Tag 7
Inhaltsverzeichnis

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

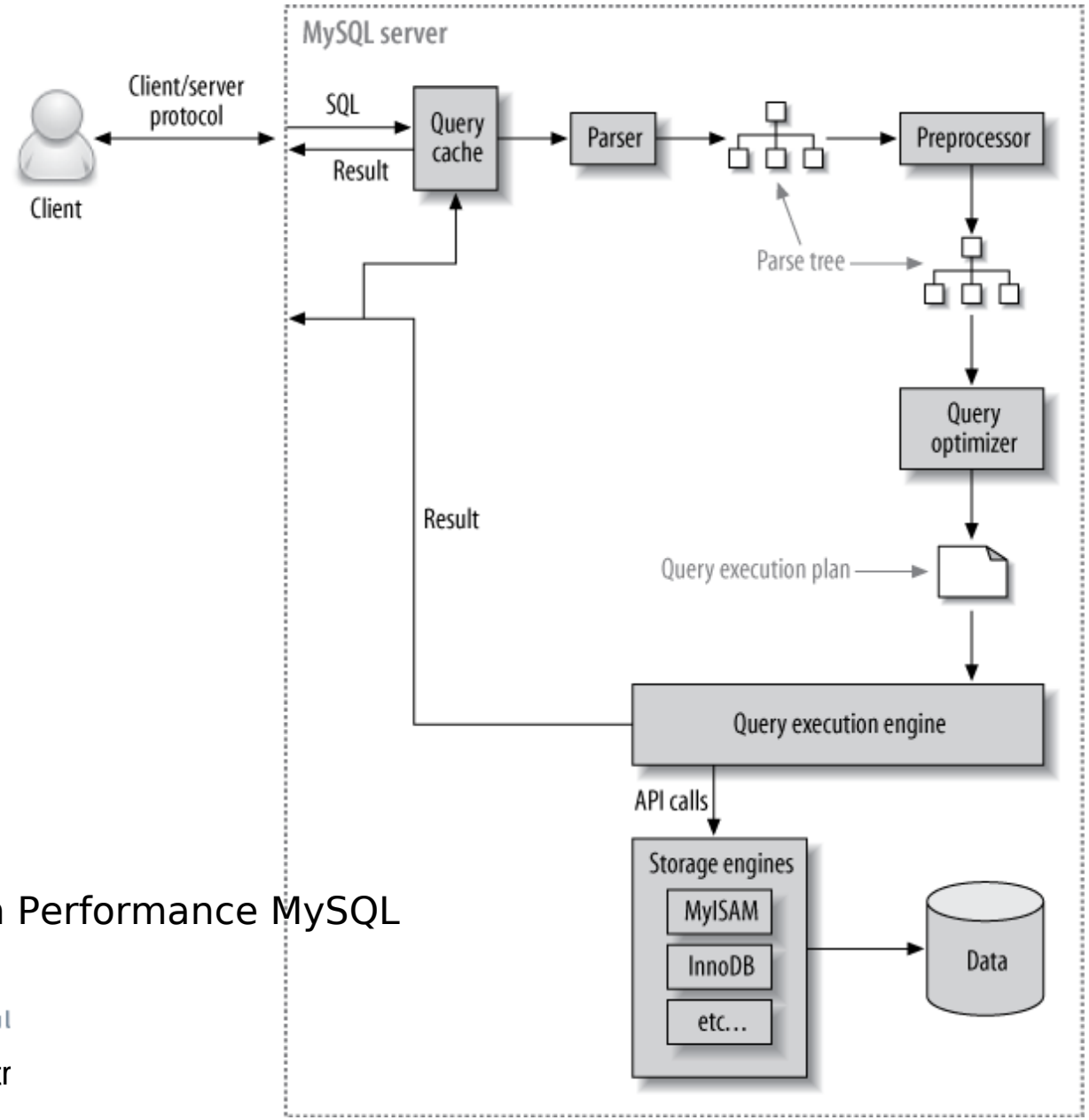
SQL-Optimierungen
Index- und Datenfragmentierung

- **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,
       musiker_instrument mi,
       instrument i
  where i.name = 'Gitarre'       and
        i.id = mi.instrument_id and
        m.id = mi.musiker_id;
  ```

- Antwort: 2

- ```
  show status like 'Last_query_cost';
  +------------------+----------+
  | Variable_name    | Value    |
  +------------------+----------+
  | Last_query_cost | 4.699248 |
  +------------------+----------+
  ```


Quelle: Last_query_cost explained

SQL-Optimierungen
MySQL Ausführungspläne vergleichen (2)

- Gleiches Beispiel wie vorher nochmals ausführen:

- ```
  select count(*)
  from musiker m,
       musiker_instrument mi,
       instrument i
  where i.name = 'Gitarre'       and
        i.id = mi.instrument_id and
        m.id = mi.musiker_id;
  ```

- 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,
       musiker_instrument mi,
       instrument i
  where i.name = 'Gitarre'       and
        i.id = mi.instrument_id and
        m.id = mi.musiker_id;
  ```

- 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,
     musiker_instrument mi,
     instrument i
where i.name = 'Gitarre'       and
      i.id = mi.instrument_id and
      m.id = mi.musiker_id;
```

```
+------+--------+------+-----------------------------------------------+
| table | type   | rows | Extra                                         |
+------+--------+------+-----------------------------------------------+
| i     | ALL    |    6 | Using where                                   |
| mi    | Index  |   10 | Using where; Using index; Using join buffer(hash join)|
| 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,
       musiker_instrument mi,
       instrument i
  where m.id = mi.musiker_id     and
        i.id = mi.instrument_id and
        i.Name = 'Gitarre';
  ```

