

# Big-Data-Technologien

## **Kapitel 6: Batch-Algorithmen**

Hochschule Trier  
Prof. Dr. Christoph Schmitz

# Überblick

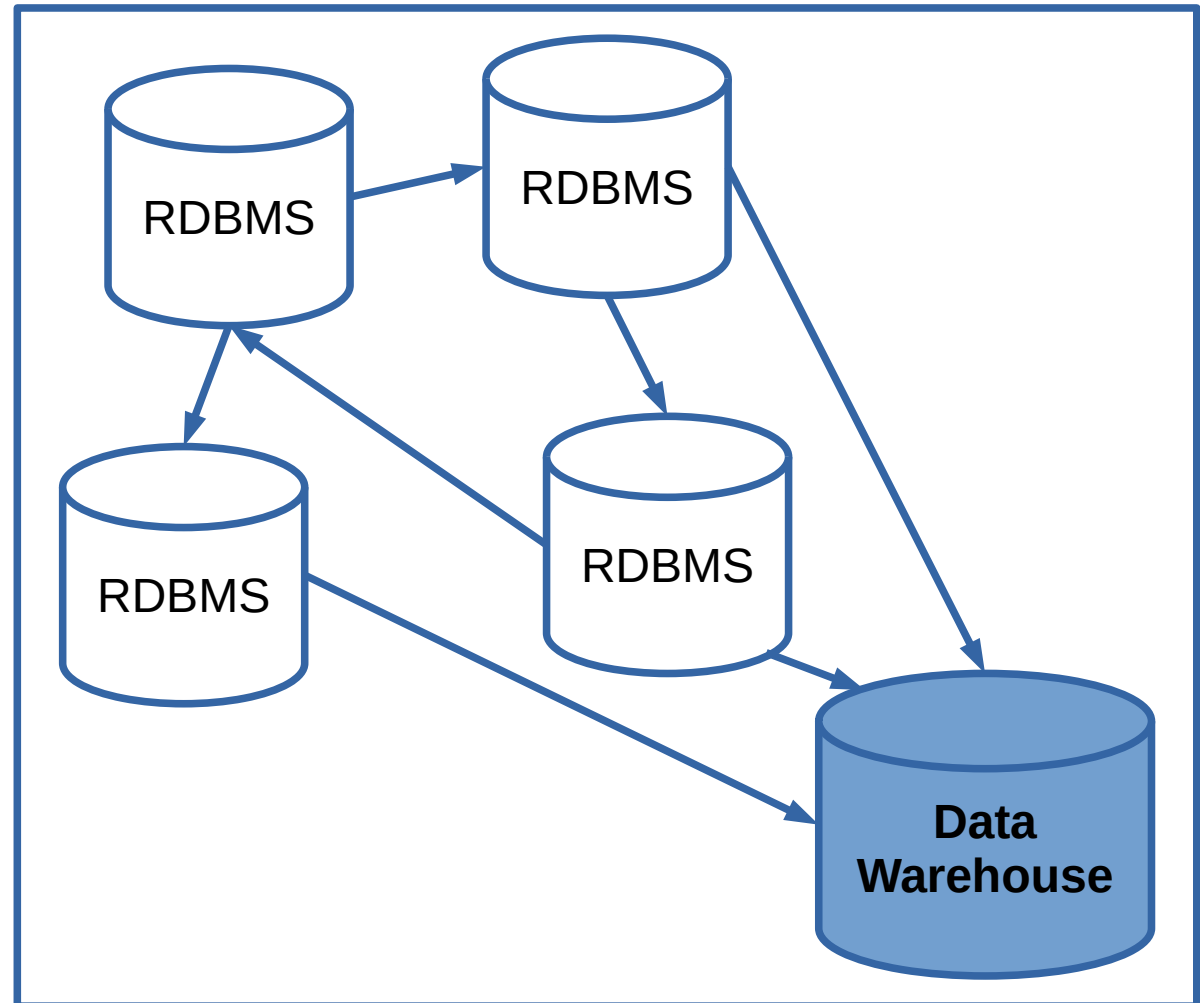
- Extract – Transform – Load
- Datenvorverarbeitung
  - Bereinigen/Filtern
  - Mappings
  - Normierung
- Invertierte Indizes
- Sortieren
- Aggregation
- Datenorganisation

# Database Offloading

- (Relationale) Datenbanksysteme entlasten
  - Lizenzkosten (pro CPU/pro GB/...)
  - teure Hardware
  - Administrationsaufwand
- „Einfache“ Operationen abnehmen
- Transfers zwischen Datenbanken

# Extract – Transform – Load (ETL)

- **Extract**
  - Abzug eines Teils des Quellsystems
- **Transform**
  - Bereinigung
  - Umkodierung/  
Anreicherung
  - Schema-  
Anpassung
- **Load**
  - Einfügen ins Zielsystem



# ETL: Extract

- Inkrementelle oder volle Abzüge
- **Vollabzug**
  - z. B. aus Backup
- **Inkrementeller Abzug**
  - `select * from tbl`  
`where timestamp >= 2017-03-01`



„Watermark“

# Transform: Bereinigung

- **Filtern**

- `!userAgent.contains("GoogleBot")`
- `date > 2017-01-01`
- `!(date > now())`

- **Bereinigen**

- `url.replace("/old/", "/new/")`
- `if (date.matches(oldDateFormat))`  
    `date = transformDate(date)`

Embarrassingly  
Parallel

# Transform: Mappings

- Mapping, engl. „Abbildung“
- Einfachster Fall:

```
switch (gender) {  
    case "m": return 0;  
    case "w": case "f": return 1;  
    default: return -1;  
}
```

**Embarrassingly  
Parallel**

# Mappings: Hash Join

- Größere Mappingtabelle, Konfigurierbarkeit → Tabelle wird **Konfiguration**

Embarrassingly  
Parallel

```
Map<Integer, String> plzOrt =  
    readMapping(...);
```

```
...  
for (Integer plz: ...) {  
    String ort = plzOrt.get(plz);  
    context.write(ort);  
}
```

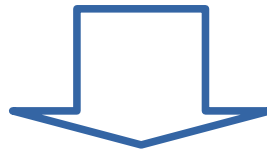
PLZ	Ort
54329	Konz
54290	Trier
54292	Trier
54293	Trier
...	...



