

Data Warehousing

Abschluss

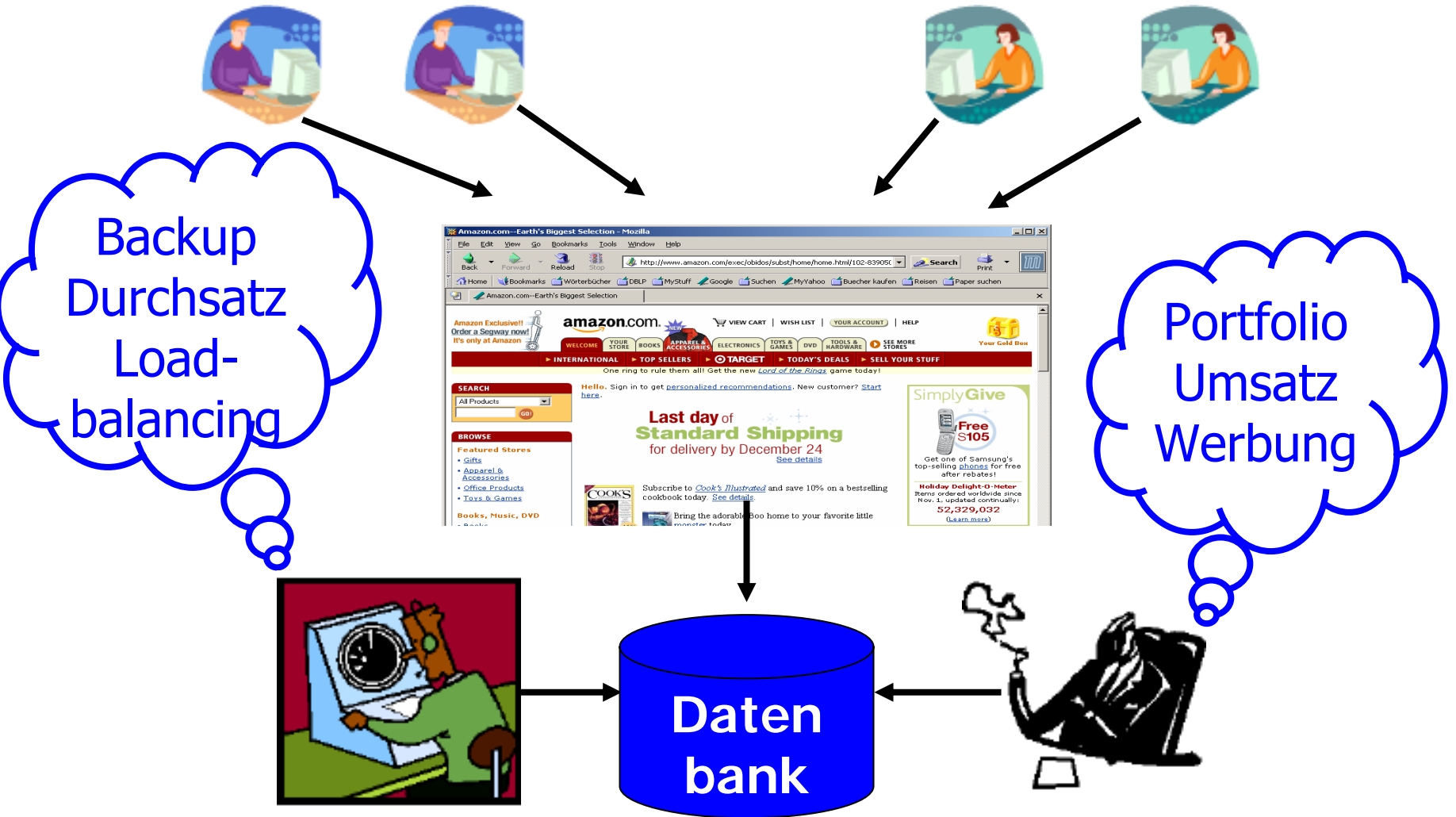
Ulf Leser

Wissensmanagement in der
Bioinformatik



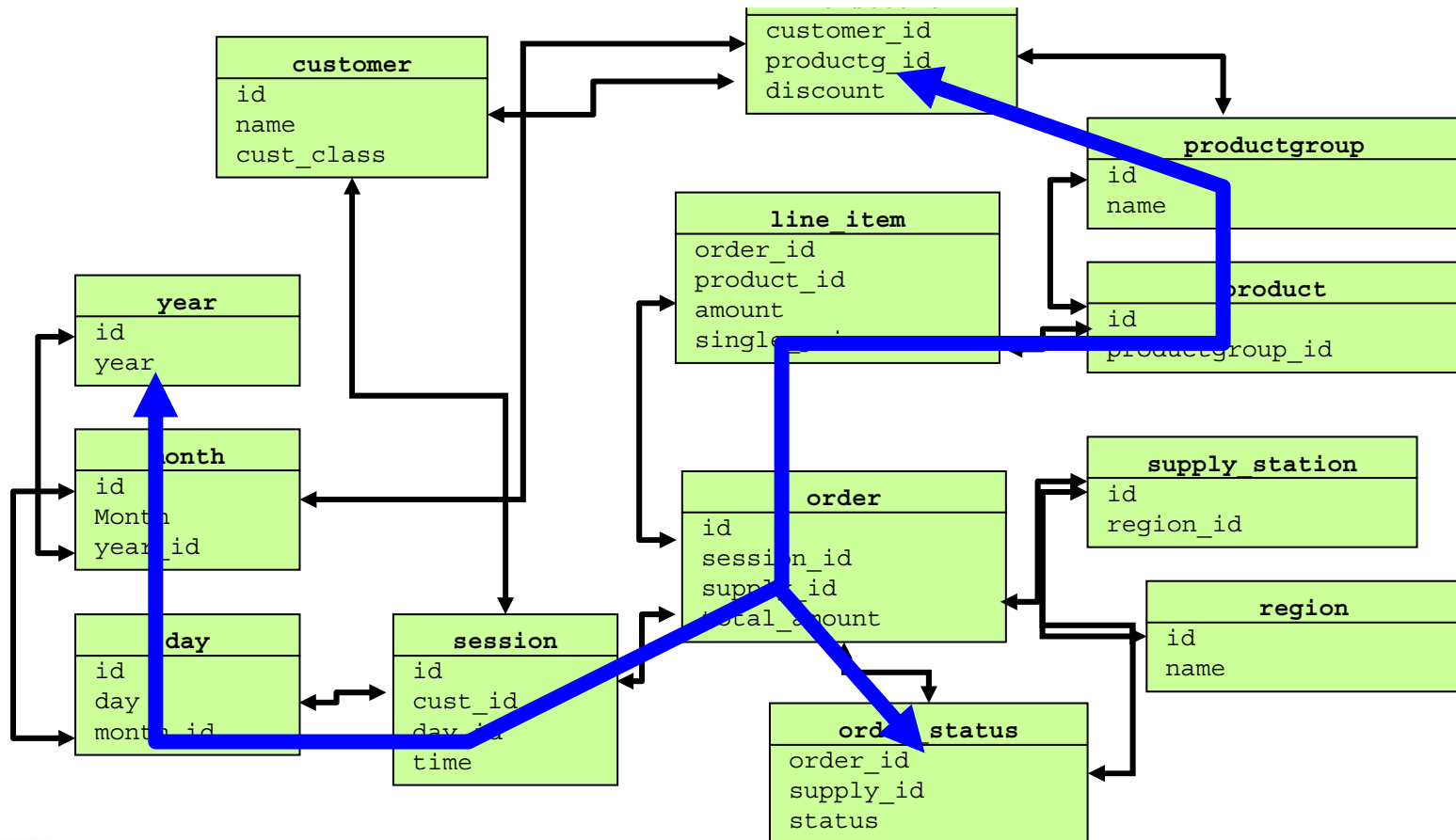
-
- Zusammenfassung
 - Ein paar Worte zur Prüfung
 - Rückschau
 - Feedback
 - Werbung

Bücher im Internet bestellen

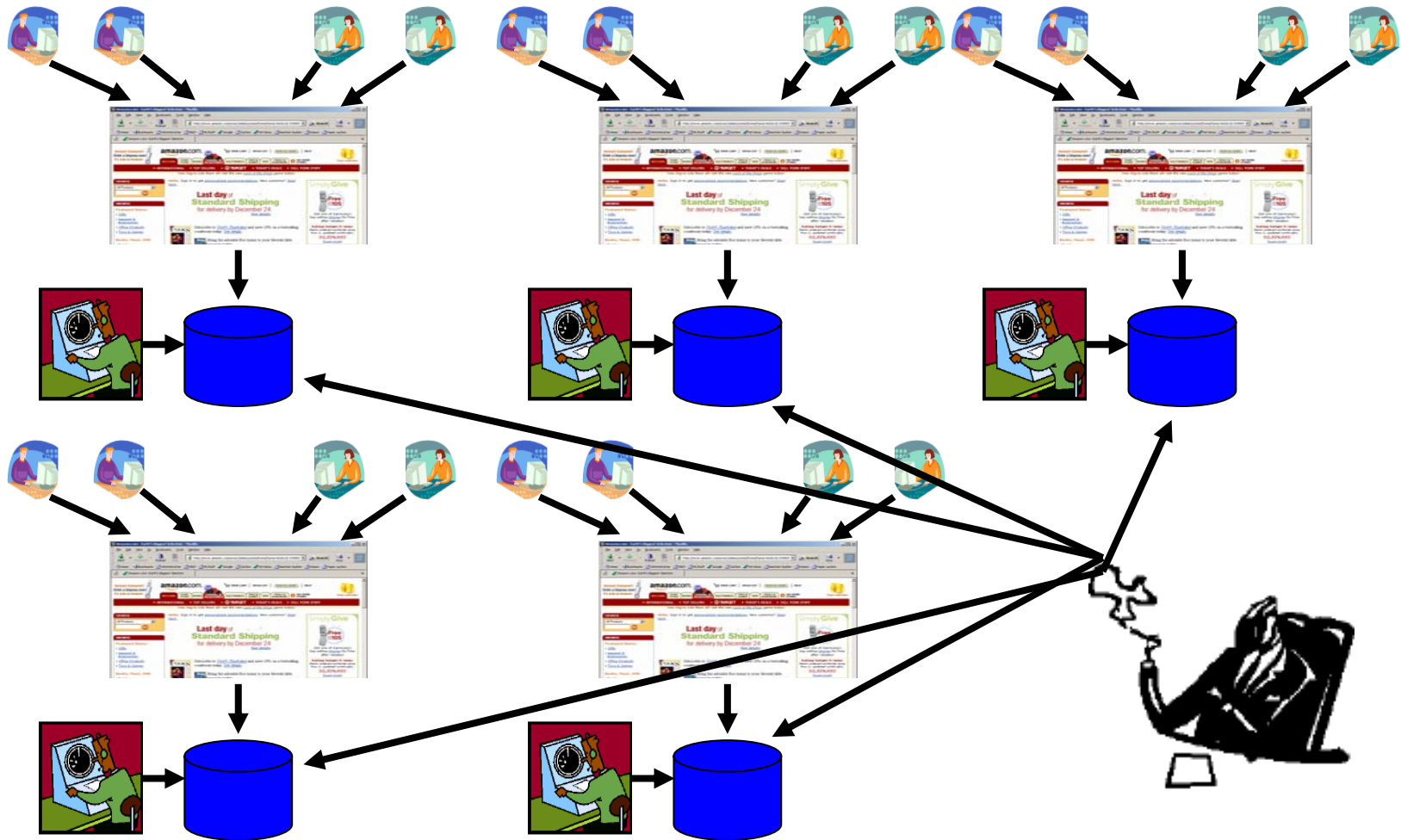


Fragen eines Marketingleiters

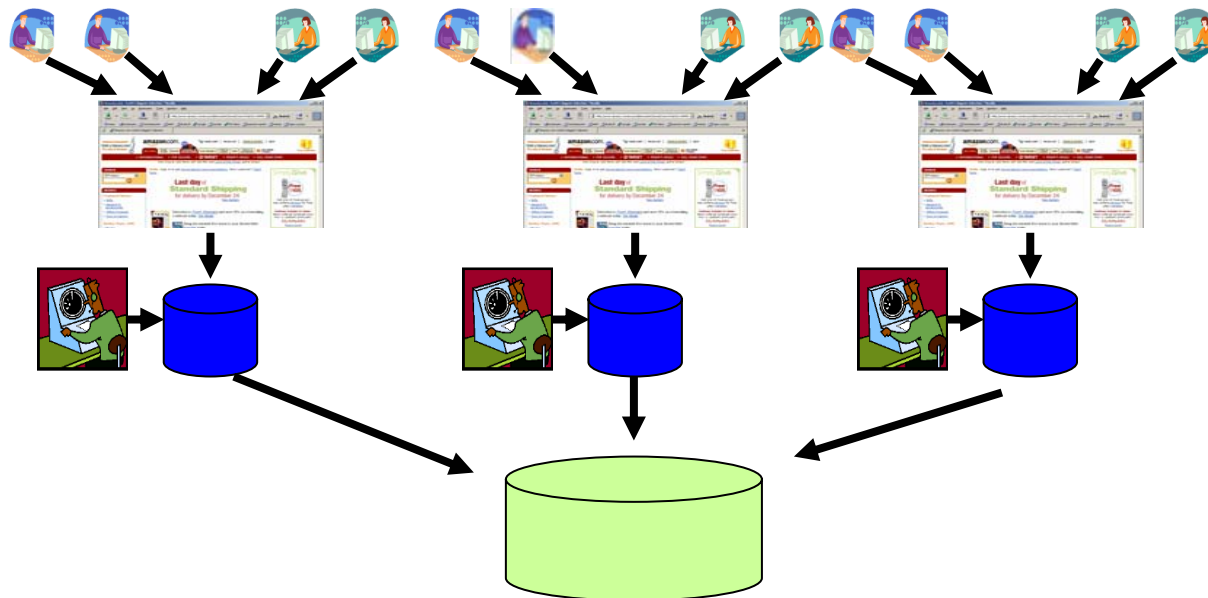
Wie viele abgeschlossene Bestellungen haben wir jeweils im Monat vor Weihnachten, aufgeschlüsselt nach Produktgruppen und Promotion?



In Wahrheit ...



Besser: Data Warehouse

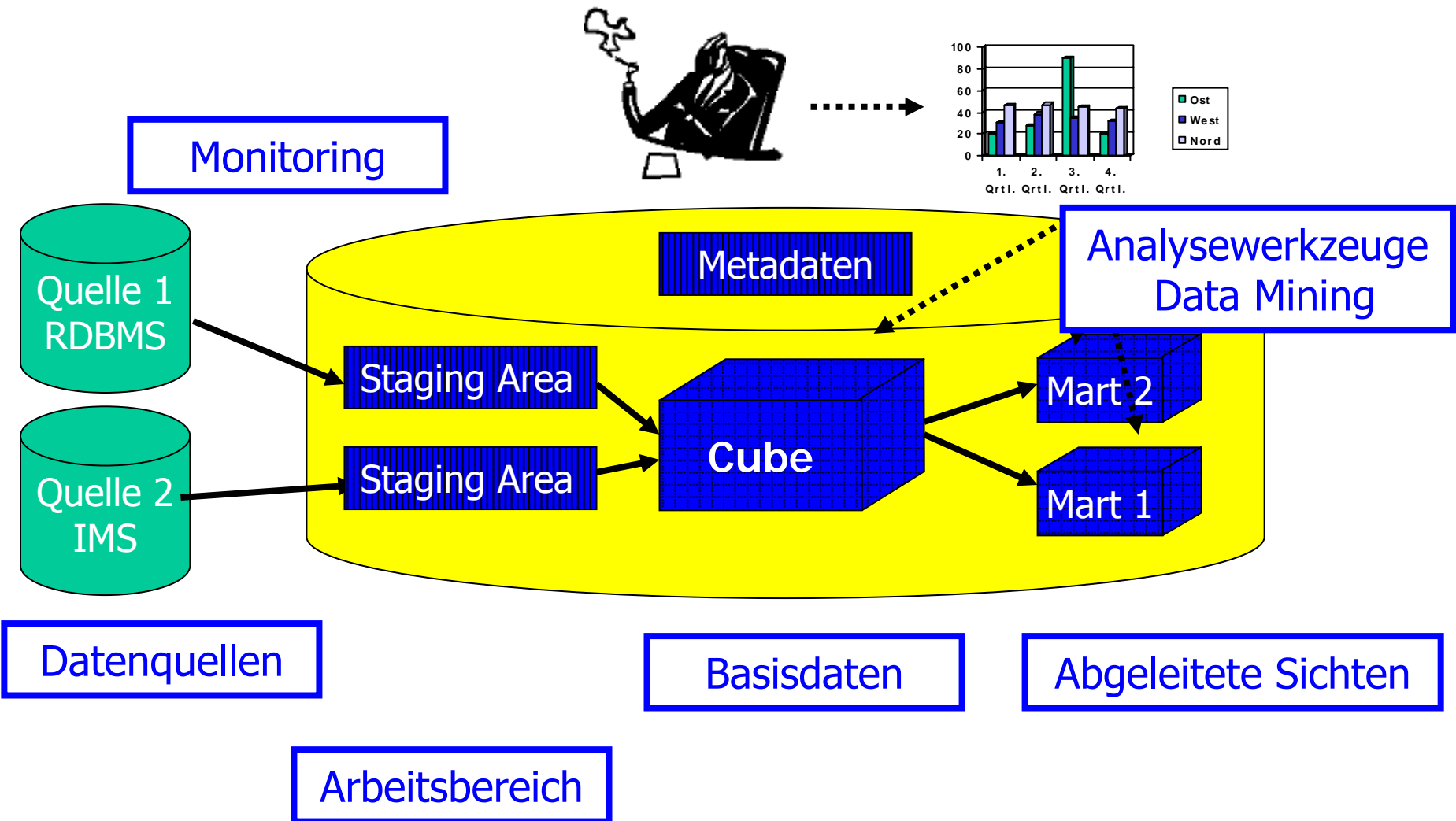


- Redundante Datenhaltung
- Spezielle Modellierung
- Transformierte und selektierte Daten
- Asynchrone Aktualisierung

