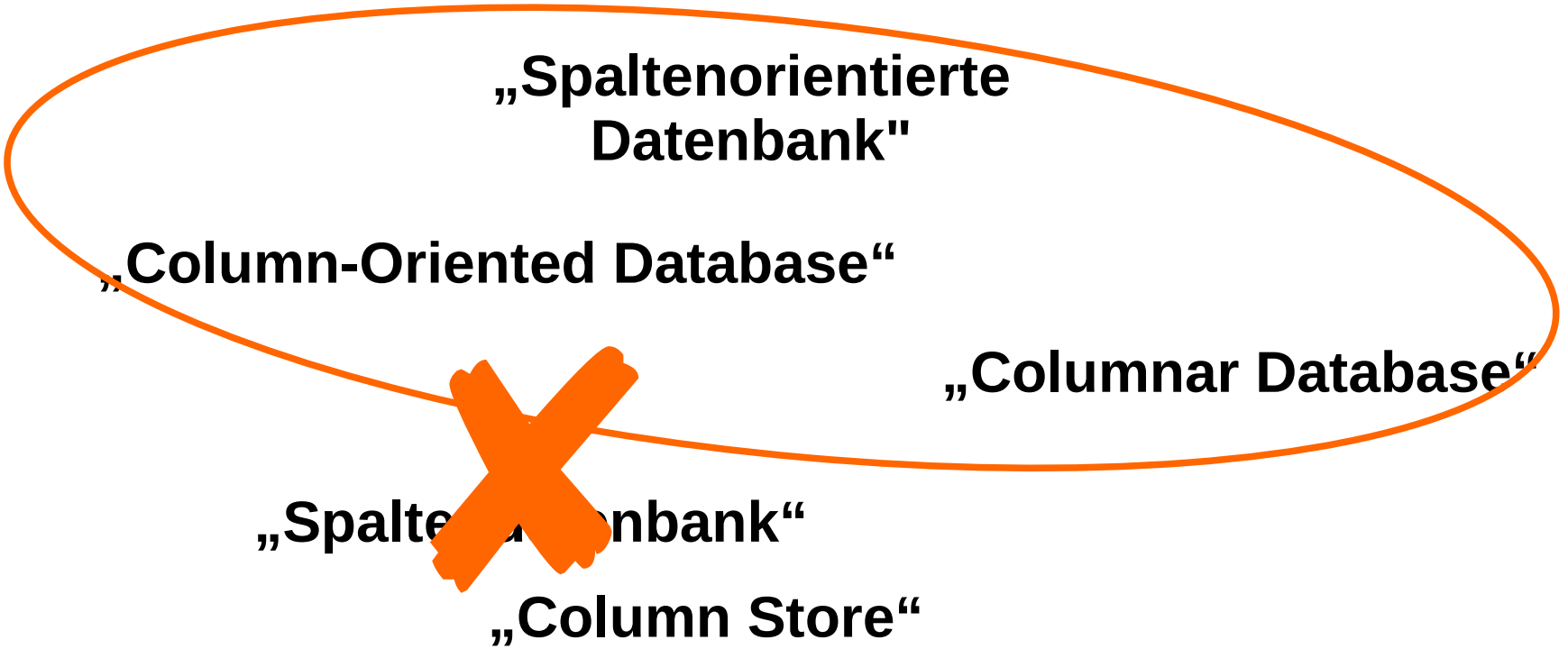
Big-Data-Technologien

Kapitel 12: NoSQL – Spaltenorientierte Datenbanken

Hochschule Trier
Prof. Dr. Christoph Schmitz

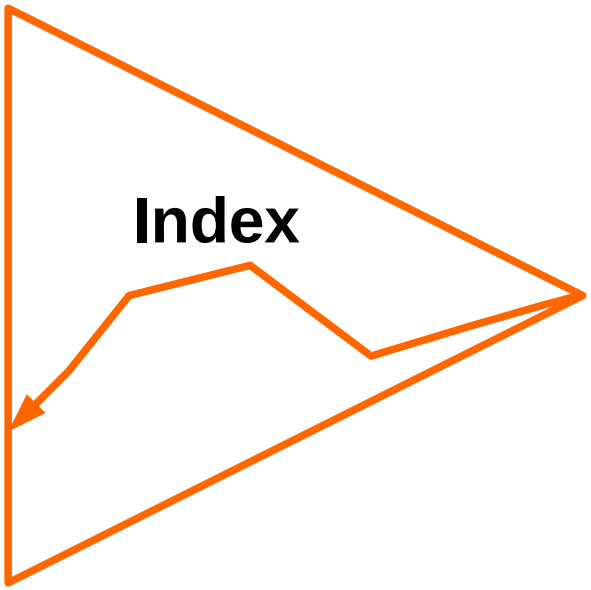
Begriffsverwirrung



„Wide-Column Store“

Zeilenorientierte Datenbanken

alice	w	red	…
bob	m	blonde	…
charlie	m	brown	…
david	m	black	…
eve	w	blonde	…
