Big Data
04
HBASE

Eric MSP Veith <veith@offis.de>

March 1, 2020

Organisatorisches

- Prüfungsleistung: Seminararbeit
  - 15 Seiten Ausarbeitung zum Thema: Arial o.ä., Schriftgröße 12pt, einfacher Zeilenabstand, 2cm Rand, keine separate Titelseite, Seitenzahl inkl. Literaturverzeichnis
  - 15 Minuten Vortrag zum Thema
- Seminarthemen & Folien online verfügbar:
  `https://vhome.offis.de/~eveith`

NoSQL

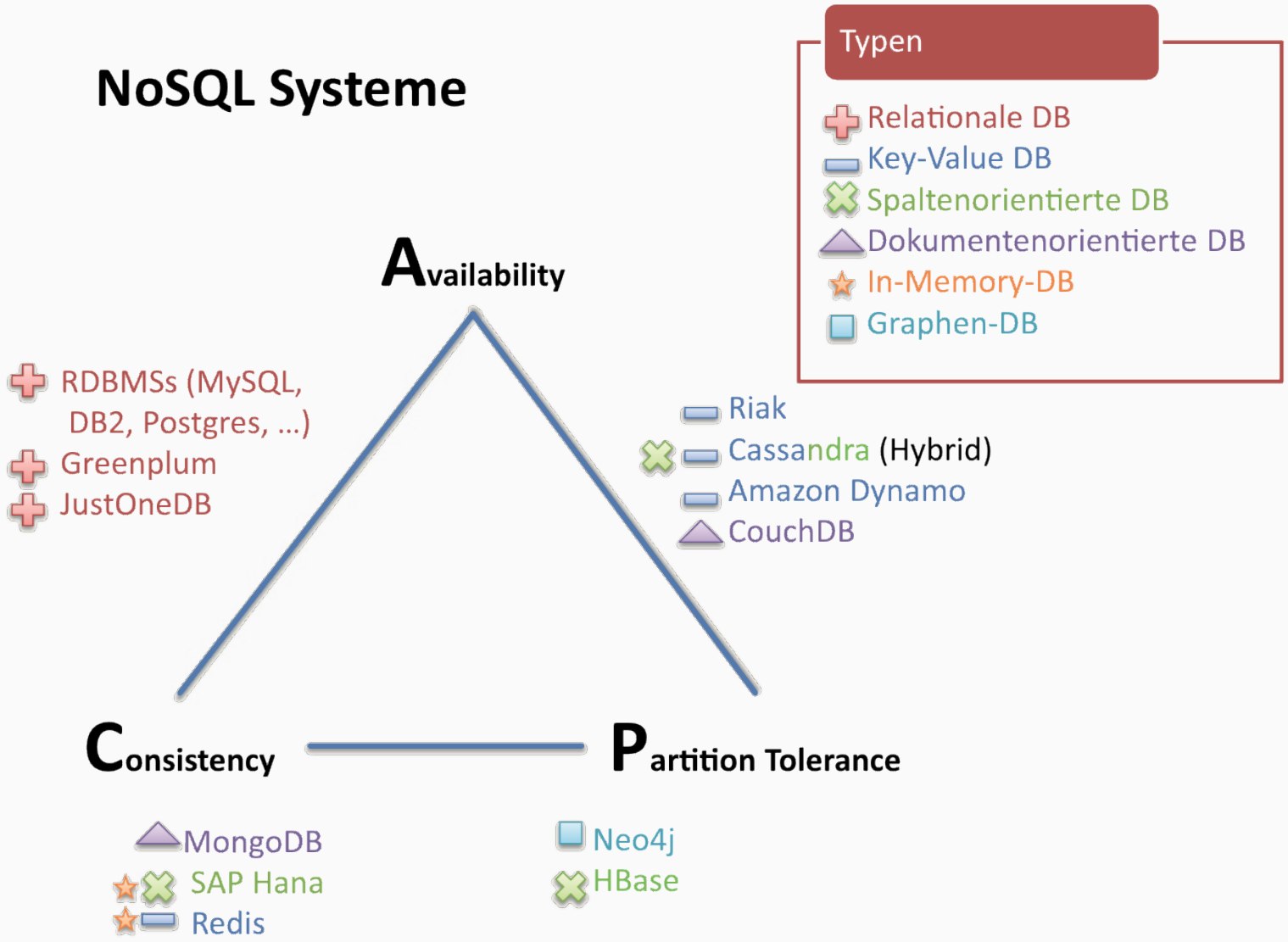
NoSQL: Noch ein Modewort...?