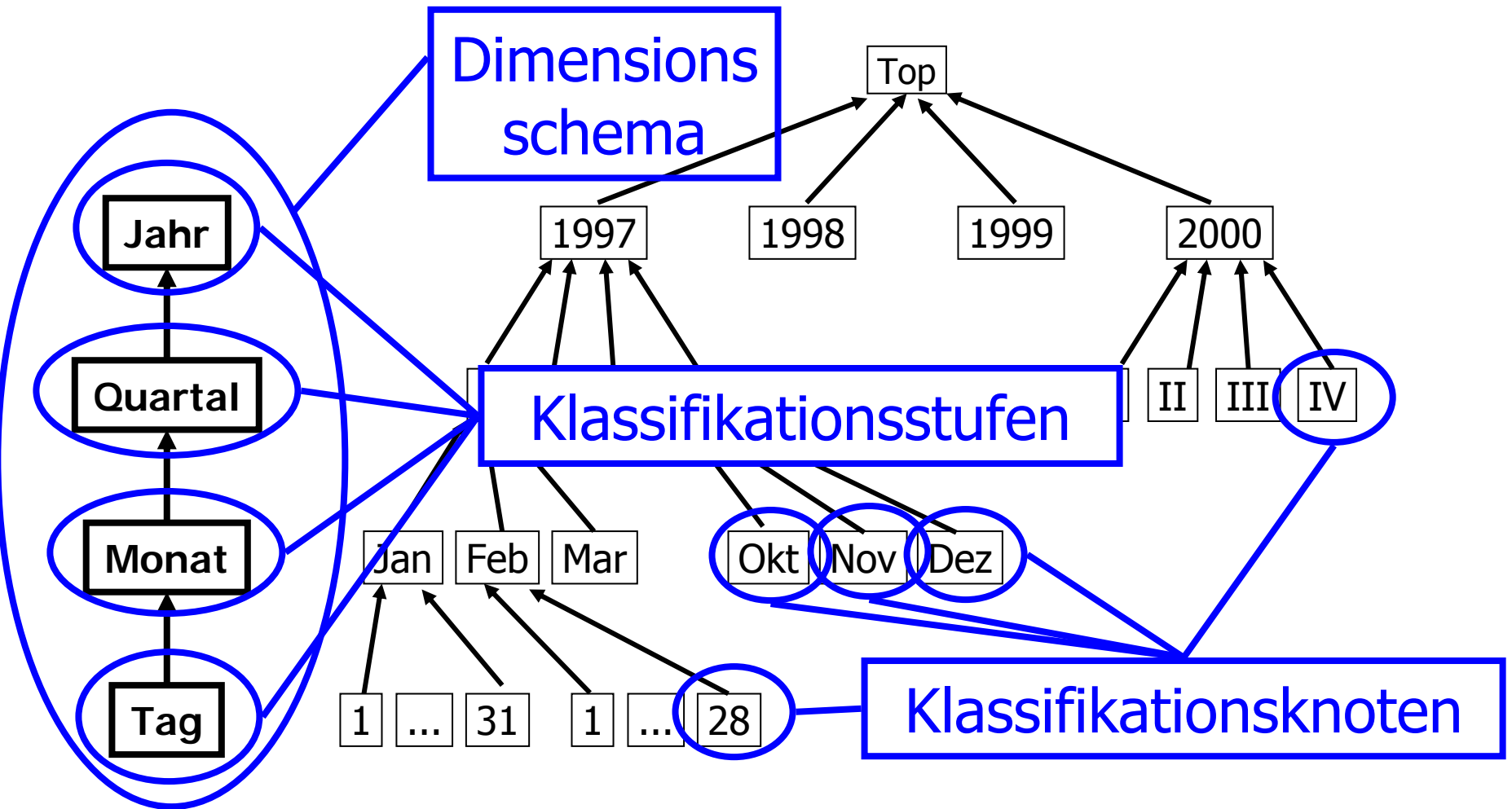
OLAP versus OLTP

	OLTP	OLAP
Typische Operationen	Insert, Update, Delete, Select	Select Bulk-Inserts
Transaktionen	Viele und kurz	Lesetransaktionen
Typische Anfragen	Einfache Queries, Primärschlüsselzugriff, Schnelle Abfolgen von Selects/inserts/updates/deletes	Komplexe Queries: Aggregate, Gruppierung, Subselects, etc. Bereichsanfragen über mehrere Attribute
Daten pro Operation	Wenige Tupel	Mega-/ Gigabyte
Datenmenge in DB	Gigabyte	Terabyte
Eigenschaften der Daten	Rohdaten, häufige Änderungen	Abgeleitete Daten, historisch & stabil
Erwartete Antwortzeiten	Echtzeit bis wenige Sekunden	Minuten
Modellierung	Anwendungsorientiert	Themenorientiert
Typische Benutzer	Sachbearbeiter	Management

DWH Architektur & Komponenten



Dimension



Würfelschema und Würfel

- Definition

Ein *Würfelschema* WS ist ein Tupel (G, F) mit

- Einer Granularität G
- Einer Menge F von Fakten mit $|F|=m$

- Ein *Würfel* W ist eine Instanz eines Würfelschema (G, F)

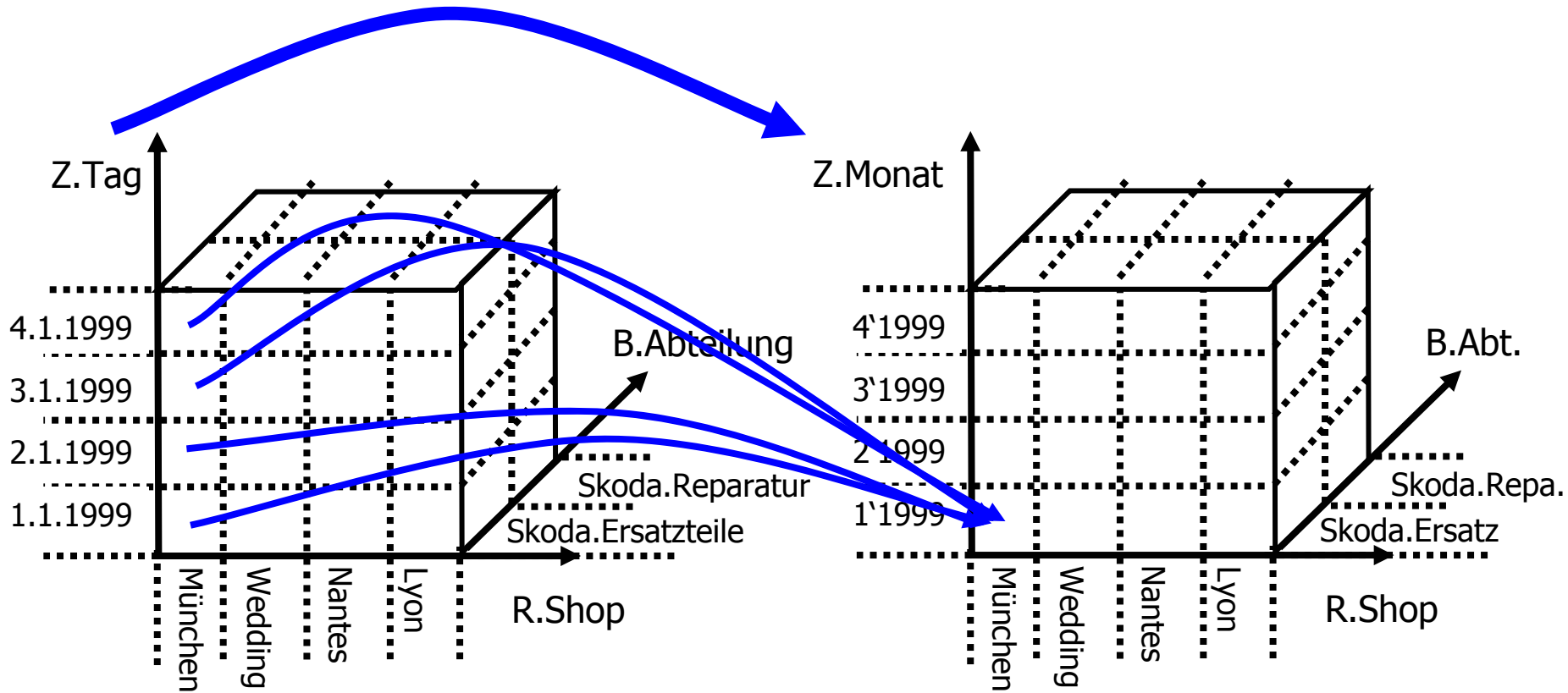
$$W = \text{dom}(G) \times \text{dom}(F)$$

$$= \text{knoten}(D_1.k_1) \times \dots \times \text{knoten}(D_n.k_n) \times \text{dom}(F_1) \times \dots \times \text{dom}(F_m)$$

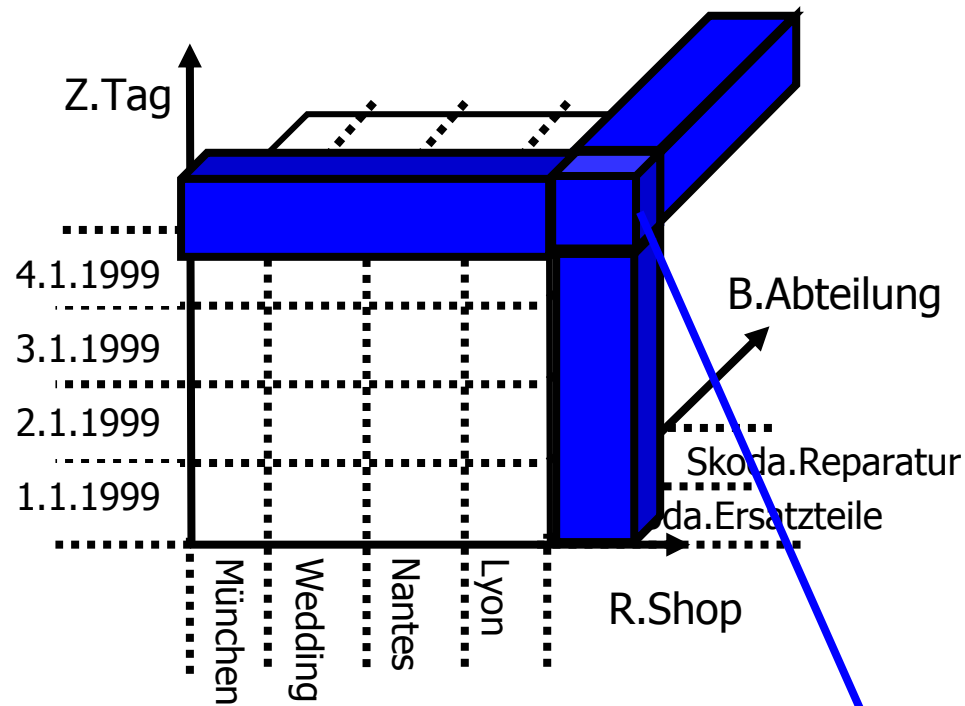
- Bemerkung

- Die Werte $\text{dom}(G)$ geben die **Koordinaten** der Werte $\text{dom}(F)$ an
- Verhältnis Würfelschema zu Würfel ist wie Relationenschema zu Relation

Beispiel Aggregation



... in mehreren Dimensionen



$$G = (Z.TOP, R.TOP, B.TOP)$$

Klassen von Aggregatfunktionen [LS97]

- Definition

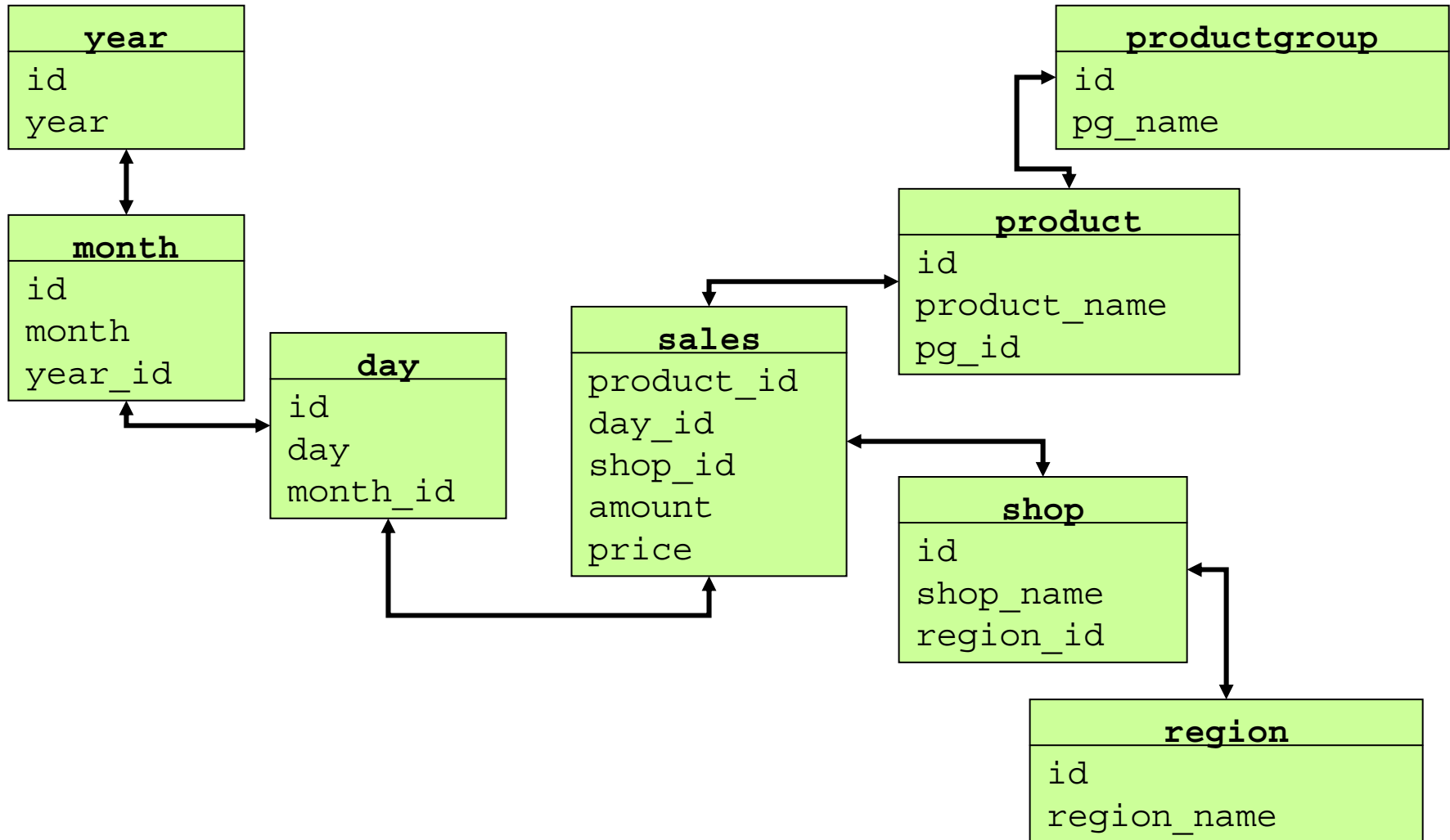
Gegeben eine Menge X und eine Partitionierung (X_1, X_2, \dots, X_n) von X . Eine Aggregatfunktion f heißt:

- *distributiv* gdw $\exists g: f(X) = f(g(X_1), g(X_2), \dots, g(X_n))$
- *algebraisch* gdw $f(X)$ berechenbar aus fester Menge von g 's
 - *Deren Zahl und Art unabhängig von X ist*
- *holistisch* gdw $f(X)$ kann nur aus den Grundelementen von X berechnet werden
 - Die Menge von g 's ist dann nur durch die Größe von X begrenzt
 - Keine Präaggregation möglich

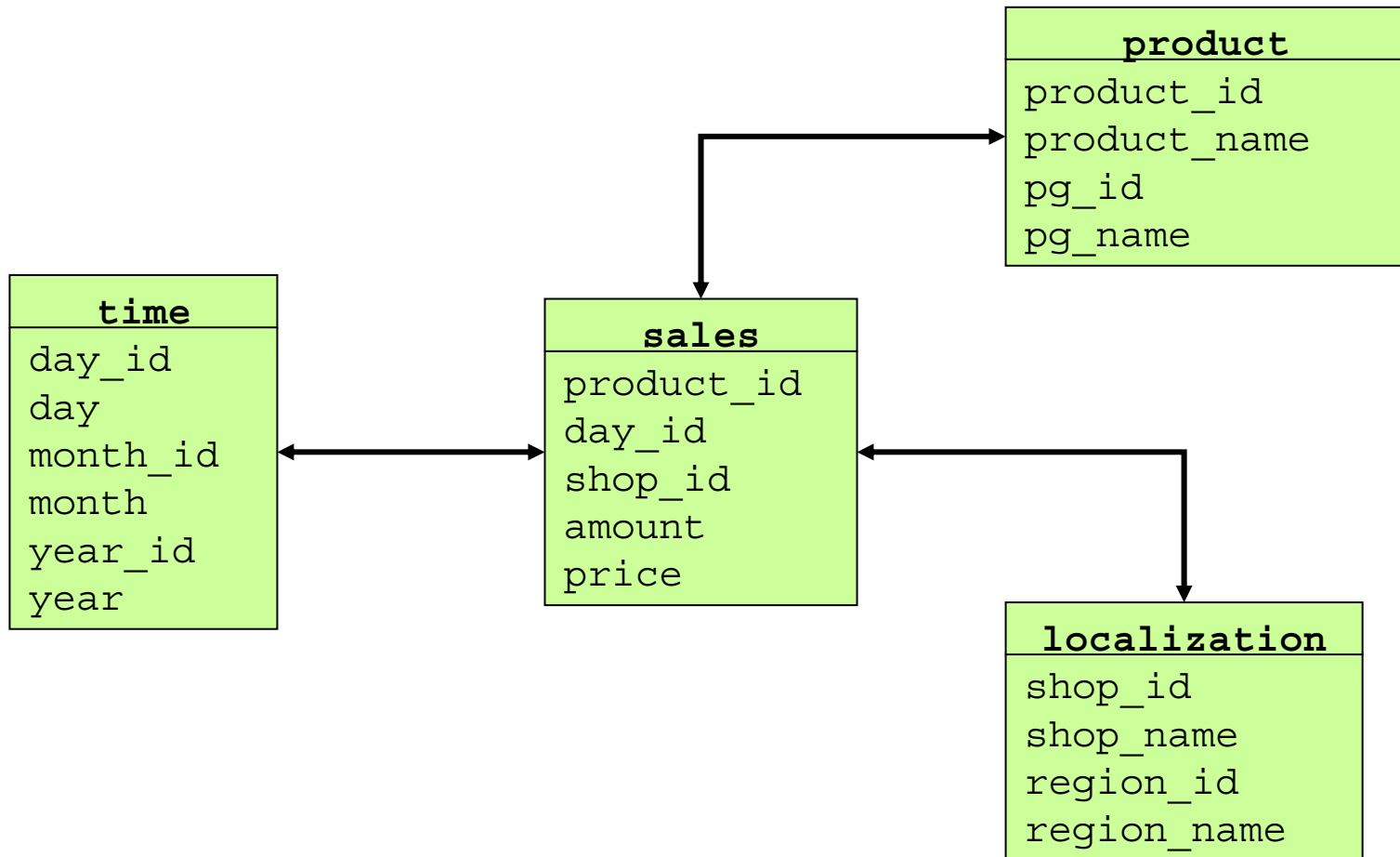
- Bemerkungen

- X entspricht einem Klassifikationsknoten, (X_1, X_2, \dots, X_n) seinen Kindern
- Die Definition verallgemeinert offensichtlich für Hierarchien

Variante 1 - Snowflake



Variante 2: Star Schema



MDX

- MDX: **Multidimensional Expressions**
 - Microsoft's Vorschlag, „OLE DB for OLAP“
- Eigene Anfragesprache
 - Standard **ohne feste Semantik** (by example)
 - MDDM Konzepte als **First-Class Elemente**
 - Dadurch kompaktere Anfragen als mit SQL
 - SQL-artige Syntax
 - Sehr mächtig und komplex
- Erzeugung der Objekte (DDL) erfolgt anderweitig
 - DSO Interface (Decision Support Objects) von SQL Server
- Wird von vielen kommerziellen Tools zur Kommunikation mit OLAP Datenbank benutzt

Member: Unique Names oder Navigation

Verschiedene Klassifikationsstufen in einer Dim möglich

```
SELECT {Wein, Bier, Limo, Saft} ON COLUMNS  
      {[2000], [1999], [1998].[1], [1998].[2]} on ROWS  
FROM Sales  
WHERE (Measures.Menge, Region.Germany)
```

Auswahl des Fakt „Menge“
Beschränkung auf Werte in BRD

	Bier	Limo	Saft
2000			
1999			
1998.1			
1998.2			

Implizite Summierung

ROLLUP Beispiel

```
SELECT      T.year_id, T.month_id, T.day_id, sum(...)
FROM        sales S, time T
WHERE       T.day_id = S.day_id
GROUP BY    ROLLUP(T.year_id, T.month_id, T.day_id)
```

1997	Jan	1	200
1997	Jan	...	
1997	Jan	31	300
1997	Jan	ALL	31.000
1997	Feb	...	
1997	March	ALL	450
1997	
1997	ALL	ALL	1.456.400
1998	Jan	1	100
1998	
1998	ALL	ALL	45.000
...	
ALL	ALL	ALL	12.445.750

CUBE - Beispiel

```
SELECT  pg_id, shop_id, sum(amount*price)
FROM    sales S ...
GROUP BY CUBE (S.pg_id, S.shop_id, T.year_id)
```

Bier	Kreuzberg	ALL	...
Bier	Charlottenburg	ALL	...
Bier	ALL	1997	...
Wein	ALL	1998	...
ALL	Kreuzberg	1997	...
ALL	Charlottenburg	1998	...
Bier	ALL	ALL	...
Wein	ALL	ALL	...
ALL	ALL	1997	...
ALL	ALL	1998	...
ALL	Kreuzberg	ALL	...
ALL	Charlottenburg	ALL	...
ALL	ALL	ALL	...