frances	w	brown	…




```
SELECT haircolor
FROM person
WHERE name = "eve"

UPDATE person
SET haircolor = "red"
WHERE name = "eve"
```

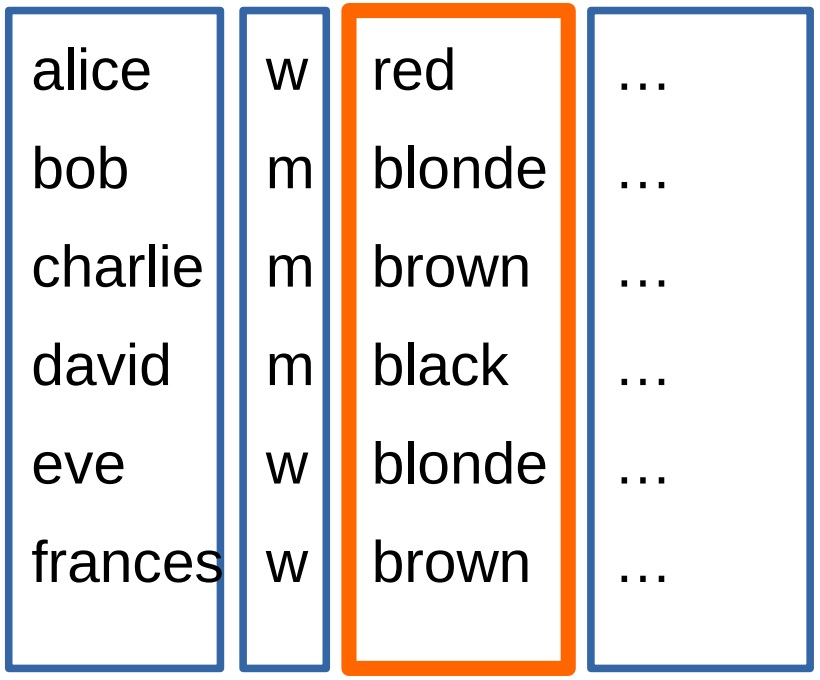
```
CREATE INDEX person_by_name
ON person(name)
```

Geeignet für OLTP

OLAP: Online Analytical Processing


```
SELECT haircolor, COUNT(*)
FROM person
GROUP BY haircolor
```