```
+------+--------+------+-----------------------------------------------+
| table | type   | rows | Extra                                         |
+------+--------+------+-----------------------------------------------+
| i     | ALL    |    6 | Using where                                   |
| mi    | Index  |   10 | Using where; Using index; Using join buffer(hash join)|
| 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,
     musiker_instrument mi,
     instrument i
where m.id = mi.musiker_id    and
      i.id = mi.instrument_id 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(hash join)
+-------+-------------------------------------------------+
(Abgekürzte Darstellung)
```

Last_query_cost: 5.101293 => Faktor 2 langsamer, wegen "i"

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 musiker_instrument mi on m.id = mi.musiker_id
  inner join instrument i on i.id = mi.instrument_id
  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(hash
join)
+-------+-----------------------------------------------------+
(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 musiker_instrument mi, instrument i, musiker m
  where i.Name = 'Gitarre'  and
        m.id = mi.musiker_id and
        i.id = mi.instrument_id;
  ```

```
+-------+--------+------+-----------------------------------------------+
| table | type   | rows | Extra
|
+-------+--------+------+-----------------------------------------------+
| mi    | Index  |   10 | Using index
| i     | ALL    |    6 | Using where; Using index; Using join buffer(hash join|
| m     | eq_ref |    1 | Using index
+-------+--------+------+-----------------------------------------------+
(Abgekürzte Darstellung)
```

Last_query_cost: 6.499229 => Früher war es ein Faktor 5... jetzt weniger

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.

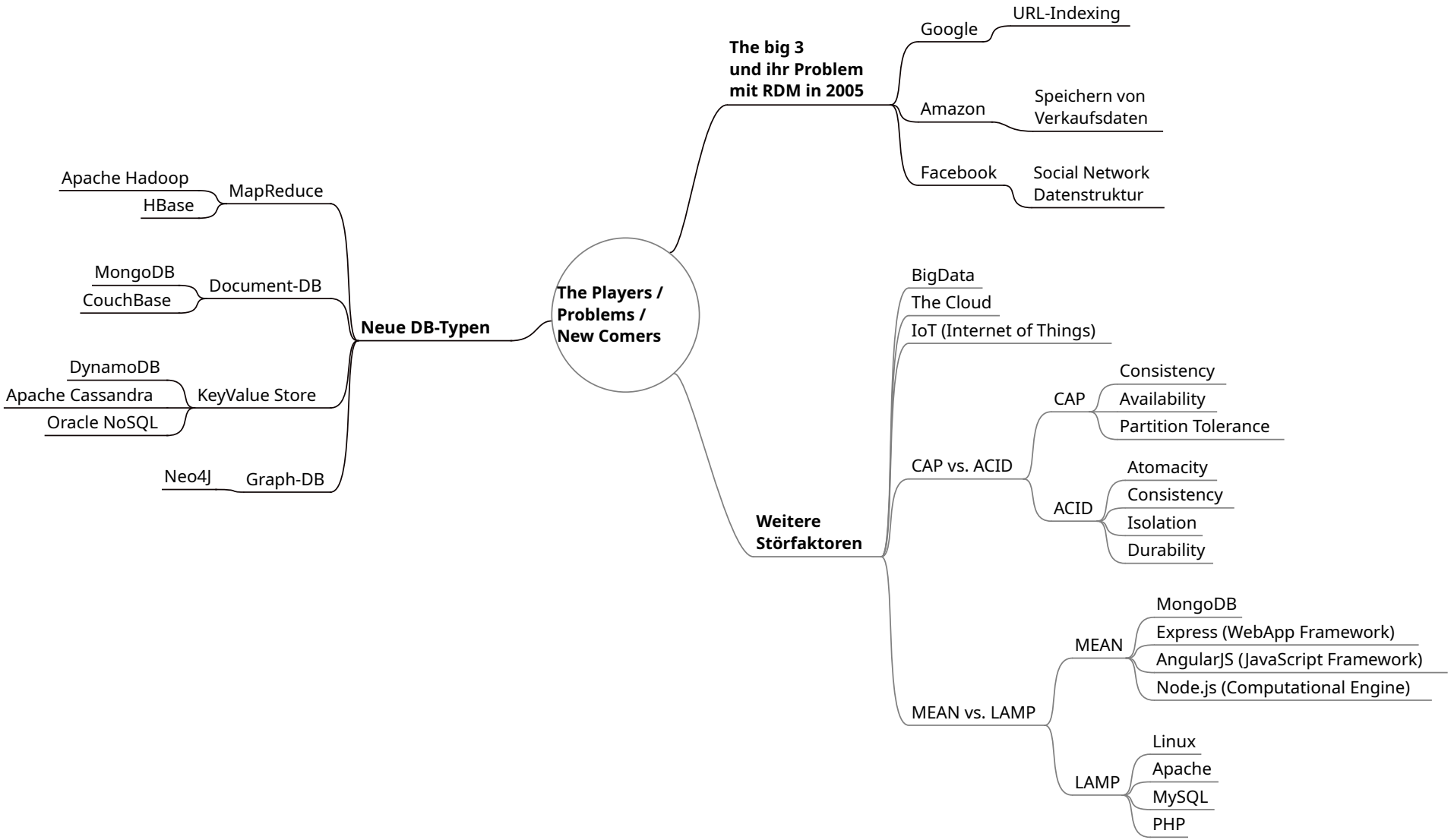
- **STRAIGHT_JOIN**
  - Bis "Faktor 1.5"-Unterschied im Beispiel.
  - => Optimizer nur nach Bedarf ausschalten!