OVER() Klausel

- Ausgabe der Summe Verkäufe eines Tages im Verhältnis zu den Gesamtverkäufen

```
SELECT      S.day_id, sum(amount) AS day_sum, day_sum/T.all_sum
FROM        sales S,
            (SELECT sum(amount) AS all_sum
             FROM sales S) T
GROUP BY    S.day_id;
```

- Kompliziert zu schreiben und potentiell **ineffizient**
 - Warum müssen wir zweimal über `sales` iterieren?
- Besser: **over() Klausel**

```
SELECT      S.day_id, sum(amount) AS day_sum,
            day_sum/ (sum(amount) OVER ())
FROM        sales S
GROUP BY    S.day_id;
```

- OVER() ohne Parameter iteriert über alle Tupel

ETL - Übersicht

- Daten werden mehrmals physikalisch bewegt
 - Von den Quellen zur Staging Area
 - Extraktion von Daten aus den Quellen
 - Erstellen / Erkennen von **differentiellen Updates**
 - Erstellen von LOAD Files
 - Von der Staging Area zur Basisdatenbank
 - Data Cleaning und Tagging
 - Erstellung integrierter Datenbestände
- Ziele
 - Kontinuierliche Datenversorgung des DWH
 - Sicherung der DWH Konsistenz bzgl. Datenquellen
- **Effiziente Methoden** essentiell -> Sperrzeiten minimieren
- **Rigorese Prüfungen** essentiell -> Datenqualität sichern

Das Differential Snapshot Problem (DSP)

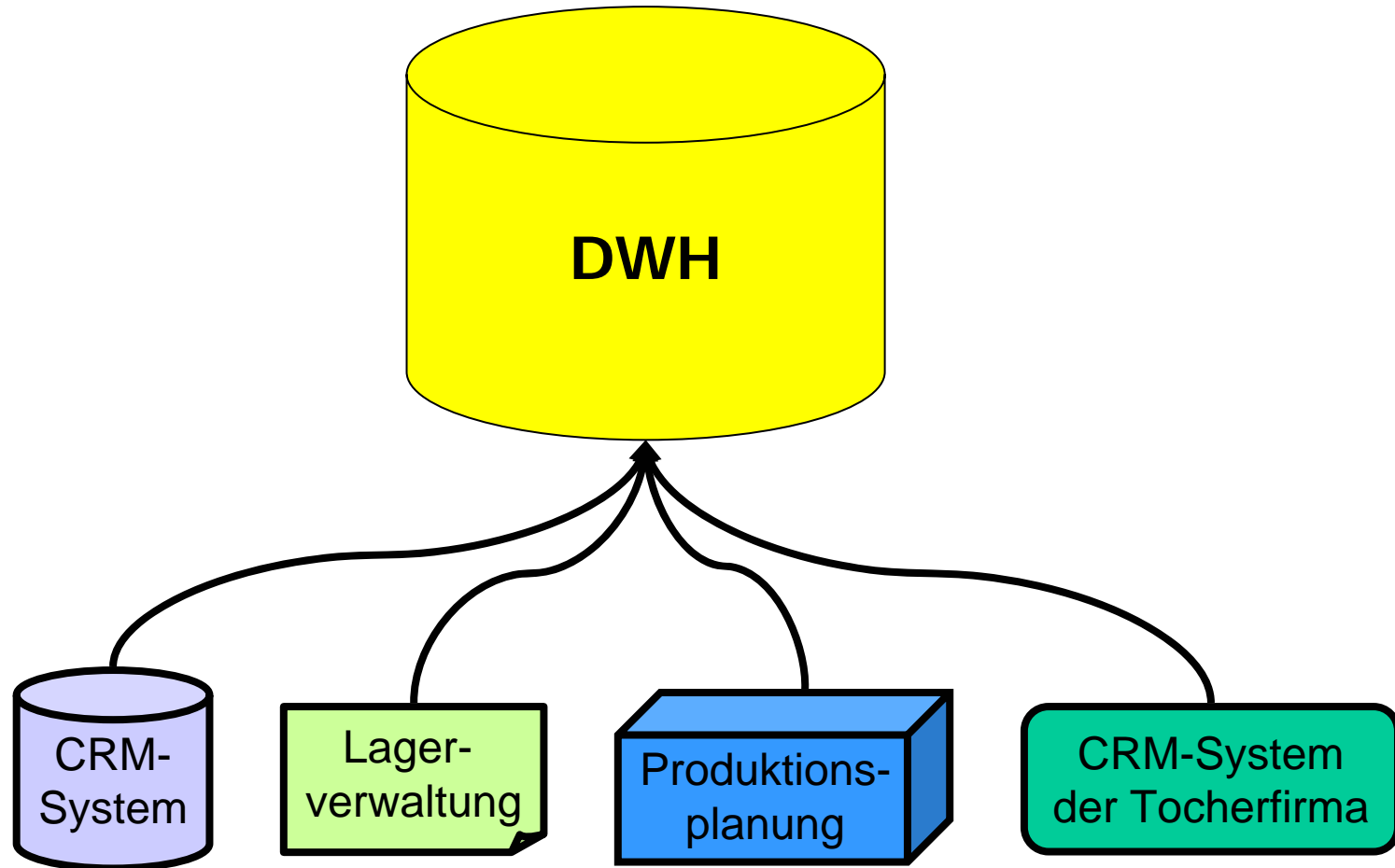
- Quelle liefert Snapshots als Files F_i
 - Einfache und identische Struktur mit Attributen A_1, \dots, A_n
 - Jedes Tupel hat einen Schlüssel k
 - File = ungeordnete Menge von Records (K, A_1, \dots, A_n)
- Definition

*Gegeben zwei Dateien F_1, F_2 gleicher Struktur mit $f_1=|F_1|, f_2=|F_2|$. Das **Differential Snapshot Problem (DSP)** besteht darin, die kleinste Menge $O=\{INS, DEL, UPD\}^*$ zu finden für die gilt: $O(F_1) = F_2$*
- Bemerkung
 - INS, DEL, UPD operieren jeweils auf genau einem Tupel identifiziert über seinen Schlüssel k
 - Änderungen in mehreren Attributen eines Tuples zählen wir nur einmal
 - O ist im Allgemeinen **nicht eindeutig**
 - $O_1=\{ (ins(X)),(del(X)) \} \equiv \{ \}$

Load

- Aufgabe
 - Effizientes Einbringen von externen Daten in DWH
- Kritischer Punkt
 - Load-Vorgänge blockieren unter Umständen die komplette DB (Schreibsperre auf Faktentabelle)
- Möglichkeiten
 - Satzbasiert
 - BULK-Load
 - Anwendungsspezifische Schnittstellen
 - Z.B. SAP
- Aspekte
 - Trigger, Integritätsconstraints, Indexaktualisierung
 - Update oder Insert?

Heterogenität



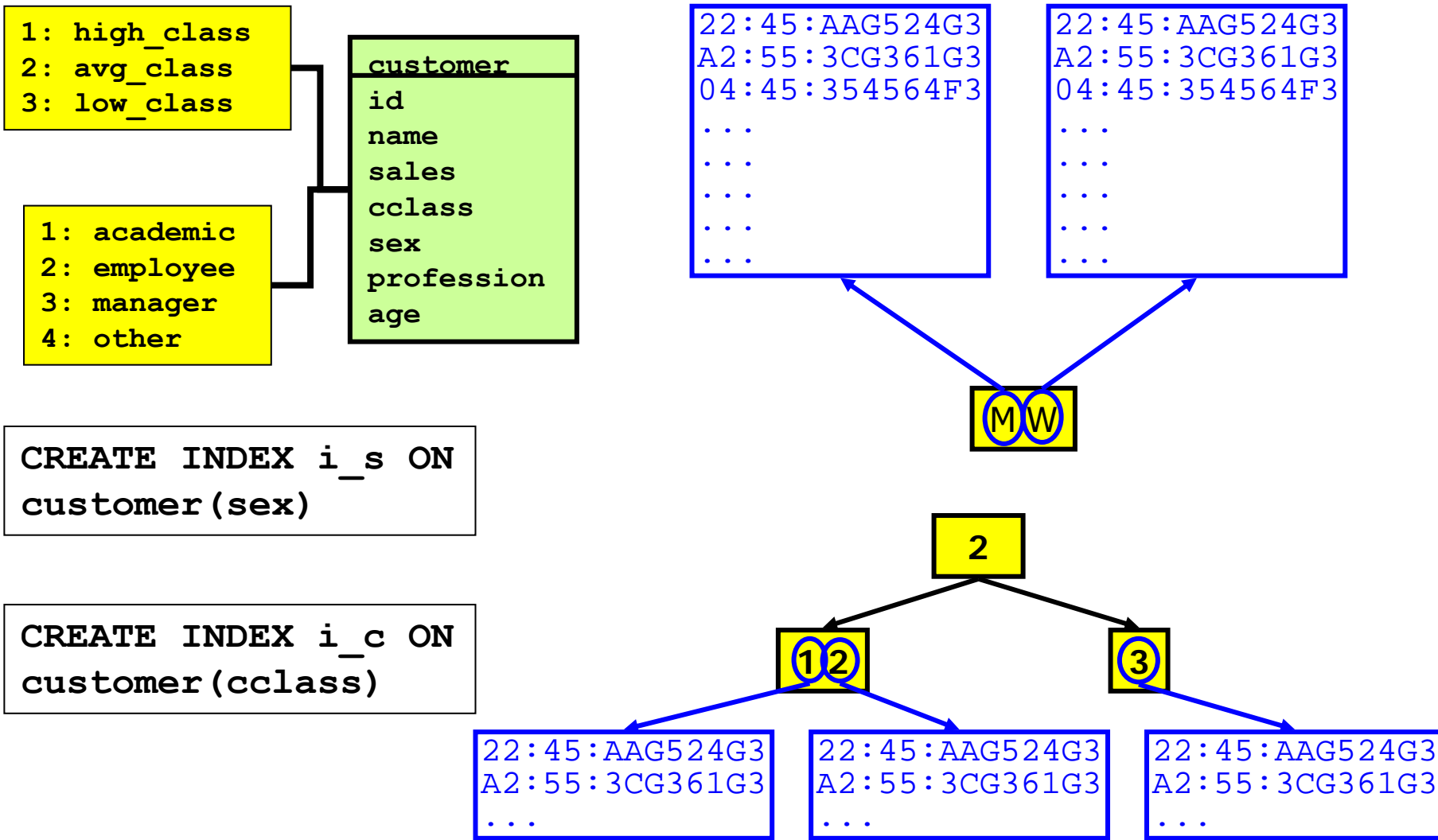
Duplikate: Das formale Problem

- Gegeben: Tupel $T = \{t_1, \dots, t_m\}$ mit Attributen A_1, \dots, A_k
 - Kein „semantischer“ Schlüssel
 - Komplexere Modelle (Bäume, Graphen) möglich
- Tupel entsprechen Objekten $O = \{o_1, \dots, o_n\}$
 - Über die wissen wir meistens nichts, **man kennt nur die Tupel**
 - Manche der Tupel sind Duplikate
- Gesucht: eine Funktion $\text{dup}: T \times T \rightarrow \text{bool}$
 - dup gibt true zurück, wenn Tupel t_1 und t_2 demselben Objekt o entsprechen
- Eine Möglichkeit: $\text{dedup}: T \rightarrow \mathbb{N}$
 - Jedem Tupel wird eine natürliche Zahl zugeordnet
 - Zwei Tupel bekommen dann und nur dann **dieselbe Zahl zugeordnet, wenn sie Duplikate** sind
 - Berechnung eines **identifizierenden Schlüssels**

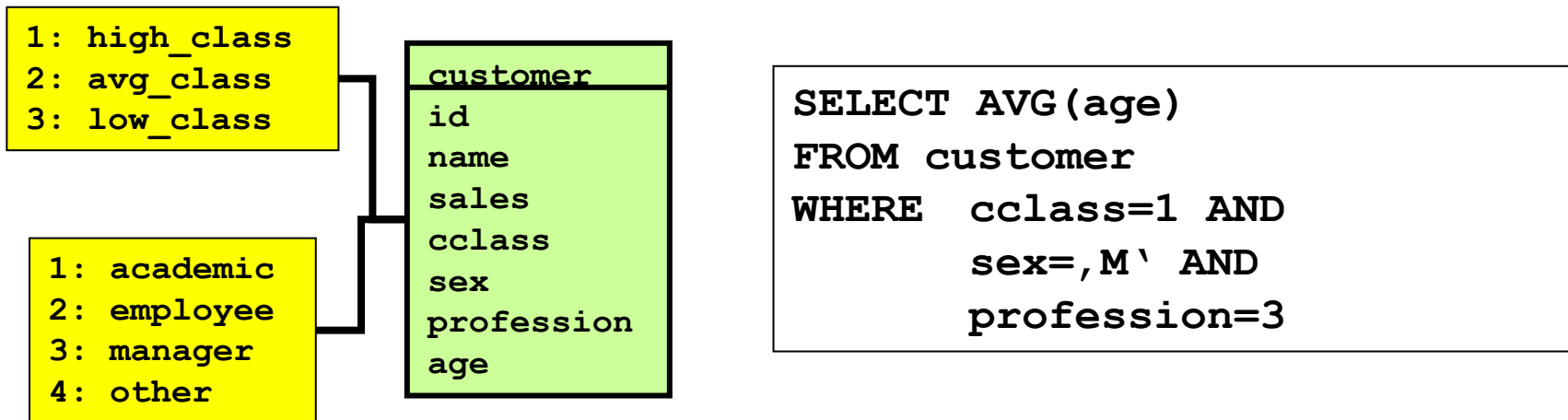
Sorted Neighborhood [HS98]

- Sorted Neighborhood Algorithmus
 - **Sortiere** Tupel so, dass Duplikate (hoffentlich) nahe beieinander liegen
 - Dazu: Erzeuge einen Sortierschlüssel aus relevanten Attributen
 - **Merge-Phase**: Fenster der Größe w über sortierte Liste schieben
 - Nur Tupel innerhalb eines Fensters vergleichen
 - Man hat also überlappende Partitionen
 - **Purge-Phase**: Duplikate werden verschmolzen, gelöscht, getagged, ...
- Komplexität
 - n Tupel
 - Schlüsselerzeugung sei linear
 - Sortieren ist $O(n \cdot \log(n))$
 - Anzahl Vergleiche ist $O(n \cdot w)$
 - n -mal werden „neue“ Tupel mit $(w-1)$ Tupeln im Fenster verglichen
 - Zum Vergleich: Naiver Ansatz war $O(n^2)$

Degenerierte B*-Bäume



Oversized Indexe



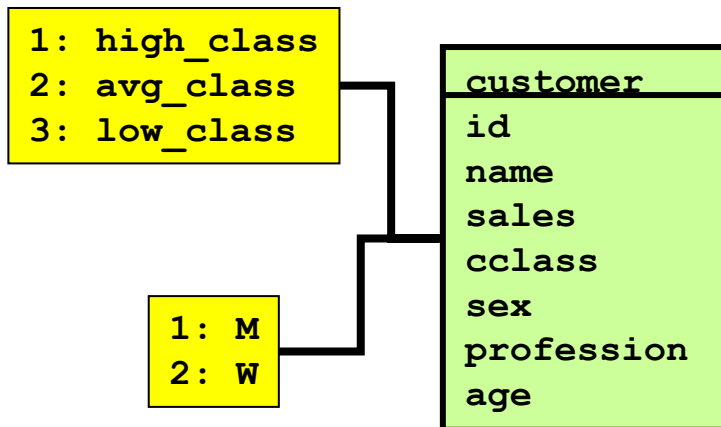
- Normaler Ablauf bei zusammengesetztem Index
 - Suche Werte „1 | |M| | 3“ in Baum
 - Ablauf TID Liste, **Datenblockzugriff für age Werte**
- Besser

```
CREATE INDEX c_scp ON
customer(sex, cclass, profession, age)
```

Grundaufbau

- Tabelle T mit Attribut A, $|T|=n$, $|A|=a$ verschiedene Werte
- Repräsentation jedes der a Werte von A als **Bitarray** der Länge n
 - Attribut X, Wert z: $X.z[i]=1$ gdw i.tes Tupel hat Wert z in Attribut X
 - Die **Ordnung der Tupel** muss feststehen (später mehr)
- Repräsentation von A: **Bitmatrix** mit $n*a$ Bits

