

# Tag 7

## Inhaltsverzeichnis

- SQL-Optimierungen
  - Was macht der Optimizer?
  - Der Abfrage Cache...
  - MySQL Ausführungspläne vergleichen
- RDB vs. NoSQL
- BigData Ausblick
  - Die Begriffe
  - Das SBB DMD-Beispiel
  - Das Hadoop Ecosystem

# SQL-Optimierungen

## Problematik

- RDB Tag 6: Performance auf Betriebssystem-, DB-Index- und Applikationsebene
- Jetzt: *Fokus auf SQL*
- Problem 1: SQL sagt "was" aber nicht "wie"...
- Problem 2: RDB-spezifisch => MySQL != Oracle

# SQL-Optimierungen

## Datentypen

- Wenn möglich, NOT NULL-Felder definieren  
(NULL sind schwerer zu optimieren, indexieren)
- MySQL-Breite für Integer-Typen (INT(1) gleich performant wie INT(20))  
Für interaktive Werkzeuge (z.B. MySQL Command-Interpreter)
- DECIMAL: mehr Speicher und langsamer als FLOAT und DOUBLE
- CHAR: gut für kleine Strings  
VARCHAR: "richtig dimensionieren", wegen Sortieren
- DATETIME vs. TIMESTAMP... nicht gleichen Anwendungszweck  
(DATETIME braucht mehr Speicher, weniger begrenzt, siehe RDB Tag 3)
- Achtung: Automatisch-generierte Schemata (z.B. ORM) überprüfen

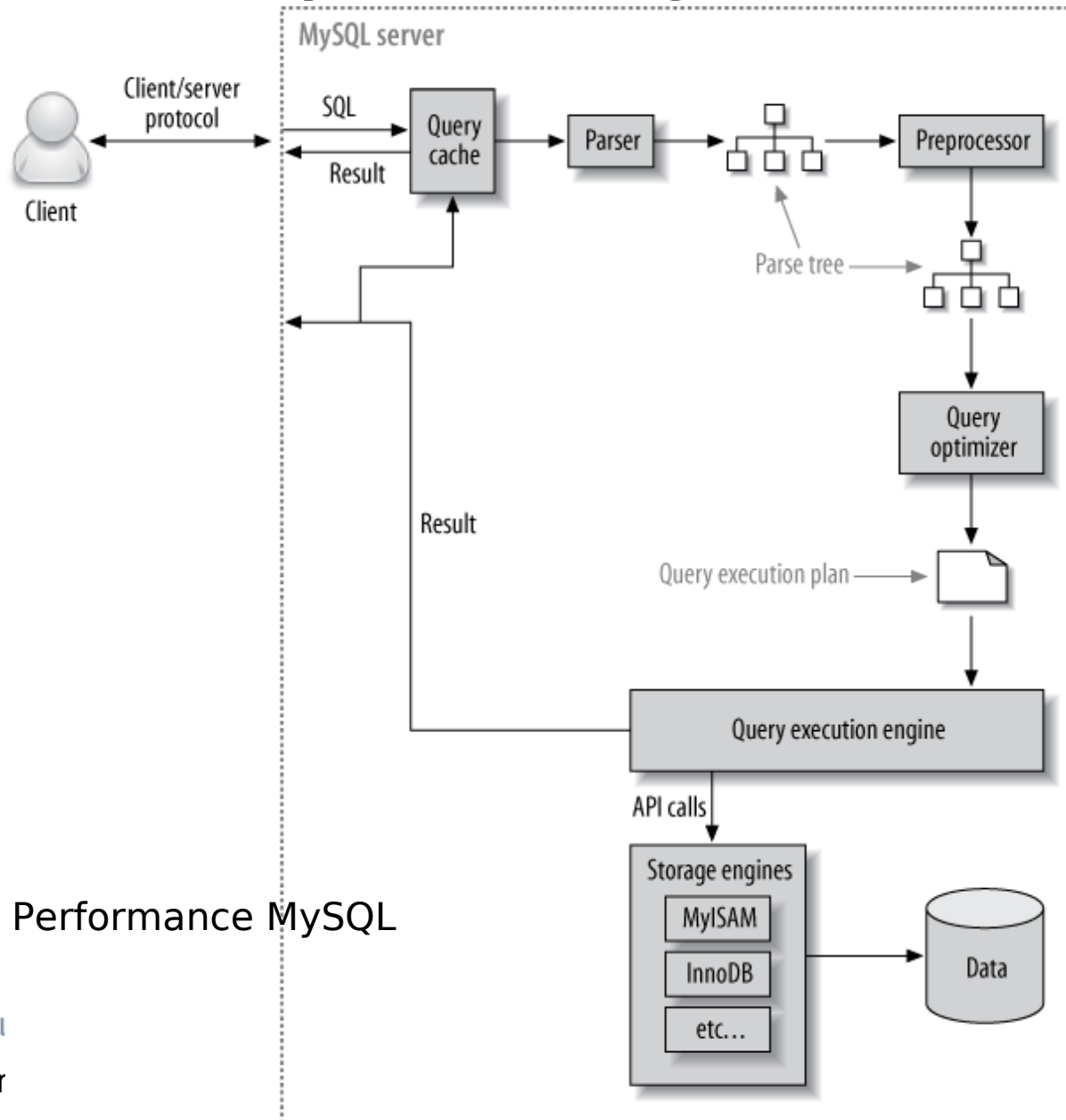
# SQL-Optimierungen

## Index- und Datenfragmentierung

- **Probleme**
  - Tabellen können beschädigt sein (wegen Crash)
  - B-Bäume Indexe können fragmentiert sein
  - InnoDB Index-Benutzung basiert auf Statistiken (diese sind vielleicht nicht mehr aktuell)
- **==> Tabellen und Indexes (Cardinality) prüfen**
- **Befehle:**
  - `analyse table <table>`
  - `show index from <table>`
  - `optimize table <table> oder`
  - `alter table <table> engine=innodb`

# SQL-Optimierungen

## MySQL Abfragebearbeitung



Quelle: High Performance MySQL

# SQL-Optimierungen

## MySQL Client/Server Protokoll

- Halb-Duplex Protokoll  
==  
Entweder Client sendet dem Server etwas, oder das Gegenteil
- <=> Keine Kommunikation "gleichzeitig" zwischen Client und Server
- ==> Keine Möglichkeit, eine Abfrage *abzubrechen*...  
Man muss warten **bis alle Daten fertig übertragen** werden!