- NoSQL: **Not only SQL**
- Problem: große, un-/polystrukturierte Daten kaum mit einem Datenbankschema erfassbar
  - Starke Vorverarbeitung nötig
  - Polystrukturierung aufgeben – Informationen verlieren?
  - Datenstrukturen können sich leicht ändern – Schemata immer migrieren?
- Stattdessen effiziente Abfragen auf polystrukturierten Daten
  - Schemafreiheit
  - Horizontale Skalierbarkeit

Das CAP-Theorem

- Drei Eigenschaften verteilter Datenbanksysteme:
  1. **Konsistenz:** Selber Datenbestand auf allen Servern
  2. **Verfügbarkeit:** Knoten und darauf befindliche Daten sind jederzeit abrufbar
  3. **Partitionstoleranz:** System funktioniert auch nach Ausfall einiger Knoten noch
- . . . nicht vollständig vereinbar
- 2 von 3 Eigenschaften bei verteilten Datenbanksystemen zumeist voll erfüllt

NoSQL-Systeme nach CAP geordnet



Freiknecht, & Papp. „Big Data in der Praxis.“

Eigenschaften von Datenbanktransaktionen: ACID

Atomicity Transaktion eine Menge von Datenbankoperationen; wird entweder ganz oder gar nicht ausgefüllt (atomos – unzerteilbar); Rollback, falls Transaktion nicht erfolgreich beendbar

Consistency Integritätsbedingungen des Datenbankschemas gelten nach der Transaktion (und müssen vorher gelten)

Isolation Nebenläufige Transaktionen beeinflussen sich nicht gegenseitig (Level: `READ COMMITTED`, `REPEATABLE READ`, `SERIALIZABLE`)

Durability Daten sind nach erfolgreichem Abschluss der Transaktion garantiert dauerhaft gespeichert.

- Wichtig bei Web Stores, hinderlich bei Chat-Servern
- Abschwächung: **BASE** (*Basically Available*, *Soft state*, *Eventually consistent*)

Typen von Datenbanken i

- **Relationale Datenbank**
  - Tabellen (zweidimensional)
  - Festes Schema mit Datentyp pro Spalte
  - Contraints/Trigger
  - Abfragesprache: SQL
- **Key-Value-Datenbank**
  - Schlüssel-Werte-Paare
  - Schlüssel eindeutig, z.B. Hash
  - Datensatz (*Value*) nicht zwangsläufig typisiert
- **Spaltenorientierte Datenbank**
  - Ablegen nach Spalten statt nach Zeilen
  - Ein Datensatz kann beliebig viele Spalten haben
  - Reduziert I/O-Last bei Aggregationen

Typen von Datenbanken ii

- **Dokumentenorientierte Datenbank**
  - Dokumente statt Tabellen
  - Keine vorgegebene Struktur von Dokumenten, z.B. JSON, PDF-Datei, Bilder, . . . in derselben Datenbank
  - Schlüssel zur Identifikation

```
{
  name: "sue",                          ← field: value
  age: 26,                              ← field: value
  status: "A",                          ← field: value
  groups: [ "news", "sports" ]          ← field: value
}
```

Typen von Datenbanken iii

- **Graphenorientierte Datenbank**
  - Verknüpfung von Entitäten über Kanten
  - Kanten sind annotiert – definiert die Beziehung

HBASE

Historie & Herkunft

- HBASE: Hadoop Database
- Erster Commit 2007
- basiert auf dem *BigTable*-Paper von Google