# Transform: Normierung (I)

- Beispiel: Clicks pro Tag pro User

UID	Mo	Di	Mi	Do	Fr	Sa	So
alice	5	3	2	3	5	2	0
bob	23	17	10	25	20	17	8



UID	Mo	Di	Mi	Do	Fr	Sa	So
alice	0.25	0.15	0.10	0.15	0.25	0.10	0.00
bob	0.23	0.17	0.10	0.25	0.20	0.17	0.08

Embarrassingly  
Parallel

# Transform: Normierung (II)

- Beispiel: Normiere Altersskala auf [0, 1]

UID	Alter
Alice	16
Bob	42
Charlie	20
David	38
Eve	67
Frank	43

$\min(\text{Alter}) = 16$   
 $\max(\text{Alter}) = 67$

UID	Alter
Alice	0
Bob	0.51
Charlie	0.08
David	0.43
Eve	1
Frank	0.53

→ **Aggregation** notwendig  
(über gesamten Datensatz!)

Embarrassingly  
Parallel

# Invertierte Indizes

# Invertierte Indizes

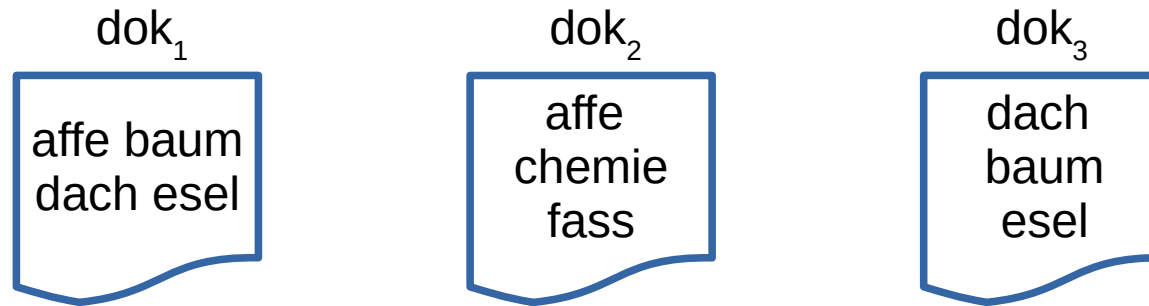
- Indexstrukturen für **Suchmaschinen**
- Ursprüngliche Verwendung von Hadoop (ca. 2005)
- Idee: gegeben Dokumente  $\text{dok}_1, \text{dok}_2, \dots$ , baue eine Struktur

$\text{wort}_1 \rightarrow \text{dok}_1, \text{dok}_5, \text{dok}_7$

$\text{wort}_2 \rightarrow \text{dok}_2, \text{dok}_4, \text{dok}_{12}, \text{dok}_{21}$

$\text{wort}_3 \rightarrow \dots$

# Aufbau eines invertierten Index



## Map

(affe,  $\text{dok}_1$ )    (baum,  $\text{dok}_1$ )    (dach,  $\text{dok}_1$ )    (esel,  $\text{dok}_1$ )    (affe,  $\text{dok}_2$ )  
(chemie,  $\text{dok}_2$ )    (fass,  $\text{dok}_2$ )    (dach,  $\text{dok}_3$ )    (baum,  $\text{dok}_3$ )    (esel,  $\text{dok}_3$ )

## Shuffle

(affe, ( $\text{dok}_1$ ,  $\text{dok}_2$ ))    (baum, ( $\text{dok}_1$ ,  $\text{dok}_3$ ))    (chemie, ( $\text{dok}_2$ ))  
(dach, ( $\text{dok}_1$ ,  $\text{dok}_3$ ))    (esel, ( $\text{dok}_1$ ,  $\text{dok}_3$ ))    (fass, ( $\text{dok}_2$ ))

## Reduce

affe  $\rightarrow$   $\text{dok}_1$ ,  $\text{dok}_2$   
baum  $\rightarrow$   $\text{dok}_1$ ,  $\text{dok}_3$   
...

# (Tera|Peta)byte Sort

# Sortieren ist doch eigentlich gelöst?

- Sortiert wird im **Shuffle**-Schritt  
(in Spark: zwischen Stages)
- Aber:
  - Globale Sortierung?
  - Verteilung der Daten → Partitionierung?
- Beispiel
  - **Sortiere Wörter in großen Mengen Text**