# SQL-Optimierungen

## MySQL Abfrage Cache

- Hash-Tabelle (Abfrage / Resultatsdaten) (Case-sensitive)

Abfrage	Resultatsdaten
select * from ...	...
select ID, XY from ...	...

- Nicht determinische Abfragen werden *nicht* gespeichert (Abfragen mit now() oder current\_user())
- Problem mit Transaktionen: Cache-Lookup erst möglich, wenn die Transaktion abgeschlossen ist
- **"Deprecated" in Version 5.7. Gelöscht ab V8.0.**

Quelle: Achtung, "High Performance MySQL" ist (zu) alt diesbezüglich

# SQL-Optimierungen

## Oracle / MySQL Abfrage Optimierer

- Oracle 10g zwei Optimierer (alt und neu):
  - Rule Based Optimizer (RBO): Regel-basiert (alt)  
<=> Basiert auf fixen Eigenschaften von Tabellen (Index, Datentypen)  
unabhängig von der Entwicklung der Daten
  - Cost Based Optimizer (CBO): Kosten-basiert (neu)  
<=> Basiert auf Tabellen-Statistiken (periodisch aktualisiert)  
abhängig von der Entwicklung der Daten
- MySQL Optimierer ist "Kosten-basiert"  
=> Statistik **müssen** stimmen (siehe Slide Fragmentierung)
- MySQL Optimierer *unabhängig* vom Engine (siehe Slide Abfragebearbeitung)



# SQL-Optimierungen

## MySQL Nested-Loop-Join's

- Alle JOIN's im MySQL als *Nested-Loop-Join's* betrachtet

```
select ... from A a join B b using(id);
```

```
foreach a in A {  
    foreach b in B {  
        if a.id == b.id => getroffen...  
    }  
}
```

- Es gibt aber andere Möglichkeiten zu "joinen"

# SQL-Optimierungen

## MySQL Ausführungspläne vergleichen (1)

- Beispiel: Wie viele Musiker in GMCD spielen Gitarre?

```
SELECT count(*)
FROM Musiker m,
     MusikerInstrument mi,
     Instrument i
WHERE i.Name = 'Gitarre'      AND
      i.ID = mi.InstrumentID AND
      m.ID = mi.MusikerID;
```

- Antwort: 2

```
SHOW STATUS LIKE 'Last_query_cost';
```

```
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Last_query_cost | 4.699248 |
+-----+-----+
```

# SQL-Optimierungen

## MySQL Ausführungspläne vergleichen (2)

- Gleiches Beispiel wie vorher nochmals ausführen:

```
• SELECT count(*)
  FROM Musiker m,
       MusikerInstrument mi,
       Instrument i
 WHERE i.Name = 'Gitarre'      AND
       i.ID = mi.InstrumentID AND
       m.ID = mi.MusikerID;
```

- Antwort: 2

```
• SHOW STATUS LIKE 'Last_query_cost';
```

```
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Last_query_cost | 2.449062 |
+-----+-----+
```

Wieso dieser Unterschied?

# SQL-Optimierungen

## MySQL Ausführungspläne vergleichen (3)

- Wir probieren noch einmal, aber *ohne Cache* diesmal:
- ```
SELECT SQL_NO_CACHE count(*)
FROM Musiker m,
      MusikerInstrument mi,
      Instrument i
WHERE i.Name = 'Gitarre'      AND
      i.ID = mi.InstrumentID AND
      m.ID = mi.MusikerID;
```
- Antwort: 2
- Last\_query\_cost: **2.449062**

Klar, der Cache gibt es offiziell nicht mehr. **Aber:**  
1) "SQL\_NO\_CACHE" wirft keine Fehlermeldung.  
2) Irgendwo ist doch ein Cache vorhanden, versteckt.  
Players: MySQL, Netzwerk, Betriebssystem selber...

# SQL-Optimierungen

## MySQL Ausführungspläne vergleichen (4)

- Wie wird überhaupt die Abfrage ausgeführt?
- **EXPLAIN** SELECT count(\*)  
FROM Musiker m,  
MusikerInstrument mi,  
Instrument i  
WHERE i.Name = 'Gitarre' AND  
i.ID = mi.InstrumentID AND  
m.ID = mi.MusikerID;

| table | type   | rows | Extra                                                          |
|-------|--------|------|----------------------------------------------------------------|
| i     | ALL    | 6    | Using where                                                    |
| mi    | Index  | 10   | Using where; Using index; Using join buffer(Block Nested Loop) |
| m     | eq_ref | 1    | Using index                                                    |

(Abgekürzte Darstellung)