Idee: Daten spaltenweise repräsentieren

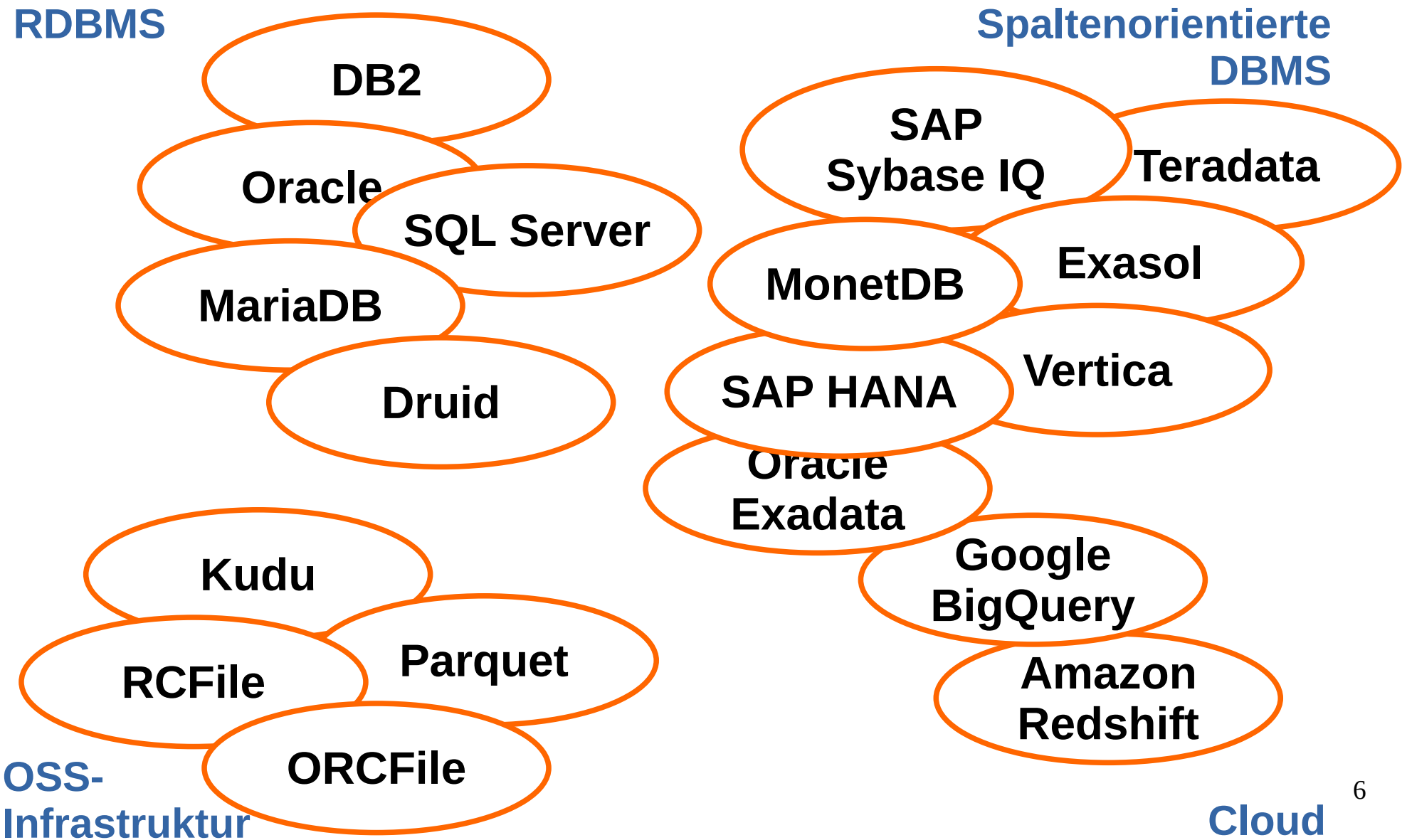


+ Aggregatfunktionen

- Zeilenweiser Zugriff

```
SELECT haircolor, COUNT(*)
FROM person
GROUP BY haircolor
```