# Idee: Partitioniere Wörter nach Anfangsbuchstaben

- **Mapper:**
  - gebe jedes Wort einzeln aus

brot aal baum  
claus ast chemie

- **Partitioner:**
  - $\text{partition} \leftarrow \text{wort}[0] - 'a'$

aal  
ast

brot  
baum

claus  
chemie

- **Shuffle** sortiert Partitionen  $\rightarrow$   $\text{sorted}_a, \dots$

aal  
ast

baum  
brot

chemie  
claus

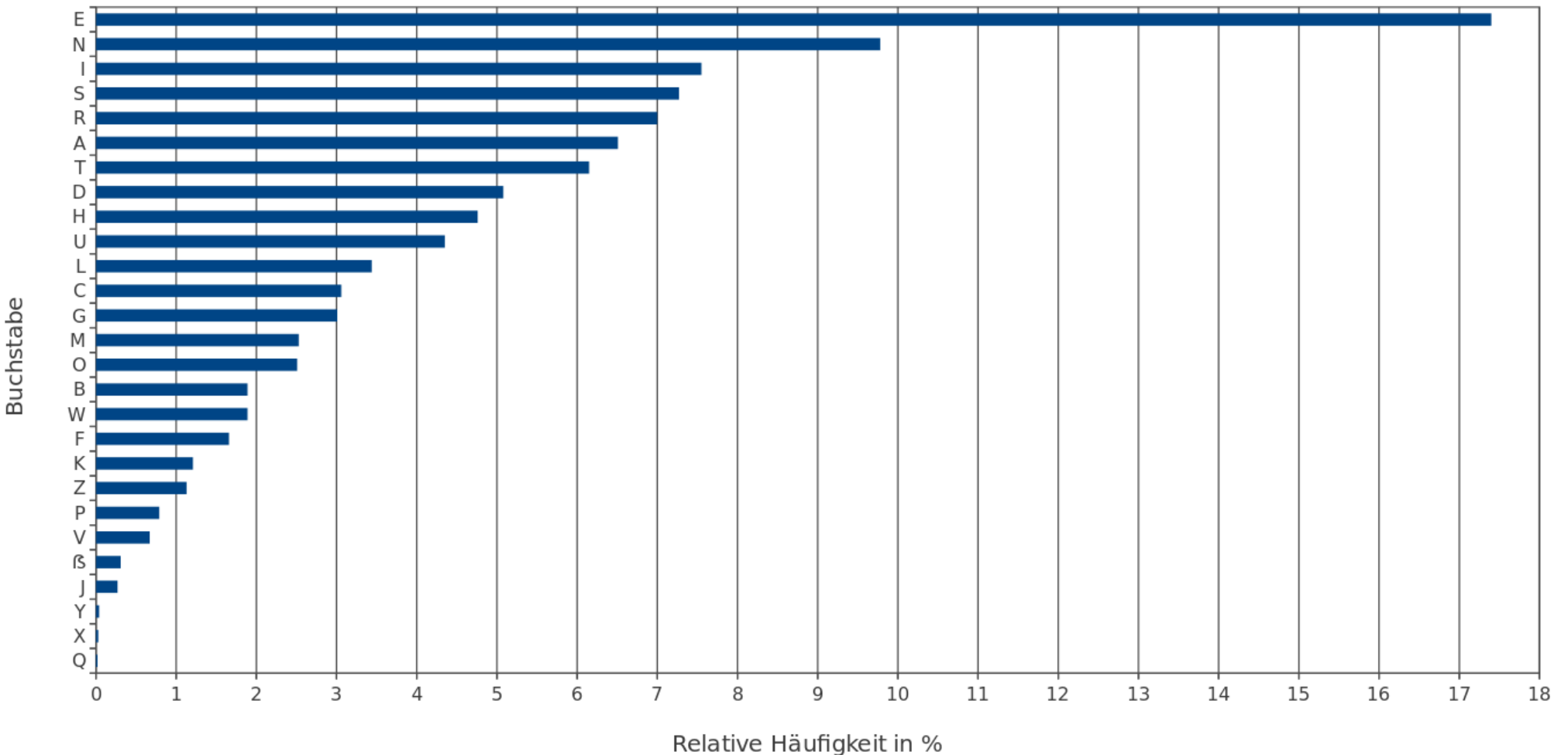
- **Reducer** tut nichts
- Ergebnis
  - $\text{concat}(\text{sorted}_a, \text{sorted}_b, \dots, \text{sorted}_z)$

aal ast baum  
brot chemie claus



# Problem: Schiefe Buchstabenverteilung

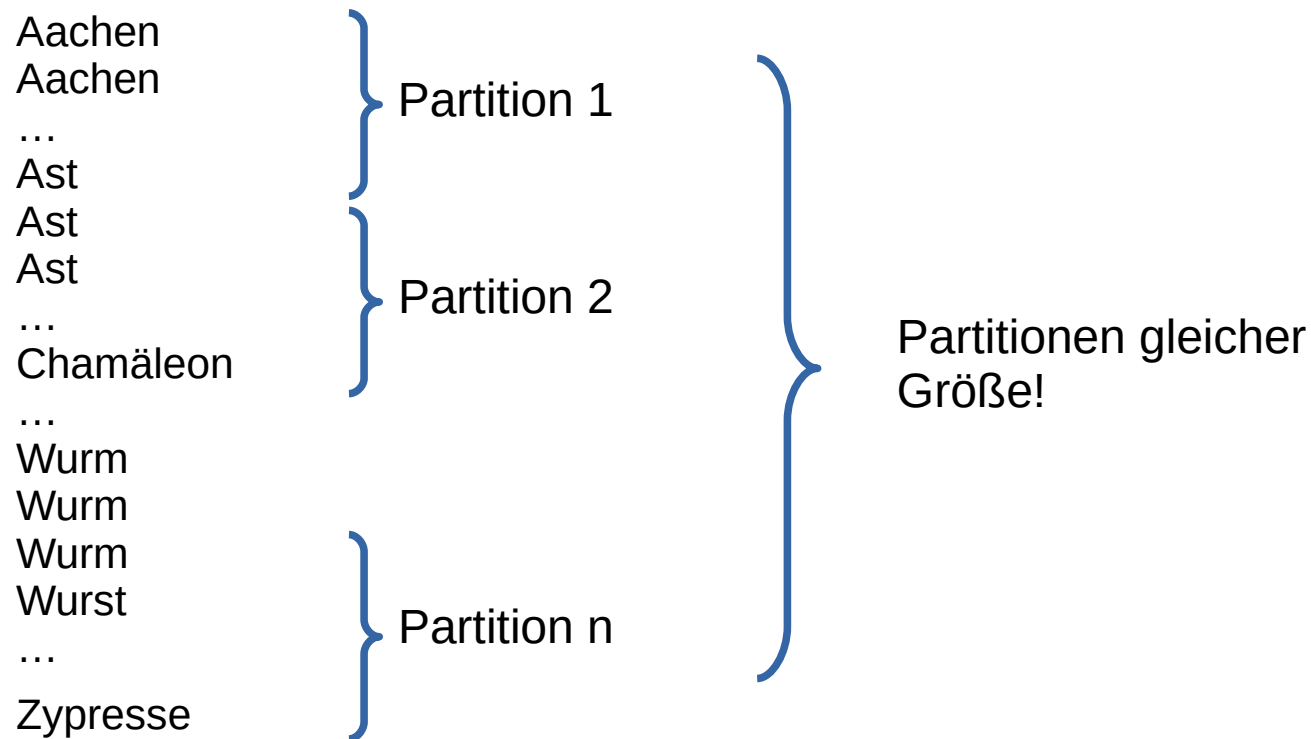
Buchstabenhäufigkeiten in deutschsprachigen Texten



Quelle: Wikipedia

# Abhilfe für schiefe Verteilung

- Erhebe **Quantile** der Eingabedaten (auf Stichprobe)



- Grenzen der Partitionen sind Konfiguration für den **Partitioner**:

quantiles  $\leftarrow$  [Aachen, Ast, Chamäleon, ..., Wurm, Zypresse]

# Probleme gelöst!

- Partitionierung nach Quantilen  
löst Probleme des naiven Ansatzes:
  - schiefe Verteilung
  - fixe Anzahl von Reducern

→ **effizienter und besser skalierbar**
- Nachteile
  - **zwei Durchläufe** nötig...
  - ... oder **repräsentative Stichprobe**

# Aggregatfunktionen

# Aggregatfunktionen

- Aggregatfunktionen fassen viele gleichartige Werte zu einem **Aggregat** zusammen:

$$\text{agg} = f(a_1, a_2, \dots, a_k)$$

- Beispiele
  - Summe/Anzahl
  - Minimum/Maximum
  - Mittelwert
  - Median
  - Varianz/Standardabweichung
  - Histogramm/empirische Verteilung
  - Mengenkardinalität

New Page  
Select Page...