Last\_query\_cost: 2.449062

# SQL-Optimierungen

## MySQL Ausführungspläne vergleichen (6)

- Wir ändern die Reihenfolge der Bedingungen in der Abfrage

- ```
EXPLAIN SELECT count(*)
FROM Musiker m,
      MusikerInstrument mi,
      Instrument i
WHERE m.ID = mi.MusikerID      AND
      i.ID = mi.InstrumentID AND
      i.Name = 'Gitarre';
```

table	type	rows	Extra
i	ALL	6	Using where
mi	Index	10	Using where; Using index; Using join buffer(Block Nested Loop)
m	eq_ref	1	Using index

(Abgekürzte Darstellung)

Last\_query\_cost: 2.449062

**Überhaupt keine Änderung im Plan... Wieso?**

# SQL-Optimierungen

## MySQL Ausführungspläne vergleichen (7)

- Wir ändern die Reihenfolge der Bedingungen in der Abfrage *und schalten den JOIN-Optimierer aus...*

- ```
EXPLAIN SELECT STRAIGHT_JOIN count(*)
FROM Musiker m,
      MusikerInstrument mi,
      Instrument i
WHERE m.ID = mi.MusikerID      AND
      i.ID = mi.InstrumentID AND
      i.Name = 'Gitarre';
```

| table | type  | rows | Extra                                                          |
|-------|-------|------|----------------------------------------------------------------|
| m     | Index | 6    | Using index                                                    |
| mi    | ref   | 1    | Using index                                                    |
| i     | ALL   | 6    | Using where; Using index; Using join buffer(Block Nested Loop) |

(Abgekürzte Darstellung)

Last\_query\_cost: 5.101293 => Faktor 2

# SQL-Optimierungen

## MySQL Ausführungspläne vergleichen (8)

- Wir ändern die Reihenfolge der Bedingungen in der Abfrage, schalten den Optimierer aus und *schreiben einen richtigen JOIN*
- ```
EXPLAIN SELECT STRAIGHT_JOIN count(*)  
FROM Musiker m  
INNER JOIN MusikerInstrument mi ON m.ID = mi.MusikerID  
INNER JOIN Instrument i ON i.ID = mi.InstrumentID  
WHERE i.Name = 'Gitarre';
```

table	type	rows	Extra
m	Index	6	Using index
mi	ref	1	Using index
i	ALL	6	Using where; Using index; Using join buffer(Block Nested Loop)

(Abgekürzte Darstellung)

Last\_query\_cost: 5.101293

**Überhaupt keine Änderung im Plan...**



# SQL-Optimierungen

## MySQL Ausführungspläne vergleichen (9)

- Wir ändern die Reihenfolge der **Tabellen** in der Abfrage, schalten den Optimierer aus
- ```
EXPLAIN SELECT STRAIGHT_JOIN count(*)  
FROM MusikerInstrument mi, Instrument i, Musiker m  
WHERE i.Name = 'Gitarre' AND  
      m.ID = mi.MusikerID AND  
      i.ID = mi.InstrumentID;
```

| table | type   | rows | Extra                                                          |
|-------|--------|------|----------------------------------------------------------------|
| mi    | Index  | 10   | Using index                                                    |
| i     | ALL    | 6    | Using where; Using index; Using join buffer(Block Nested Loop) |
| m     | eq_ref | 1    | Using index                                                    |

(Abgekürzte Darstellung)