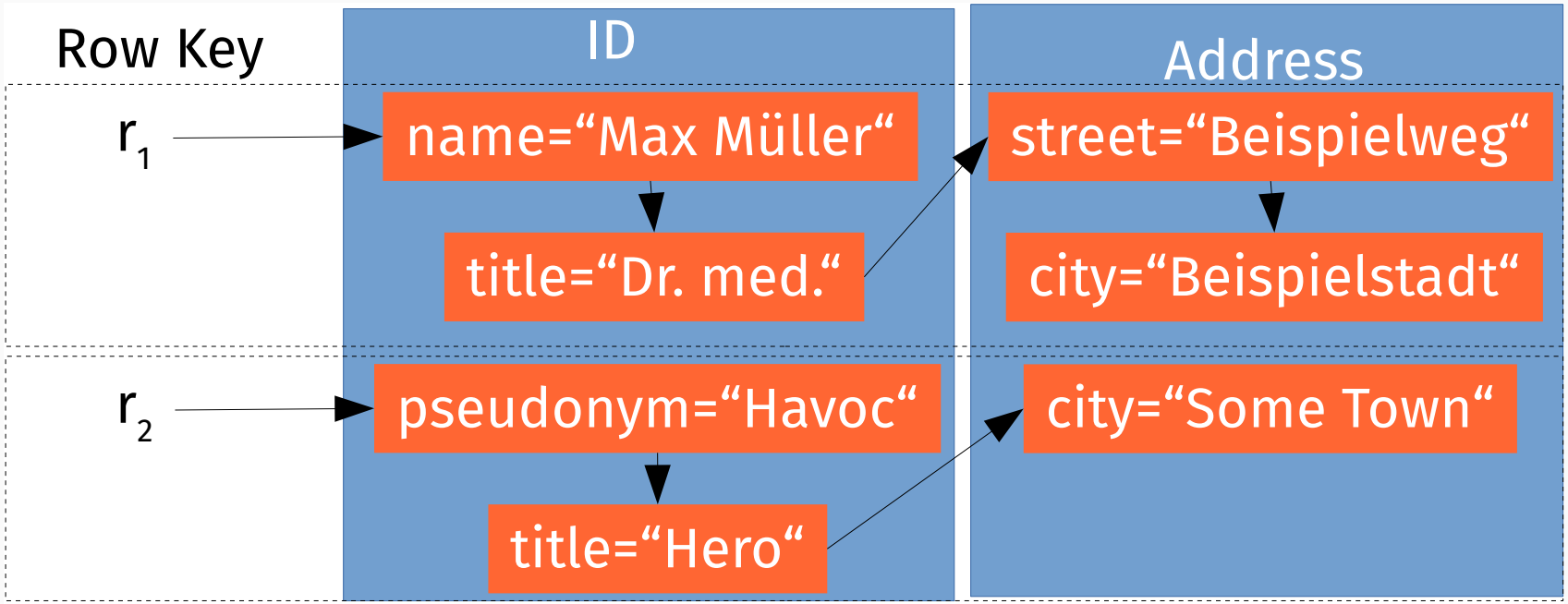
Datenmodell von HBASE

- Zeilenschlüssel immer vorhanden
  - Vom Benutzer vorgegeben
  - Alphabetische Sortierung
  - Kein autoinkrement möglich
  - Gefahr: **Hot Spotting** (z. B. bei führender Null)
- Beliebig viele Spalten pro Zeile (**Spaltenorientierte Datenbank**)
- Spalten nach *Column Families* gruppiert
- Intern als verkettete Liste implementiert
- Vorteil:
  - Leere Spalten verbrauchen keinen Speicherplatz
  - Speicherung je nach vorliegenden Informationen (Beispiel: Adressdatenbank)

