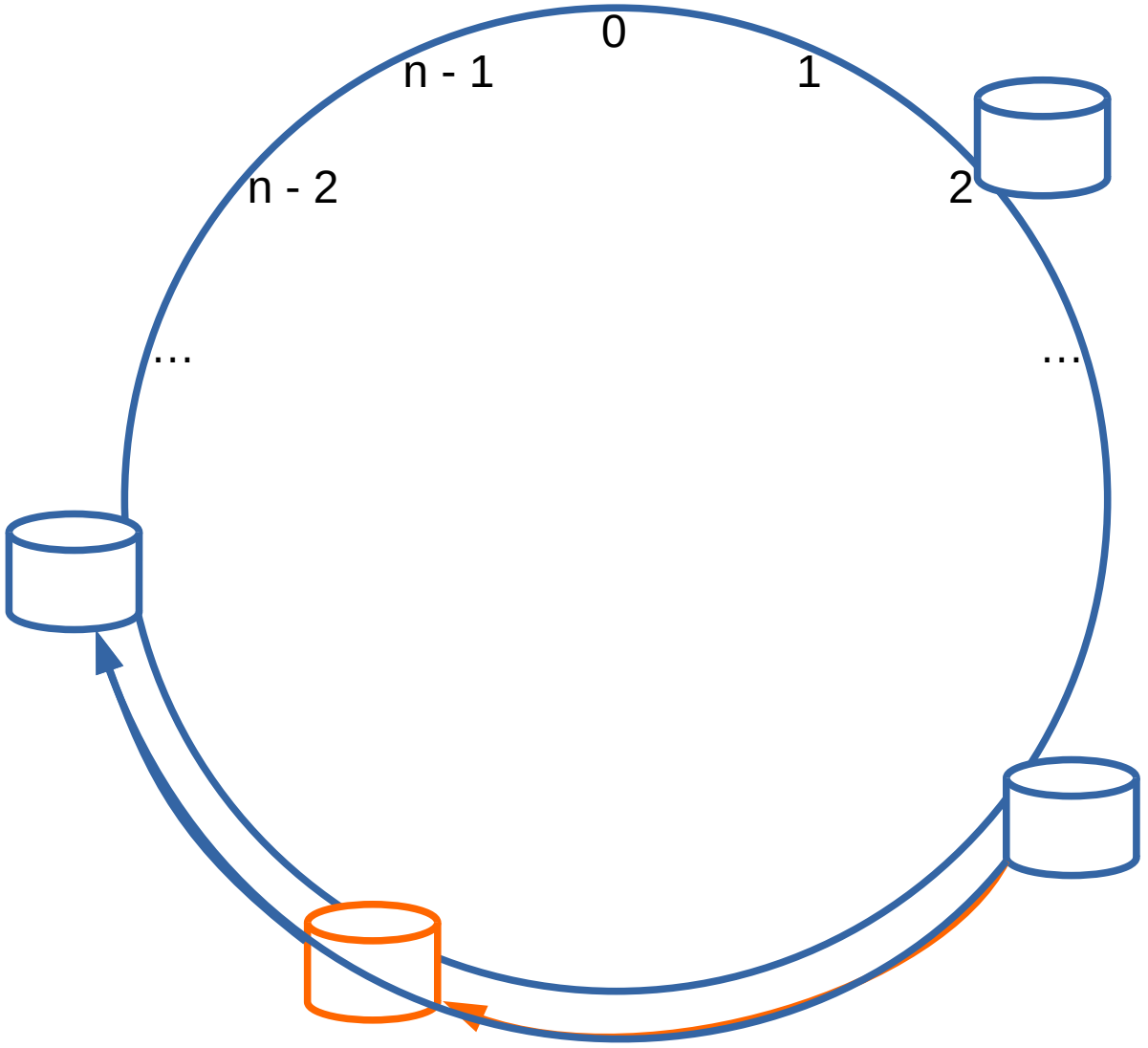
Consistent Hashing



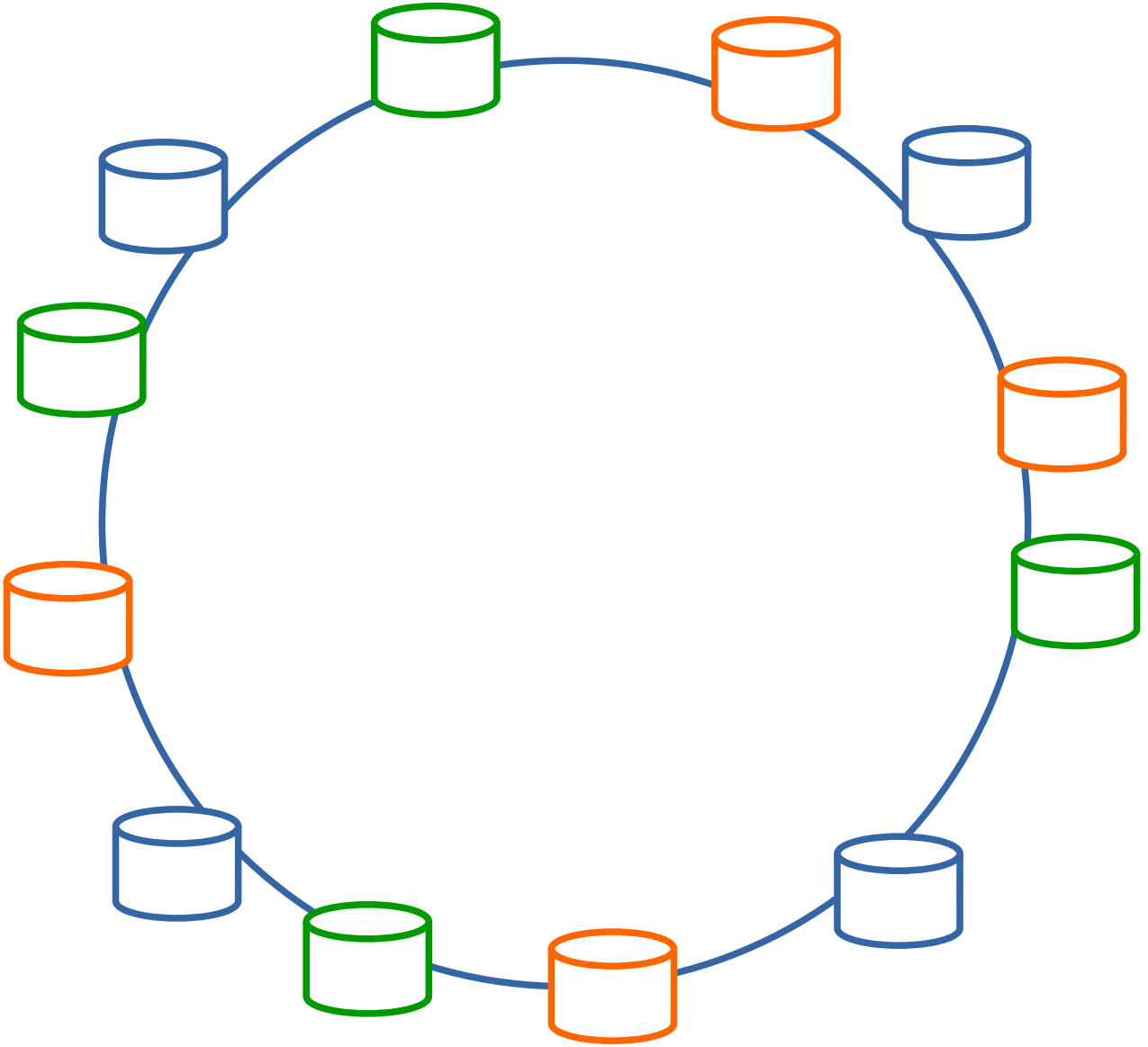
Hinzufügen eines Knotens



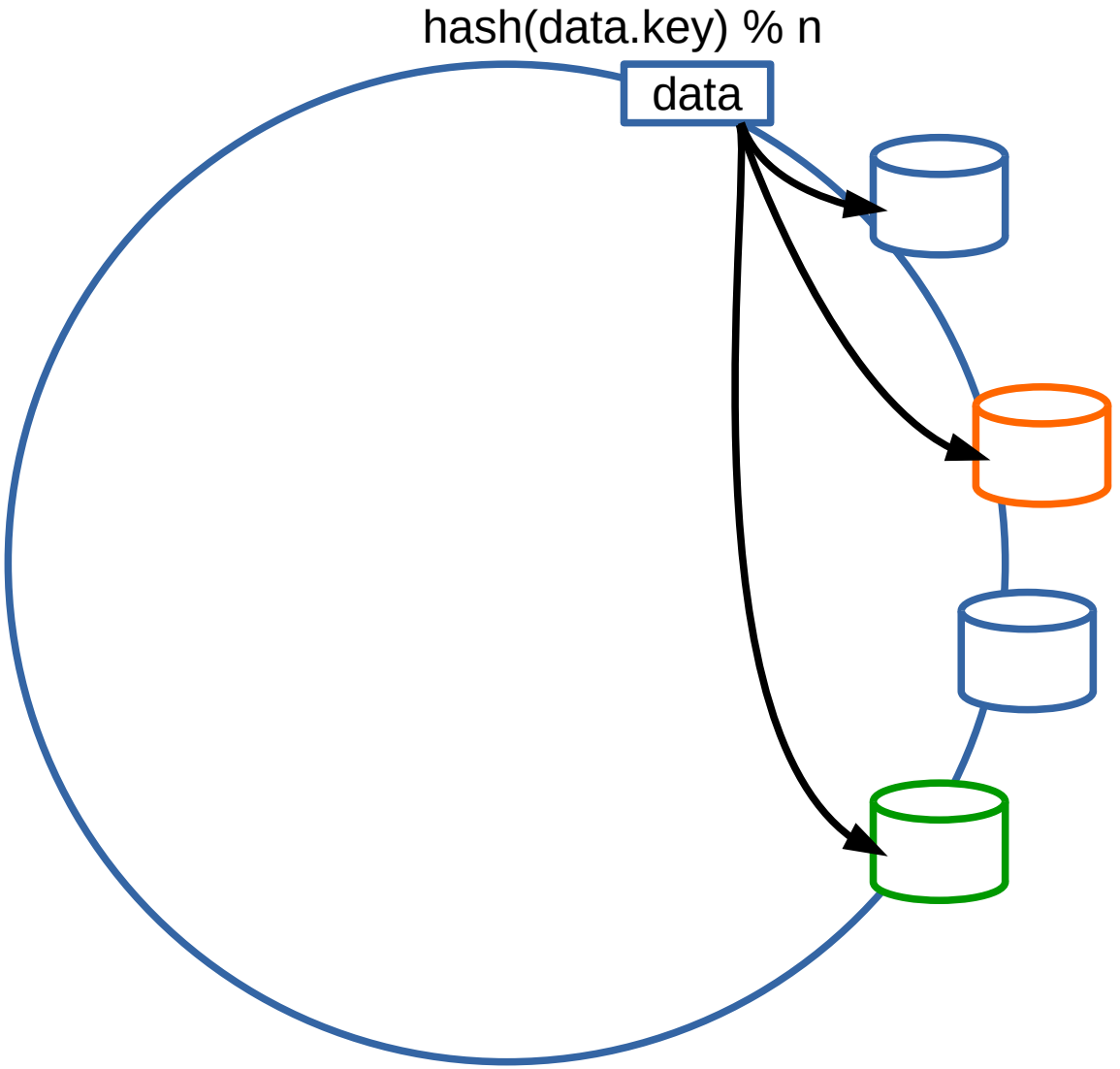
Entfernen eines Knotens



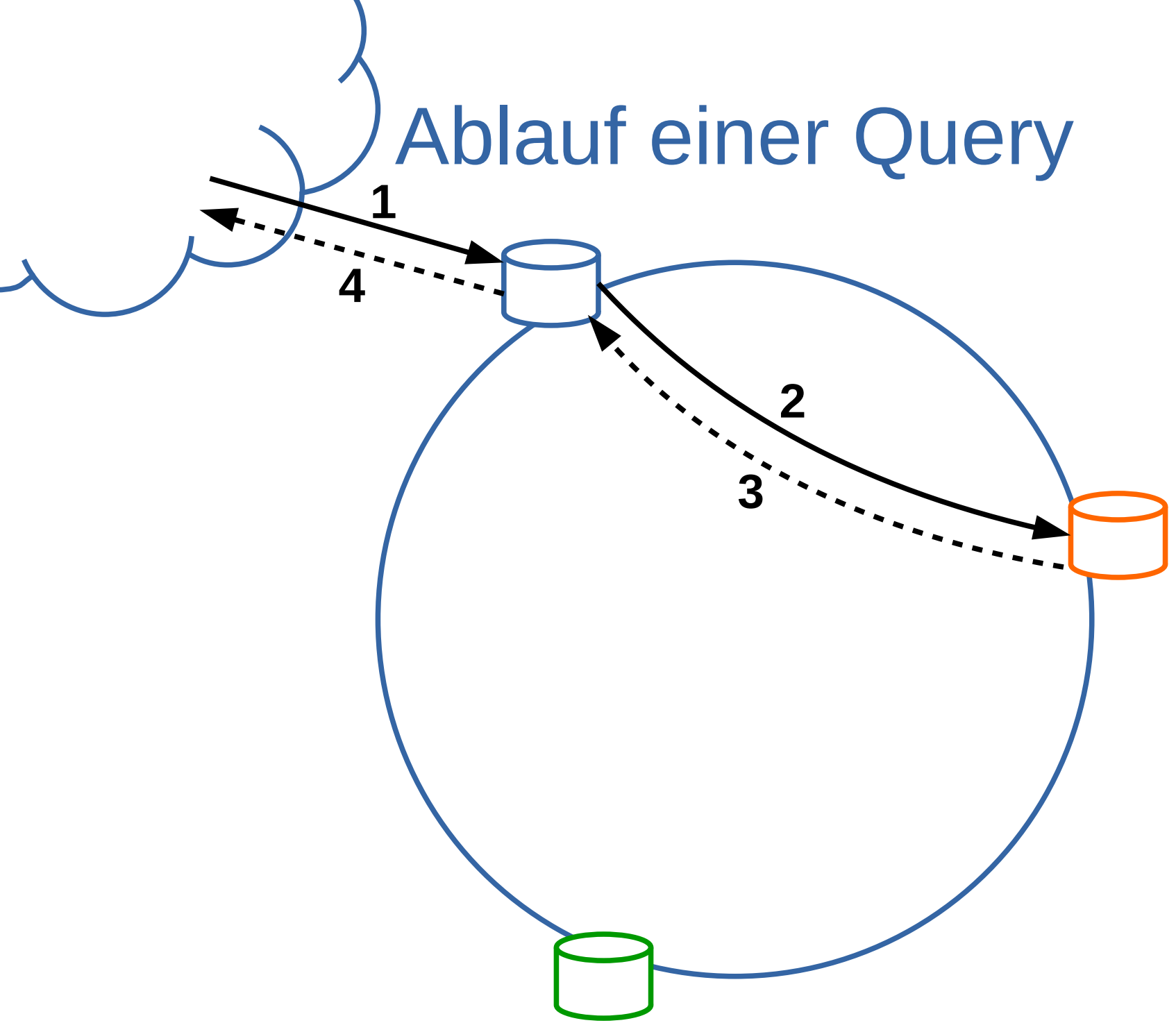
Virtuelle Knoten



Datenreplikation