- **Reihenfolge der Bedingungen**
  - Spielt keine Rolle, wenn Optimizer on.

- **INNER JOIN vs. "Join von Hand"**
  - Spielt keine Rolle, wenn Optimizer on.

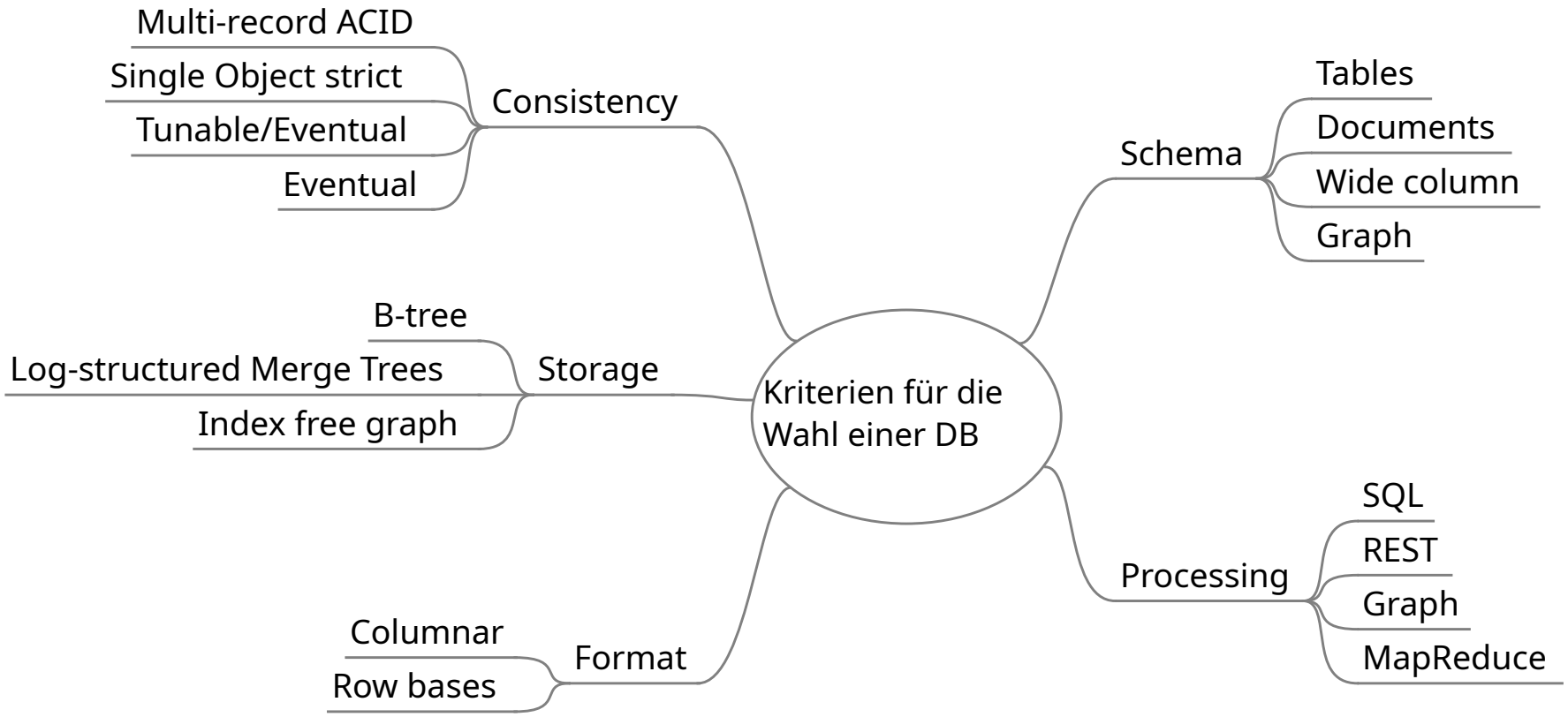
- **Im Zweifelsfall**
  - Selber prüfen. Das alles ist aber DB-spezifisch!
  - RTFM ("excuse my french"…).

RDB vs. NoSQL
Die Problematik (1)



