Hive

- Datenbank / Data Warehouse basierend auf Hadoop
- Hive = MapReduce + SQL
  - SQL ist einfach und weit verbreitet
  - MapReduce skaliert sehr gut
- Automatische Übersetzung von SQL nach MapReduce nötig
  - Programme schwer zu warten, kaum Reuse
  - Barriere für Nicht-Experten
  - Fehlende Ausdrucksmächtigkeit, z.B. hoher Zeitaufwand, um simple Count/Avg-Anfragen in MapReduce zu realisieren

Quellen für diese und folgende Folien: [Hive], [Hive1], [Hive2], [Hive3]

Hive: Übersicht

- Verwaltung und Analyse strukturierter Daten mit Hilfe von Hadoop
  - keine Online-Datenbank, hohe Latenzzeit
- Datenhaltung im HDFS, Metadaten für Abbildung auf Tabellen
  - Komplexe Datentypen (u.a. Listen, Maps)
  - Direkter Zugriff auf Dateien und unterschiedliche Datenformate
- Anfragen mit HiveQL, Ausführung mit MapReduce
  - Einbindung von Skripten (z.B. Python) in Anfragen
  - Metadaten u.a. für Optimierung (u.a. Join, Group By)
- Skalierbarkeit und Fehlertoleranz
  - durch HDFS + MapReduce
- Erweiterbarkeit
  - User-Defined Table-Generating Functions (UDTF)
  - User-Defined Aggregate Functions (UDAF)
- Alternative Query Execution Engines möglich
  → Hive on Spark, Hive on Tez

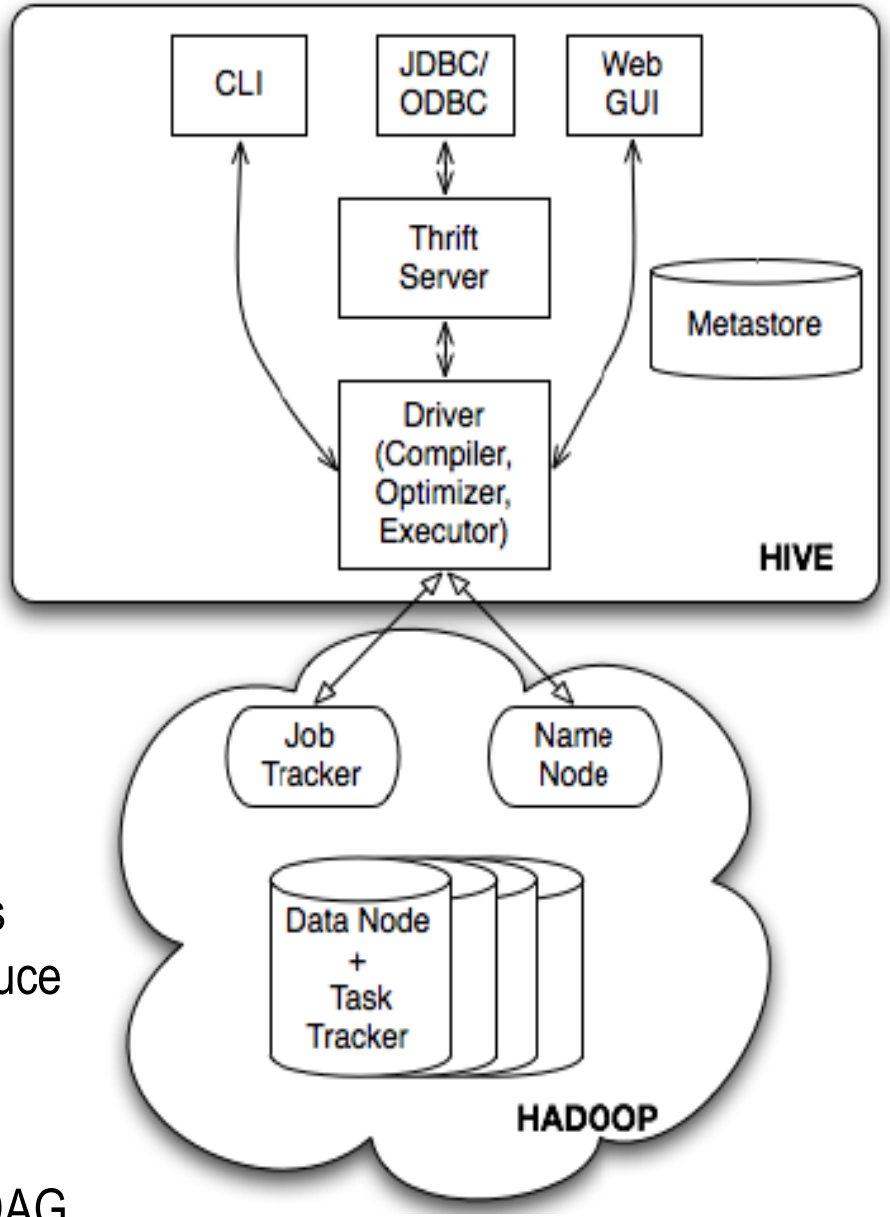
Hive 2.0 - Releases

- Seit Juni 2015 Aufspaltung in zwei "main lines“
- **Hive 2.x** – *Master Branch*
  - Kein Support einiger älterer Features (deprecated)
    → Nicht(!) zwingend abwärts kompatibel zu 1.x Versionen
  - Enthält alle **neuen Features und Bug Fixes**
- **Hive 1.x –** *Branch-1*
  - Build von **stabilen, abwärtskompatiblen** Releases
  - Enthält (aktuell) alle kritischen Bug-Fixes des Masters bis mind. Juni 2016
  - Schrittweise Aufnahme neuer Features aus dem Master
    - Features, die Abwärtskompatibilität verletzen, werden grundsätzlich nicht akzeptiert