Tabelle



```
22:45,1,Meier,20.000,2,M,...
A2:55,2,Müller,15.000,3,W,...
33:D1,3,Schmidt,25.000,1,M,...
1A:0E,4,Dehnert,22.000,2,M,...
...
```

Bitmap-Index cclass?

```
1:0010...
2:1001...
3:0100...
```

Bitmap-Index sex?

```
M:1011...
W:0100...
```

Run-Length-Encoding (RLE)

- RLE1: **Explizites Speichern der 1-Positionen**
 - Beispiel: $n=1.000.000$, $a=100$, TID: 4 Byte
 - Pro Bitarray ist nur einer von 100 Werten eine „1“
 - Annahme: Gleichverteilung der Werte von A über T
 - Bitmap ohne RLE: $1.000.000 * 100/8 = 12,5 \text{ MB}$
 - Bitmap mit RLE
 - $1.000.000$ ist durch 20 Bit adressierbar
 - $1.000.000 * (20/8) / 100 * 100 = 2.5 \text{ MB}$
- **Nachteil**
 - Wir müssen die **Größe des Arrays** von vorneherein festlegen
 - Um die Anzahl der Bits für Kodierung festlegen zu können
 - Wenig adaptiv für schrumpfende / wachsende Tabellen

Vertikale Komprimierung

- Attribut A mit $|A|=a$ verschiedenen Werten

t_1 t_2 t_3

```
1:00100000001010100000001010000...
2:10011110000001000011000000010...
3:01000000001000000000000010000...
...
a: ...
```

- Komprimierung bisher

```
1:14,33,45...
2:3,55, ...
3:17,34,41,...
...
a: ...
```

- Können wir auch **vertikal komprimieren** ($b < a$)?

t_1 t_2 t_3

```
1:001001100010101000000010100110...
...
b: ...
```

Beispiel 1

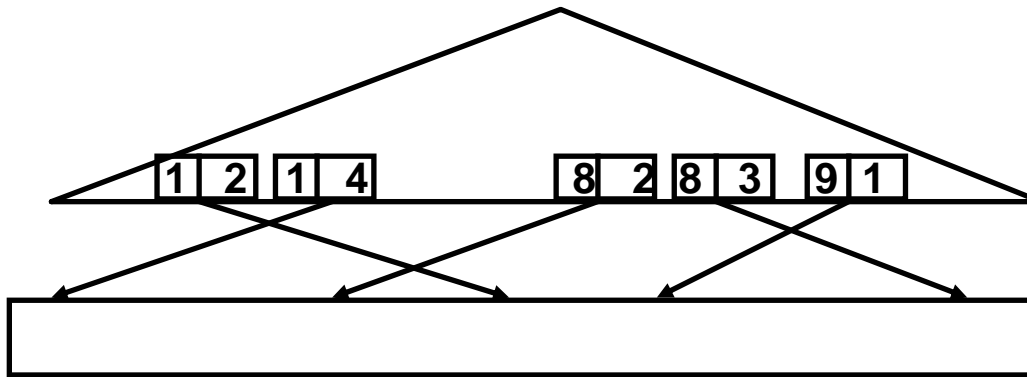
- Darstellung von 20 Werten
- Bitmap
 - Speicherverbrauch (pro Tupel) 20 Bit
 - Vergleich mit Konstanter 1 Bitarray lesen
- Bitmapped zur Basis $\langle 4,4 \rangle$
 - Speicherverbrauch 8 Bit
 - Vergleich mit Konstanter 2 Bitarrays
- Bitmapped zur Basis $\langle 2,4,3 \rangle$
 - Speicherverbrauch 9 Bit
 - Vergleich mit Konstanter 3 Bitarrays
- Bitmapped zur Basis $\langle 2,2,2,2,2 \rangle$
 - Speicherverbrauch 10 Bit
 - Vergleich mit Konstanter 5 Bitarrays
- Binärdarstellung
 - Speicherverbrauch 5 Bit
 - Vergleich mit Konstanter 5 „Bitarrays“

Multidimensional Indexing

- **Multidimensional queries**
 - Conditions on **more than one attribute**
 - Combined through AND (intersection) or OR (union)
 - Partial queries: Conditions on some but not all dimensions
 - Selects sub-cubes
 - 2D: "All beverage sales in March 2000"
 - 4D: "All beverage sales in 2000 in Berlin to male customers"

Composite Index

- One index with two attributes (X, Y)



- General
 - Prefix of attribute list in index must be present in query
 - The longer the prefix in the query, the better
- Alternatives
 - Also index (Y, X)
 - 5 attributes: 5! orders
 - Combinatorial explosion, infeasible for more than 2 attributes
 - Use independent indexes on each attribute

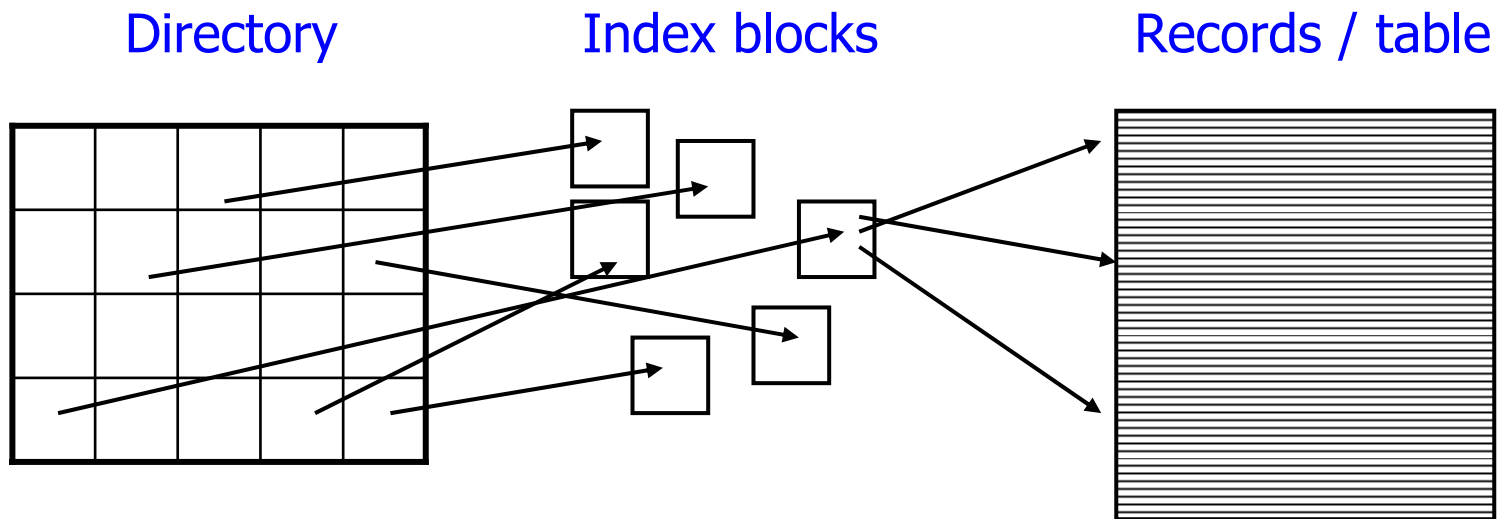
Independent versus Composite Index

- Consider 3 dimensions of range $1, \dots, 100$
 - 1.000.000 points, uniformly (and randomly) distributed
 - Index blocks hold 50 keys or records
 - Index on each attribute has height 4
- Find points with $40 \leq x \leq 50, 40 \leq y \leq 50, 40 \leq z \leq 50$
 - Using x-index, we generate TID list $|X| \sim 100.000$
 - Using y-index, we generate TID list $|Y| \sim 100.000$
 - Using z-index, we generate TID list $|Z| \sim 100.000$
 - For each index, we have $4 + 100.000/50 = 2004$ IO
 - TIDs are sorted in sequential blocks, each holding 50 TIDs
 - Hopefully, we can keep the three lists in main memory
 - Intersection yields app. 1.000 points with 6012 IO
 - Why 1000 points?
- Using composite index (X,Y,Z)
 - Number of indexed points doesn't change
 - Key length increases – assume blocks hold only 30 (10) keys or records
 - Index has height 5 (6)
 - This is worst case – index blocks only 50% filled
 - Total: $5 (6) + 1000/30 (10) \sim 38 \text{ IO } (104)$



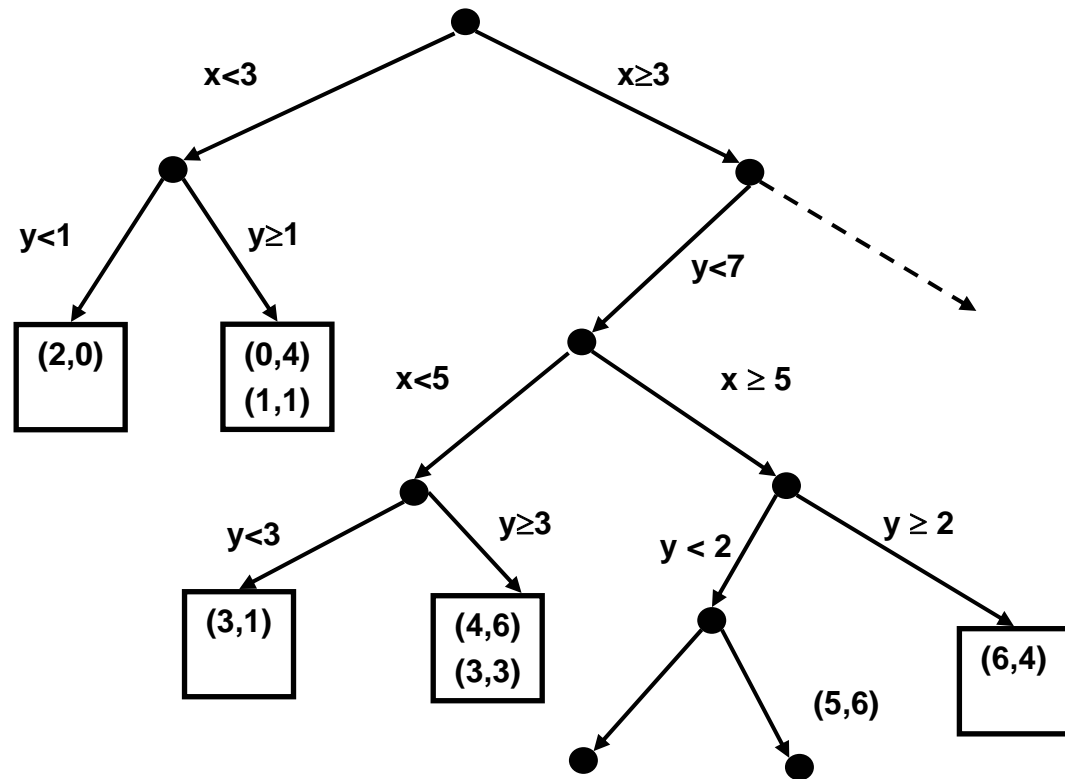
Fixed Grid-File

- [Does not adapt to data distribution at all]
- Idea
 - Split space into equal-spaced cuboids or **cells**
 - We need maximal and minimal values in all dimensions
 - **Directory** stores one pointer for each cell
 - Points to block storing pointers to records

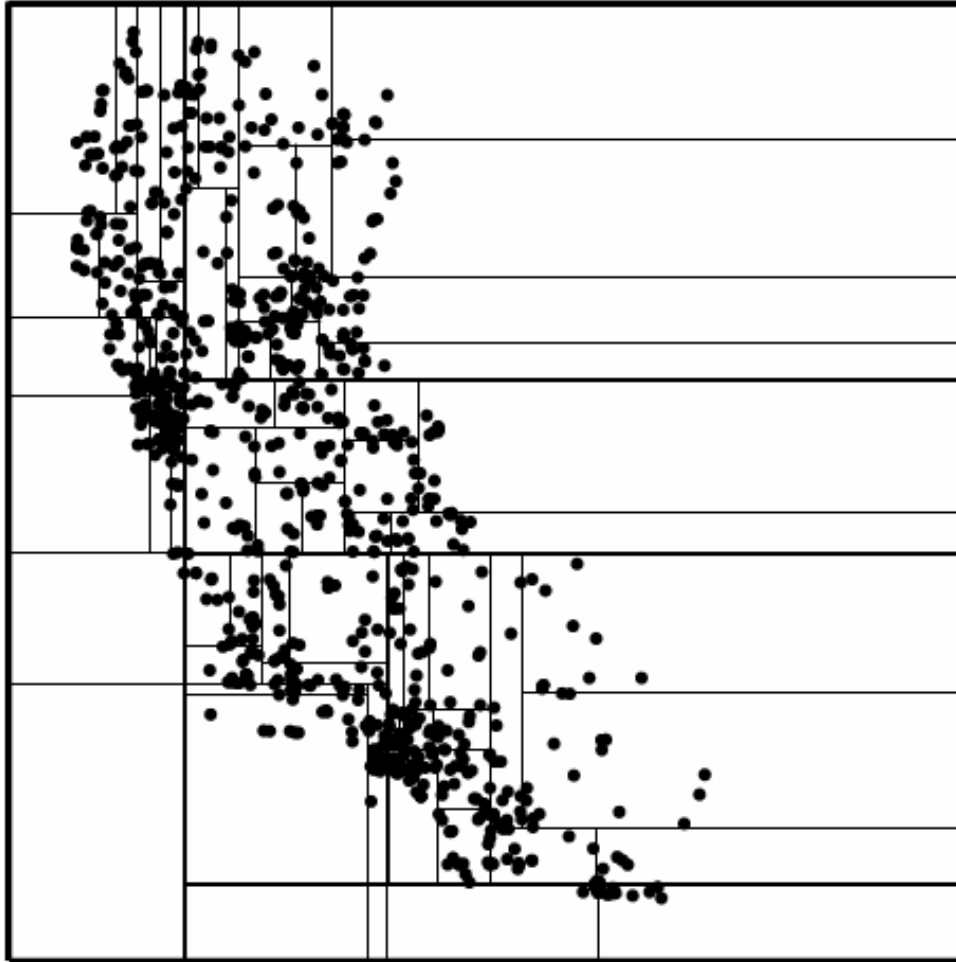


kd-tree: General Idea

- Binary, rooted tree
- Each inner node has two children
- Path is selected based on a pair (**dimension / value**)
- Dimensions need not be statically assigned to levels of the tree
 - Can be rotating, random ...
- Data points are only stored in leaves
- Each leaf stores points in a n -dimensional hypercube with **m border planes** ($m \leq n$)

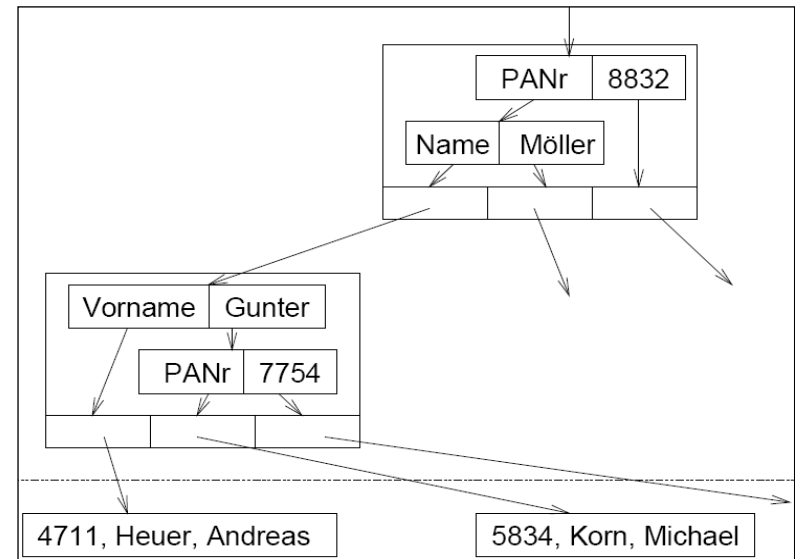


Local Adaptation

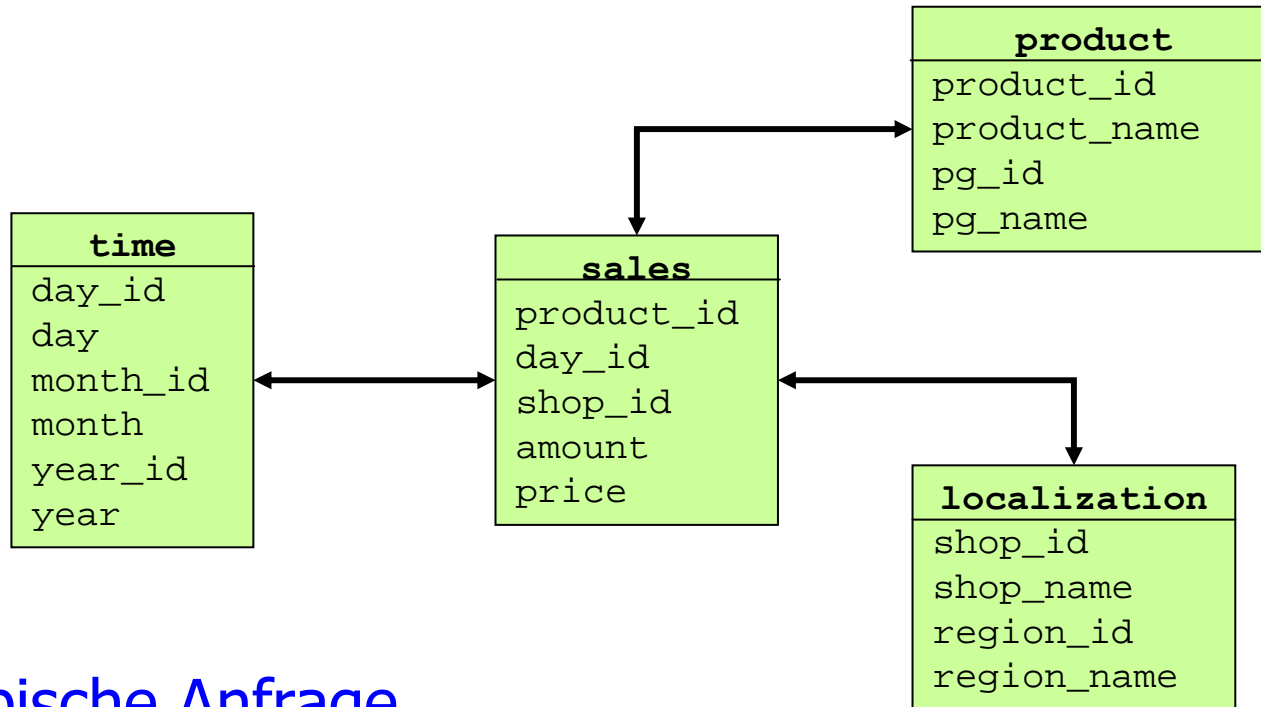


kdb-trees

- Option 2: Store **entire subtrees** in one block
 - Inner nodes still have only two children
 - But those are (usually) stored in the same block
 - We need to “map” nodes to trees
 - kdb-tree: **inner nodes store kd-trees**
- Operations
 - Searching: As with kd-trees
 - But better IO complexity
 - Insertion/Deletion
 - **Complex scheme** for keeping tree balanced