Spaltenorientierte Datenbanken



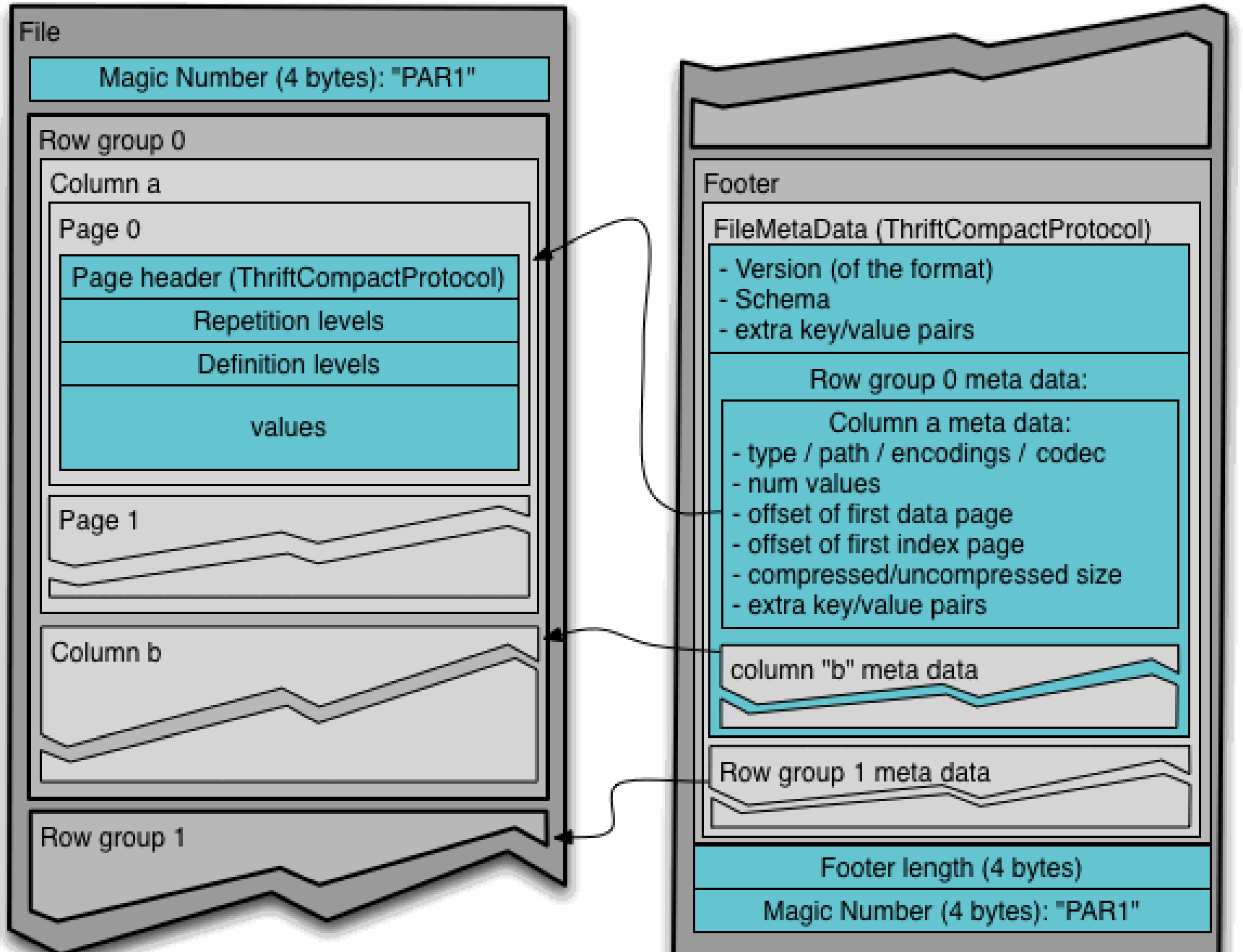
Apache Parquet

- Twitter/Cloudera 2012
- Spaltenorientiertes **Dateiformat** für das **Hadoop-Ökosystem**
- Verwendbar als Format für
  - Hive
  - MapReduce
  - Spark
  - …

Apache Parquet

- **Spaltenweise** Speicherung
- Spaltenweise **Kompression**
- Spaltenweiser **Zugriff**
- Effiziente **Kodierung**