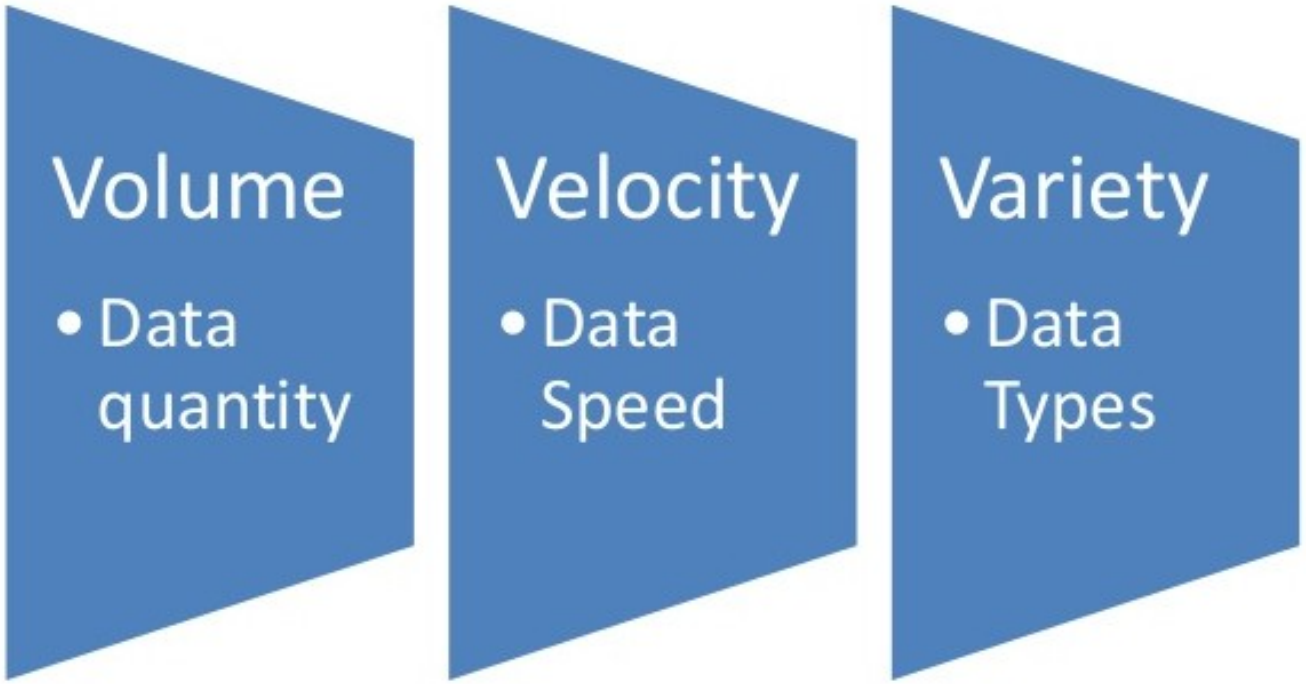
Quelle: Next Generation Databases, S. 15-73

RDB vs. NoSQL
Wahl einer DB (2)