## Customer Satisfaction

93.13%

Previous  
79.82% Change  
+14.29Trend  
▲

## Gross Profit

\$192.13M

Previous  
\$183.61M% Change  
+4.43%Trend  
▲

## New Customers

10,719

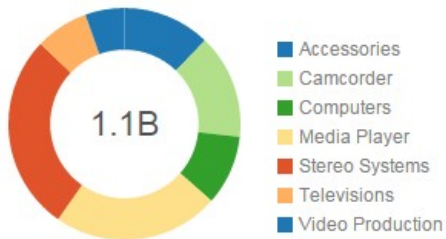
Previous  
11,918% Change  
-11.21%Trend  
▼

## Revenue

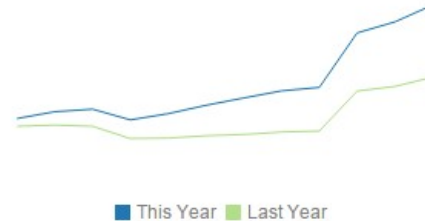
\$1,061M

Previous  
\$906M% Change  
+14.61%Trend  
▲

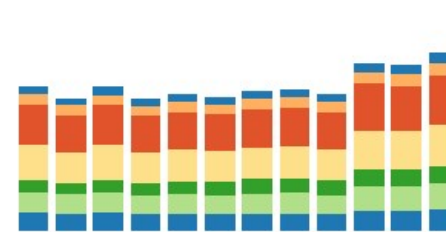
## Sales by Product Category



## Sales Comparison



## Sales by Month



## Brand Profitability



## Sales by Country

Sales by Country | Sales by Month | Sales by State



2 Comments

## Products

Products	Region	Stores	Customers	Comments			
Subcategory	Gross Profit	Discount	MSRP	COGS	Qty		
Blu Ray	\$51,771,195	\$10,895,633	243,779,705	\$181,112,921	679,495		
Speaker Kits	\$25,819,242	\$4,954,243	112,169,618	\$81,396,140	244,199		
Headphones	\$24,523,024	\$3,516,913	79,703,501	\$51,663,564	228,349		
Handheld	\$21,393,655	\$1,959,624	43,930,192	\$20,576,916	250,167		
Standard	\$19,369,668	\$3,214,787	71,656,083	\$49,071,633	192,205		
Video Editing	\$17,947,620	\$2,695,891	60,749,162	\$40,105,657	199,749		
Tablet	\$17,674,116	\$2,018,135	45,464,132	\$25,771,890	146,728		
Receivers	\$16,555,836	\$2,643,045	59,528,536	\$40,329,668	150,568		
Flat Panel TV	\$15,885,499	\$3,478,829	78,441,670	\$59,077,345	92,501		
Smartphone	\$15,834,702	\$2,790,776	62,661,241	\$44,035,774	205,049		
Professional	\$8,835,523	\$1,933,997	45,987,828	\$35,218,308	12,872		
Charger	\$1,970,124	\$187,486	4,210,324	\$2,052,711	105,257		
Streaming	\$1,936,587	\$338,560	7,339,881	\$5,064,730	67,910		

# Wünschenswerte Eigenschaften von Aggregatfunktionen

- **kommutativ:**  
 $f(a, b) = f(b, a)$
- **assoziativ:**  
 $f(a, f(b, c)) = f(f(a, b), c) =: f(a, b, c)$
- **algebraisch:**  
 $h(a, b, c, d, \dots) = g(f(a, b, c, d)),$   
f kommutativ und assoziativ
- Kommutativ und assoziativ  $\rightarrow$  algebraisch
- Algebraisch  $\nrightarrow$  kommutativ und assoziativ

# Eigenschaften konkreter Aggregatfunktionen

- **kommutativ und assoziativ:**
  - Summe/Anzahl
  - Minimum/Maximum
  - Histogramm/empirische Verteilung
- **algebraisch:**
  - Mittelwerte
  - Varianz, Standardabweichung
- **keins von beiden:**
  - Mengenkardinalität
  - Quantile/Median
  - Top-k (häufigste Werte)



# Beispiele

- **Summe** ist kommutativ und assoziativ


$$(a + b) + (c + d) = (b + a + d) + c = a + b + c + d$$

- **Maximum** ist kommutativ und assoziativ

$$\max(\max(4, 5), 2) = \max(4, \max(2, 5)) = 5$$

# Beispiele

- **Arithmetisches Mittel** ist algebraisch, aber nicht assoziativ:

$$\begin{aligned}\text{mean}(\text{mean}(3, 4, 5), 6) &= \text{mean}(4, 6) = 5 \\ \text{mean}(3, 4, 5, 6) &= 4.5\end{aligned}$$


- Aber:

$$\text{mean}(3, 4, 5, 6) = \text{sum}(3, 4, 5, 6) / |\{3, 4, 5, 6\}|$$

Summe, Anzahl sind assoziativ und kommutativ!

# Berechnung algebraischer Aggregatfunktionen

- **Map:** Überführen der Daten zu atomaren Aggregaten

```
map(key, value):  
    write(key, atom(value))
```

4 9 7 5 5

(4, 1) (9, 1) (7, 1)  
(5, 1) (5, 1)

- **Reduce:** Berechnen der Aggregatfunktion

```
reduce(key, values):  
    agg ← 0  
    foreach value in values:  
        agg ← f(agg, value)  
    write(key, agg)
```

$(4 + 9 + 7 + 5 + 5, 1 + 1 + 1 + 1 + 1)$   
 $= (30, 5)$

- **Zusammenfassen:**

```
finish(result):  
    write(key, g(result))
```

(30, 5)

$30/5 = 6$

# Beispiele

Aggregatfunktion	Atomares Aggregat für x	Neutrales Element	Aggregation	Zusammenfassung
Anzahl	1	0	$u, v \rightarrow u + v$	-
Summe	x	0	$u, v \rightarrow u + v$	-
Arithmetisches Mittel	(x, 1)	(0, 0)	$(x, c), (y, d) \rightarrow (x + y, c + d)$	$(x, c) \rightarrow x/c$
Minimum	x	$\infty$	$u, v \rightarrow \min(u, v)$	-