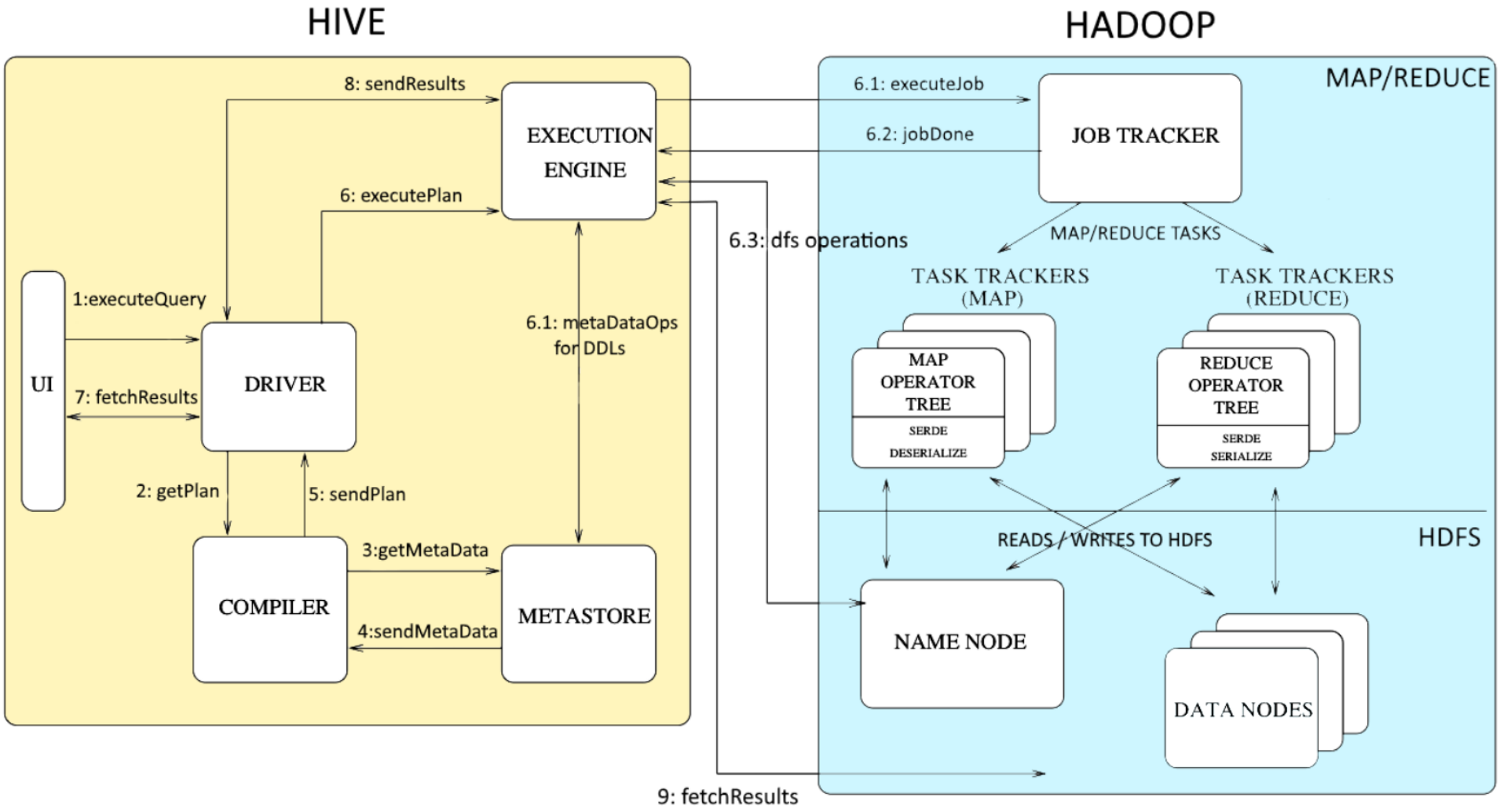
https://cwiki.apache.org/confluence/display/Hive/HowToContribute#HowToContribute-UnderstandingHiveBranches

Hive: Architektur

- Metastore
  - Tabellen, Spalten/Typ
  - Location, Partitionen
  - (De)Serialisierungsinformationen
- CLI / Web-GUI
  - Browse Metastore
  - Absetzen von Abfragen
- Thrift
  - Cross-language Service → HiveQL
- Compiler + Optimizer
  - Anfrageoptimierung und Übersetzung des HiveQL-Statements in DAG von MapReduce Jobs
- Executor / Execution Engine
  - Ausführung der MR-Jobs entsprechend DAG



Hive: Architektur



Quelle: https://cwiki.apache.org/confluence/display/Hive/Design

Hive: Datentypen & Datenzugriff

- Datentypen
  - einfache und zusammengesetzte Datentypen
  - Listen, Maps
- Flexible (De)Serialisierung von Tabellen
  - unterschiedliche (nutzerdefinierte) Formate, z.B. XML, JSON, CSV
  - unterschiedliche Speicherung, z.B. Datei, ProtocolBuffer (geplant)
- Vorteil
  - keine Konvertierung (und Replizierung!) der Originaldaten in relationale Form sondern direkter Hive-Zugriff
- Nachteil
  - keine Vorverarbeitung (z.B. Indexierung) möglich
  - immer full Table/File Scans nötig

Hive: Tabellen, Partitionen und Dateien

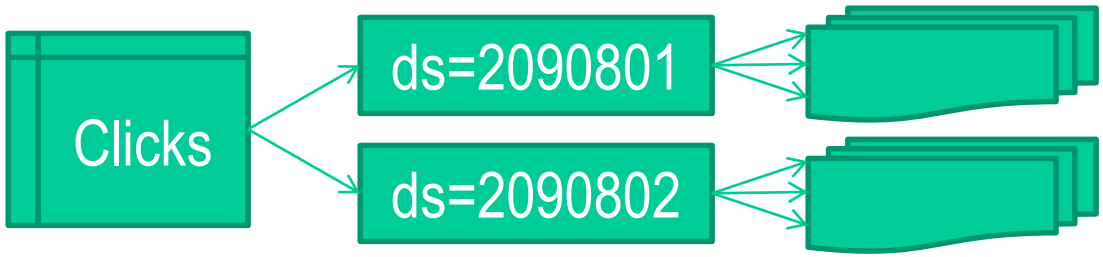
- Tabelle kann auf exist. Daten im HDFS verweisen
  - Tabelle hat korrespond. HDFS-Verzeichnis : `/wh/pvs`
  - Definition von Spalten, anhand welcher die Daten partitioniert werden