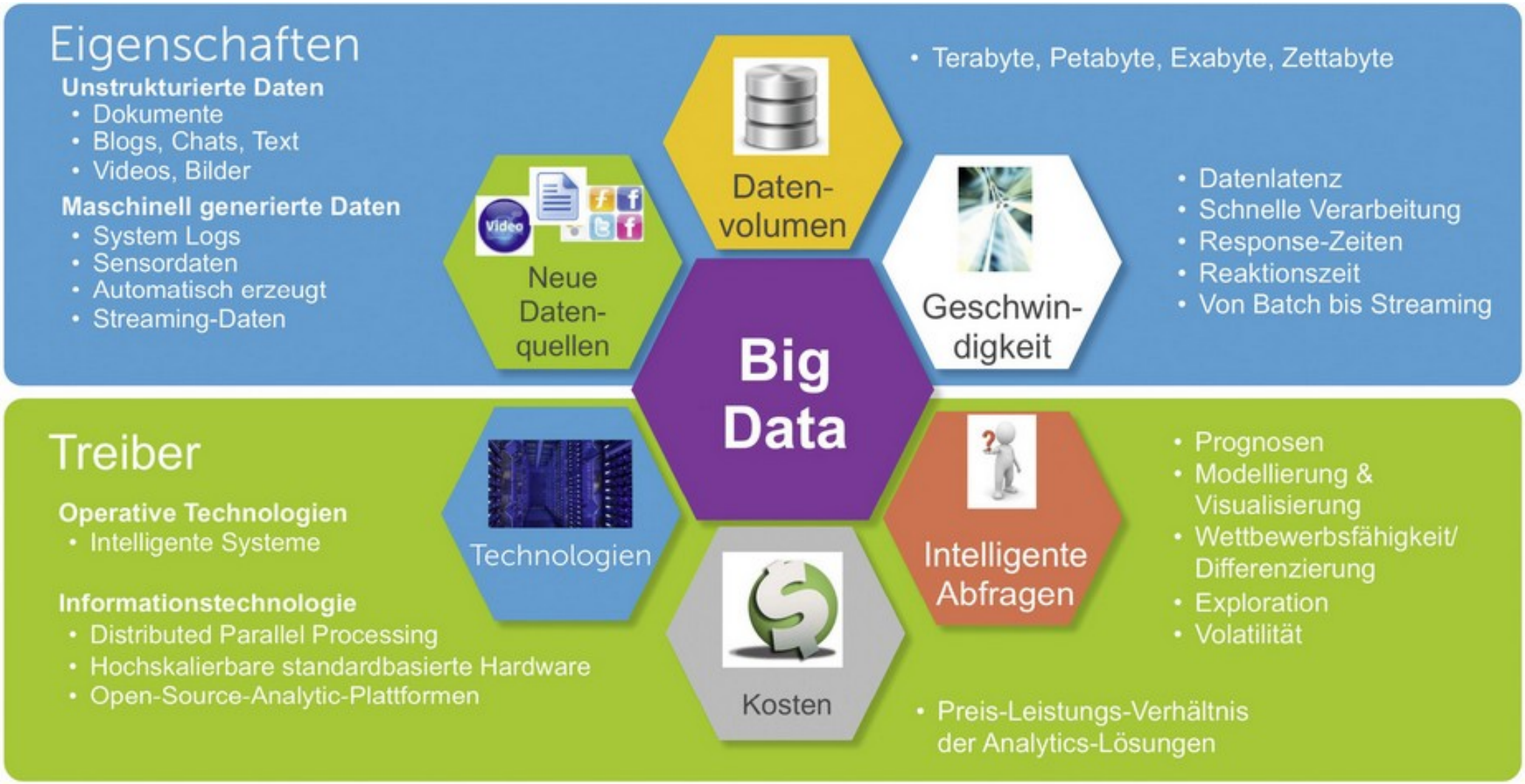


Quelle: Next Generation Databases, S. 194

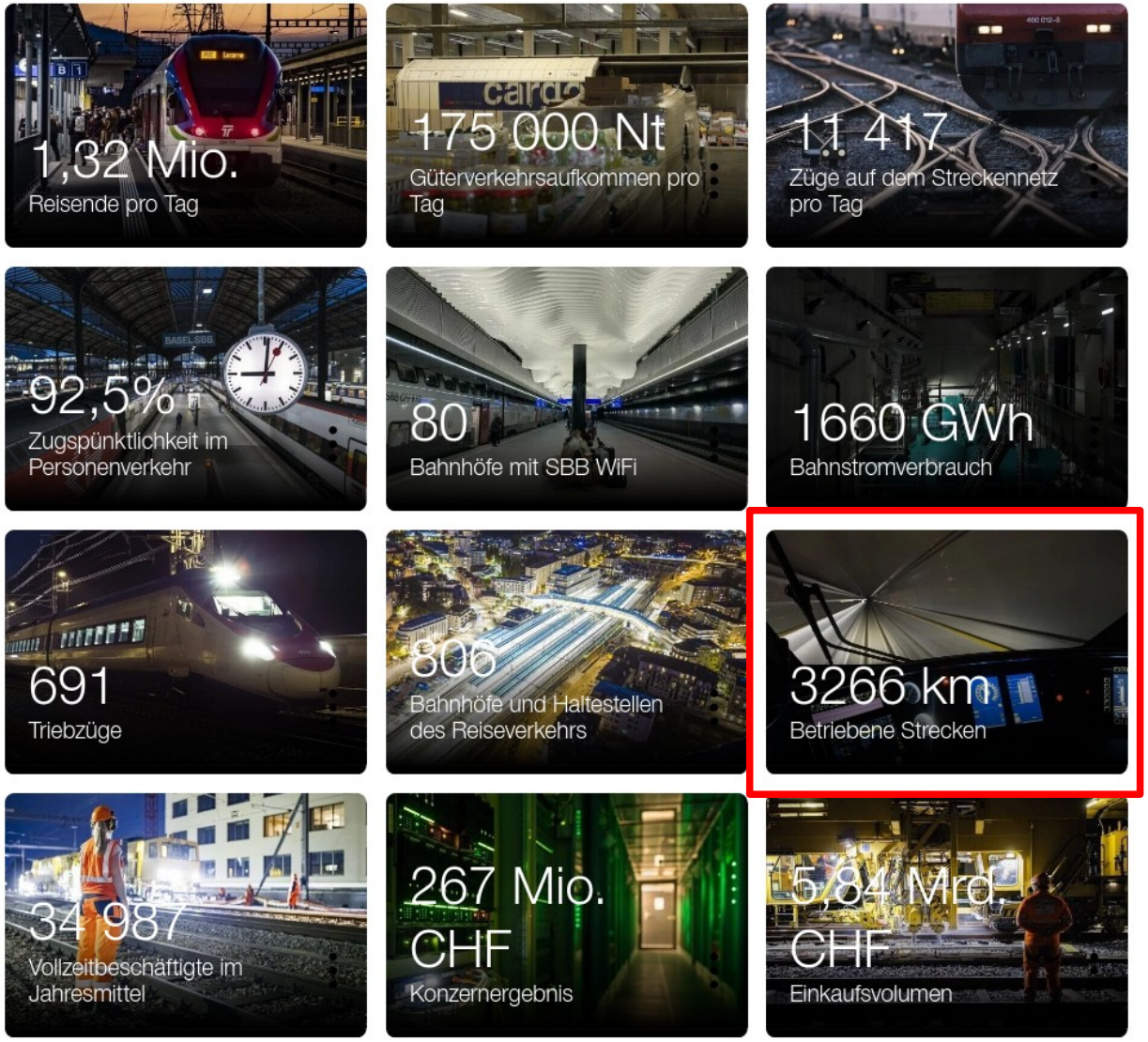
BigData Ausblick (1)
Die Begriffe



BigData Ausblick (2)
Die Begriffe



BigData Ausblick (3)
Die SBB in Zahlen und Fakten



Quelle: SBB / 2024, mehr hier

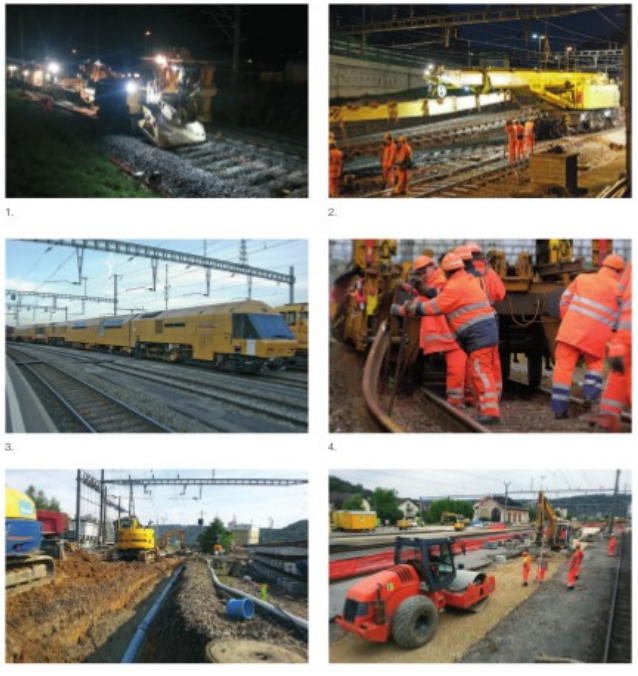
BigData Ausblick (4)
Das SBB DMD-Beispiel