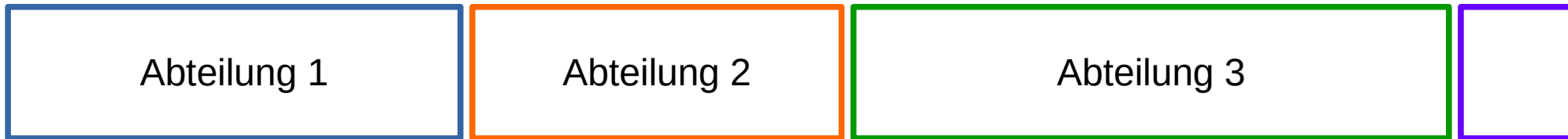
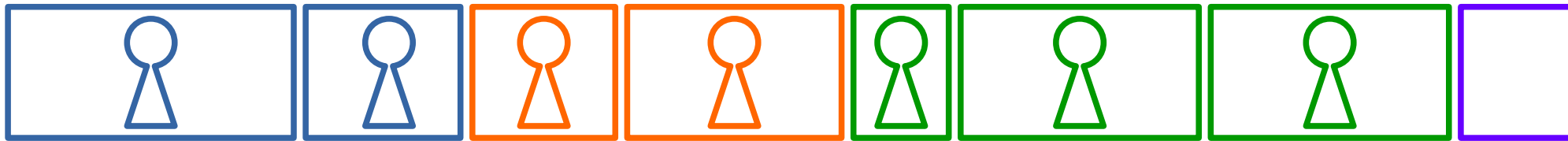
# Zwischenfazit: Aggregatfunktionen

- Zusammenfassung vieler Werte zu einem
- Möglichst kommutative und assoziative Funktionen
- Zur Not algebraische Funktionen
- **Alle anderen sind unangenehm!**

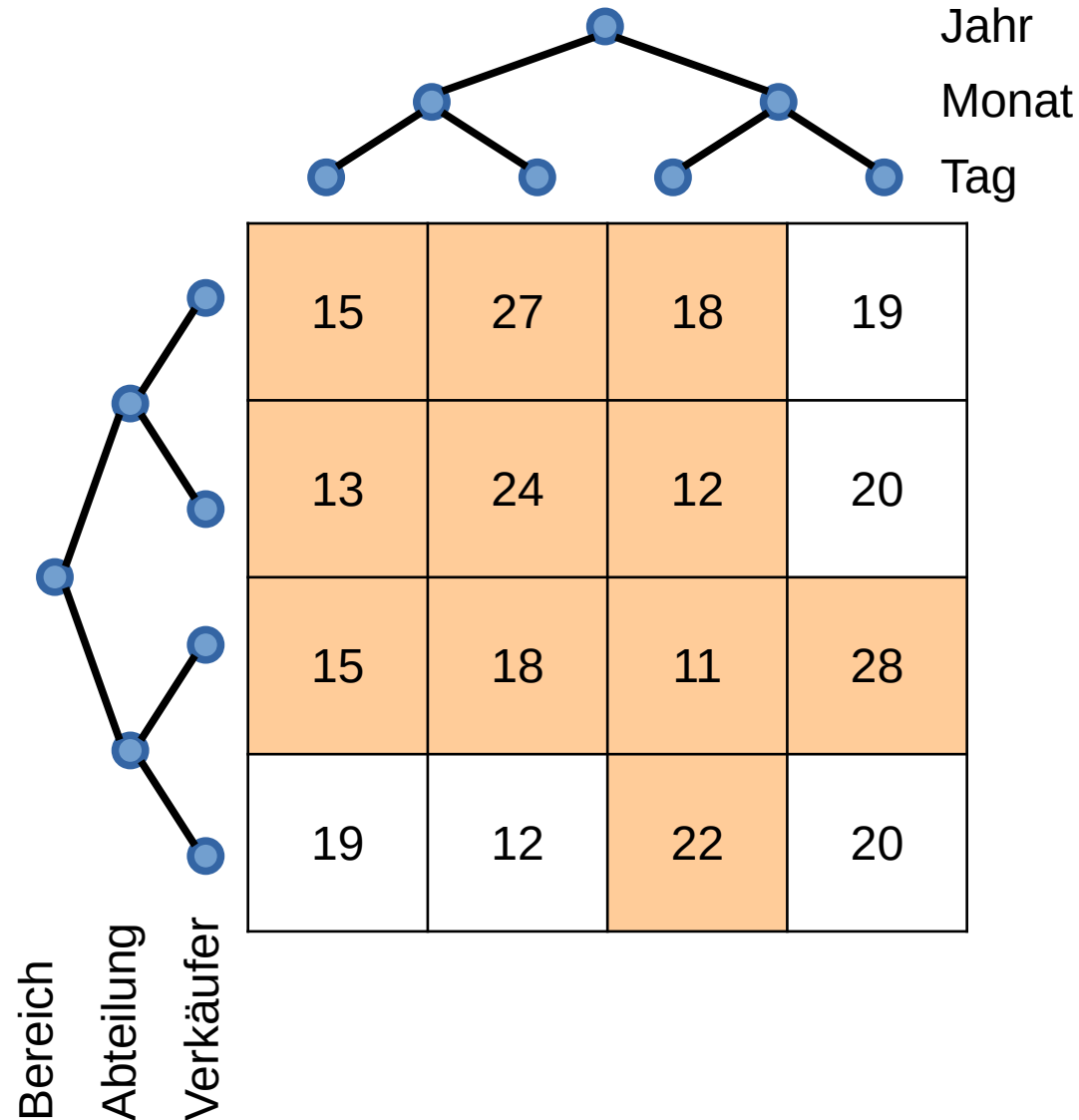
# Zwischenergebnisse

- Viele Domänen sind **hierarchisch** gegliedert
  - Stunde → Tag → Woche
  - Tag → Monat → Jahr
  - Stadt → Kreis → Land → Staat
  - Mitarbeiter → Abteilung → Bereich → Firma
- Idee: Vorberechnen von **Teilaggregaten**