Anfragen an Star-Schema



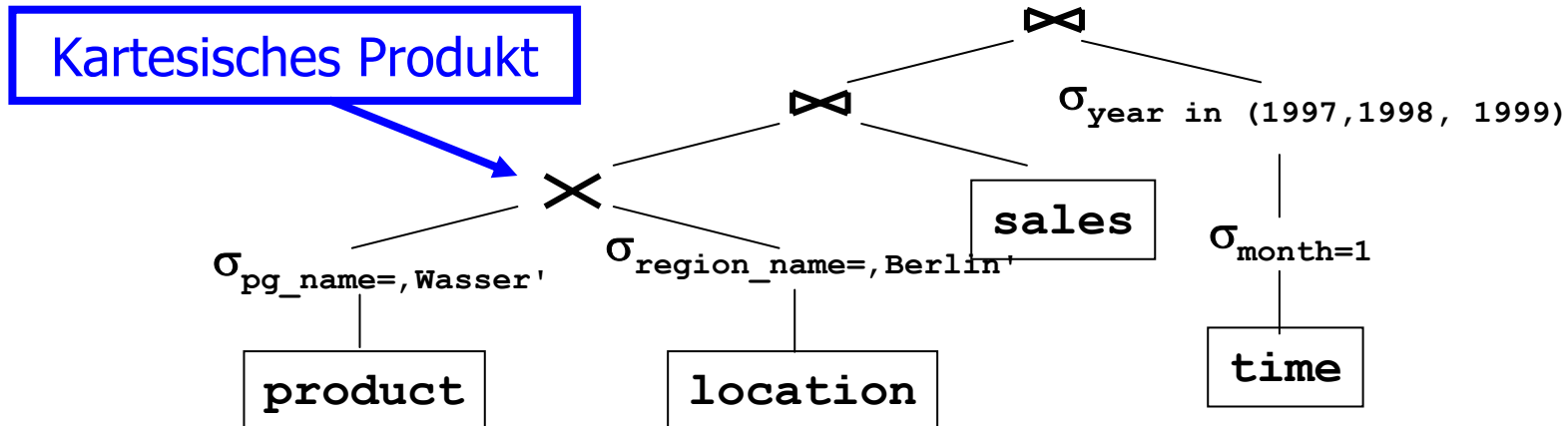
- **Typische Anfrage**

- Aggregation und Gruppierung
- Bedingungen auf den Werten der Dimensionstabellen
- Joins zwischen Dimensions- und Faktentabelle

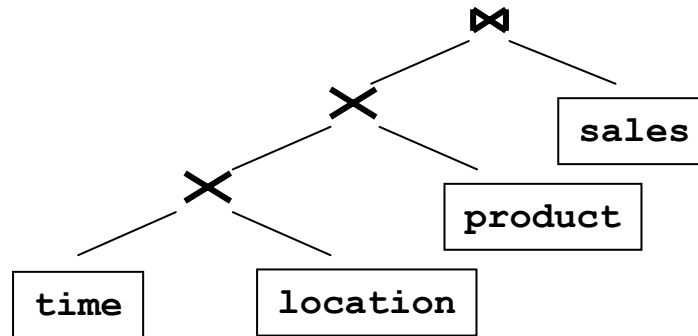
Heuristiken

```
SELECT T.year, sum(amount*price)
FROM sales S, product P, time T, localization L
WHERE P.pg_name=,Wasser` AND
      P.product_id = S.product_id AND
      T.day_id = S.day_id AND
      T.year in (1997, 1998, 1999) AND
      T.month = ,1` AND
      L.shop_id = S.shop_id AND
      L.region_name=,Berlin`
GROUP BY T.year
```

- Typisches Vorgehen
 - Auswahl des Planes nach Größe der Zwischenergebnisse
 - Keine Beachtung von Plänen, die **kartesisches Produkt** enthalten



Plan mit kartesischen Produkten



	Zwischenergebnis
1. time x location (3*20 * 100)	6.000
2. ... x product (P ₁ *20)	120.000
3. ... ⋈ sales	3.333

- Es gibt mehr „Zellen“ als Verkäufe
- Nicht an jedem Tag wird jedes Produkt in jedem Shop verkauft

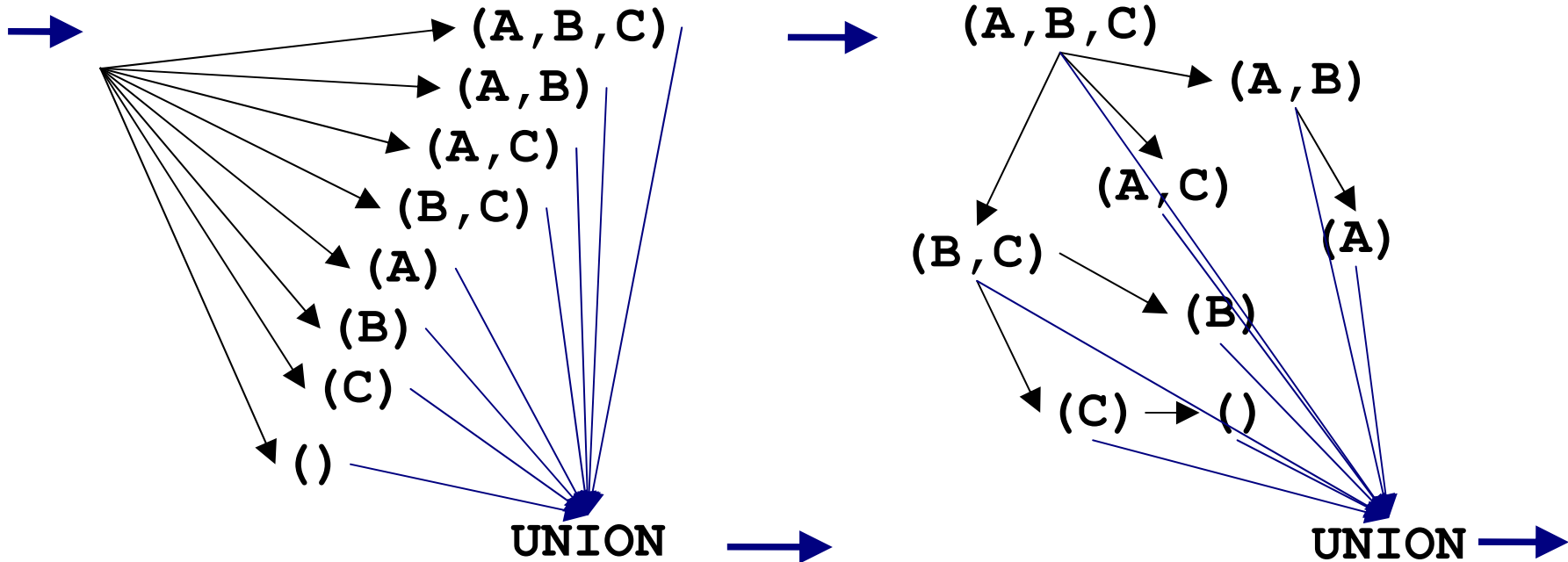
Star-Join in Oracle 8i – 9i

- Neue Star-Join Strategie seit Oracle 8i
- Benutzung komprimierter **Bitmapindexe**
- Phasen
 1. Berechnung aller FKs in Faktentabelle gemäß Dimensionsbedingungen einzeln für jede Dimension
 2. Anlegen/laden von bitmapped Join-Indexen auf allen FK Attributen der Faktentabelle
 3. Merge (AND) aller Bitmapindexe
 4. Direkter Zugriff auf Faktentabelle über TID
 5. Join **nur der selektierten Fakten** mit Dimensionstabellen zum Zugriff auf Dimensionswerte
- Zwischenergebnisse sind nur Bitlisten

Partitionierung

- Aufteilung der Daten **einer Tabelle** in Untereinheiten
 - **Physische Partitionierung** (Allokation): Für den Benutzer transparent
 - Nach Definition der Partitionen
 - Logische Partitionierung: Für den Benutzer nicht transparent
 - Explizites Anlegen mehrerer Tabellen
 - Anfragen müssen entsprechend formuliert werden
- Ursprünglich für verteilte Datenbanken entwickelt
 - Verteilung der Partitionen auf verschiedenen Knoten
 - Vereinfachte Synchronisation & Lastverteilung
 - Wichtig ist **Lokalität Anfrage – Partition**

Realisierung CUBE



Naiver Ansatz

Ausnutzen von
(direkter) Ableitbarkeit

Mögliche Tricks

1. Smallest-parent

- Berechne Gruppierung aus dem **Elternteil mit den wenigsten Tupeln**
- Beispiel: (A) kann aus (AB), (AC) oder (ABC) berechnet werden
 - (ABC) hat sicher am meisten Tupel, aber $|AB| \neq |AC|$
- Gewinn: Weniger Rechenaufwand, eventuell können abgeschlossene Gruppierungen früher verdrängt werden

2. Cache-results

- Berechne Ketten von ableitbaren Gruppierungen
- Halte dazu **Ergebnisse im Hauptspeicher**, um IO zu sparen
- Beispiel: (ABC) \rightarrow (AB) \rightarrow (B) \rightarrow ()
 - Dazu muss (ABC) und (AB) in den Hauptspeicher passen (oder sortieren)
- Gewinn: Weniger IO

3. Amortize-scans

- Wenn schon eine Gruppierung geladen werden muss, berechne **möglichst viele der Kinder** gleichzeitig
- Beispiel: Aus (ABC) berechne mit einem Scan (AB), (BC), (AC)
- Gewinn: Weniger Scans, weniger IO

Grundidee

- Bei den Kosten der Berechnung gehen mit ein
 - **Kosten für Sortierung** – wenn notwendig
 - **Kosten für Gruppierung** – abhängig von der Größe des Eltern
- Kosten werden an Kanten im Gitter geschrieben
- PipeSort geht **ebenenweise** durch das Gitter
 - Start bei der Gruppierung mit der höchsten Auflösung
 - Jede Gruppierung wird von einem unmittelbaren Vorfahr abgeleitet
 - Das ist eine Heuristik zur Komplexitätsreduktion
- Für jeden Ebenenwechsel wird eine Lösung durch **biartites Matching** berechnet
 - Global ist das i.d.R. nicht optimal
 - PipeSort ist Greedy

Iceberg Cubes

- Immer noch: Es gibt exponentiell viele Gruppierungen in einem CUBE Operator
 - 10 Dimensionen a 10 Ausprägungen = 1024 Gruppierungen mit zwischen 1 und 10^{10} Partitionen
 - „High dimensionality“ – die allermeisten Kombinationen müssen leer sein
 - So viel kann man gar nicht verkaufen
- Iceberg Cubes
 - Oftmals interessieren nur Aggregate oberhalb einer gewissen Grenze
 - ```
SELECT product_id, day_id, shop_id, SUM(amount)
FROM sales S
GROUP BY CUBE(product_id, day_id, shop_id)
HAVING COUNT(*) > threshold
```

# Materialisierte Sichten

---

- Grundidee
  - Viele **Anfragen wiederholen sich häufig**
    - Bilanzen, Bestellungsplanung, Produktionsplanung
    - Kennzahlenberechnung
  - Viele Anfragen sind Variationen derselben Anfrage
    - Alle Verkäufe nach Produkt, Monat und Region
    - Alle Verkäufe in Region X nach Produkt, Monat und Shop
    - Alle Verkäufe in Region Y nach Produkt, Monat und Shop
- **Materialisierte Views (MV)**
  - Berechnen und **Speichern** einer Sicht
  - **Transparente Verwendung** in späteren Anfragen
    - Transparent = Benutzer muss die MV nicht direkt adressieren

# Themen

---

- Welche Views soll man materialisieren?
  - MVs kosten: Platz und Aktualisierungsaufwand
  - Wahl der optimalen MVs hängt von Workload ab
  - Algorithmen zur **Auswahl der optimalen Menge**
- Wie hält man MV aktuell?
  - MV nachführen, wenn sich Basistabellen ändern
  - U.U. schwierig: Aggregate, Joins, Outer-Joins, ...
    - Algorithmen zur **inkrementellen Aktualisierung**
- **Wann kann/soll man welche MV verwenden?**
  - **Wann** kann man MV verwenden?
    - Query Containment und Ableitbarkeit
  - **Welche** MV kann man verwenden?
    - Logische Optimierung, Query Rewriting
  - Wann **soll** man MV verwenden?
    - Kostenbasierte Optimierung

# Definition

---

- Definition

*Sei  $S$  ein Schema und  $q, v$  Anfragen gegen  $S$ :*

- Eine **Instanz** von  $S$  ist eine beliebige Datenbank  $D$  mit Schema  $S$ , geschrieben  $D^S$
- Das **Ergebnis** einer Query  $q$  in einer Datenbank  $D^S$ , geschrieben  $q(D^S)$ , ist die Menge aller Tupel, die die Ausführung von  $q$  in  $D^S$  ergibt
- $q$  ist **äquivalent zu  $v$** , geschrieben  $q \equiv v$ , gdw.  
 $q(D^S) = v(D^S)$  für jede mögliche Instanz  $D^S$  von  $S$
- $q$  ist **enthalten in  $v$** , geschrieben  $q \subseteq v$ , gdw.  
 $q(D^S) \subseteq v(D^S)$  für jede mögliche Instanz  $D^S$  von  $S$

- Bemerkung

- **Semantische Definition:** Es zählt das Ergebnis einer Query
  - Natürlich können wir nicht alle Instanzen aufzählen
- Wir wollen Enthalten-Sein beweisen, indem wir nur auf die **Definition** der Anfragen sehen

# Containment Mappings

---

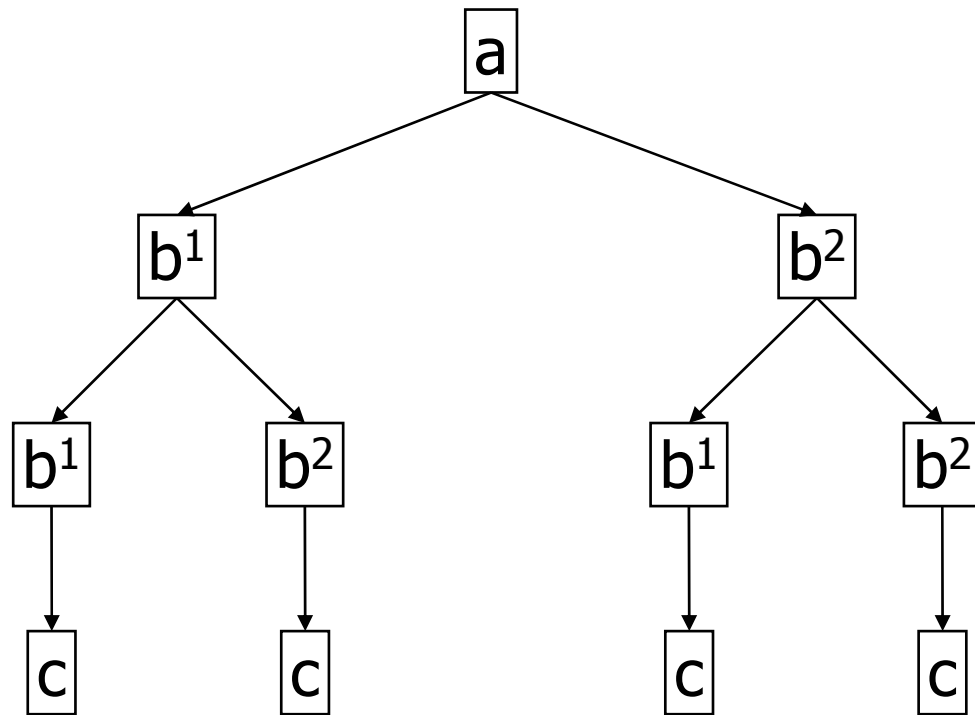
- Definition

*Ein **Containment Mapping** (CM)  $h$  von Anfrage  $q_2$  nach Anfrage  $q_1$  ist ein Symbol Mapping von  $q_2$  nach  $q_1$  für das gilt:*

1.  $\forall c \in \text{const}(q_2)$  gilt:  $h(c) = c$ 
  - Jede Konstante in  $q_2$  wird auf dieselbe Konstante in  $q_1$  abgebildet
2.  $\forall l \in q_2$  gilt:  $h(l) \in q_1$ 
  - Jedes Literal in  $q_2$  wird auf (mindestens) ein Literal in  $q_1$  abgebildet
3.  $\forall e \in \text{exp}(q_2)$  gilt:  $h(e) \in \text{exp}(q_1)$ 
  - Der Kopf von  $q_2$  wird auf den Kopf von  $q_1$  abgebildet
4.  $\text{cond}(q_1) \rightarrow \text{cond}(h(q_2))$ 
  - Die Bedingungen von  $q_1$  sind logisch restriktiver als die von  $q_2$