Diagnosefahrzeug der SBB

Messdaten
und Bilder

Daten upload
und transformation
(DMD)

Preventive
Maintenance

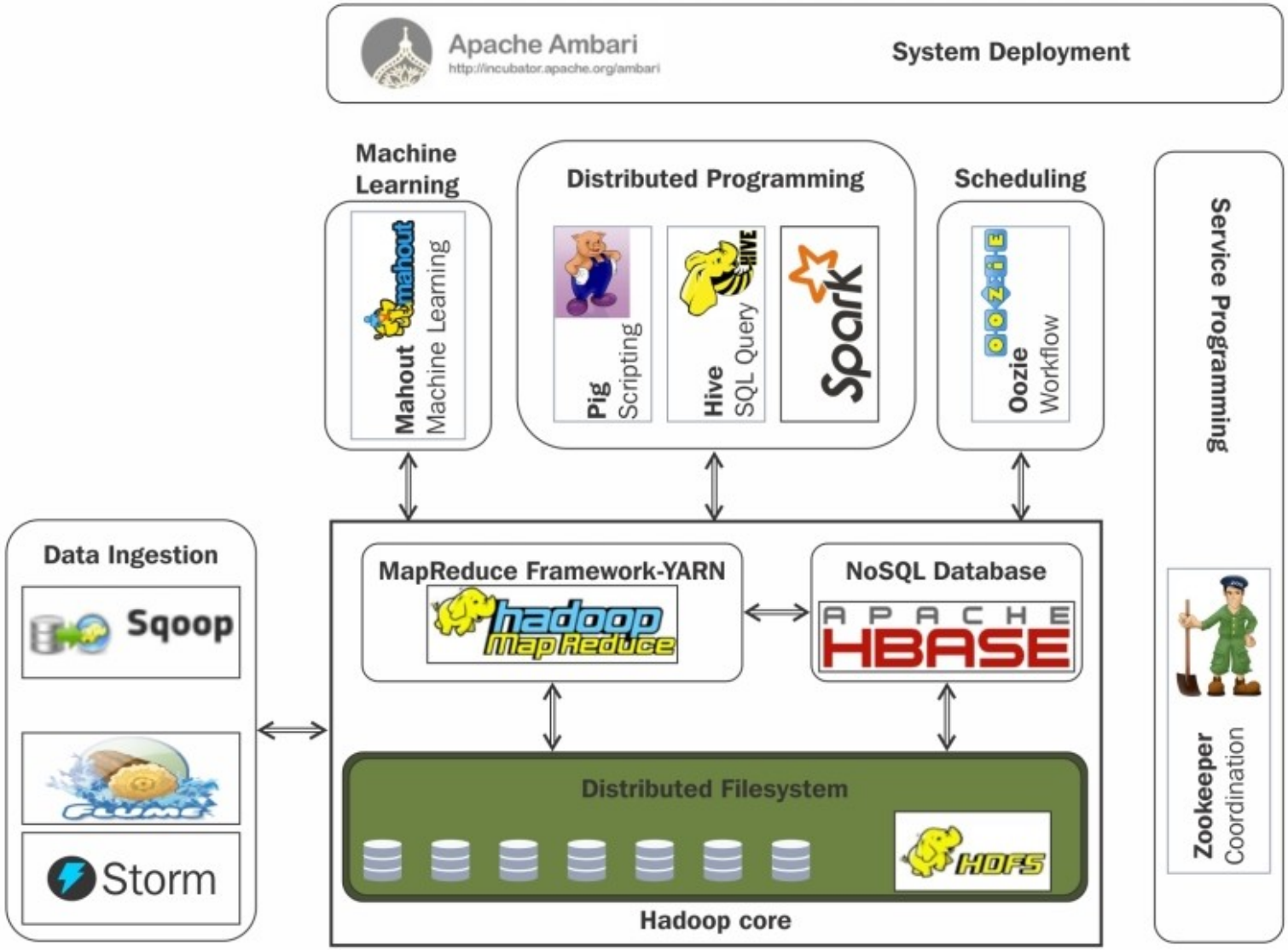


Wartung der
Infrastruktur

Heute: 32 TB
Morgen: 700 TB

RDB 7 - 24
Version 3.13

BigData Ausblick (5)
Das Hadoop Ecosystem



Quelle: https://hadoop.apache.org/