# Beispiel: Umsätze pro Verkäufer, Abteilung, ...



# Kombination hierarchischer Dimensionen

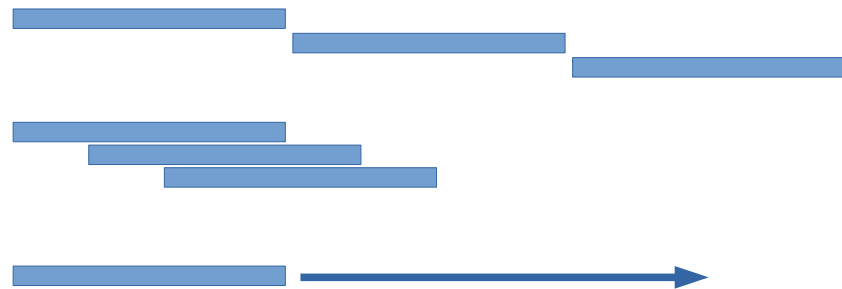


„Cube“



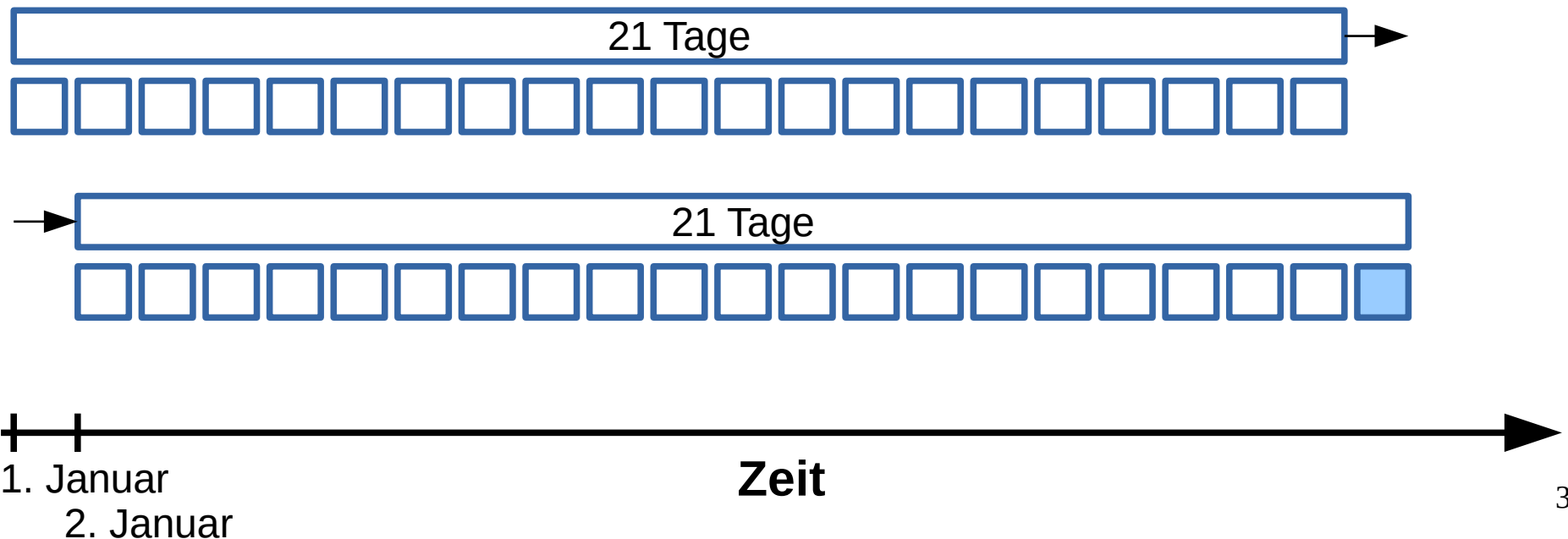
# Zwischenergebnisse (II): Zeitfenster

- Aggregiere Daten über Zeitraum
- Zeitraum verschiebt sich jeden Tag/jede Stunde/jede Woche/...
- Verschiedene Arten von Fenstern:
  - Tumbling Windows
  - Hopping Windows
  - Sliding Windows



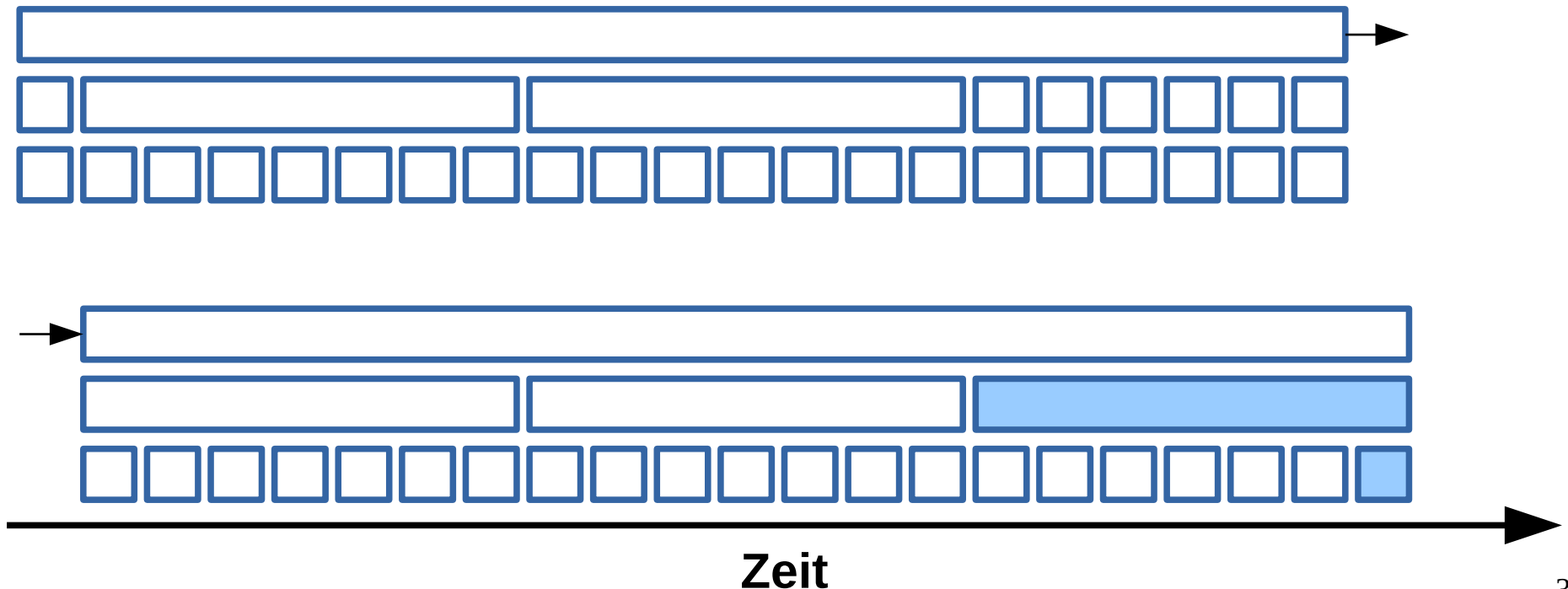
# Gleitende Zeitfenster

- Typische Metriken:
  - „Anzahl der Klicks in den letzten 21 Tagen“
  - „Verkäufe in den letzten 12 Monaten“