    `/wh/pvs/ds=20150801/ctry=US`

    `/wh/pvs/ds=20150801/ctry=CA`
  - Bucketing: Aufteilen der Dateien eines Verz. anhand Hash-Wert (Datenparallelität)

    `/wh/pvs/ds=20150801/ctry=US/part-00000` …

    `/wh/pvs/ds=20150801/ctry=US/part-00020`



Tabelle

Partitionen (evtl. mehrere Level)

HDFS Dateien (evtl. als Hash-Bucket)

Hive: Tabellen

- Erstellung

```
CREATE EXTERNAL TABLE pvs
(userid int, pageid int, ds string, ctry string)
PARTITIONED BY(ds string, ctry string)
STORED AS textfile
LOCATION ‘/path/to/existing/file’
```

- Laden von Daten

```
status_updates
(user_id int, status string, ds string)
LOAD DATA LOCAL
INPATH ‘/logs/status_updates’
INTO TABLE status_updates
PARTITION (ds=’2009-03-20’)
```

Hive-QL

- SQL-ähnliche Anfragesprache
  - Selektion, Projektion, Equi-Join, Union, Sub-Queries, Group By, Aggregatfunktionen
- Erweiterung von Queries um
  - MapReduce Skripte
  - UDF, auch auf komplexen Objekten (Lists, Map)

```
FROM (
    FROM pv_users
    SELECT TRANSFORM(pv_users.userid, pv_users.date)
    USING 'map_script'
    AS(dt, uid)
    CLUSTER BY(dt)
) map
INSERT INTO TABLE pv_users_reduced
SELECT TRANSFORM(map.dt, map.uid)
USING 'reduce_script'
AS (date, count);
```

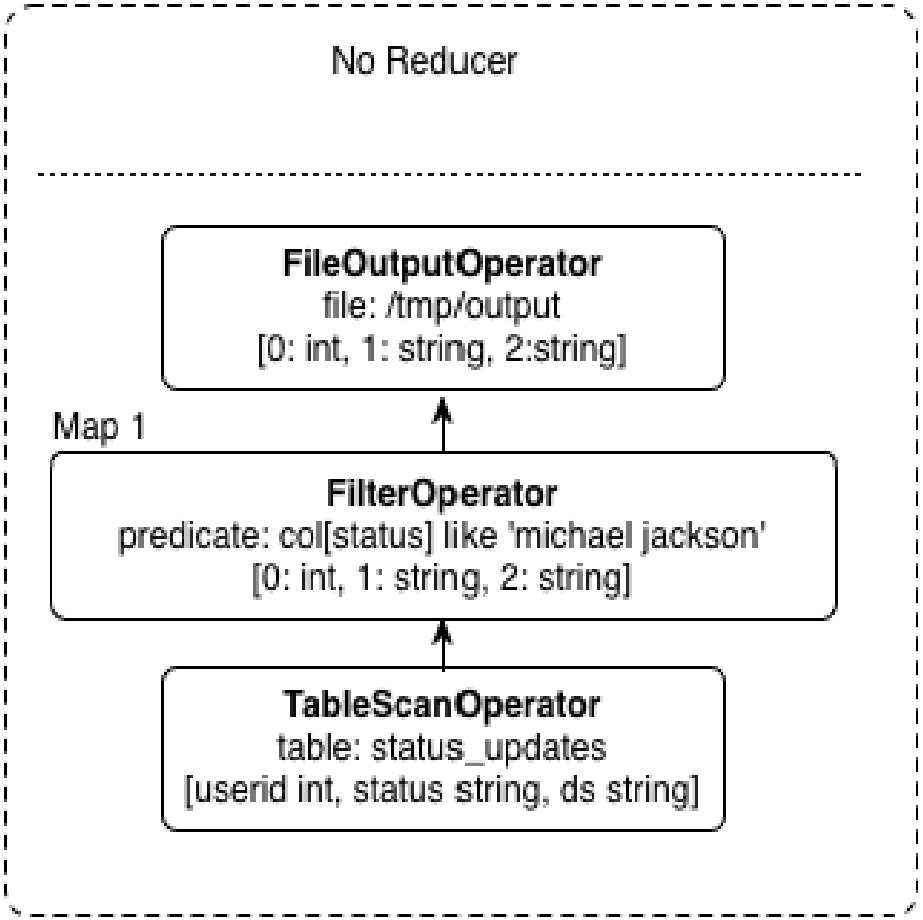
Hive-QL: Anfrageübersetzung

- Hive-QL Query wird in DAG (directed acyclic graph) übersetzt
- Knoten: Operatoren
  - TableScan
  - Select, Extract
  - Filter
  - Join, MapJoin, Sorted Merge Map Join
  - GroupBy, Limit
  - Union, Collect
  - FileSink, HashTableSink, ReduceSink
  - UDTF
- Graph repräsentiert Datenfluss
- Mehrere (parallele) Map/Reduce Phasen möglich

Hive-QL: Anfrageübersetzung (Beispiel)