BigData Ausblick (5)
Die Hadoop und YARN Architektur

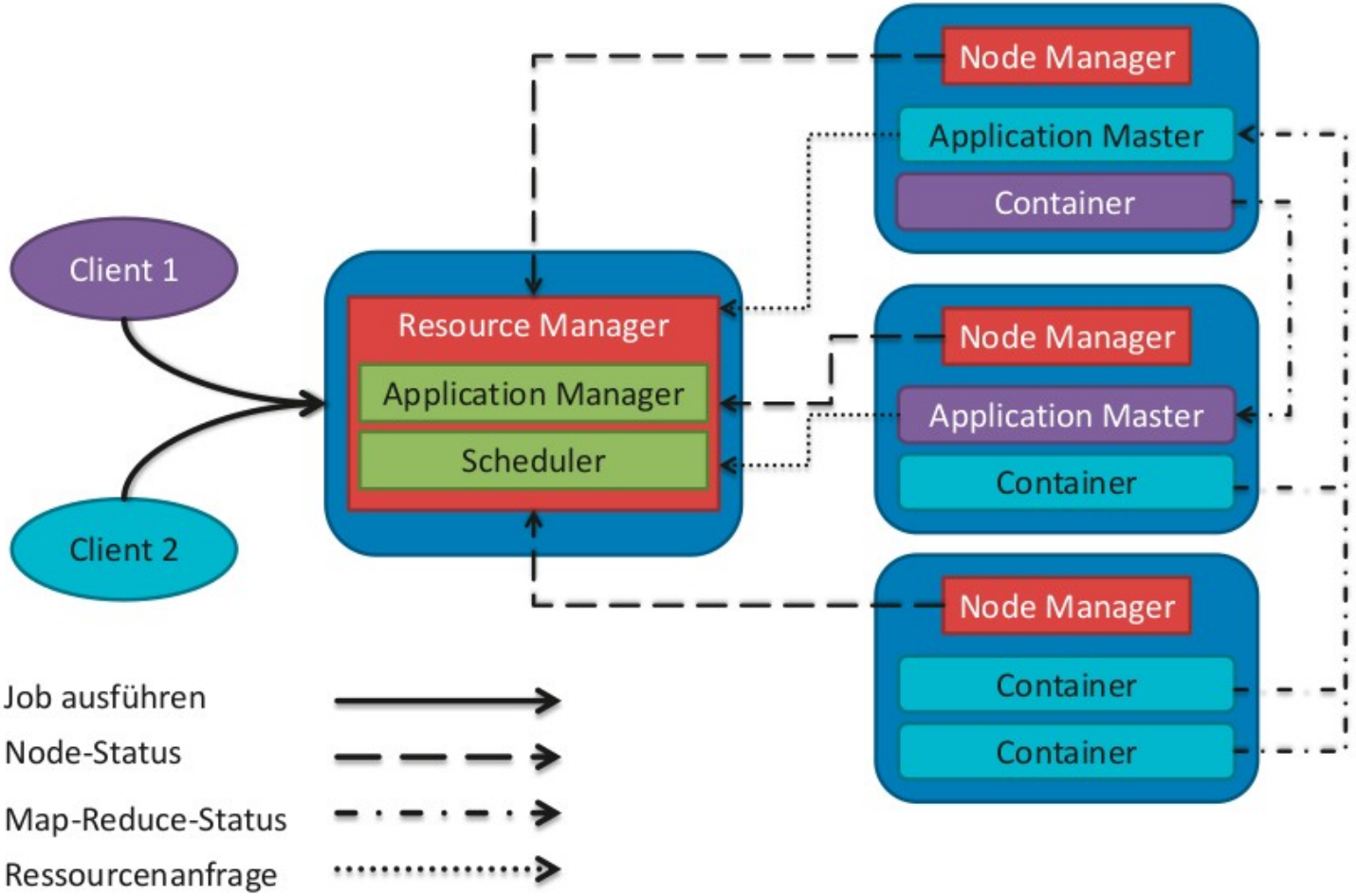


Bild 3.2 Architekturübersicht zu YARN

Yet Another Resource Negotiator

BigData Ausblick (5)
Das HDFS

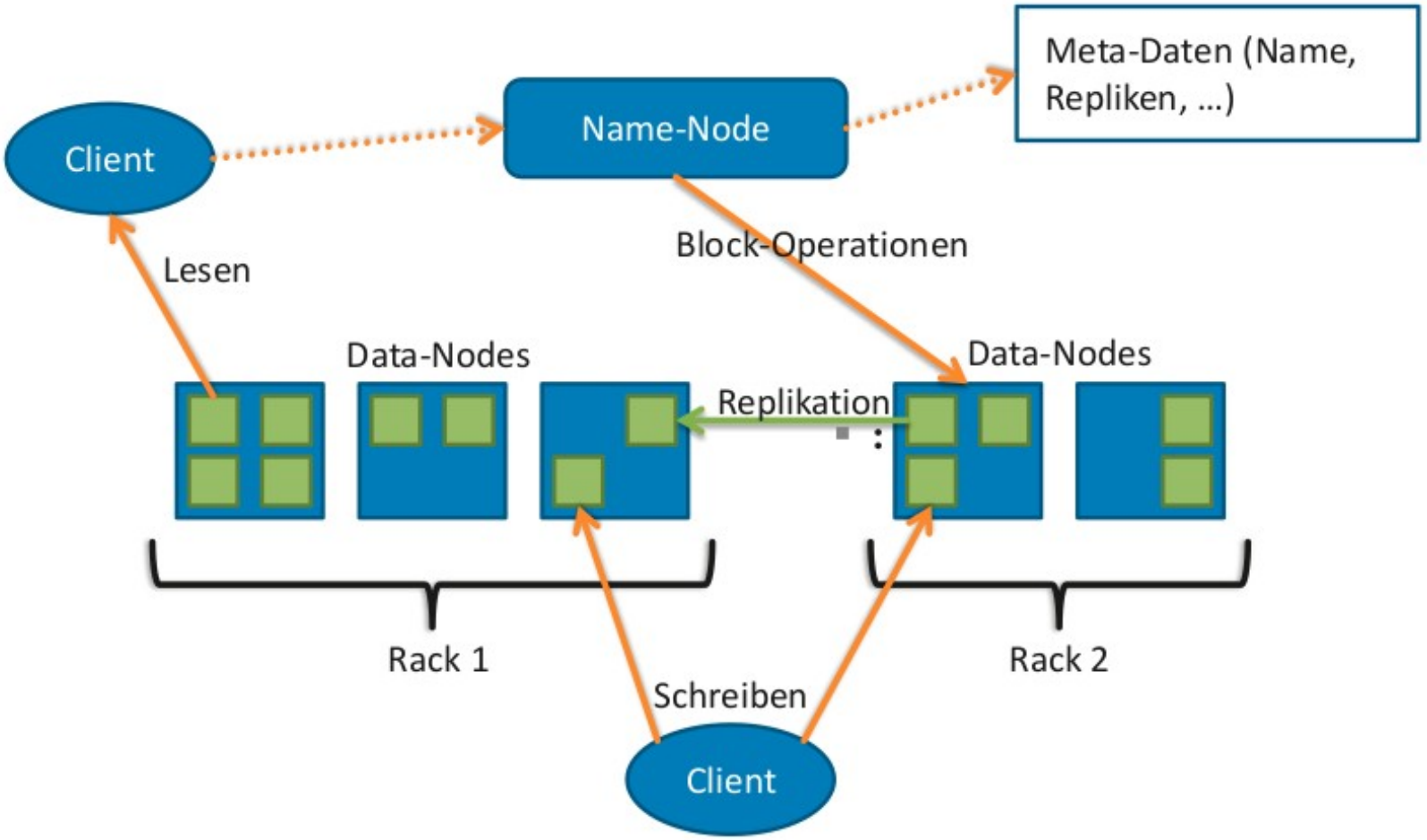
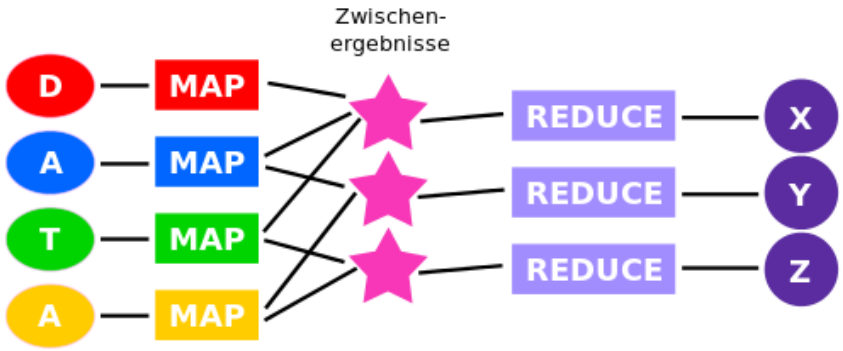