Last\_query\_cost: 15.499458 => Faktor 5

### Grosse Änderung im Plan...

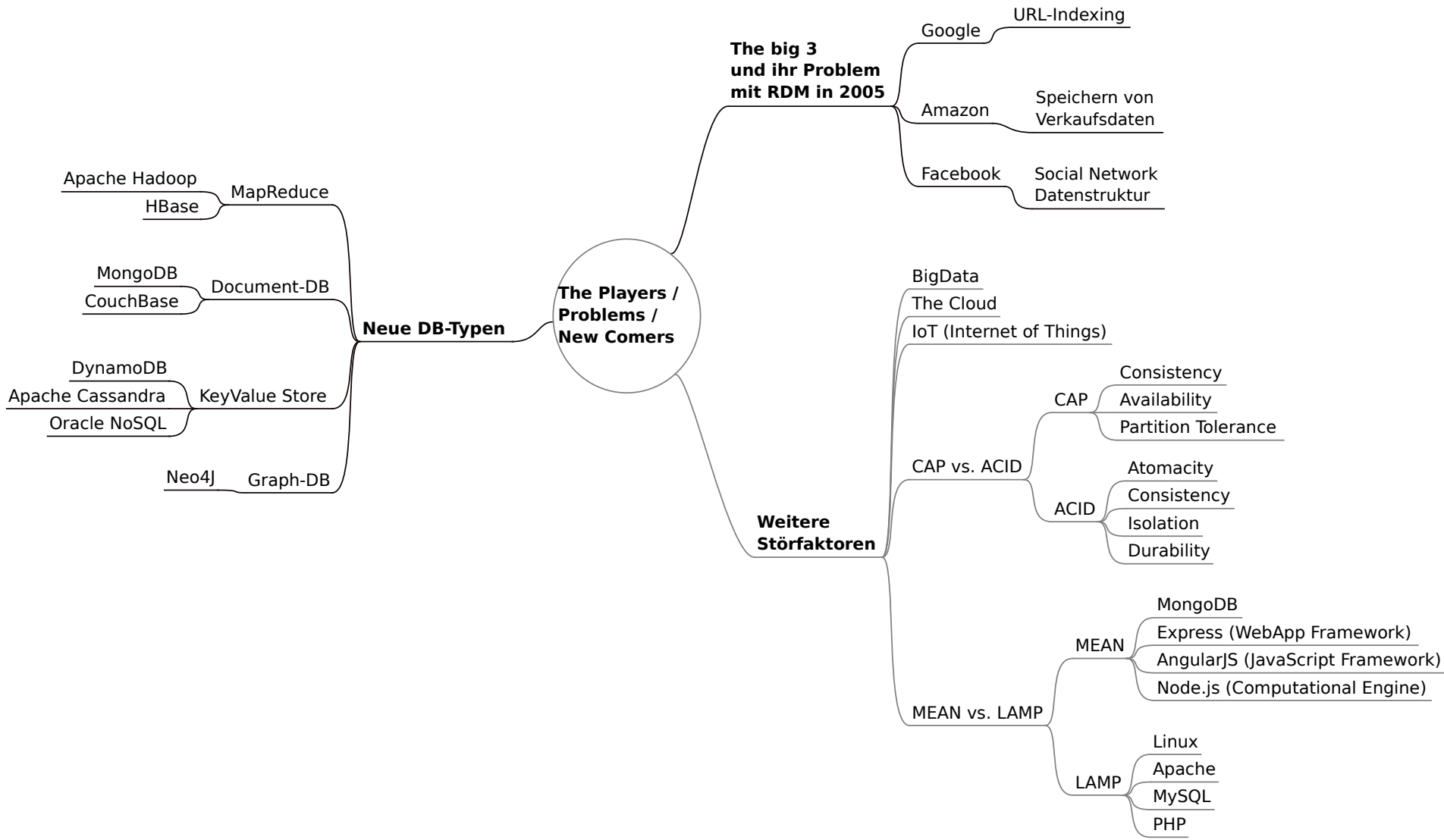
# SQL-Optimierungen

## MySQL Ausführungspläne, Zusammenfassung

- **SQL\_NO\_CACHE**
  - Hat ab V8.0 keine Wirkung mehr.
  - Wird nach wie vor akzeptiert.
  - Ein "verstecktes" Cache gibt es immer noch.
- **STRAIGHT\_JOIN**
  - Bis "Faktor 5"-Unterschied im Beispiel.
  - => Optimizer nur nach Bedarf ausschalten!
- **Reihenfolge der Bedingungen**
  - Spielt keine Rolle.
- **INNER JOIN vs. "Join von Hand"**
  - Spielt keine Rolle.
- **Im Zweifelsfall**
  - Selber prüfen.
  - **RTFM** ("excuse my french"...).

# RDB vs. NoSQL

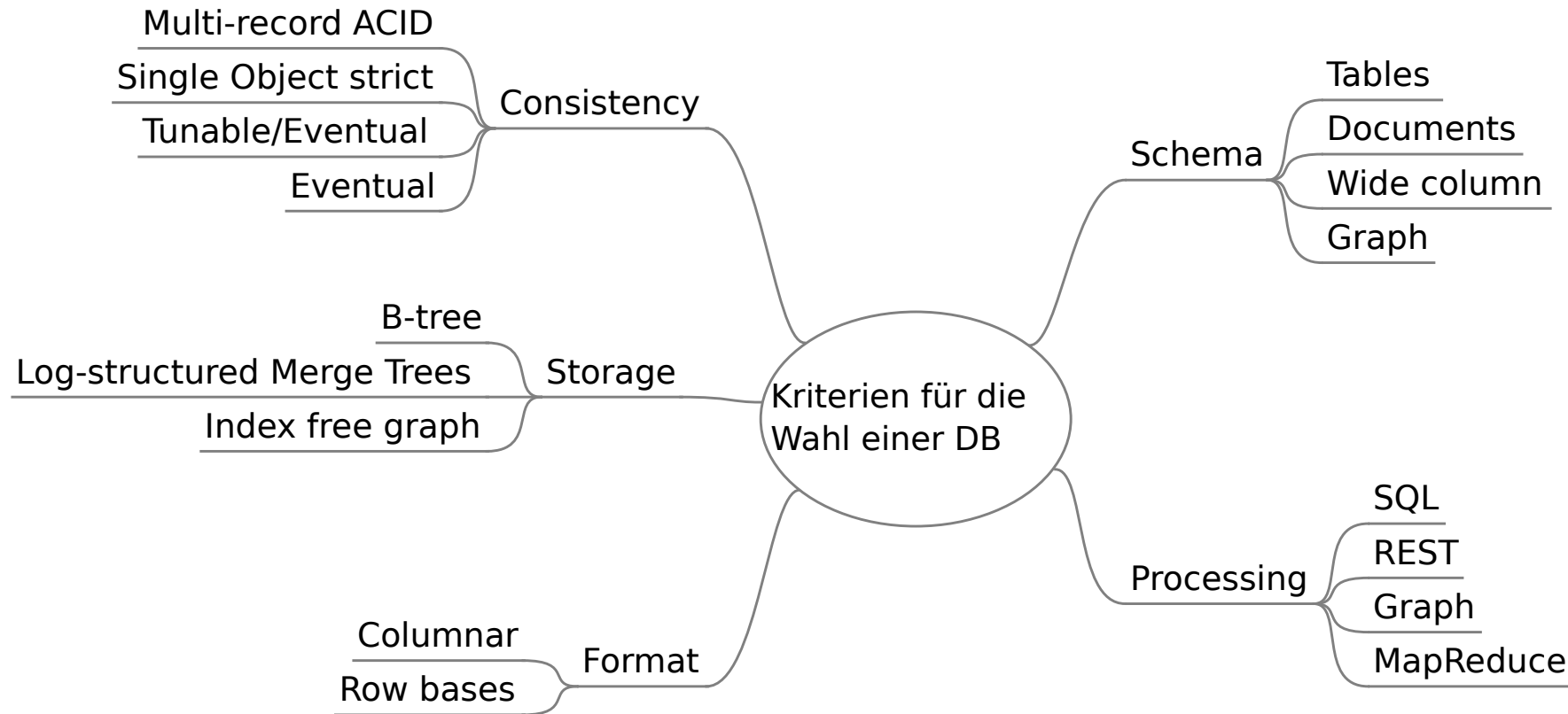
## Die Problematik (1)