# Suchraum

$v =$   
 $a(\dots),$   
 $b^1(\dots),$   
 $b^2(\dots),$   
 $c(\dots)$

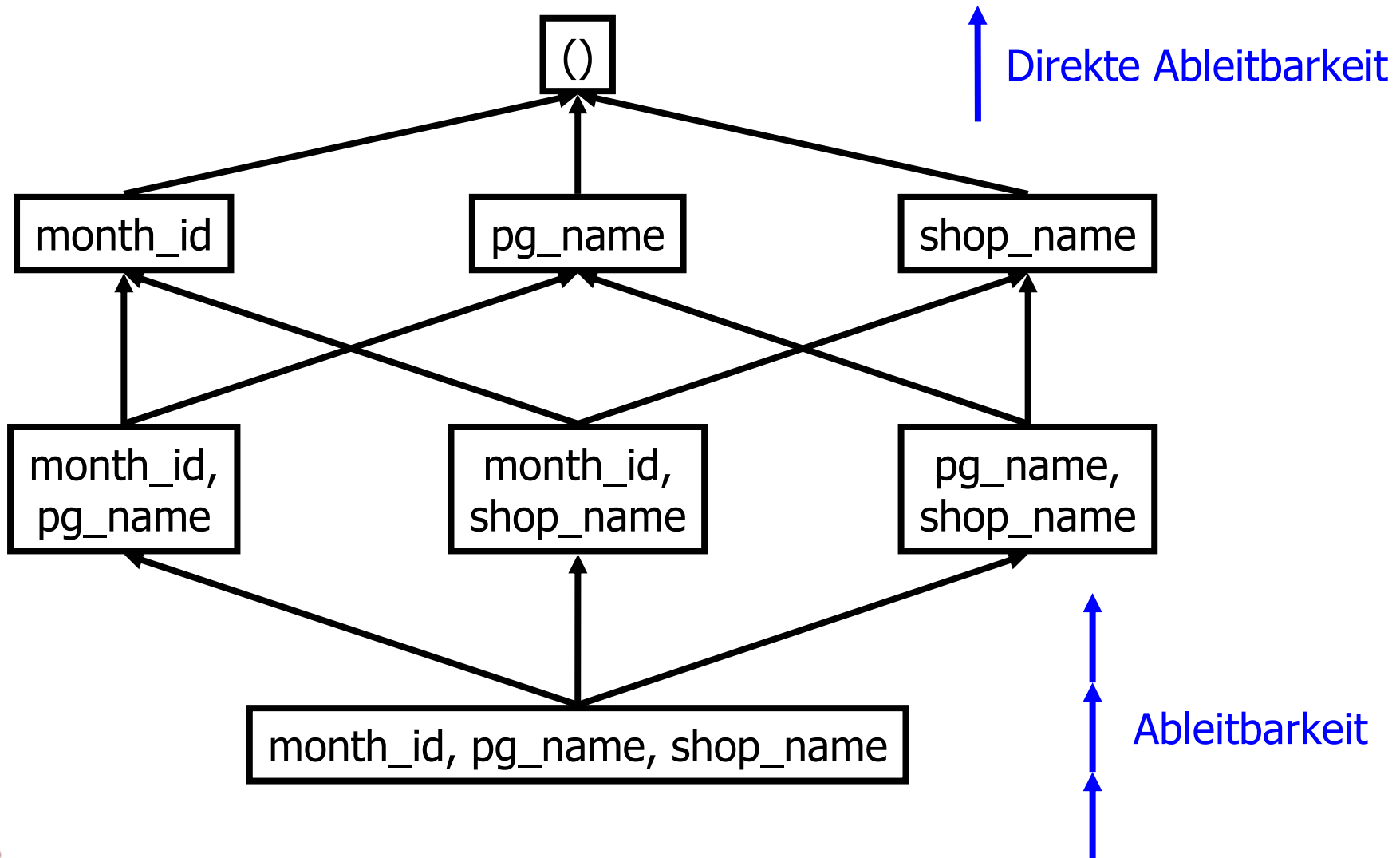


# Ableitbarkeit

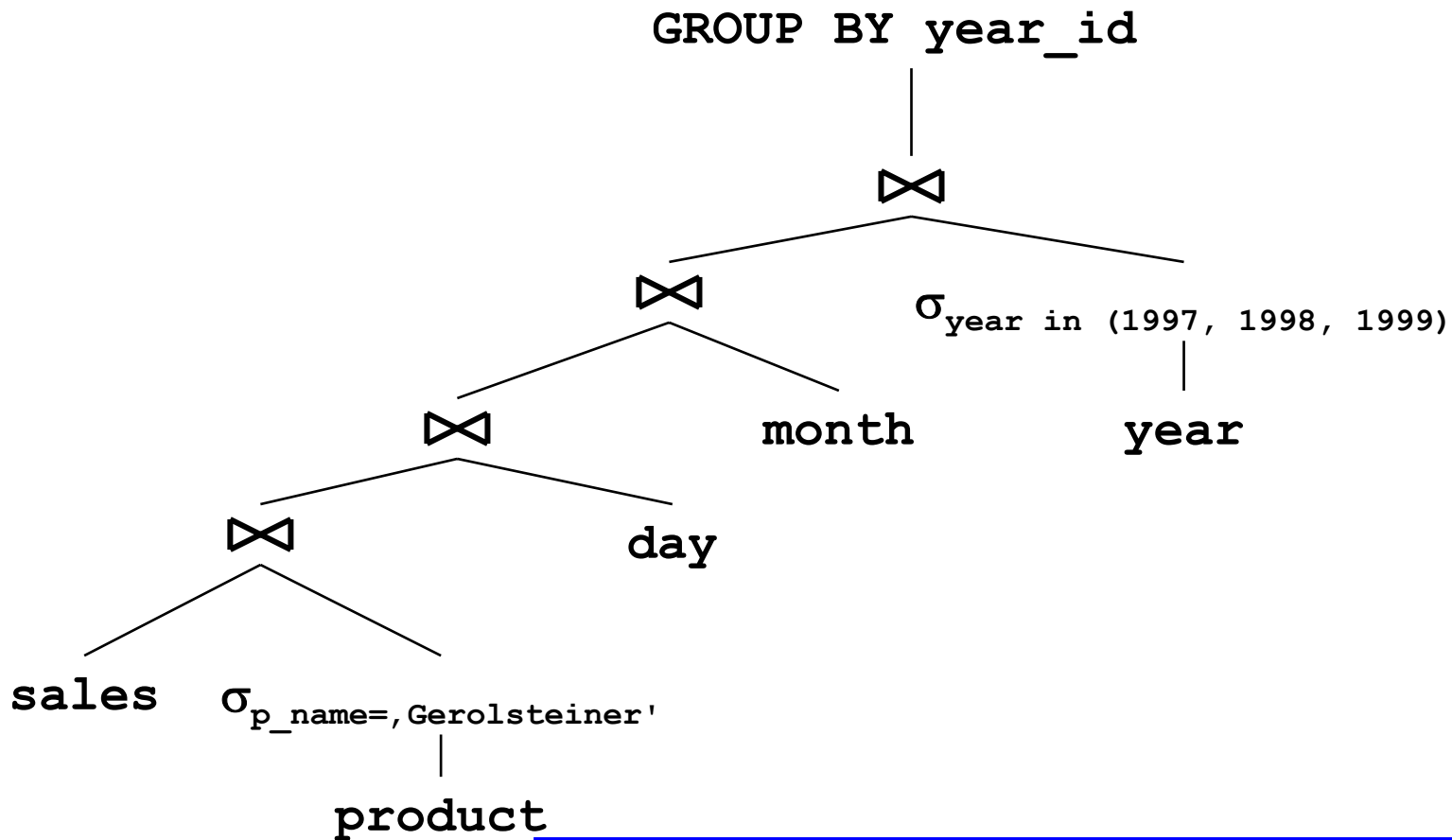
---

- Jetzt hat man alle Attribute, aber zu viele Tupel
- Wie findet man die richtigen?
- **Ableitbarkeit**
  - Wenn  $q \subseteq v$ , dann gilt:  $q \rightarrow h(v)$ 
    - $h$ : Containment Mapping von  $v$  nach  $q$
    - $\rightarrow$  bezeichnet hier die **logische Implikation** zwischen Formeln in Prädikatenlogik
  - Gesucht: Ausdruck  $F$  für den gilt:  $q \equiv h(v) \wedge F$
  - Im Allgemeinen **unentscheidbar**
    - Wegen Unentscheidbarkeit der Prädikatenlogik
  - Aber wir betrachten **nur konjunktive Anfragen**

# Aggregationsgitter



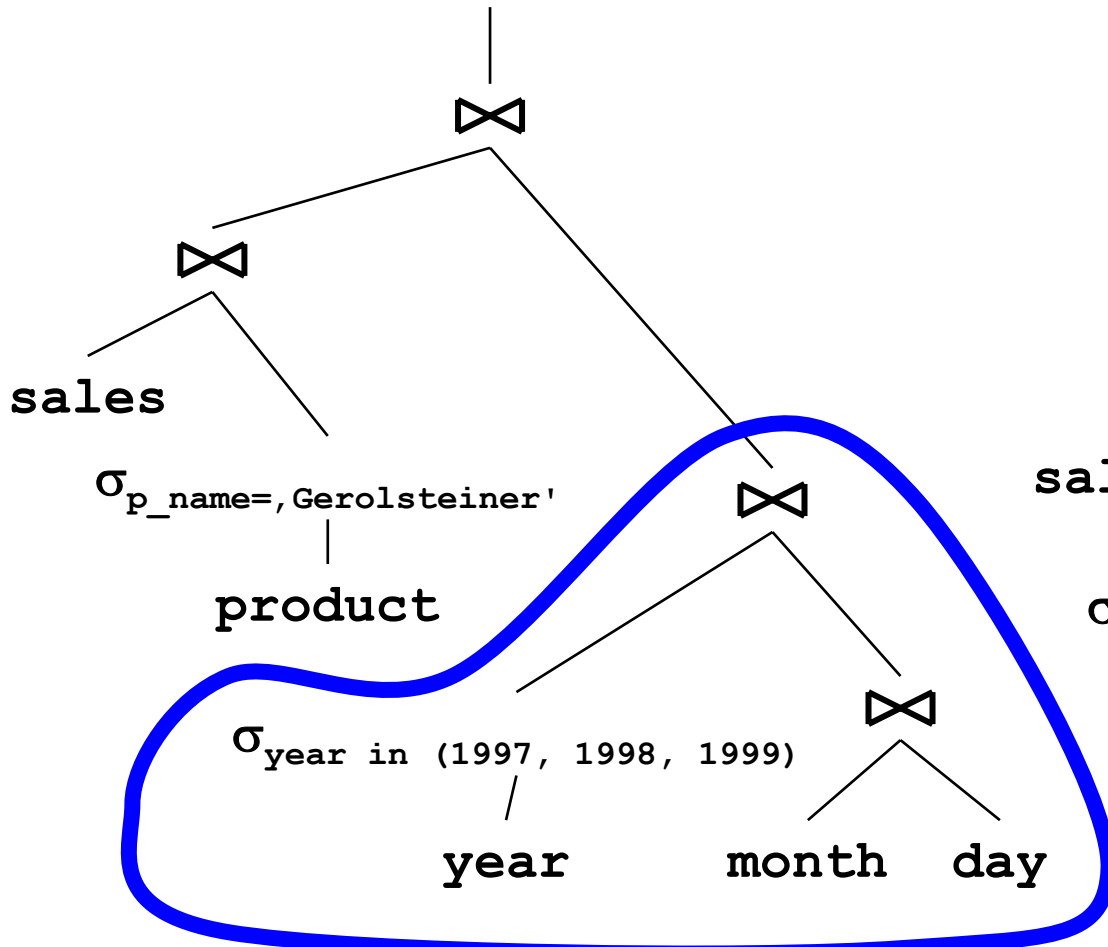
# Ausführungsplan



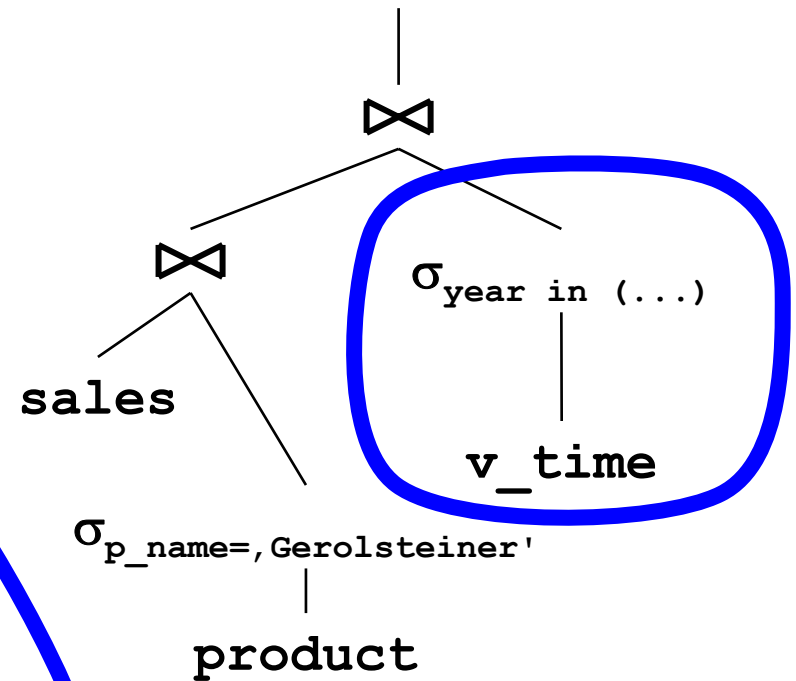
```
CREATE MATERIALIZED VIEW v_time AS
SELECT Y.id, Y.year, M.id, M.month, D.id, D.day
FROM year Y, month M, day D
WHERE Y.id = M.year_id AND
M.id = D.month_id
```

# Alternativen bewerten

GROUP BY year\_id

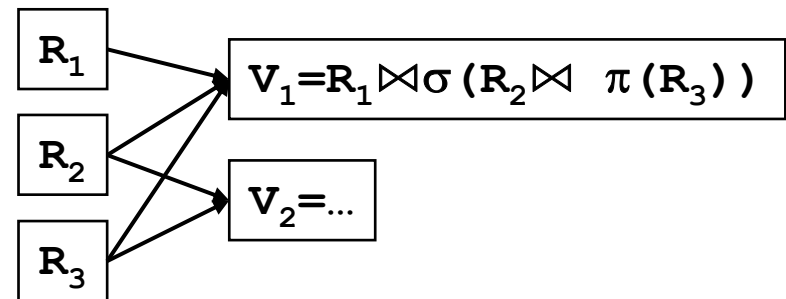


GROUP BY year\_id



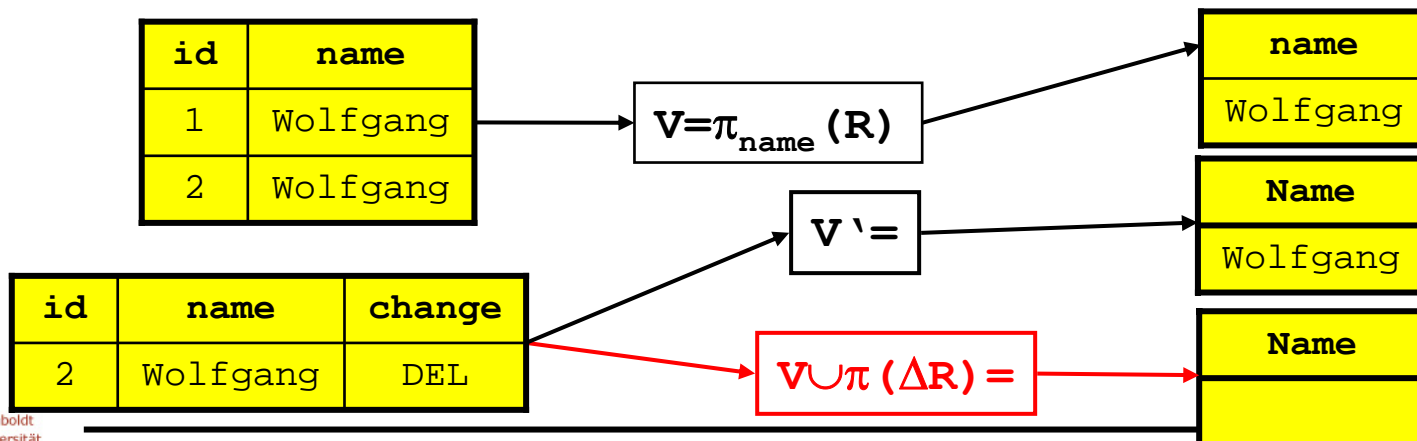
# Das Problem

- Materialisierte Sichten sind Anfragen auf **Basisdaten**
- Wenn sich Basisdaten ändern, müssen auch die **gespeicherten MV angepasst** werden
  - Sonst führt die Verwendung zu falschen Ergebnissen
- Möglichkeit
  - **Synchrone Aktualisierung**
    - Sowie sich Basisdaten ändern, MV's anpassen
    - Z.B. über Trigger
    - Teuer im laufenden Betrieb
  - Führt schnell zu **inkonsistenten Daten**
    - Wie?



# MV: Selektion und Projektion

- MV ohne Join aber mit Selektion und/oder Projektion
- Selektion
  - Ausnutzung der Distributivität der Selektion
  - Sei also  $V = \sigma(R)$
  - $V' = \sigma(R') = ?$
  - $V' = \sigma(R') = \sigma(R \cup \Delta R) = \sigma(R) \cup \sigma(\Delta R) = V \cup \sigma(\Delta R)$
  - Sehr schön: Wir müssen **nur das Delta** betrachten
- Projektion
  - Gilt  $V' = \pi(R') = \pi(R \cup \Delta R) = \pi(R) \cup \pi(\Delta R) = V \cup \pi(\Delta R) ?$



# Auswahl materialisierter Sichten

---

- Wir betrachten nur MV **mit Aggregation**
  - Berechnung von Aggregaten sind tendenziell teuer
  - Wiederholtes Scannen sehr großer Tabellen
  - Vorberechnung ist daher attraktiv
- Problem: Der vollständige CUBE benötigt viel zu viel Platz
  - Beispiel: 5 Dimensionen a 50 Knoten
    - Gleichverteilung der Werte auf alle Knoten
  - Die höchste Granularität hat  $\sim 50^5 = 3 \cdot 10^8$  Tupel
  - Dazu:  $\sim 5 \cdot 50^4 + 20 \cdot 50^3 + 20 \cdot 50^2 + 5 \cdot 50 = 3.3 \cdot 10^7$  Tupel mit weniger Gruppierungsattributen
  - Tatsächliche Zahlen abhängig vom **Füllgrad des Cube** ab
    - Wie schätzen?