Bild 3.1 Architektur und Funktionsweise des HDFS

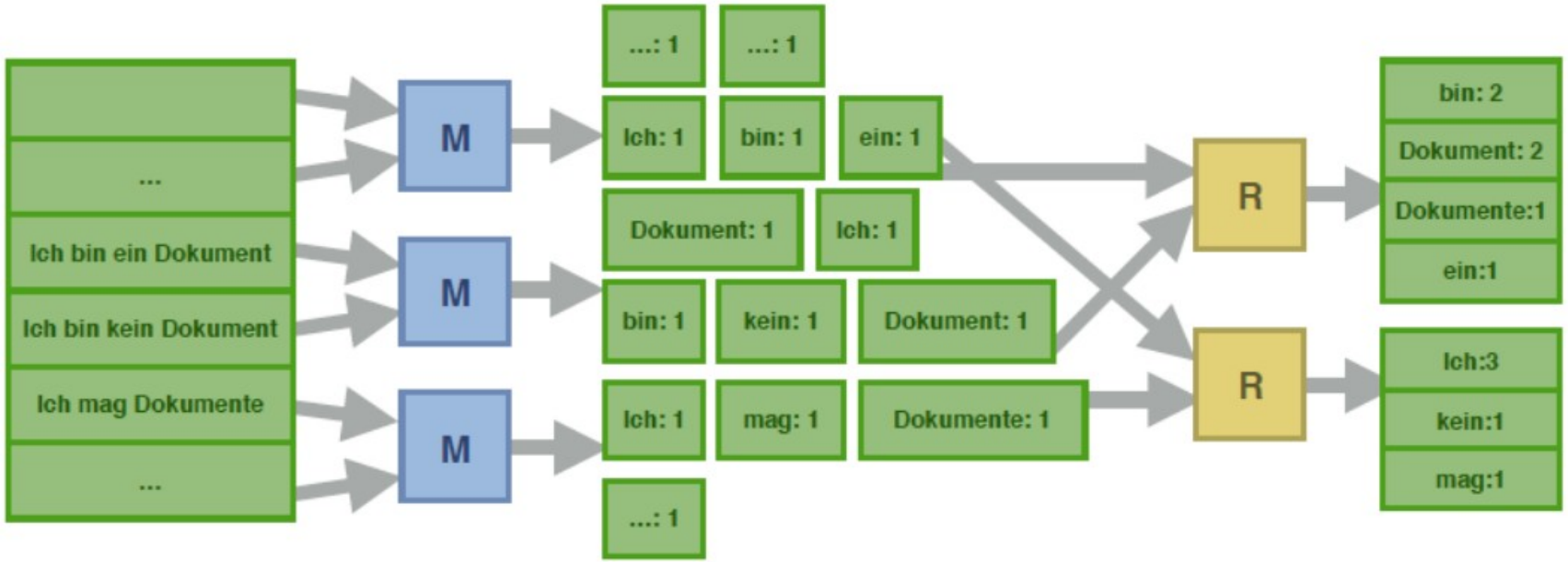
Hadoop Distributed File System

BigData Ausblick (5)
Map / Reduce

Illustration des Datenflusses [Bearbeiten | Quelltext bearbeiten]



Nur für Batch!
Kein Streaming



Eingabe Zwischenergebnisse Ergebnis

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

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.