Spaltenweise Speicherung

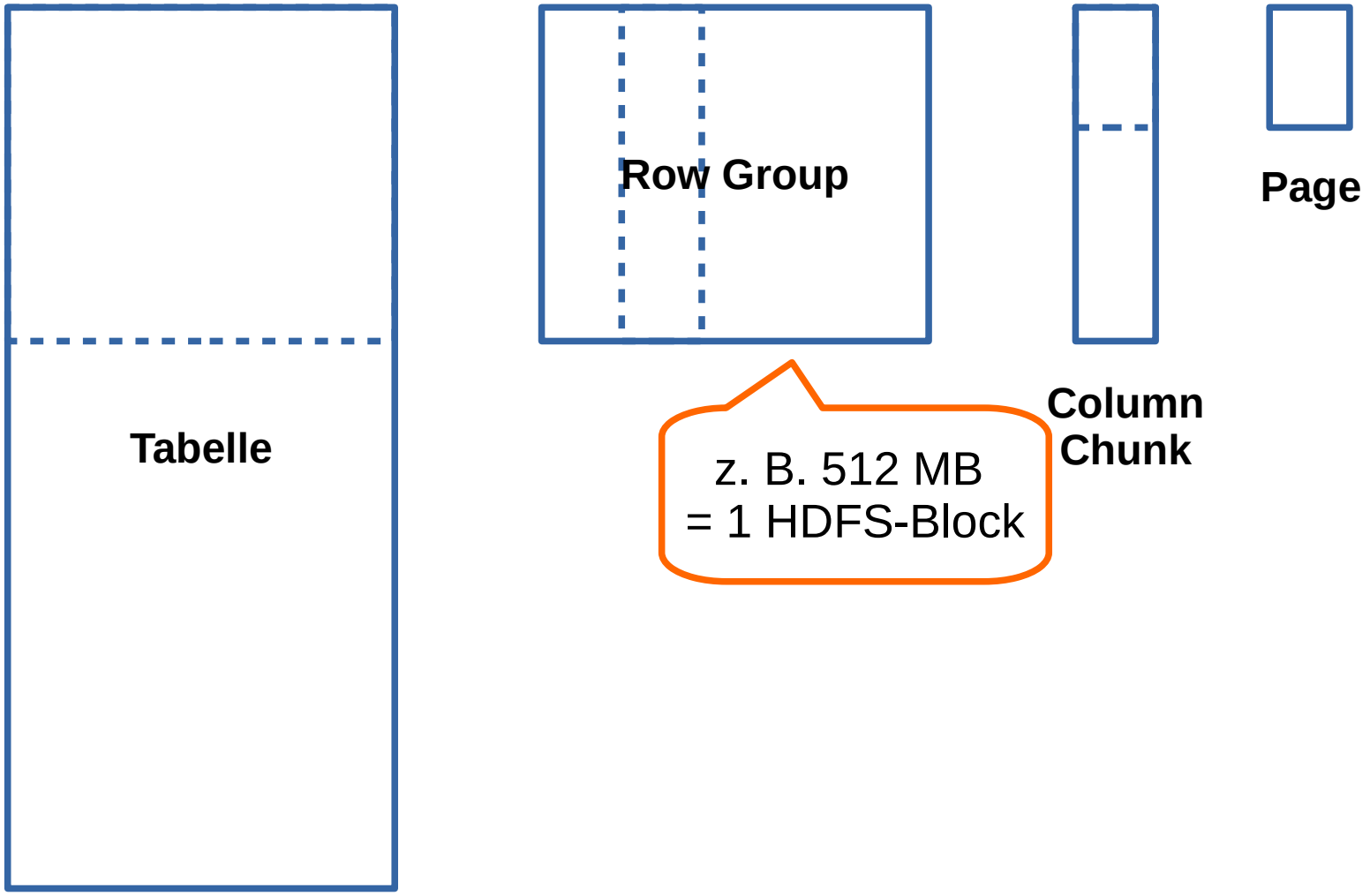


Quelle: parquet.apache.org

Spaltenweise Speicherung

- **Row Groups**
  - Partition von Zeilen in einem Datensatz
- **Column Chunk**
  - Teil einer Spalte innerhalb einer Row Group
  - Garantiert zusammenhängend
- **Seite (Page)**
  - Kleinste, unteilbare Einheit in einem Column Chunk