# Problem

---

- Definition

*Gegeben eine Workload  $Q$ ,  $|Q|=n$ , mit Anfragefrequenzen  $h$ , ein Aggregationsgitter  $A$ , eine Kostenmatrix  $c$  und eine Konstante  $m$ . Das **MV-Selektion Problem** sucht die Menge  $M \subseteq A$ , die den folgenden Ausdruck minimiert*

$$c(Q, M) = \sum_{i=1 \dots n} h_i * \min_{a_j \in M} (c(q_i, a_j))$$

*unter Beachtung der **Nebenbedingung***

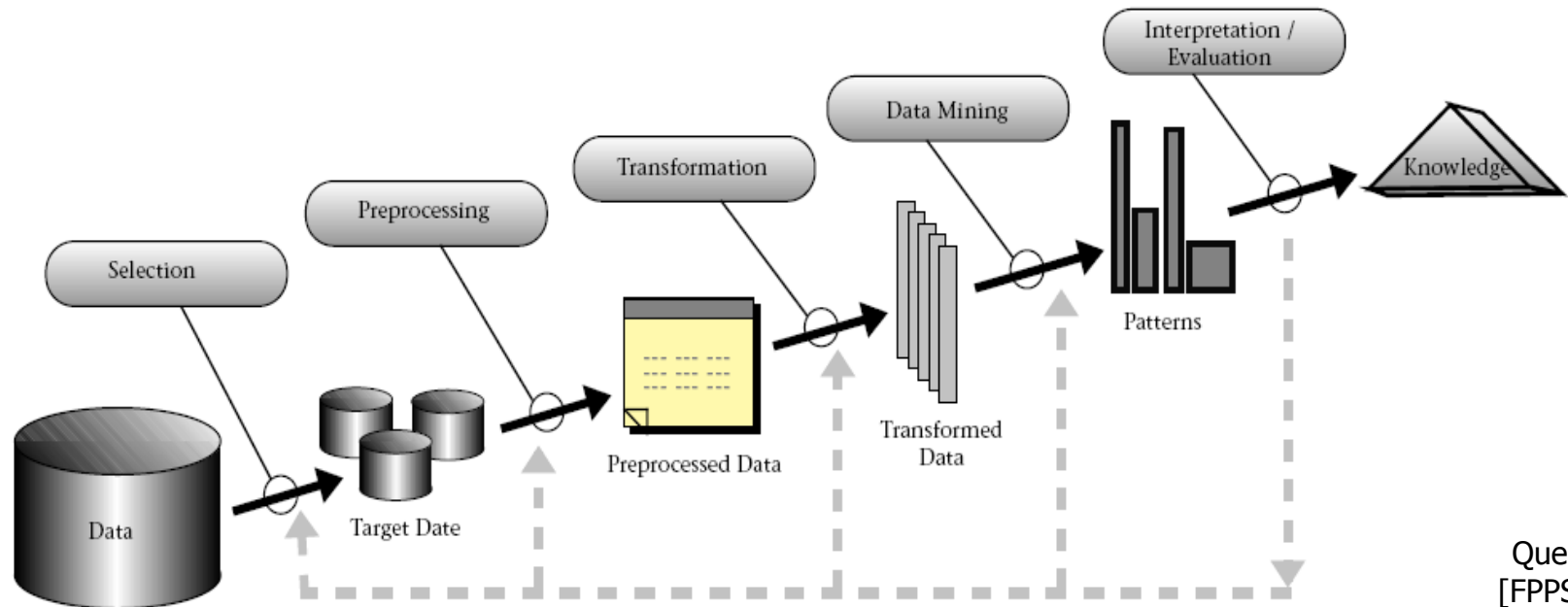
$$\sum_{a_i \in M} |a_i| \leq m$$

# Heute: Data Mining

---

- Riesige **Datenberge**
  - Business: Weblogs, e-commerce, Telekoms, Transaktionen, Börsendaten, Kundendaten, ...
  - Forschung: Astronomie, Teilchenphysik, Bioinformatik, ...
  - Jeder: Nachrichten, Blogs, Webseiten, Fotos, ...
- Manuelle Analyse ausgeschlossen
  - Millionen oder **Milliarden von Datensätzen**
  - Hochdimensionale Daten mit Hunderten von Attributen
  - „**Data-Driven**“: Data Mining generiert die Hypothese und soll sie auch gleich testen
    - Vorsicht: Irgendwas findet man immer
- „We are drowning in data and starving for knowledge“
  - „Was machen Kunden eigentlich auf meiner Webseite?“

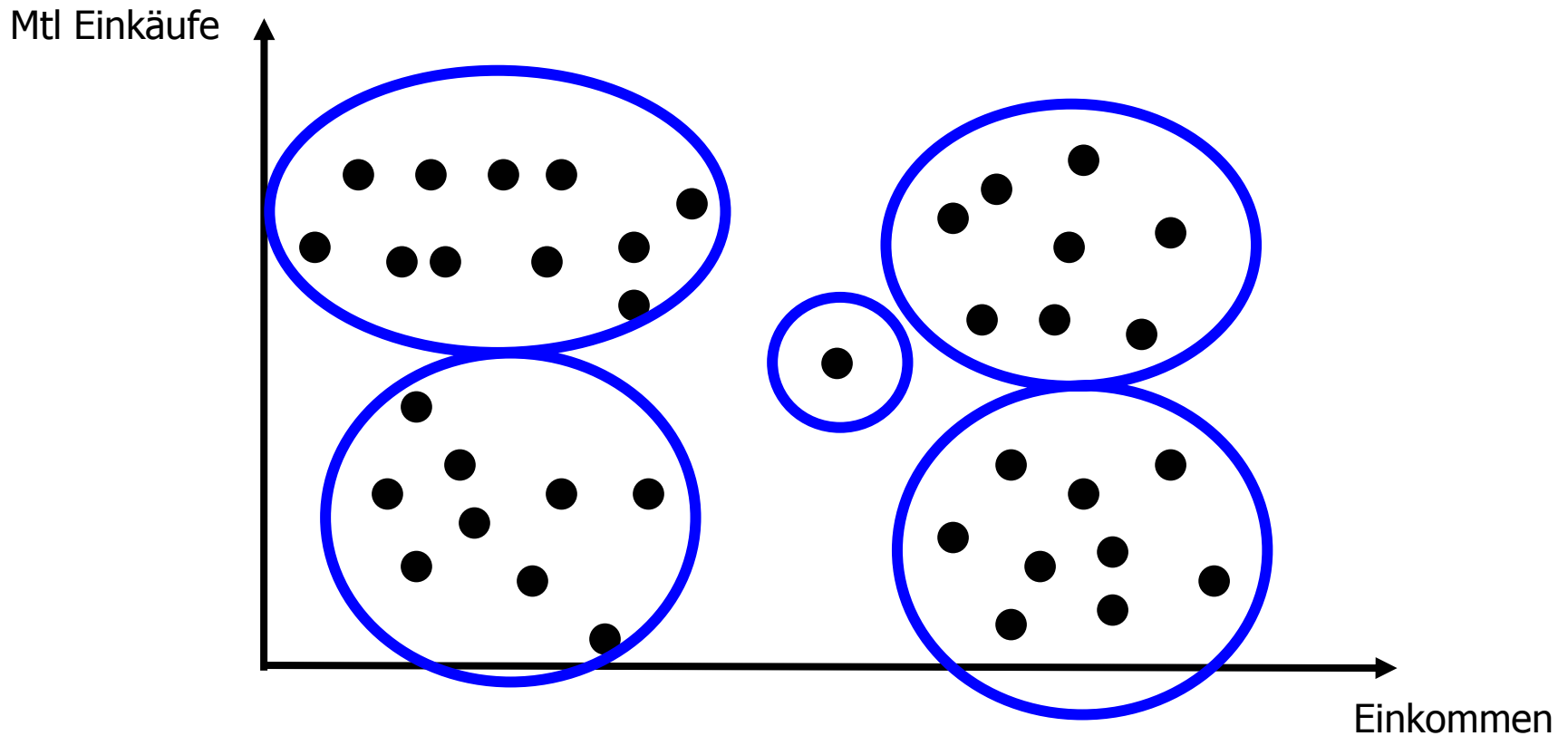
# KDD als Prozess



Quelle:  
[FPPS96]

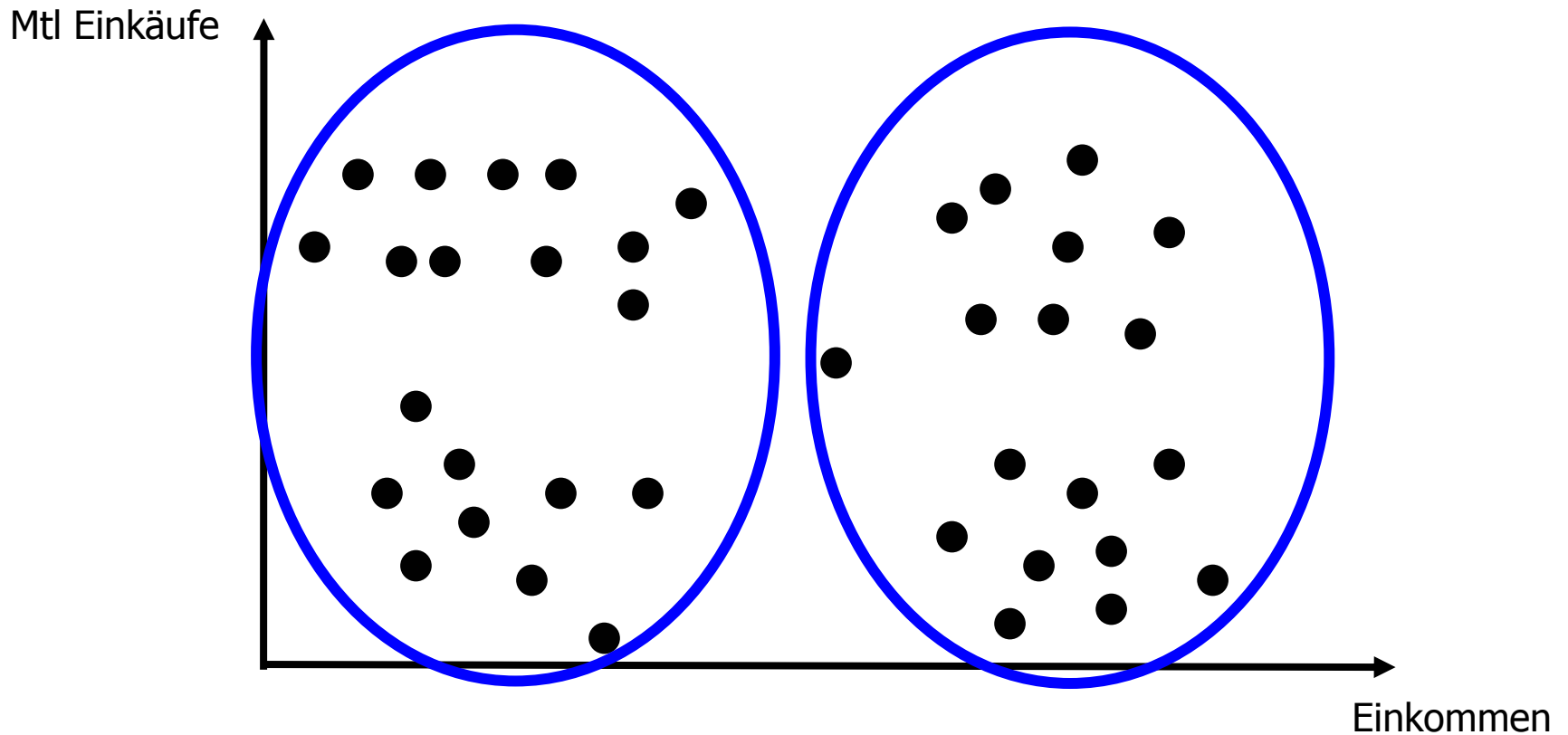
- Anwendung verstehen
- Daten auswählen
- Data cleaning und preprozessieren
- Datenreduktion (Dimensionen, Gruppierung, ...)
- Explorative Datenanalyse
- Data Mining
- Interpretation und Anwendung der Ergebnisse

# Beispiel 1



- Vier Cluster und ein **Ausreißer**(-Cluster)
- Überlappungsfreie, konvexe Cluster

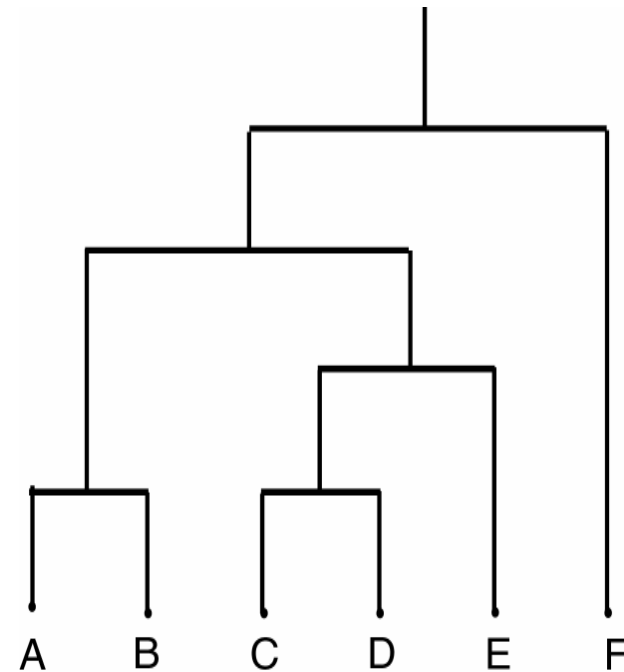
# Beispiel 2



- Zwei Cluster
- Besser?

# Hierarchisches Clustering

- Auch: UPGMA
  - Unweighted Pair-group method with arithmetic mean
- Bottom-Up Berechnung eines binären Baums (**Dendogramm**)
- Algorithmus
  - Berechne **Abstandsmatrix M**
    - Ähnlichkeit aller Objekte zu allen anderen Objekten
  - Wähle Paar  $o_1, o_2$  mit dem kleinsten Abstand
  - Berechne  $x = m(o_1, o_2)$  (den Mittelpunkt)
  - Lösche  $o_1, o_2$  aus M; füge x ein
  - Berechne Abstand von x zu allen verbleibenden Objekten/Clustern in M
    - Als Mittelwert der Abstände von  $o_1$  und  $o_2$
  - Iteriere, bis M leer ist



# K-Means

---

- K-Means ist wahrscheinlich die bekannteste Clusteringmethode
  - Mit sehr vielen Varianten
- Benötigt die **Anzahl  $k$  von Clustern als Parameter**
- Berechnet ein lokales Optimum bezüglich  $k$ -Güte
  - Keine Garantien für globale Optimalität bzgl.  $k$ -Güte
- Algorithmus
  - Wähle zufällig  $k$  Clustermittelpunkte
  - Iteriere
    - Für alle Objekte
      - Berechne Abstand jedes Objekts zu jedem Clustermittelpunkt
      - Weise Objekt seinem **nächsten Clustermittelpunkt** zu
    - Wenn sich keine Objektzuordnung mehr geändert hat, dann STOP
    - Für alle Cluster
      - Berechne **neues Clusterzentrum**

# Dichtebasiertes Clustering [EKSX96]

---

- Sucht nach **Regionen hoher Dichte**
  - Anzahl Cluster ist nicht vorbestimmt
  - Findet auch **nicht konvexe** Cluster
- Bekanntester Vertreter: **DBSCAN**
- Wie definiert man „dichte“ Bereiche?
  - Jeder Punkt eines Clusters hat **viele nahe Nachbarn**
  - Alle Punkte eines Clusters sind über nahe Nachbarn voneinander erreichbar



# Algorithmus

---

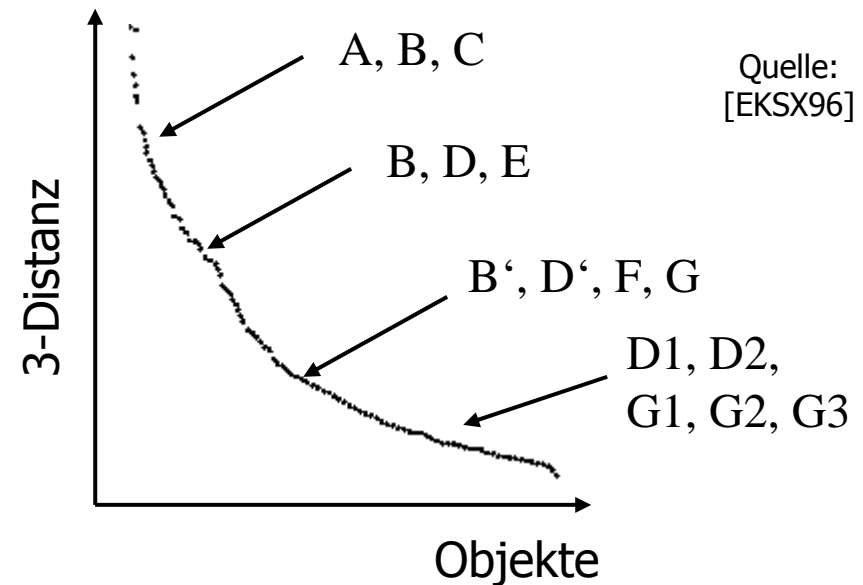
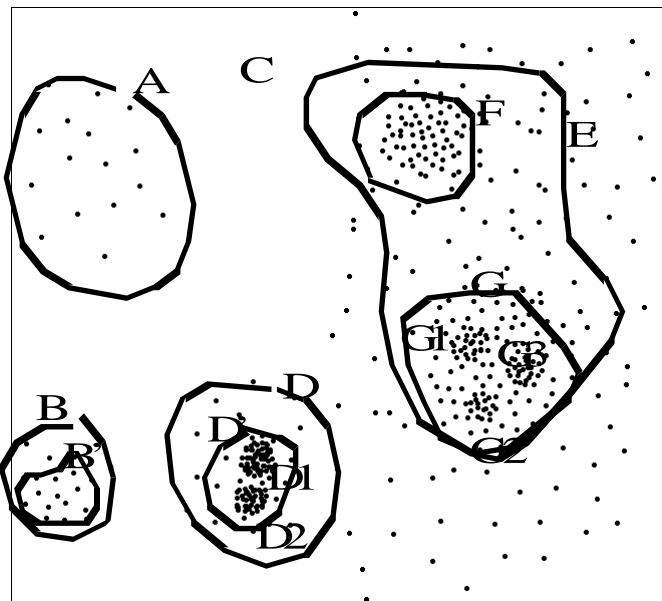
- Aus der Definition ergibt sich unmittelbar ein Algorithmus zum Finden **des dichtebasierten Clusterings** einer Objektmenge  $O$ 
  - Das Clustering ist eindeutig

```
clusterCount := 1;
for i from 1 to |O| do // Alle Punkte ansehen
 o := O.get(i);
 if o.clusterID = NULL and // Punkt in keinem Cluster
 kernobjekt(o) then // ... und im Kernbereich
 // eines (neuen) Clusters

 if expCl(o, O, clusterCount) != 0 // Cluster berechnen
 then clusterCount++; // Nächster Cluster
```

# Probleme

- Und wenn es keinen Knick gibt?
- Schwierig: **Stark unterschiedlicher Dichte** in verschiedenen Bereichen des Raumes
  - Viele (kleine) Knicks
  - Mit einem Parameterpaar  $\minpts, \varepsilon$  kann man das nicht beschreiben
- Auch schwierig: Cluster und Rauschen sind nicht gut getrennt
  - Kein Knick



Quelle:  
[EK SX96]

# Etwas komplexer

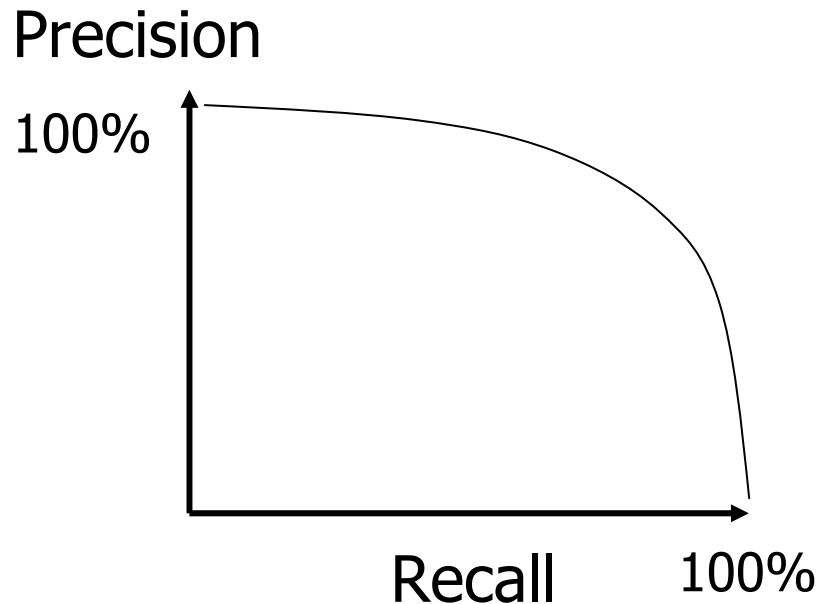
---

| ID | Alter | Einkommen | Risiko  |
|----|-------|-----------|---------|
| 1  | 20    | 1500      | Hoch    |
| 2  | 30    | 2000      | Niedrig |
| 3  | 35    | 1500      | Hoch    |
| 4  | 40    | 2800      | Niedrig |
| 5  | 50    | 3000      | Niedrig |
| 6  | 60    | 6000      | Hoch    |

- Welches Risiko schätzen wir für eine Person von 45 Jahren mit 4000 Euro Einkommen?