Quelle: Next Generation Databases, S. 15-73

# RDB vs. NoSQL

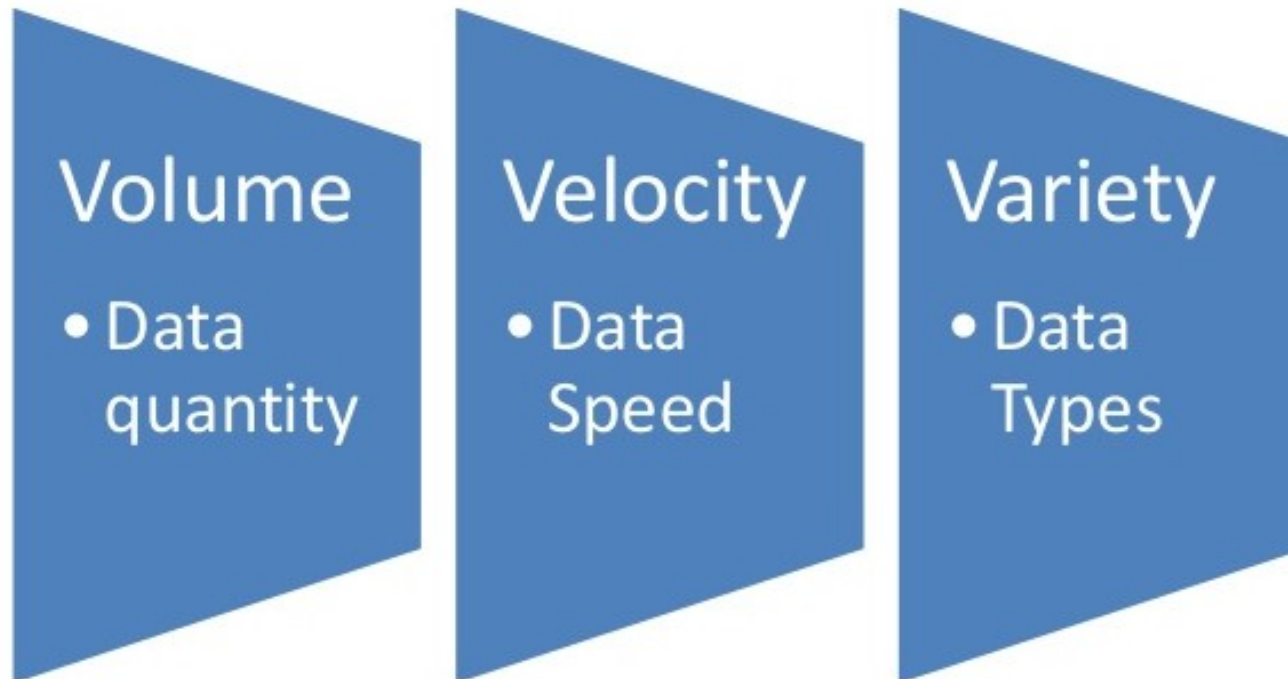
## Wahl einer DB (2)



# BigData Ausblick (1)

## Die Begriffe

### Three Characteristics of Big Data V3s



# BigData Ausblick (2)

## Die Begriffe

### Eigenschaften

#### Unstrukturierte Daten

- Dokumente
- Blogs, Chats, Text
- Videos, Bilder

#### Maschinell generierte Daten

- System Logs
- Sensordaten
- Automatisch erzeugt
- Streaming-Daten



- Terabyte, Petabyte, Exabyte, Zettabyte

- Datenlatenz
- Schnelle Verarbeitung
- Response-Zeiten
- Reaktionszeit
- Von Batch bis Streaming

**Big  
Data**

### Treiber

#### Operative Technologien

- Intelligente Systeme

#### Informationstechnologie

- Distributed Parallel Processing
- Hochskalierbare standardbasierte Hardware
- Open-Source-Analytic-Plattformen

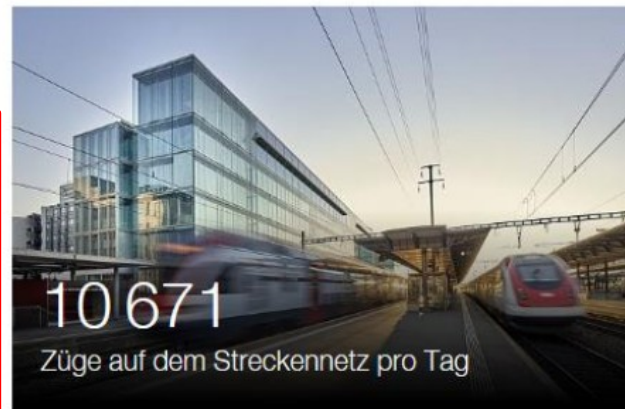


- Prognosen
- Modellierung & Visualisierung
- Wettbewerbsfähigkeit/ Differenzierung
- Exploration
- Volatilität

- Preis-Leistungs-Verhältnis der Analytics-Lösungen

# BigData Ausblick (3)

## Das SBB DMD Beispiel



# BigData Ausblick (4) Das SBB DMD-Beispiel



Diagnosefahrzeug der SBB

Messdaten  
und Bilder

Daten upload  
und transformation  
(DMD)