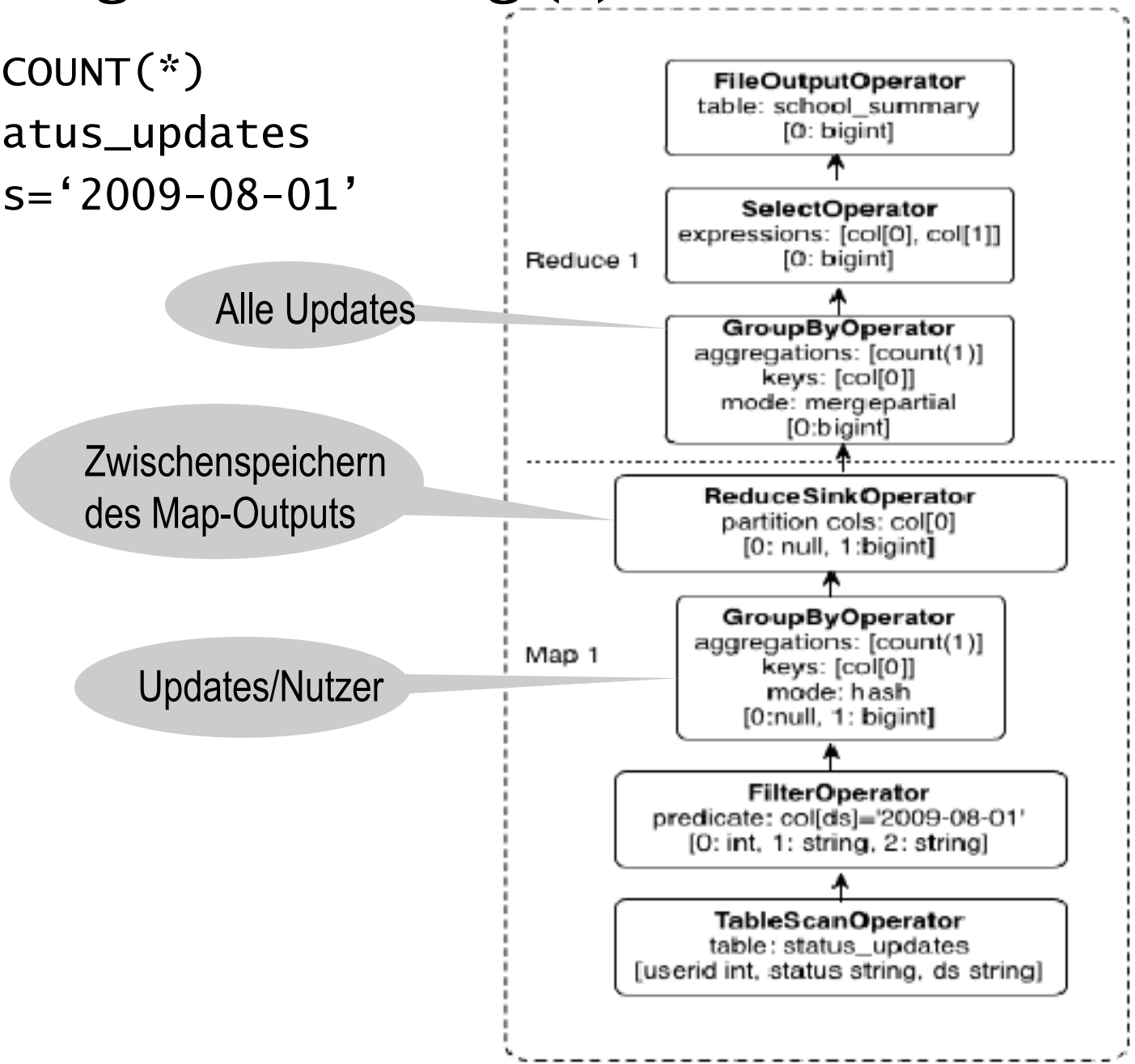
- Beispiel

```
SELECT *
FROM status_updates
WHERE status
    LIKE ‘michael jackson’
```



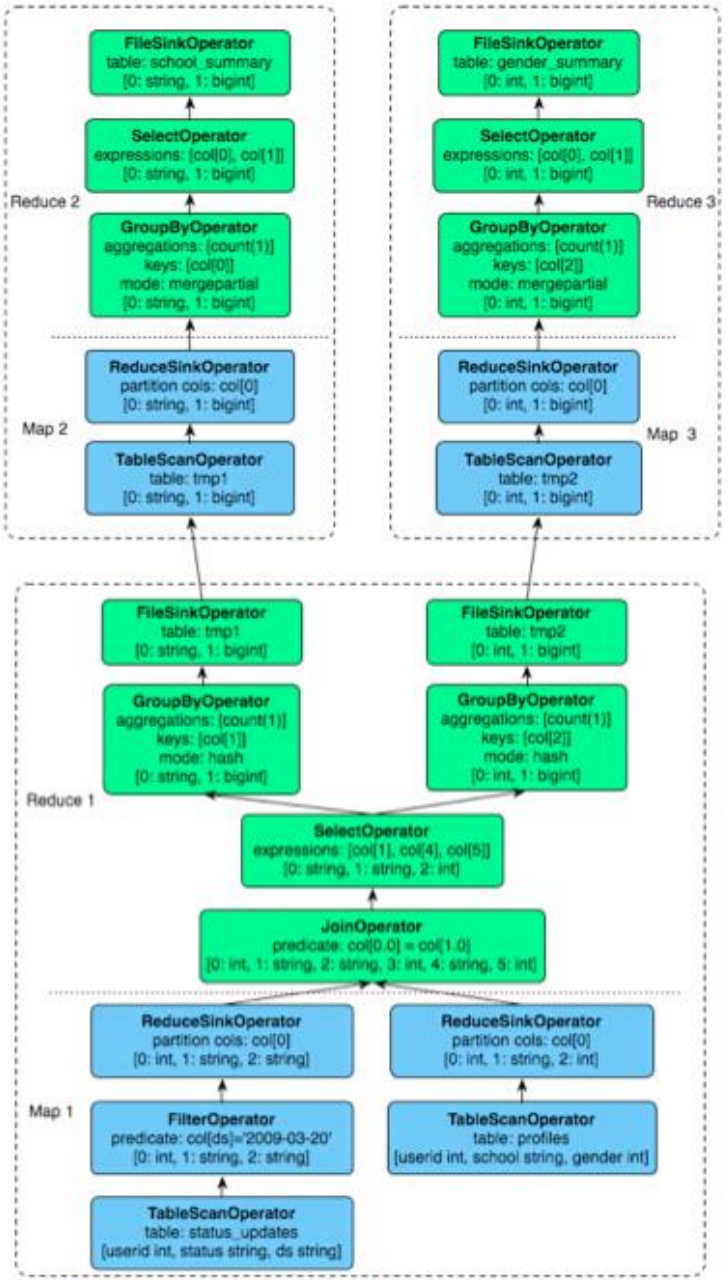
Hive: Anfrageübersetzung (2)

```
SELECT COUNT(*)
FROM status_updates
WHERE ds='2009-08-01'
```