Ablauf einer Query



Schweizer Taschenmesser: nodetool

```
$ nodetool status

Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address         Load       Tokens  Owns   Host ID                               Rack
UN  143.93.53.143   296.03 KB  256     59.1%  1f61803a-20ef-4bcd-ae6d-65849b69d52f  rack1
UN  143.93.53.142   330.13 KB  256     61.5%  0974394a-4fc9-46c3-a1f2-14dc246af5d4  rack1
UN  143.93.53.145   307.79 KB  256     58.2%  d2e55352-4263-4a64-984c-e7849475501c  rack1
UN  143.93.53.144   296.97 KB  256     61.0%  60f66be5-5ba2-44b4-a5fc-a34007c62ed0  rack1
UN  143.93.53.146   312.92 KB  256     60.2%  8a7df3d3-4578-4091-a04f-2f5feaad7ed0  rack1
```

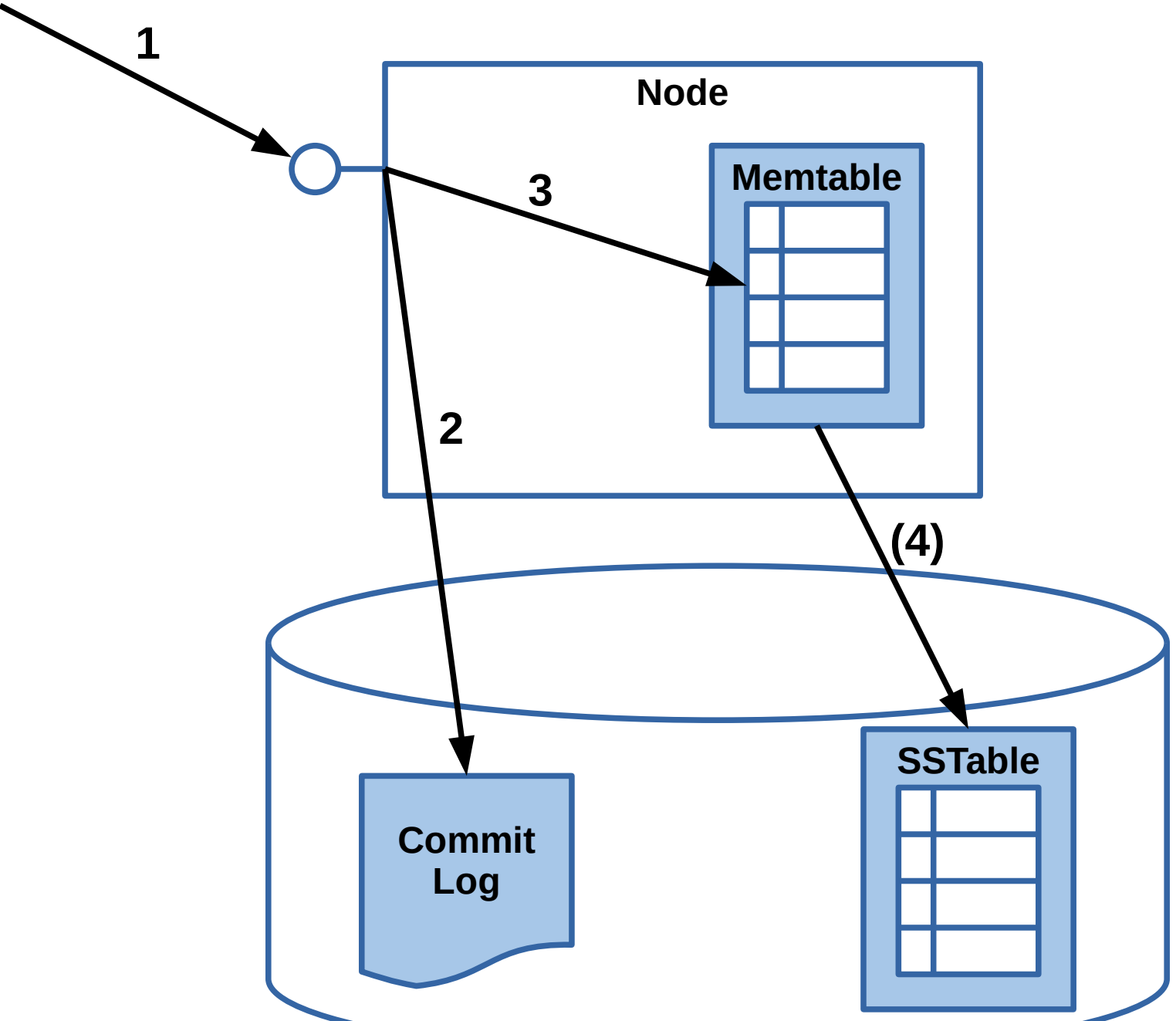
Zwischenfazit: Consistent Hashing

- Daten und Knoten abbilden auf **Ring** [0, …, n – 1]
- Knoten übernehmen je ein **Teilintervall**