Preventive  
Maintenance



Wartung der  
Infrastruktur



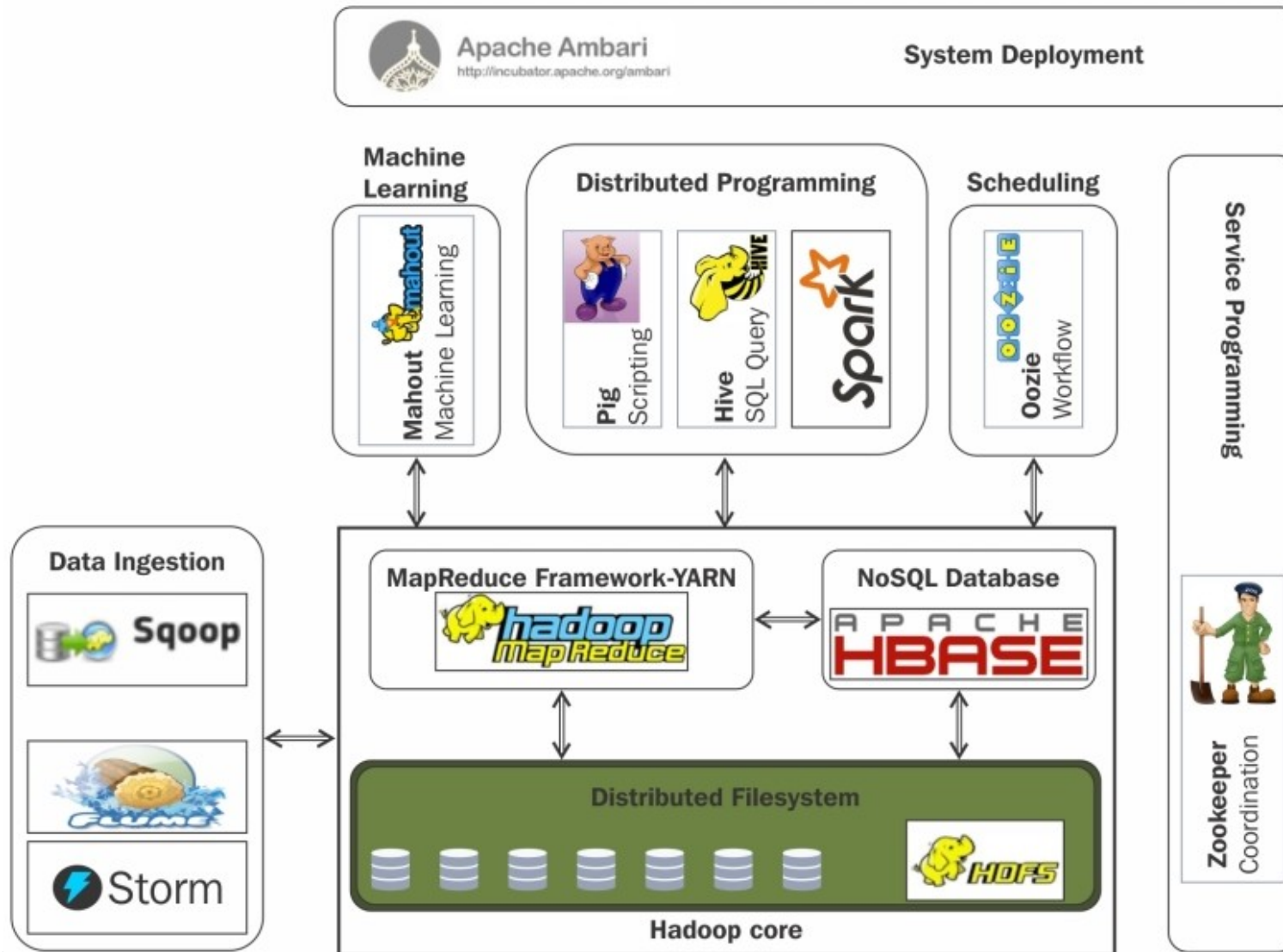
Heute: 32 TB  
Morgen: 700 TB

RDB 7 - 24  
Version 3.10