# Gleitende Zeitfenster

- Typische Metriken:
  - „Anzahl der Klicks in den letzten 21 Tagen“
  - „Verkäufe in den letzten 12 Monaten“



# Zusammenfassung: Aggregatfunktionen

- Unterschiedliche Arten von Aggregatfunktionen
  - **kommutativ und assoziativ**
  - **algebraisch**
  - weder noch
- Algebraische Aggregatfunktionen ermöglichen **Zwischenaggregate**
- Zwischenaggregate sind Grundlage für
  - analytische Anwendungen
  - gleitende Zeitfenster

# Datenorganisation

- **Strukturierung** von
  - Eingangsdaten
  - Zwischenergebnissen

- **Grundlage** für

- Beschaffung
- Verwendung
- Entsorgung

von Daten

} **Informationslebenszyklus**

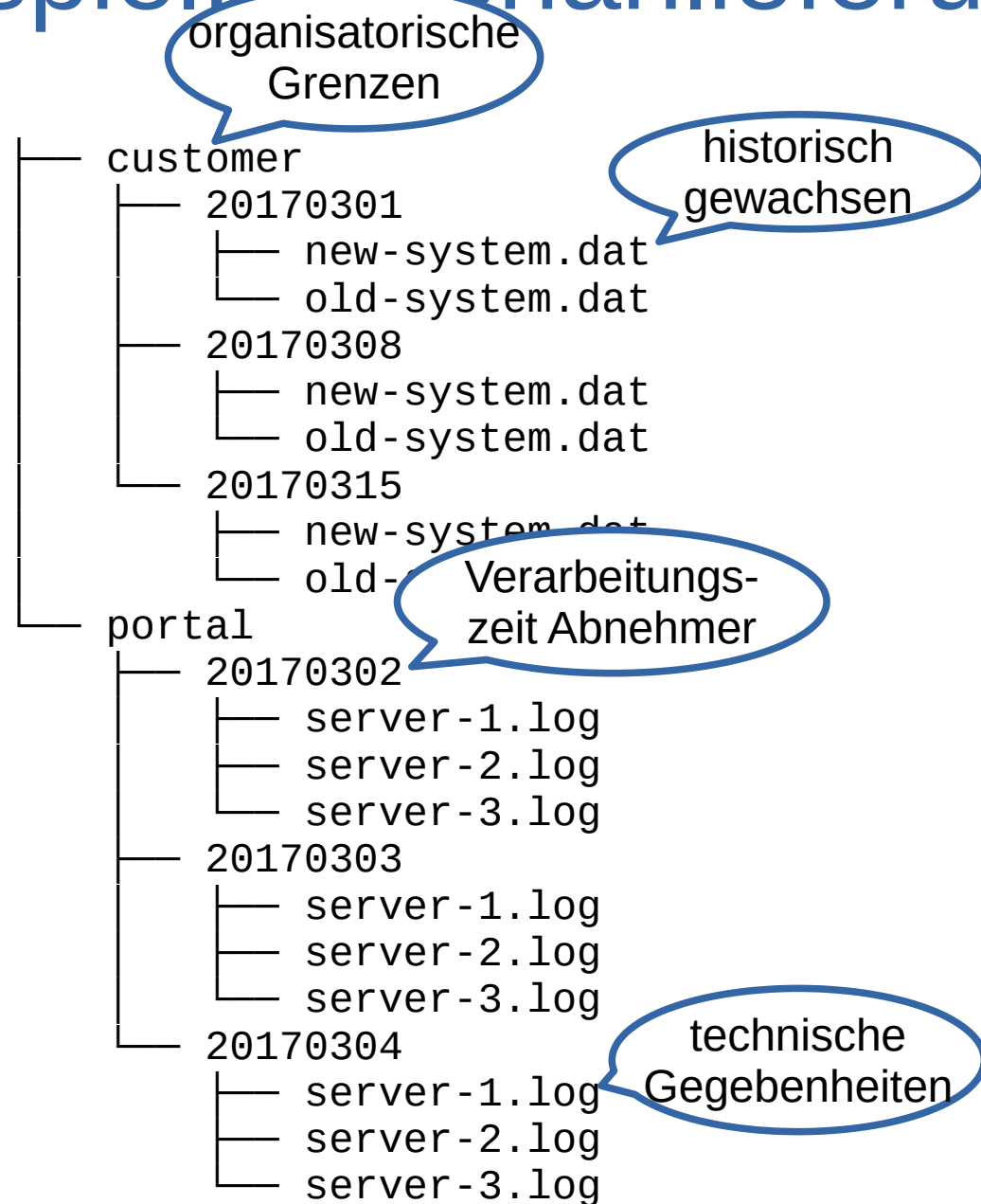
# Datenorganisation: Dimensionen

- **Fachliche** Dimensionen
  - Fachliche Objekte (Kunden, Artikel, ...)
  - Zeitstempel
- **Technische** Dimensionen
  - Lieferndes System
  - Zeitstempel
- **Organisatorische** Dimensionen
  - Lieferant
  - Abnehmer

# Datenorganisation: Beschaffung

- Anlieferung typischerweise **periodisch**
  - täglicher/wöchentlicher Datenbankabzug
  - stündliche/tägliche Logfiles
- Partitionierung oft **technisch** motiviert
  - ein Logfile pro Server pro Stunde
- Verschiedene **Zeitbegriffe**
  - Eventzeit
  - Verarbeitungszeit Lieferant
  - Verarbeitungszeit Abnehmer

# Beispiel: Datenanlieferung





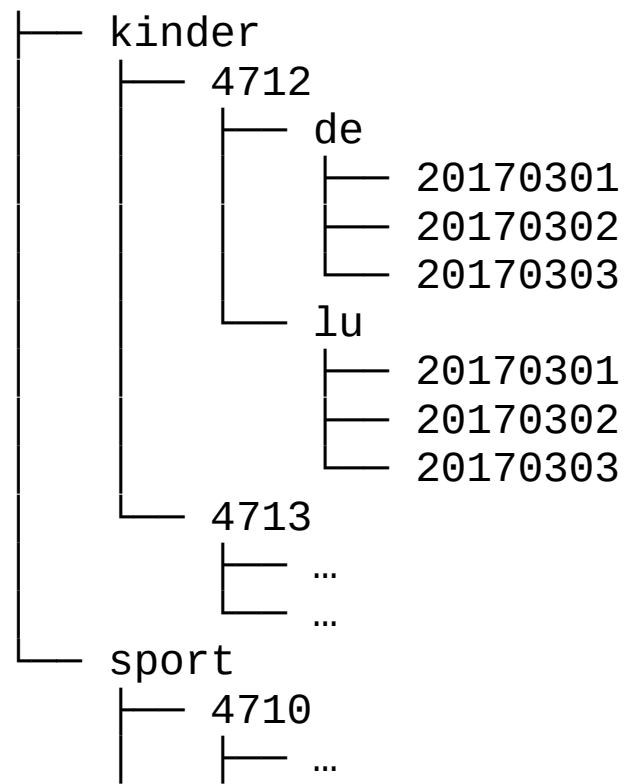
# Datenorganisation: Verwendung

- **Partitionierung** bestimmt
  - mögliche Parallelität
  - Anzahl der Tasks
  - (un)günstige Dimensionen
- notfalls
  - **Umpartitionieren**
  - **Redundanz** in Kauf nehmen