Speicherstruktur



Vorteile bis hierhin

- Nur **notwendige** Daten müssen gelesen werden
- Günstig für **Aggregatfunktionen**

red
blonde
brown
black
blonde
brown

Effiziente Codierung – Delta Coding

100	98	100	101	98	102	99	100	101	100

100	-2	2	1	-3	4	-3	1	1	-1

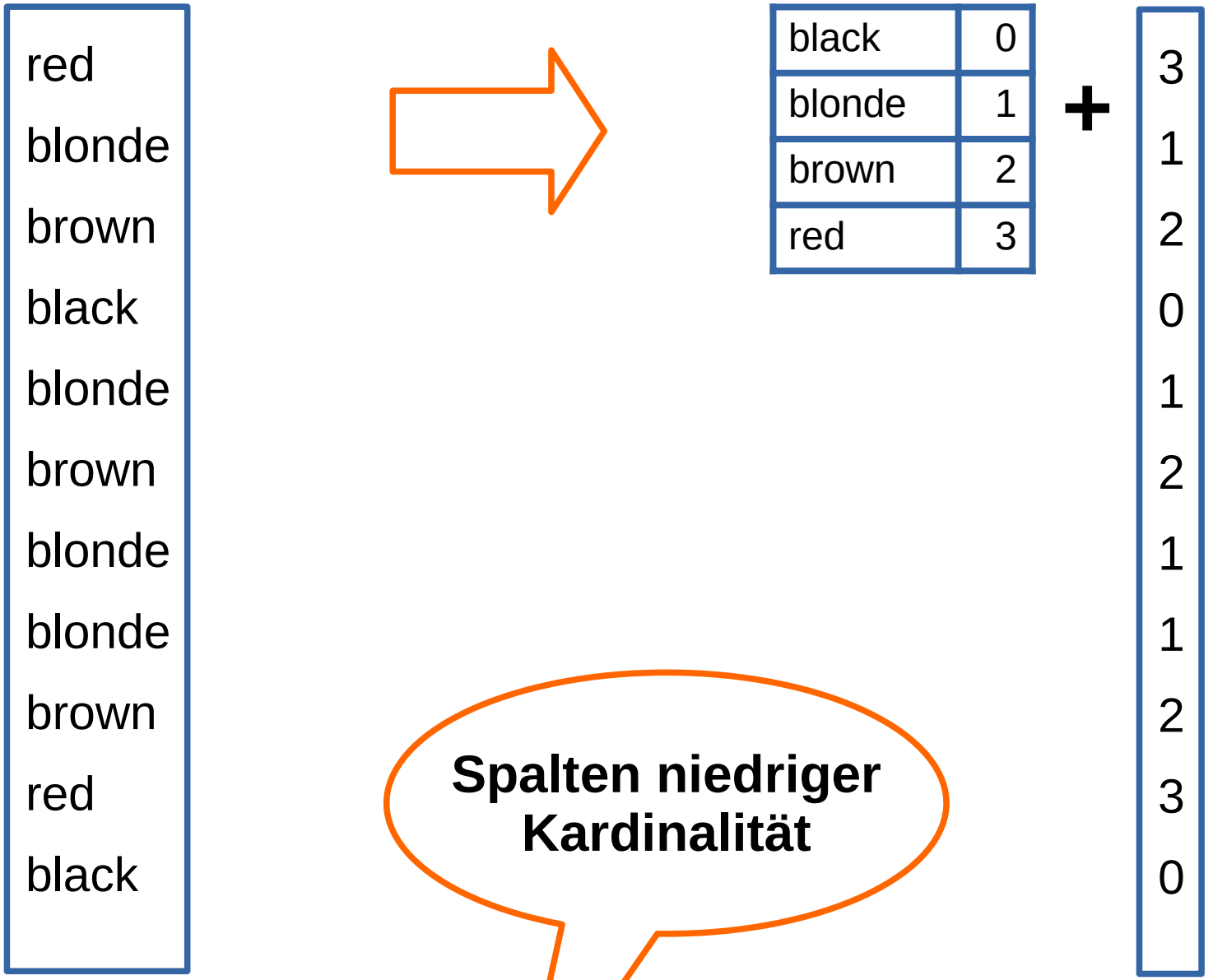
100	1	5	4	0	7	0	4	4	2

mindelta = -3

100	001	101	100	000	111	000	100	100	010

mindelta = -3
bits = 3

Effiziente Codierung – Dictionary

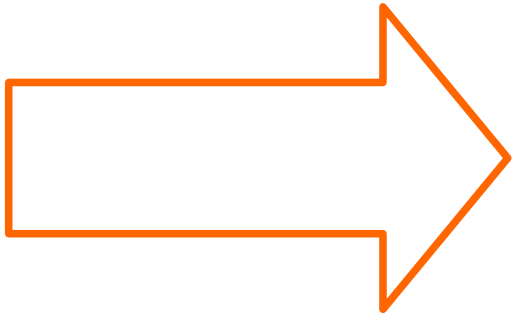


Effiziente Codierung – Run-Length Encoding

black	0
blonde	1
brown	2
red	3

+

3
3
3
3
1
1
1
1
2
2
2



black	0
blonde	1
brown	2
red	3

+

3	4x
1	4x
2	3x

Effiziente Codierung – Bit Packing

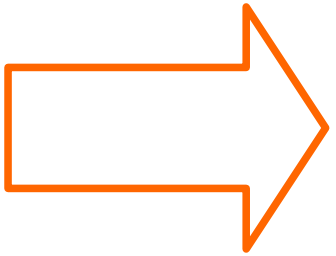
- Codiere kleine Zahlen mit **weniger Bits**
  - Markierung notwendig – wie viele Bits folgen?
- Beispiel:
  - Zahl beginnt mit 0 → 2 Bits folgen