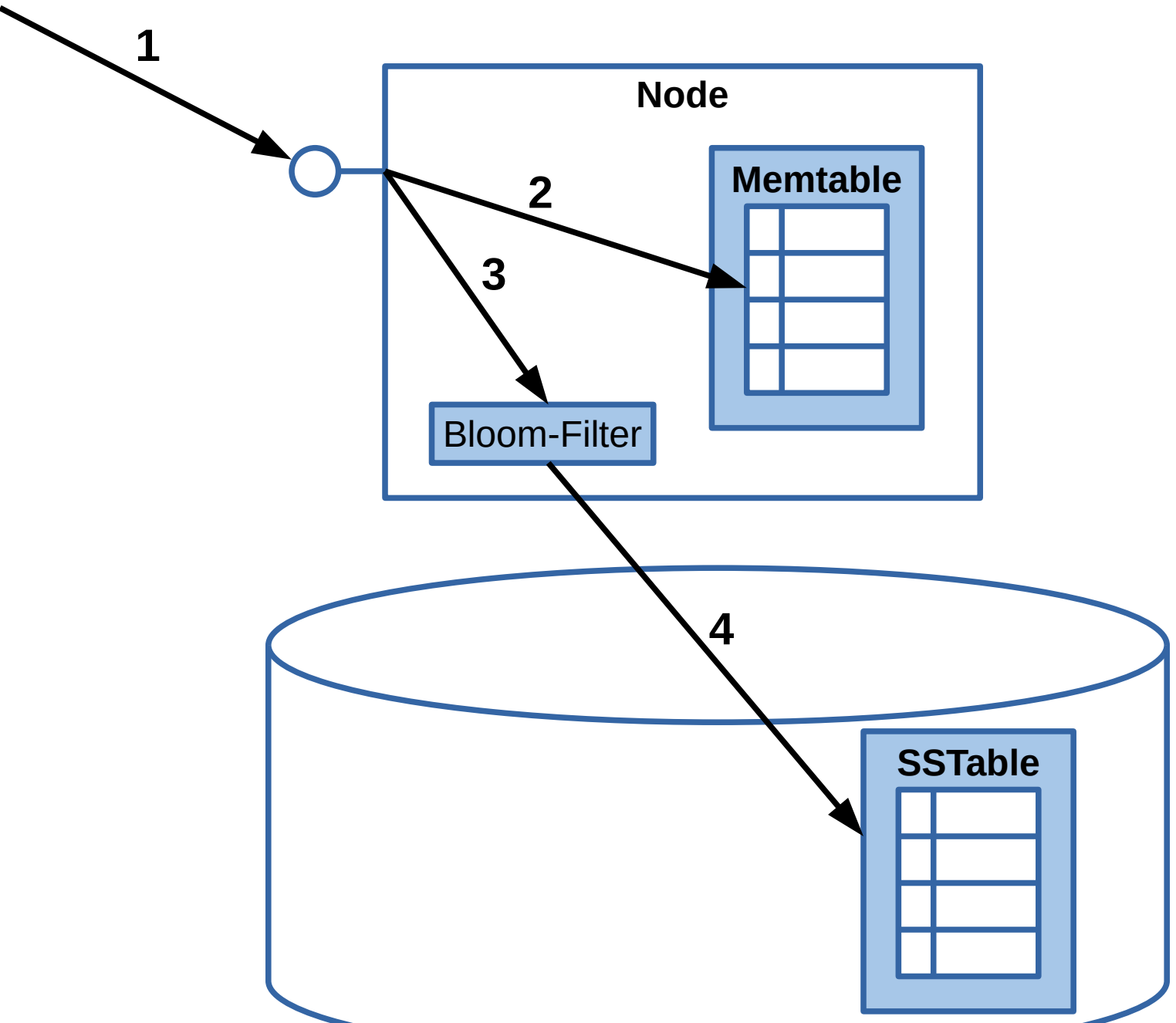
- Ausfall, Hinzufügen betreffen nur **1/n des Rings**

- Knoten vervielfältigen → **Gleichmäßigere Auslastung**
- Daten vervielfältigen → **Schutz vor Ausfällen**