Hive: Anfrageübersetzung und -optimierung

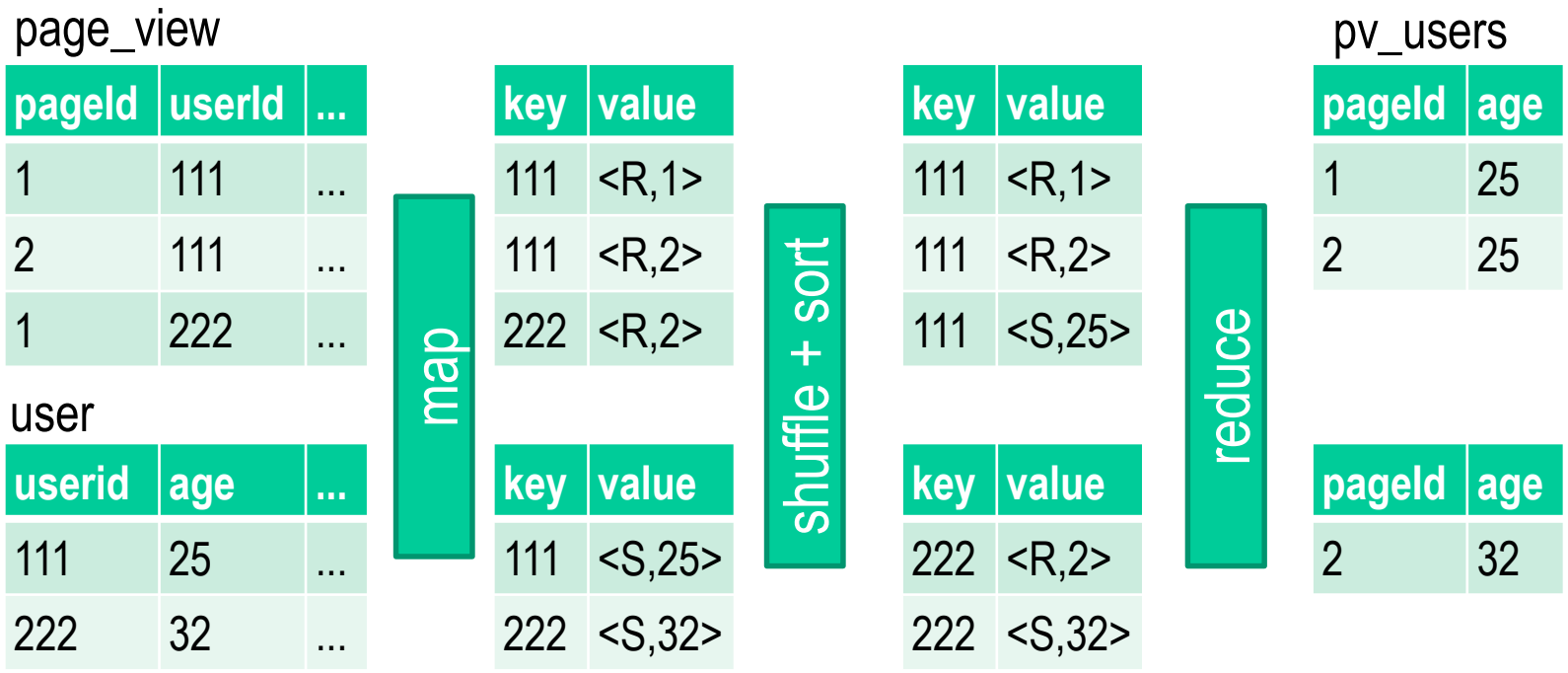
- Anfragepläne können sehr komplex werden
- Anfrageoptimierung
  - Verwerfen nicht benötigter Spalten
    - Berücksichtigung von (Outer-)Join- und Selektionsattributen
  - Frühes Anwenden von Selektionsprädikaten
  - Verwerfen nicht benötigter Partitionen



Hive: Join

= Shuffle Join (reduce-side join)

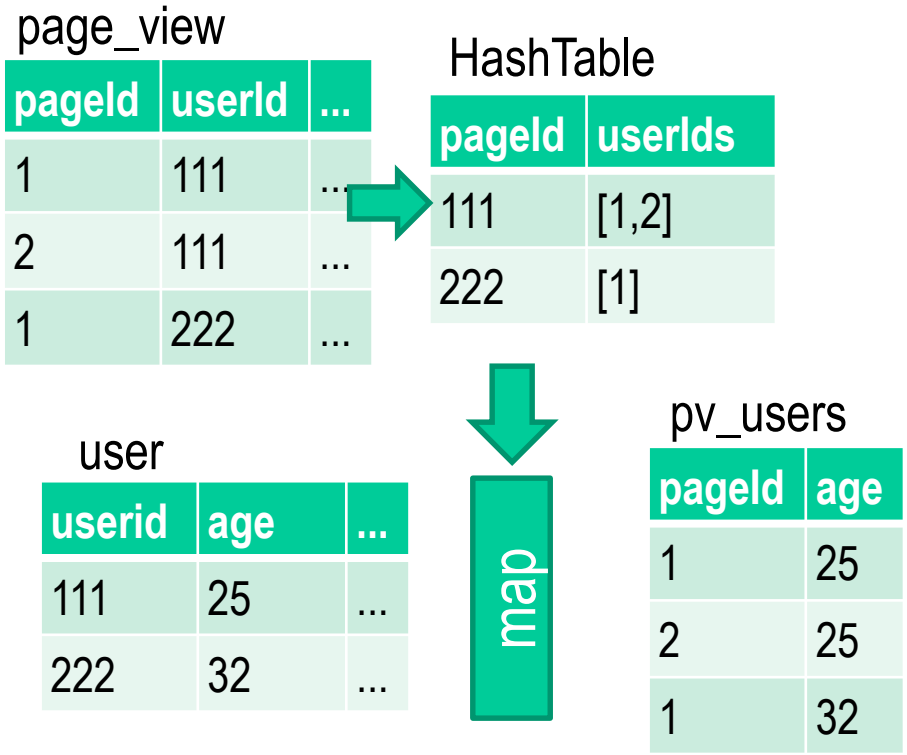
```
INSERT INTO TABLE pv_users
SELECT pv.pageid, u.age
FROM page_view pv
JOIN user u ON (pv.userid = u.userid)
```