  - Zahl beginnt mit 10 → 4 Bits folgen
  - Zahl beginnt mit 11 → 8 Bits folgen

Effiziente Codierung – Bit Packing

- Beispiel:
  - Zahl beginnt mit 0 → 2 Bits folgen
  - Zahl beginnt mit 10 → 4 Bits folgen
  - Zahl beginnt mit 11 → 8 Bits folgen
- 1 → **0**01
- 4 → **10**0000 (subtrahiere 4)
- 9 → **10**0101 (subtrahiere 4)
- 34 → **11**00001110 (subtrahiere 4 + 16)

Effiziente Codierung – Incremental Encoding für Strings

absurd
absyrtus
abundance
abundant
abundantly
abuse
abused
abuser
abuser
abusing
abuts



0 absurd
3 yrtus
2 undance
7 t
8 ly
3 se
5 d
5 r
5 r
4 ing
3 ts

Fazit: Kodierung

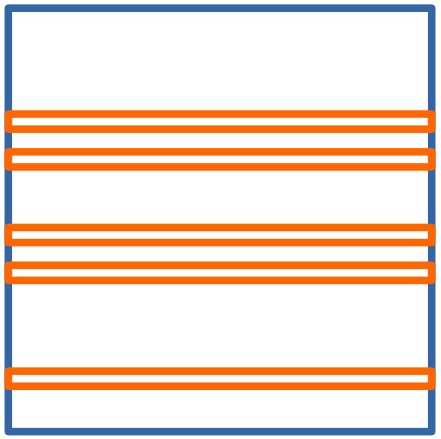
- **Kleine Zahlen** effizient kodieren
- **Wiederholungen** ausnutzen
- **Geringe Kardinalität** effizient kodieren

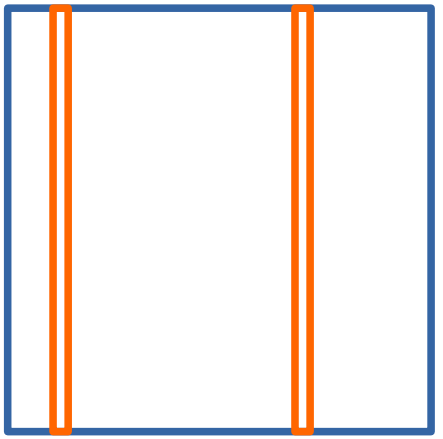
- Geringe Abstände → kleine Zahlen → …
- Verfahren kombinieren