Schreiboperationen

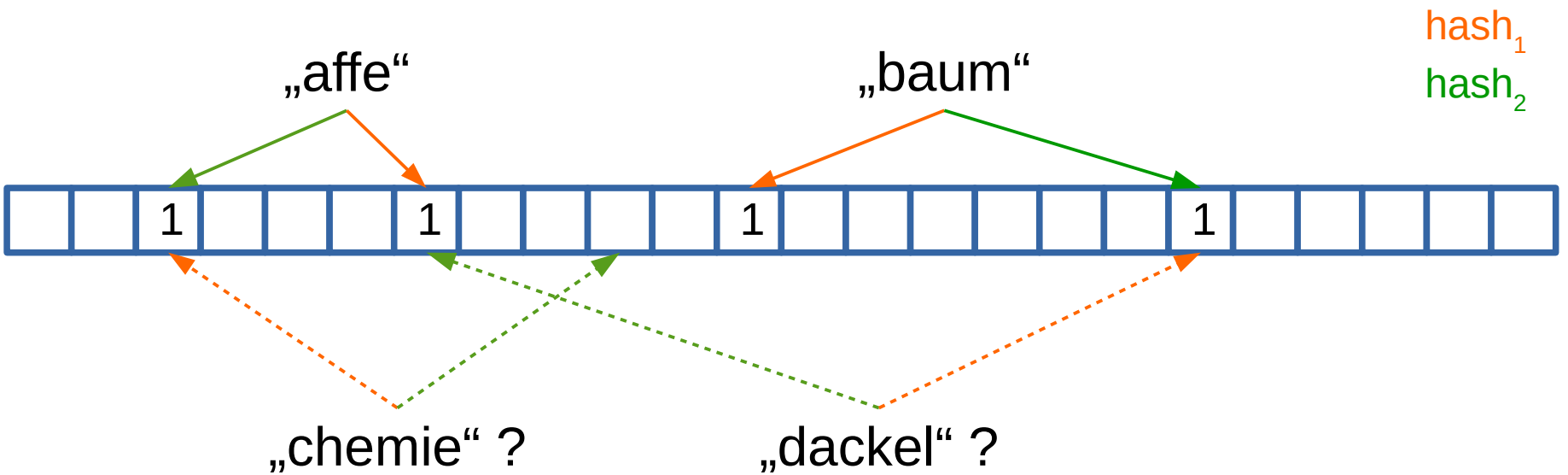


Leseoperationen



Bloom-Filter

- Existiert der Schlüssel in der SSTable?
- Mögliche Antworten:
  - **Nein**
  - **Möglicherweise**



SSTable Compaction

- SSTables sind unveränderlich („immutable“)
- Neuere Werte machen alte Einträge überflüssig

TS	Key	Value
1	123	aaa
2	234	bbb
3	345	ccc
4	456	ddd

TS	Key	Value
6	123	fff
5	456	eee
8	456	hhh
7	567	ggg

TS	Key	Value
6	**123**	**fff**
2	234	bbb
3	345	ccc
8	**456**	**hhh**
7	**567**	**ggg**

Zwischenfazit: Datenstrukturen

- Schreiben zur Sicherheit in **Commit-Log**
- … dann in **Memtable**
- Memtables als **SSTables** persistiert
- **Bloom-Filter** prüfen vor Lesen aus SSTables
- **SSTable-Compaction** → alte Werte verwerfen

Umgang mit Fehlerfällen

Gossiping

- „epidemisches“ Protokoll für **Zustandsupdates**
- Austausch von Informationen über
  - **eigenen** Zustand
  - Zustand **bekannter Knoten**
- **versioniert**
- jede Sekunde…
- … mit bis zu drei Nachbarn

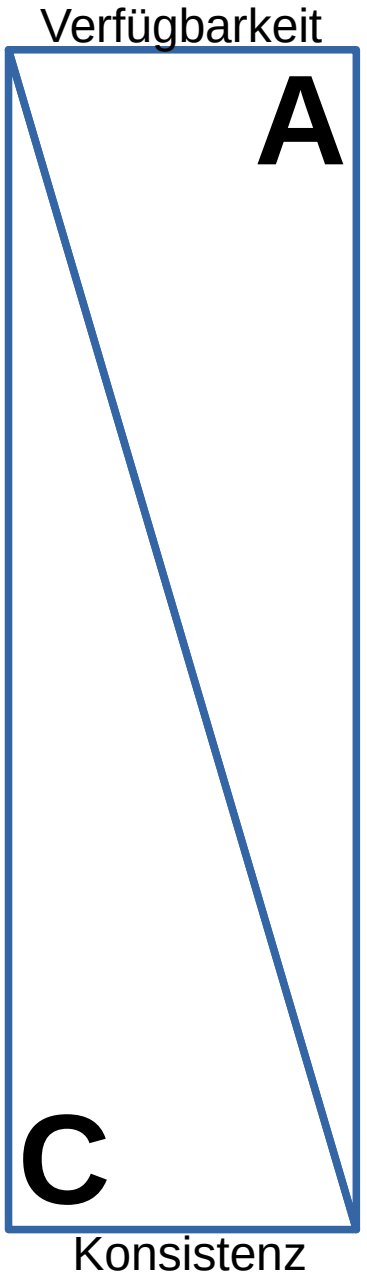
Einstellbare Garantien

- *„Tuneable Consistency“*

- Einstellbarer **Replikationsfaktor**
- Einstellbare Garantien beim **Schreiben**
- Einstellbare Garantien beim **Lesen**