# BigData Ausblick (5)

## Das Hadoop Ecosystem



# BigData Ausblick (5)

## Die Hadoop und YARN Architektur

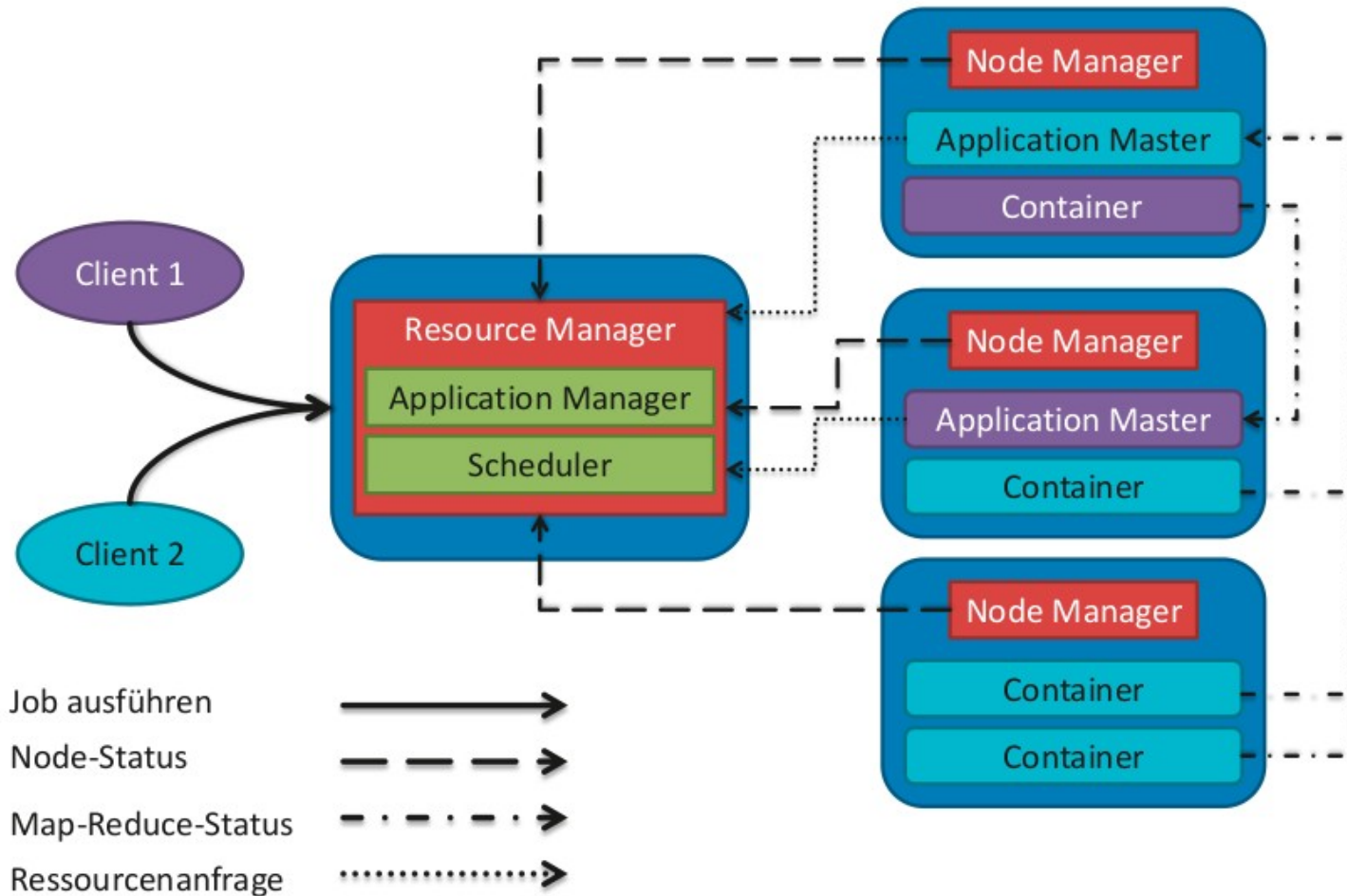
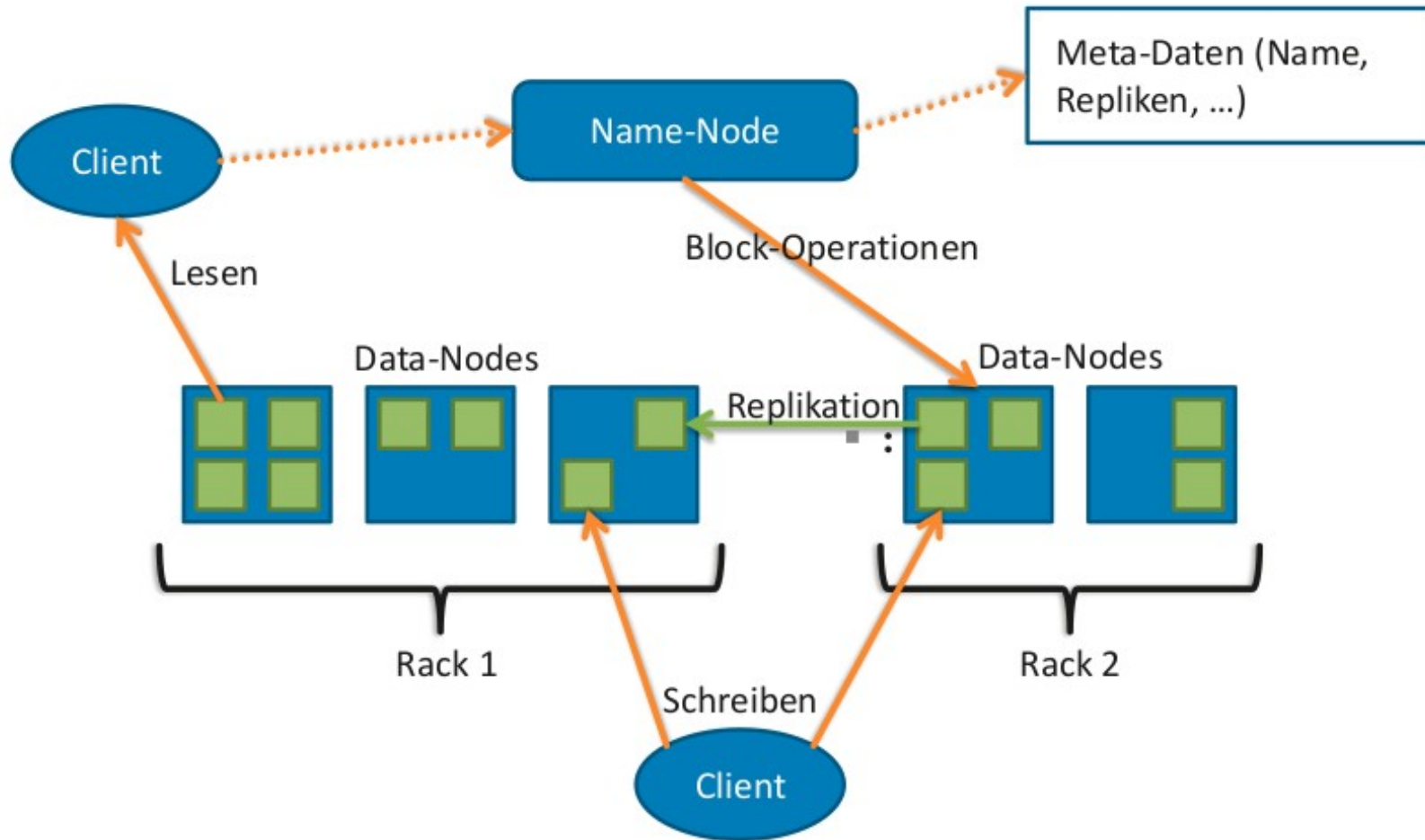