Spalten vs. Zeilen

- **Zeilen**
- Einzel-Updates
- Einzel-Abfragen
- Frühe **Selektion**

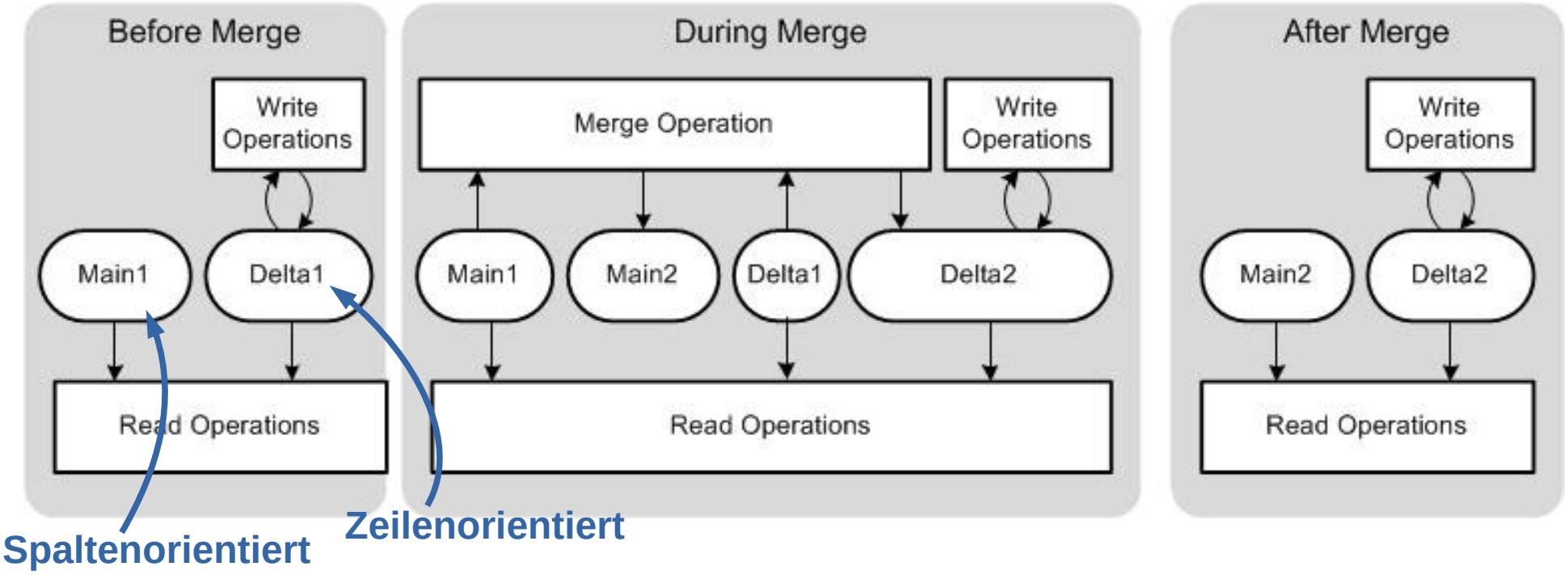
- **Spalten**
- Batch-Beladung
- Aggregatfunktionen
- Frühe **Projektion**





Konsolidieren von Spalten- und Zeilenorientierung

- Beispiel: **SAP HANA®**



Quelle: HANA®-Dokumentation (hana.ondemand.com) © SAP®

Zusammenfassung: Spaltenorientierung

- Spaltenweise Speicherung für **analytische Anwendungen**
- Ersparnis von I/O-Operationen bei **Aggregationen**
- **Effiziente Speicherung** durch Kompression

- Verwendung für **OLTP** schwierig
  → Kombinieren mit Zeilenorientierung
  → Redundanz, Umkopieren