- Key = Join-Key, Value mit Flag (R oder S) zur Unterscheidung d. Tabellen
- Mehrweg-Join mit selbem Join-Key → 1 MapReduce job
- Mehrweg-Join mit *n* Join-Keys → *n* MapReduce job

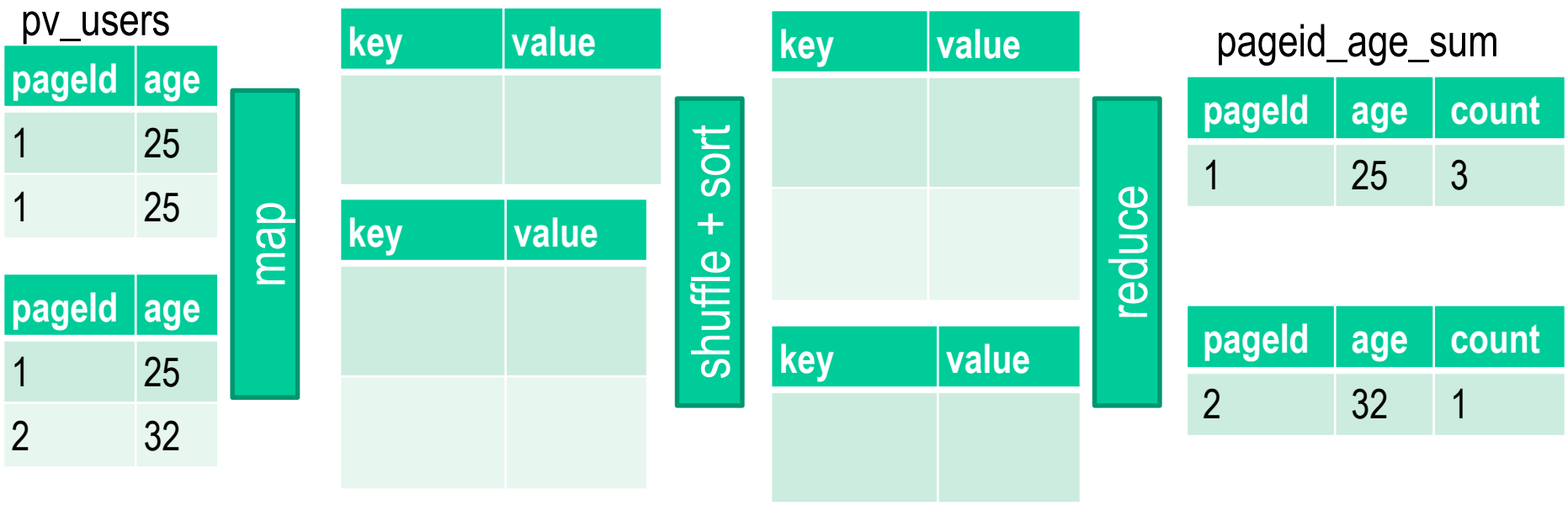
Join: Performanzsteigerung durch MapJoin

- MapJoin (aka Broadcast Join)
  - kleine Tabelle als zusätzlicher Map-Input
  - kann vorher zu Hash-Tabelle umgewandelt werden (ggf. zusätzlich komprimiert)
  - kein Reduce notwendig
  - *n* Wege-Join möglich, wenn *n*-1 Tabellen für map verfügbar
- Dynamische Join-Entscheidung
  - Bestimmung großer/kleiner Tabelle zur Laufzeit
  - Anwendung von MapJoin falls kleine Tabelle(n) “klein genug”



Hive: Group By

```
INSERT INTO TABLE pageid_age_sum
SELECT pageid, age, count(*)
FROM pv_users
GROUP BY pageid, age
```