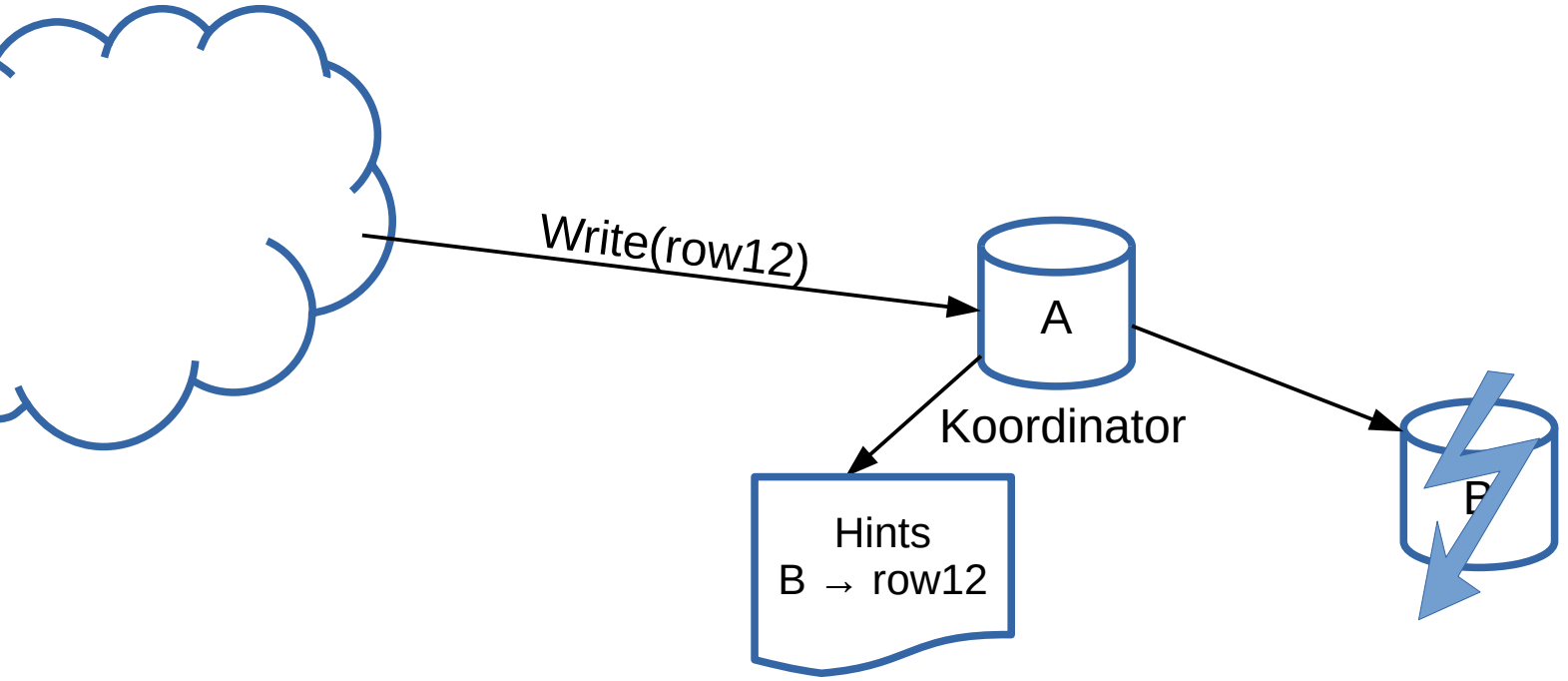
Einstellbare Garantien (Auswahl)

- **ANY**
  - irgendein Knoten, inkl. Hinted Handoff
- **ONE/TWO/THREE**
  - 1/2/3 Replikas
- **QUORUM**
  - Mehrheitsentscheidung
- **LOCAL_…**
  - wie oben, in *einem* Data Center
- **EACH_QUORUM**
  - wie QUORUM, in *allen* Data Centers
- **ALL**
  - alle Replikas