# Beispiel: Verkäufe Online-Shop

Datum	Artikel	Kategorie	Kunde	Region	...
2017-03-02	4711	Sport	cust-1	DE	...
2017-03-02	4712	Kinder	cust-2	LU	...
...	...	...	...	...	...

Idee: Partitioniere über **alle** relevanten Dimensionen!



- ✓ Alle Kombinationen direkt wählbar
- ✓ Keine weitere Infrastruktur nötig
- ✗ Fragmentierung
- ✗ Reihenfolge der Dimensionen?

# Datenorganisation: Entsorgung

- Lösche alle Daten **älter als x Tage**
  - einfach bei zeitlicher Partitionierung
  - ... aber sonst?
- Werden diese Daten **noch gebraucht?**
  - Metadaten über Abhängigkeiten  
→ **Lineage**
  - organisatorische Regelungen  
→ **Data Governance**
- **Archivieren** statt Löschen

```
customers/  
├── 20170301  
├── 20170308  
└── 20170315
```

# Datenorganisation: Metadaten

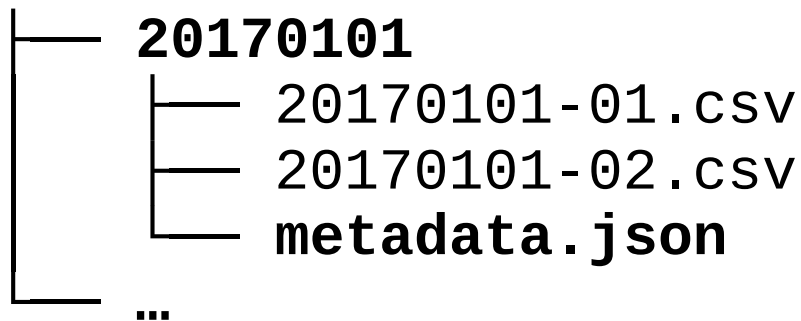
- Metadaten = „Daten über Daten“
- Beispiele
  - Schema
  - Zeitstempel
  - Größe
  - ...

```
$ ls -l data.txt
```

```
-rw-rw-r-- 1 schmi schmi 231 Feb  6 13:05 data.txt
```

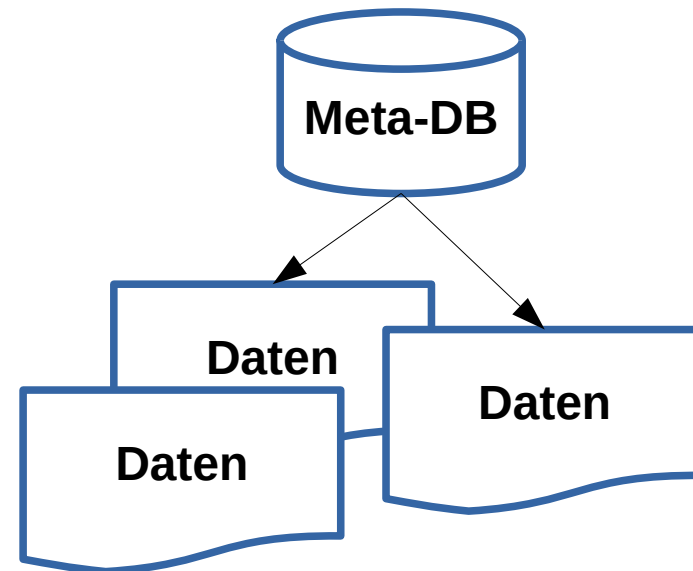
# Datenorganisation: Metadaten

- **Filesystem**



- ✓ testbar
- ✓ Metadaten „immer dabei“
- ✓ keine gesonderte Infrastruktur
- ✗ umständlich
- ✗ keine zentrale Verwaltung

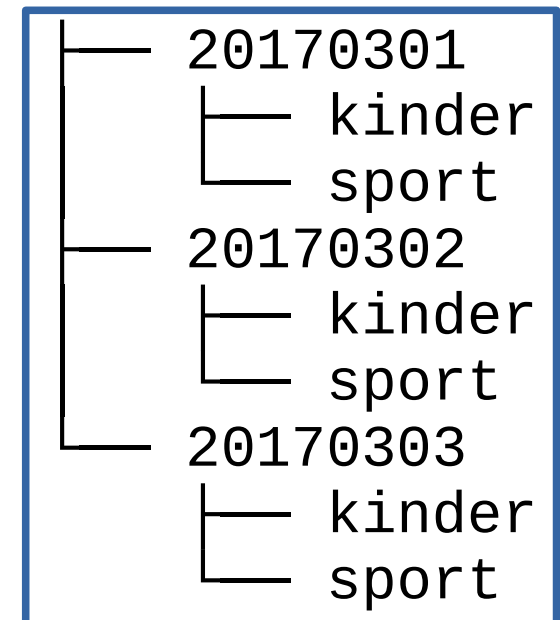
- **Datenbank**



- ✓ beliebiges Datenmodell
- ✓ anfragbar
- ✗ schlechter testbar
- ✗ Infrastruktur notwendig

# Datenorganisation: Vorschlag

- Partitioniere nach dem **wichtigsten fachlichen Kriterium**
- **Zeitstempel** muss sein wegen Entsorgung
- Zur Not **unpartitionieren**/Redundanz zulassen
- **Metadaten** im Filesystem



# Überblick

- Extract – Transform – Load
- Datenvorverarbeitung
  - Bereinigen/Filtern
  - Mappings
  - Normierung
- Invertierte Indizes
- Sortieren
- Aggregation
- Datenorganisation