Bild 3.2 Architekturübersicht zu YARN

Yet Another Resource Negotiator

# BigData Ausblick (5) Das HDFS



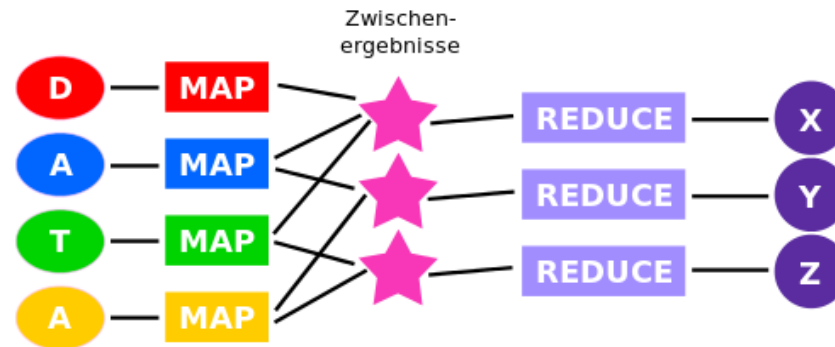
**Bild 3.1** Architektur und Funktionsweise des HDFS

Hadoop Distributed File System

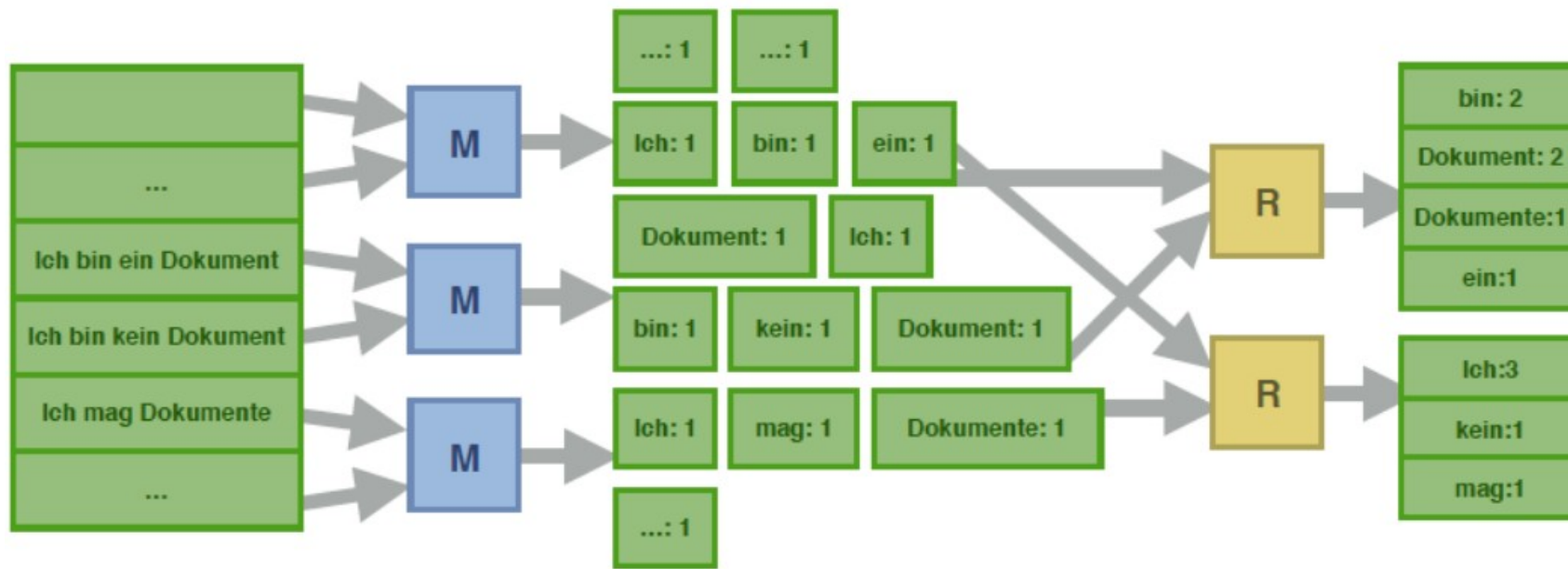
# BigData Ausblick (5)

## Map / Reduce

Illustration des Datenflusses [ Bearbeiten | Quelltext bearbeiten ]



Nur für Batch!  
Kein Streaming



Eingabe

Zwischenergebnisse

Ergebnis



Berner  
Fachhochschule

Gilles Maitre

Quelle: <https://de.wikipedia.org/wiki/MapReduce> + SBB intern

RDB 7 - 28  
Version 3.10

# Übungen

1) Probieren Sie diese SQL-Optimierungsbeispiele zu reproduzieren und vor allem sie zu verstehen.

Parameter, die eine Rolle spielen:

1) SQL JOIN oder nicht

2) JOIN-Reihenfolge forciert oder nicht

"EXPLAIN SELECT" und "last\_query\_cost" ausprobieren

2) Probieren Sie den Ausführungsplan einer Abfrage aus Ihrer eigenen Fantasie/Erfahrung abzubilden, zu verstehen und am Schluss von Hand zu optimieren.