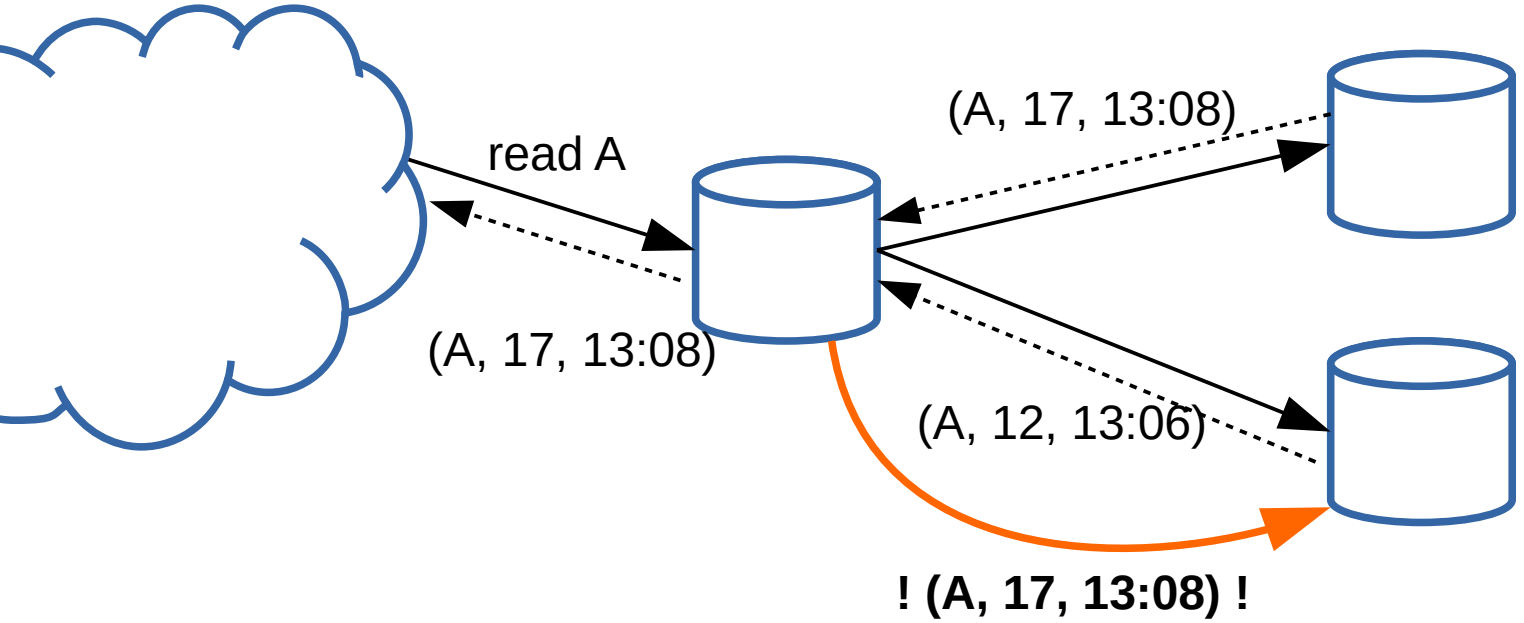
Hinted Handoff

- „Übergabe mit Hinweis“
- Puffern und **Wiederholen fehlgeschlagener Schreibzugriffe**
- Neuer Versuch, wenn Knoten sich erholt



Read Repair

- Reparieren von inkosistentem Zustand
- Level TWO oder höher



Anti-Entropy Repair

- Kompletter Abgleich mit Replikas im **Wartungsbetrieb** (`nodetool repair`)
- Abgleich der Stände aller Replikas
  - Hash-Bäume (→ Merkle-Trees)

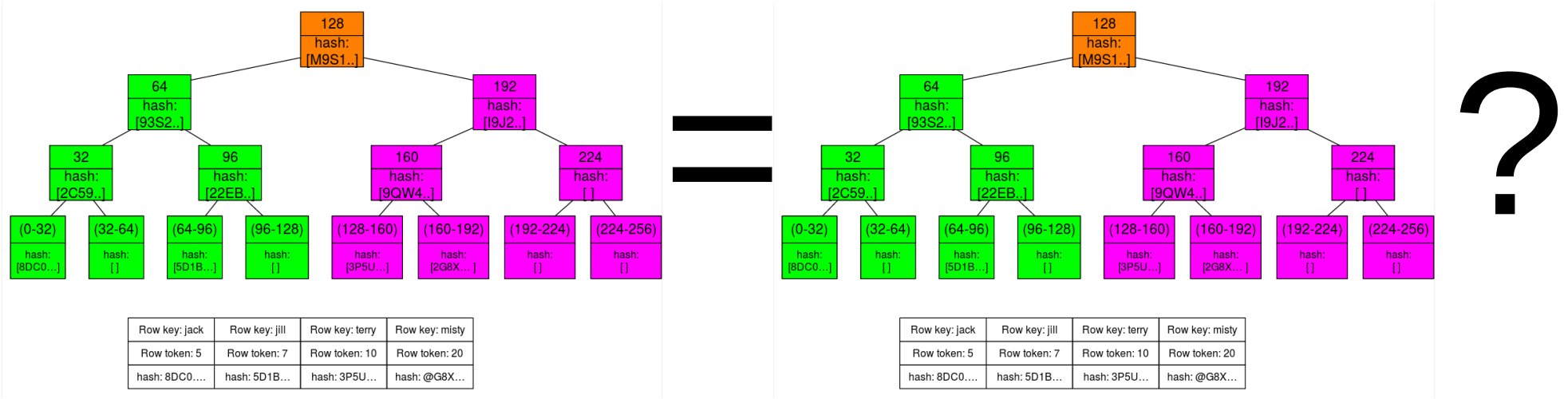


Bild: datastax.com

Erweiterte Anfragemöglichkeiten

Zusammengesetzte Schlüssel und Clustering Columns

- Schlüssel bestimmt
  - **Partitionierung** der Daten im Hash-Ring
  - **Sortierung** der Daten innerhalb einer Partition
- Zusammengesetzte Schlüssel

```
… PRIMARY KEY (uid, name)
```

Partitionierung **Sortierung**

```
… PRIMARY KEY ((company, dept), name, uid)
```

Partitionierung **Sortierung**

Sekundärindizes

- Effiziente Suche über **Nicht-Schlüsselspalten**
- Unsichtbare zusätzliche **Tabellen**
- Nicht für
  - **viele Updates**
  - Spalten mit **hoher Kardinalität**

Sekundärindizes

```
> SELECT * FROM person WHERE age = 34;
InvalidRequest: code=2200 [Invalid query]
message="No secondary indexes on the restricted
columns support the provided operators: "

> CREATE INDEX person_age ON person(age);
> SELECT * FROM person WHERE age = 34;

 name | age | friends
------+-----+---------
  bob |  34 |    null
```

Denormalisierung

- Hinnehmen von **Redundanz** für effiziente Anfragen
- Beispiel: speichere Anzahl Käufe bei der **Bestellung** und beim **Artikel**

```
CREATE TABLE order (
    …
    MAP<INT, INT> article_counts
)

CREATE TABLE article (
    …
    sales COUNTER
)
```

Beide aktualisieren!

Cassandra 3 → Materialized Views

Zusammenfassung

- **Wide-Column-Datenmodell**, abgebildet auf Mengen/Listen/Maps
- Datenmodell bestimmt **mögliche Operationen**
- Hohe **Schreibleistung:** Commit Log, Memtables, SSTables
- Verteilung über **Consistent Hashing**
- **Tuneable Consistency**
- **Eventual Consistency:** Reparaturstrategien