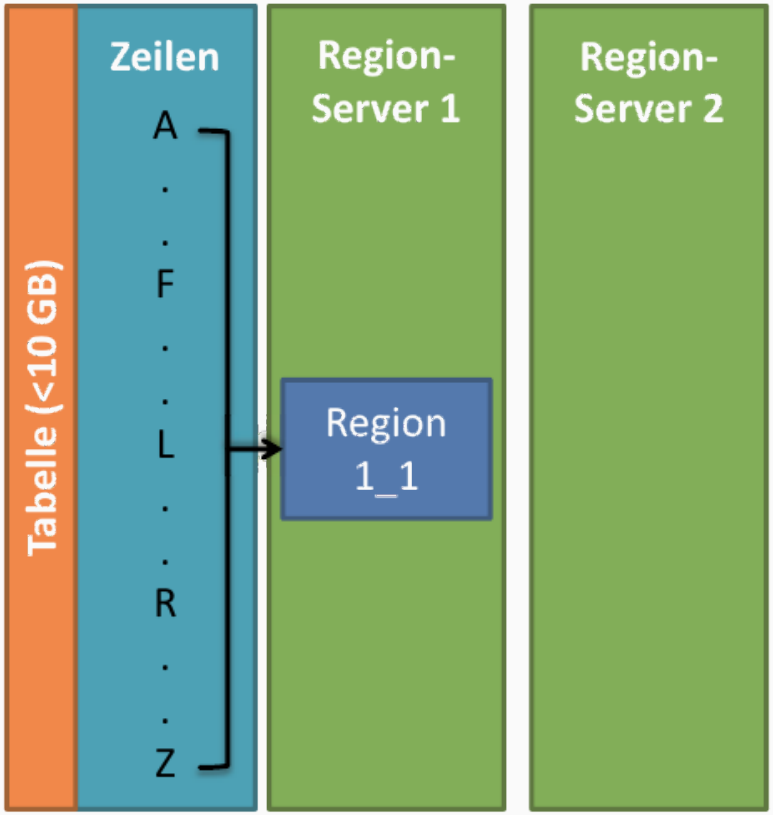
Datenmodell von HBASE – Ein Beispiel

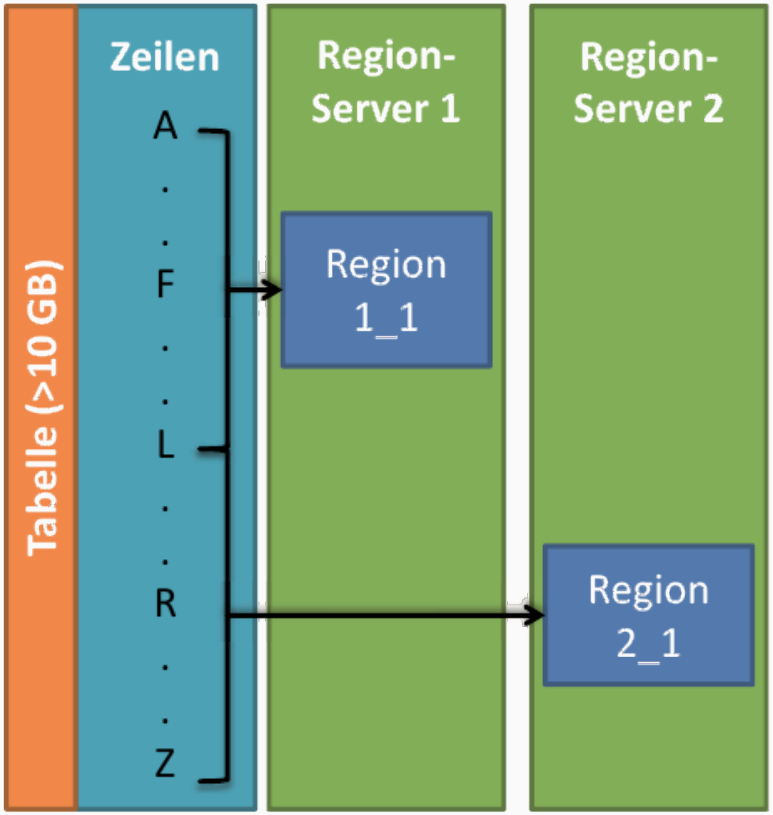
Schlüssel	Zeitstempel	Column Family **ID**	Column Family **Address**
r_1	1582979740	**ID:** name="Max Müller"	**Address:** street="Beispielweg"
r_2	1582979745	**ID:** pseudonym="Havoc"	**Address:** city="Some Town"
r_3	1582979751	**ID:** name="Jane Doe"	**Address:** country = "Washington, DC, USA"
r_4	1582979751	**ID:** id="0xdeadbeef"	**Address:** pobox="2341"

Verkettete Liste in HBASE



Regionen und Splitting





Schlüssel in HBASE

- Schlüssel in HBASE alphabetisch sortiert
- Schlüssel legen Regionen fest – Gefahr von **Hot Spotting**
- Alternativen:
  - Checksums (SHA-512, . . .)
  - Zufallszahlen
  - *Counter Table*: Tabelle mit Zählvariablen für verschiedene Domänen
  - Zeitstempel + eindeutiger Bezeichner (z.B. Name, Adresse, etc.)
  - Kombination der obigen

Umgang mit HBASE

Konsole:

```
% hbase shell
> create 'people', 'id', 'address'
> put 'people', 'r1', 'id:name', 'Max Müller'
> put 'people', 'r1', 'address:street', 'Beispielweg'
> ...
> get 'people', 'r1'
COLUMN    CELL
id:name         timestamp=1582979740, value="Max Müller"
address:street timestamp=1582979740, value="Beispielweg"
    ...
```

HBASE mit Python

```
import happybase

connection = happybase.Connection('hostname')
table = connection.table('people')

table.put(b'r1', {b'id:name': b'Max Müller',
                  b'address:street': b'Beispielweg'})

row = table.row(b'r1')
print(row[b'address:street'])  # => Beispielweg

for key, data in table.rows([b'r1', b'rr2']):
    print(key, data)

for key, data in table.scan(row_prefix=b'r1'):
    print(key, data)

row = table.delete(b'r3')
```