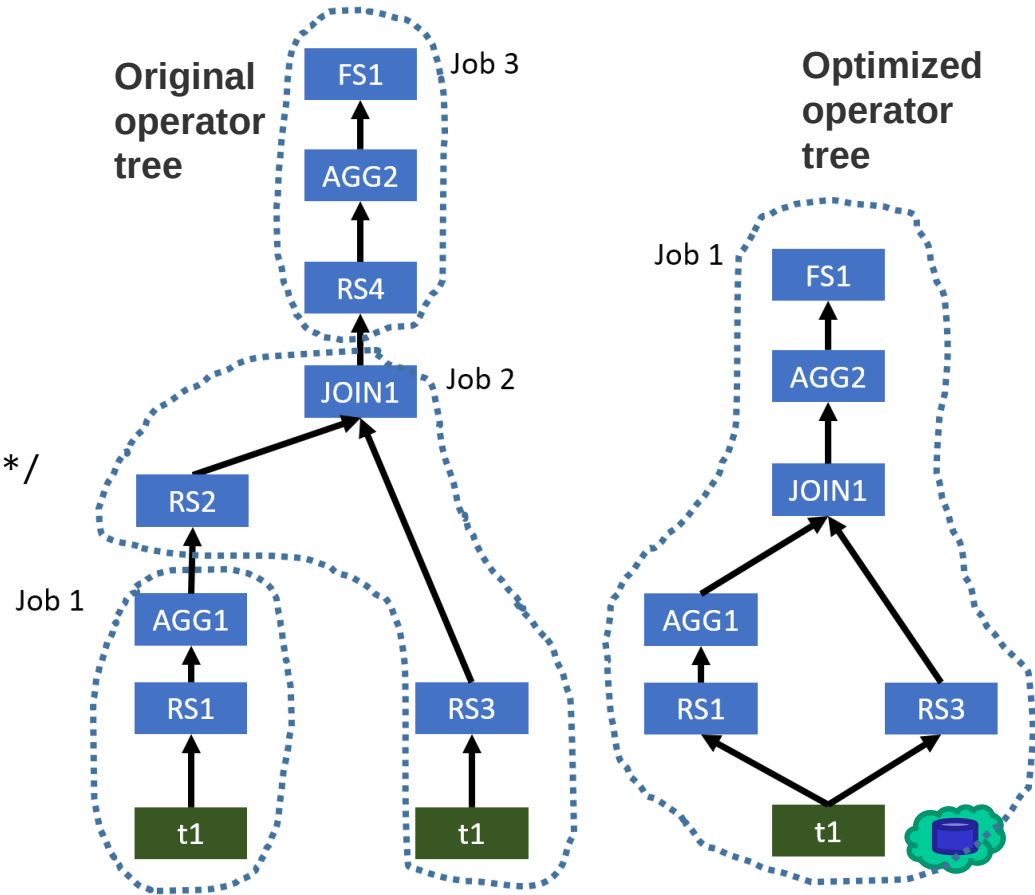


- Key = Gruppierungsattribute
- Reduce = Aggregationsfunktion
  - “Voraggregation” durch Combiner in Map möglich (z.B (<1,25>,2))

HIVE: Correlation Optimizer

- Ausnutzen von Korrelationen u.a. zwischen Join und GroupBy Operationen
- Vermeiden teurer Shuffle Operationen
- Input Correlation: Input Tabelle wird orig. Operatorbaum in mehreren MR tasks verwendet
- Job Flow Correlation: zwei abhängige MR tasks shufflen die Daten in gleicher Weise

```
SELECT tmp1.key, count(*)
FROM (SELECT key, avg(value) AS avg
      FROM t1
      GROUP BY key) tmp1     /*AGG1*/
JOIN t1 ON (tmp1.key = t2.key) /*JOIN1*/
WHERE t1.value > tmp1.avg
GROUP BY tmp1.key;           /*AGG2*/
```



Nutzer-definierte Skripte

- Verwendung von Skripten in HiveQL-Anfragen mittels TRANSFORM-Operator
  - Daten(de-)serialisierung
  - Austausch per stdin/stdout

id	title
1	Body Snatcher
2	Armageddon
3	AI

firstl	n
B	1
A	2

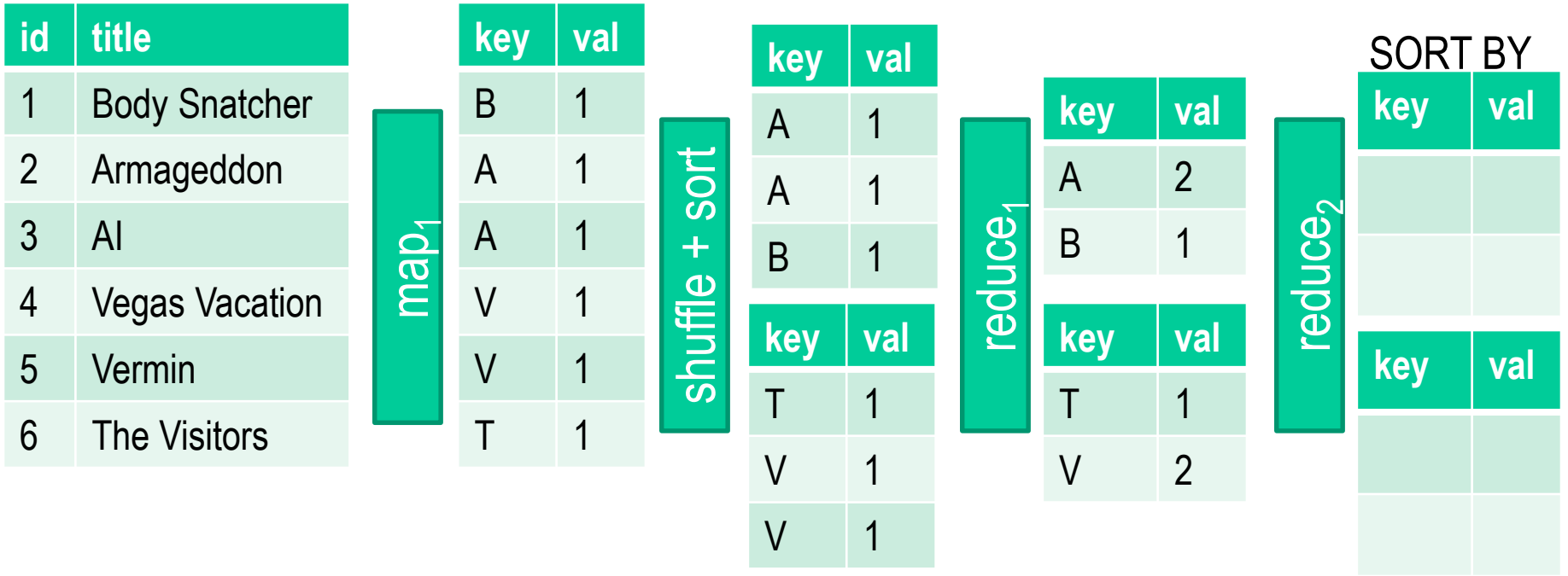
firstletter.py

```
import sys
for line in sys.stdin:
  line = line.strip()
  id, title = line.split('\t')
  firstl = title[:1]
  print '\t'.join([id, title, firstl])
```

```
ADD FILE firstletter.py;
SELECT firstl, count(id) AS n
FROM (
  SELECT
    TRANSFORM (id, title)
    USING 'python firstletter.py'
    AS id, title, firstl
  FROM item ) f
GROUP BY firstl;
```

Hive: Sortierung

```
SELECT firstl, count(id) AS n
FROM ... GROUP BY firstl
SORT BY | ORDER BY n DESC
```



- Sortierung durch zusätzlichen Reduce-Step
- SORT BY = Sortierung pro Reducer
- ORDER BY = globale Sortierung
  = SORT BY mit nur einem Reducer

Pig: Übersicht

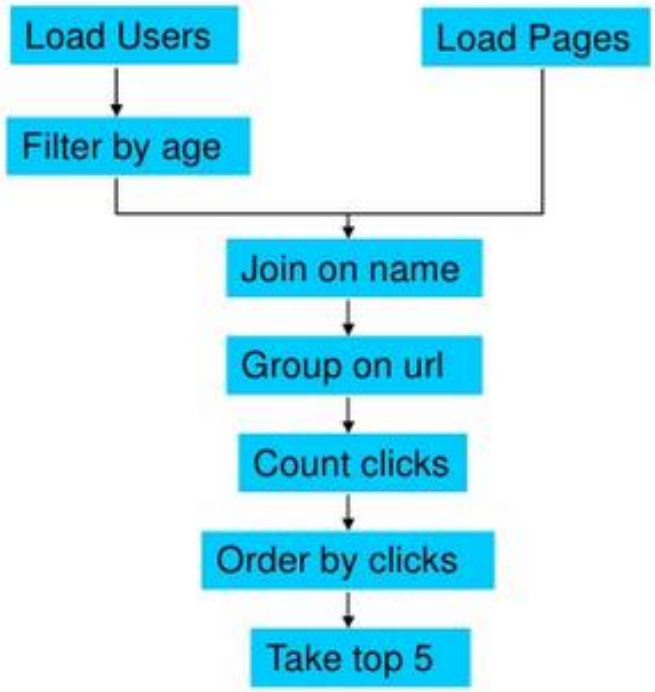
- System zur Verarbeitung großer, semi-strukturierter Daten auf Basis von Hadoop/MapReduce
- Prozedurale High-Level-Skriptsprache: PigLatin
  - Nutzung von Variablen mit komplexe Datentypen (Tupel, Bag, Map)
- Skript-Ausführung
  - Pig-Skript wird geparst und ggf. optimiert
  - Transformierung in MapReduce-Workflow
  - Ausführung mittels Hadoop
- Vorteile
  - Vereinfachte Definition komplexer Workflows
  - Automatische Konfiguration/Tuning vom MR-Jobs / -Workflows
  - Optionales Type-Checking falls Schema vorhanden

Quellen: Diese und nachfolgende Folien:
http://www.slideshare.net/hadoop/practical-problem-solving-with-apache-hadoop-pig
http://www.slideshare.net/xefyr/pig-making-hadoop-easy
http://pig.apache.org/docs/r0.10.0/

Pig: Beispiel

- Webnutzungs-Analyse: „Die fünf am meisten besuchten Webseiten junger Nutzer (18-25 Jahre)“

- Pig-Skript:




```
Users = load 'users' as (name, age);
Fltrd = filter Users by age >= 18 and age <= 25;
Pages = load 'pages' as (user, url);
Jnd = join Fltrd by name, Pages by user;
Grpd = group Jnd by url;
Smmd = foreach Grpd generate group, COUNT(Jnd) as clicks;
Srtd = order Smmd by clicks desc;
Top5 = limit Srtd 5;
store Top5 into 'top5sites';
```

Pig: Relationale Operatoren

- Load / Store / Dump
  - Daten von File System lesen / in FS schreiben / auf stdout ausgeben
- Foreach … Generate
  - Anwendung eines Ausdrucks auf jeden Datensatz → Generierung neuer Datensätze
  - Ähnlich: map
- Group
  - Gruppierung von Datensätzen nach Schlüssel
- Join
  - Join zweier oder mehr Eingabedateien
  - Unterstützung verschiedener Join-Implementierungen inkl. Skew-Behandlung
- Weitere: Stream, Order, Filter, Distinct, Sample, Split, …
- Zusätzlich: String-Funktionen, mathematische Funktionen (count, avg, …), Diagnose-Methoden (describe, explain, …), UDF-Handling, …

Pig: Abgrenzung SQL und Hive

	Pig	SQL
Definition		Deklarativ
Datenmodell		(flaches) Relationenmodell
Schema		Voraussetzung
Workloads		OLTP+OLAP
Query-Performanz		++, Optimierungen

- Anwendungsdomänen
  - Pig: Datenpipelines, Iterative Berechnungen, …
  - Hive: OLAP-artige Anfragen, BI Tools, …
- Pig geeignet zur Programmierung von ETL-Strecken
  - Datenvorverarbeitung für Data Warehousing

Zusammenfassung

- MapReduce ist kein DBMS, kann aber zur “datenbank-artigen” Verarbeitung großer Datenmengen genutzt werden
  - SQL-Anfragen können automatisch in MapReduce-Programme transformiert werden
  - MR kann flexibel auf die (semi-strukturierten) Originaldaten (d.h. Dateien) zugreifen
- RDBMS sind “pro Knoten” effizienter als MapReduce
  - ... aber MapReduce skaliert deutlich besser und ist fehlertoleranter
- Kombination der Stärken von RDBMS und MapReduce sinnvoll

Quellen & Literatur

- [MRJoin] Blanas et al.: A Comparison of Join Algorithms for Log Processing in MapReduce. SIGMOD 2010
- [Hive] http://hadoop.apache.org/hive/
- [Hive1] http://www.slideshare.net/zshao/hive-data-warehousing-analytics-on-hadoop-presentation
- [Hive2] http://www.slideshare.net/ragho/hive-user-meeting-august-2009-facebook
- [Hive3] http://www.slideshare.net/jsichi/hive-evolution-apachecon-2010