# Area under the Curve, ROC

---



- Fast alle Klassifikationsalgorithmen haben einen **Parameter**, an dem man zwischen Precision und Recall verschieben kann
- Welcher Faktor wichtiger ist – anwendungsabhängig
  - Konservative Bank: Bloß keine Risiken, lieber mal einen Kredit ablehnen
  - Aggressive Bank: Lieber mal ein Risiko eingehen, als Kredite ablehnen

# Was tun gegen Overfitting?

---

- Wenn man auf den Trainingsdaten lernt und von den Testdaten die Klassen nicht kennt – wie kann man dann den Grad an Overfitting beurteilen?
- **Cross-Validierung** (Leave-one-out)
  - **Teilen der Trainingsdaten** in  $k$  zufällige Partitionen
  - $k$ -maliges Lernen auf  $k-1$  Partitionen und Anwendung des Modells auf der  $k$ 'ten Partition
  - Die Ergebnisse (Precision, Recall) sollten sehr nahe beieinander liegen
    - Und die Ergebnisse sollten besser sein, je größer  $k$  – größere Trainingsmenge
  - Man nimmt meist den Mittelwert als **erwartete Performance** des Klassifikators auf neuen Daten

# Wahrscheinlichkeiten

---

- Gesucht:  $p(c|o)$
- Wir haben
  - $O$  mit binären Features  $f_1, \dots, f_n$
  - Die **A-Priori Wahrscheinlichkeit**  $p(f)$  jedes Features  $f$ 
    - Bei vielen Objekten ist  $f$  wahr?
  - Die **A-Priori Wahrscheinlichkeiten**  $p(c)$  aller Klassen  $c \in C$ 
    - Wie viele Objekte aus  $O$  gehören zu jeder Klasse?
  - Die **bedingten Wahrscheinlichkeiten**  $p(f|c)$ , dass Feature  $f$  bei Klasse  $c$  wahr ist
    - Wie oft ist  $f$  bei allen Objekten der Klasse  $c$  wahr?
- Wir formulieren um und wenden Bayes' Theorem an

$$p(c | f_1, \dots, f_n) = \frac{p(f_1, \dots, f_n | c) * p(c)}{p(f_1, \dots, f_n)} \approx p(f_1, \dots, f_n | c) * p(c)$$

# SQL

---

- Wir brauchen  $p(f_i|c_j)$ ,  $p(c_j)$ ,  $p(f_i)$
- Wie kriegen wir die **effizient mit SQL?**
  - Schema: `obj(oid, fid, fvalue)`, `class(oid, cid)`

- $p(c_j)$

```
SELECT cid, count(*)
FROM class
GROUP BY cid;
```

- $p(f_i)$

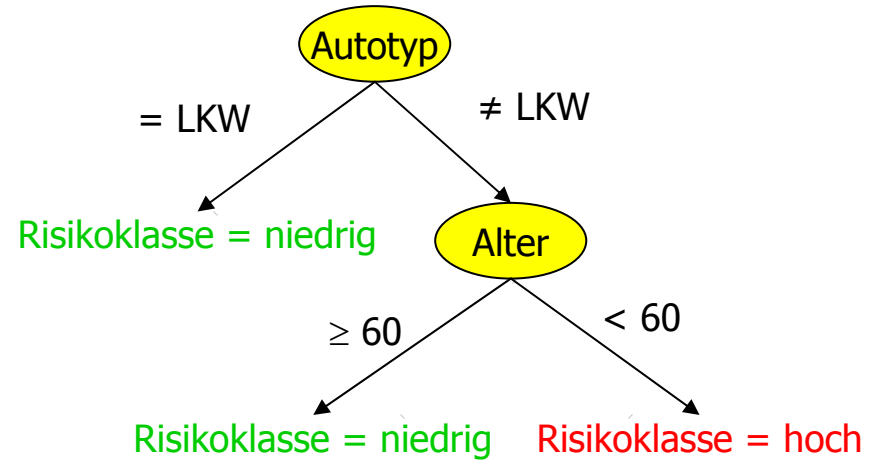
```
SELECT fid, count(*)
FROM object
WHERE fvalue = ?
GROUP BY fid;
```

- $p(f_i|c_j)$

```
SELECT cid, fid, count(*)
FROM object o, class c
WHERE o.oid=c.oid AND
 fvalue = ?
GROUP BY cid, fid;
```

# Entscheidungsbäume

| ID | Alter | Autotyp | Risiko  |
|----|-------|---------|---------|
| 1  | 23    | Familie | hoch    |
| 2  | 17    | Sport   | hoch    |
| 3  | 43    | Sport   | hoch    |
| 4  | 68    | Familie | niedrig |
| 5  | 32    | LKW     | niedrig |



- Definition

Ein *Entscheidungsbaum* ist ein Baum mit

- Jeder innere Knoten ist mit einem Attribut beschriftet
- Die ausgehenden Kanten eines inneren Knoten sind mit *Bedingungen an die Werte des Vaterknotens* beschriftet, deren Wertebereiche sich nicht überschneiden
- Jedes Blatt ist mit einer Klasse beschriftet
- Die Bedingungen einer Kante  $e$  widersprechen nie der Konjunktion der Bedingungen des Pfades von Root zum Vater von  $e$

# ID3 etc.: Entropy [Qui86, Qui93]

---

- Klassische DTI Algorithmen: ID3 / CART / C4.5 / C5.0
- Ziel: Anzahl der **erwarteten Vergleiche** minimieren
  - Also Baum balanciert und niedrig halten
  - Heuristik dazu: Wähle Split so, dass die **Entropie der neuen Partitionen maximal kleiner** wird

- Definition

Die **Entropie** einer Trainingsmenge  $T$  bzgl. Klassen  $C=\{c_1, \dots, c_n\}$  ist definiert als

$$\text{entropy}(T) = - \sum_{i=1}^n p_i * \log(p_i)$$

mit  $p_i = \text{Häufigkeit der Klasse } c_i \text{ in } T$

- Bemerkung

- Entropie misst die **Reinheit** von  $T$  bzgl.  $C$ 
  - $\text{entropy}(T) = 1$ : maximale Verwirrung
  - $\text{entropy}(T) = 0$ : homogene Menge (ein  $p_i=1$ , alle anderen  $=0$ )

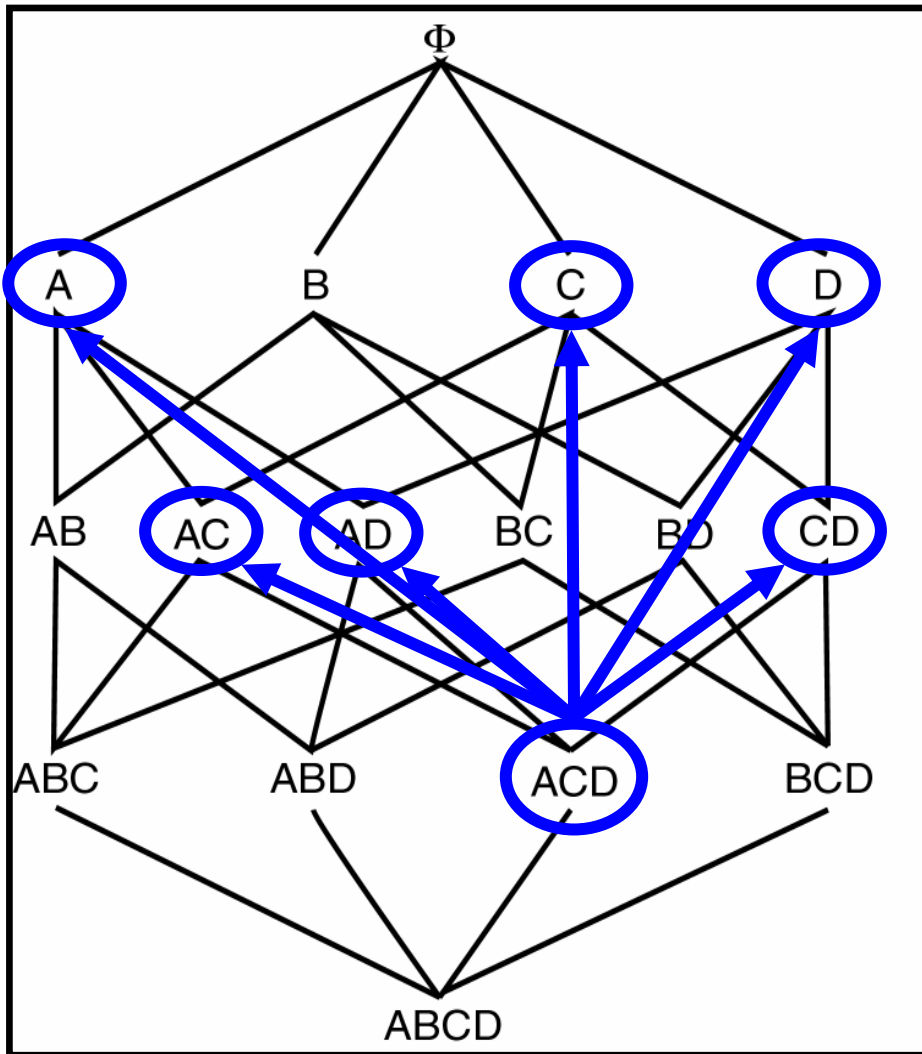
# Aufgabe

- Klassisches Beispiel:  
**Warenkorbanalyse**
- Transaktionen mit binären Attributen
  - Produkt (Item) gekauft oder nicht



- Frage: **Welche Produkte werden häufig zusammen gekauft?**
  - Frequent itemsets
  - Bier und Windeln?
- Data Warehouse Kontext
  - Viele Millionen Transaktionen (T)
  - Viele tausend Produkte (I)
  - Man kann nicht  $2^I$  **verschiedene Itemsets** zählen
  - Optimierte Algorithmen notwendig

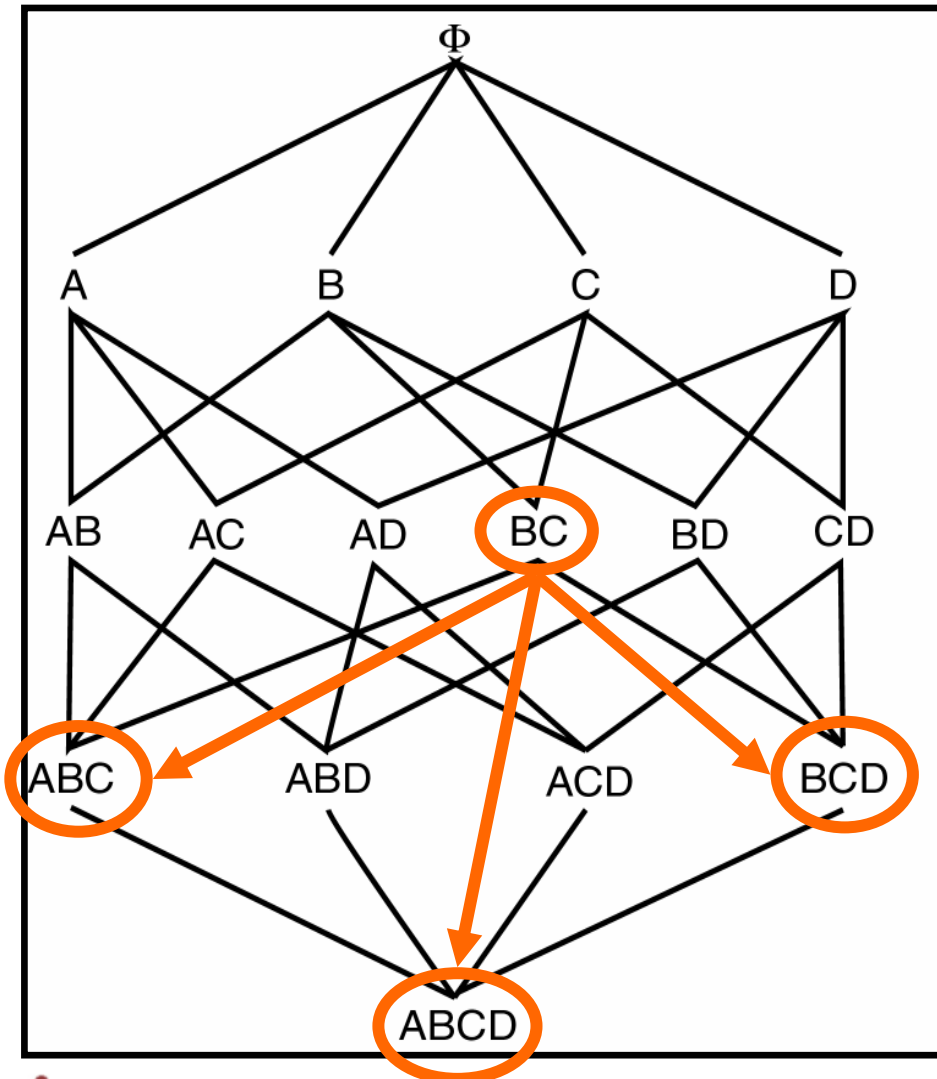
# Beispiel



- Downward

- Wenn ACD häufig ist, muss AC, AD, CD, A, C, D häufig sein
- ABCD kann häufig sein

# Beispiel



- Downward
  - Wenn ACD häufig ist, muss AC, AD, CD, A, C, D häufig sein
  - ABCD kann häufig sein
- Upward
  - Wenn BC nicht häufig ist, kann ABC, BCD und ABCD nicht häufig sein
  - B und C können häufig sein

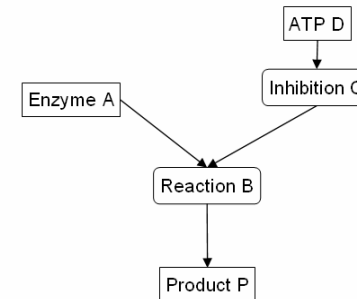
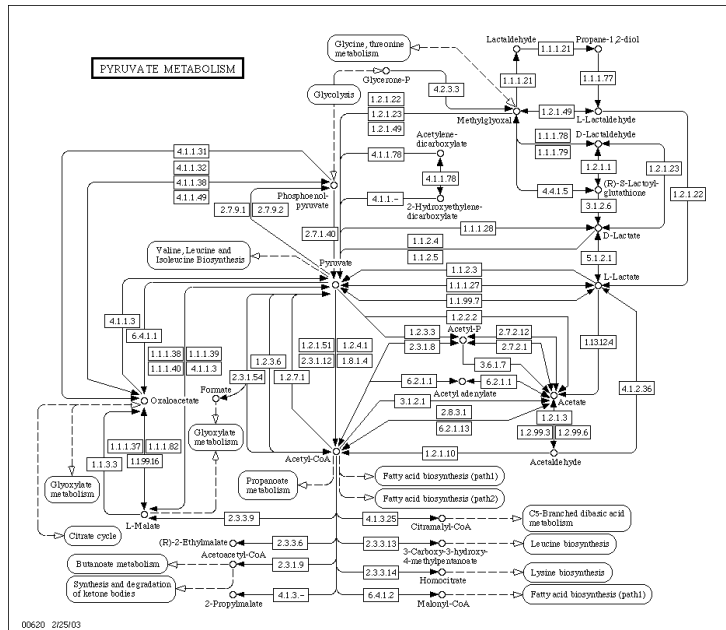
# Algorithmus: Häufige Itemsets

```
L1 := frequent 1-Itemsets aus I; // Ein Scan von T
k := 2; // k: Größe der Itemsets
while Lk-1 ≠ ∅ { // Noch Obermg sinnvoll?
 Ck := generateCandidates (Lk-1); // später
 for each t ∈ T // Viele weitere Scans
 C := frequent(Ck, t); // Welche Kandidaten
 for each c ∈ C { // ... sind häufig?
 c.count++;
 }
 }
 Lk := {c ∈ Ck | (c.count ≥ minsup)}; // Merken
 k++;
}
return ∪k Lk; // Alle Häufigen
```

- Benötigt **viele Scans** der Datenbank
- Verbesserungen notwendig (und viele bekannt)

- 
- Zusammenfassung
  - Eigene Rückschau
  - Ein paar Worte zur Prüfung
  - Feedback
  - Werbung

# Graphanfragen



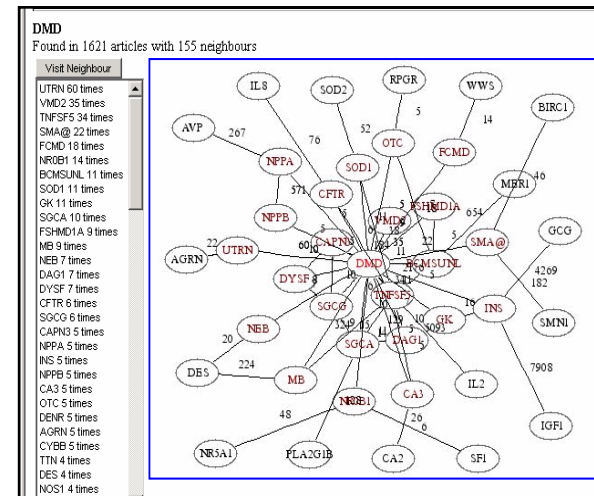
SELECT A  
 FROM A, B, C, D, E, P  
 WHERE A ISA 'enzyme' AND D ISA 'reaction'  
 AND E ISA 'reaction' AND  
 B[-1]D AND A[-1]D AND A[-1]E AND  
 C[-1]E AND D[-1]P AND E[-1]P

- Spracherweiterung und Implementierung
- Visualisierung von Anfragen und Ergebnissen
- Skalierbarkeit, Graphindizierung
- Integration mit weitere Datenquellen

# Text Mining

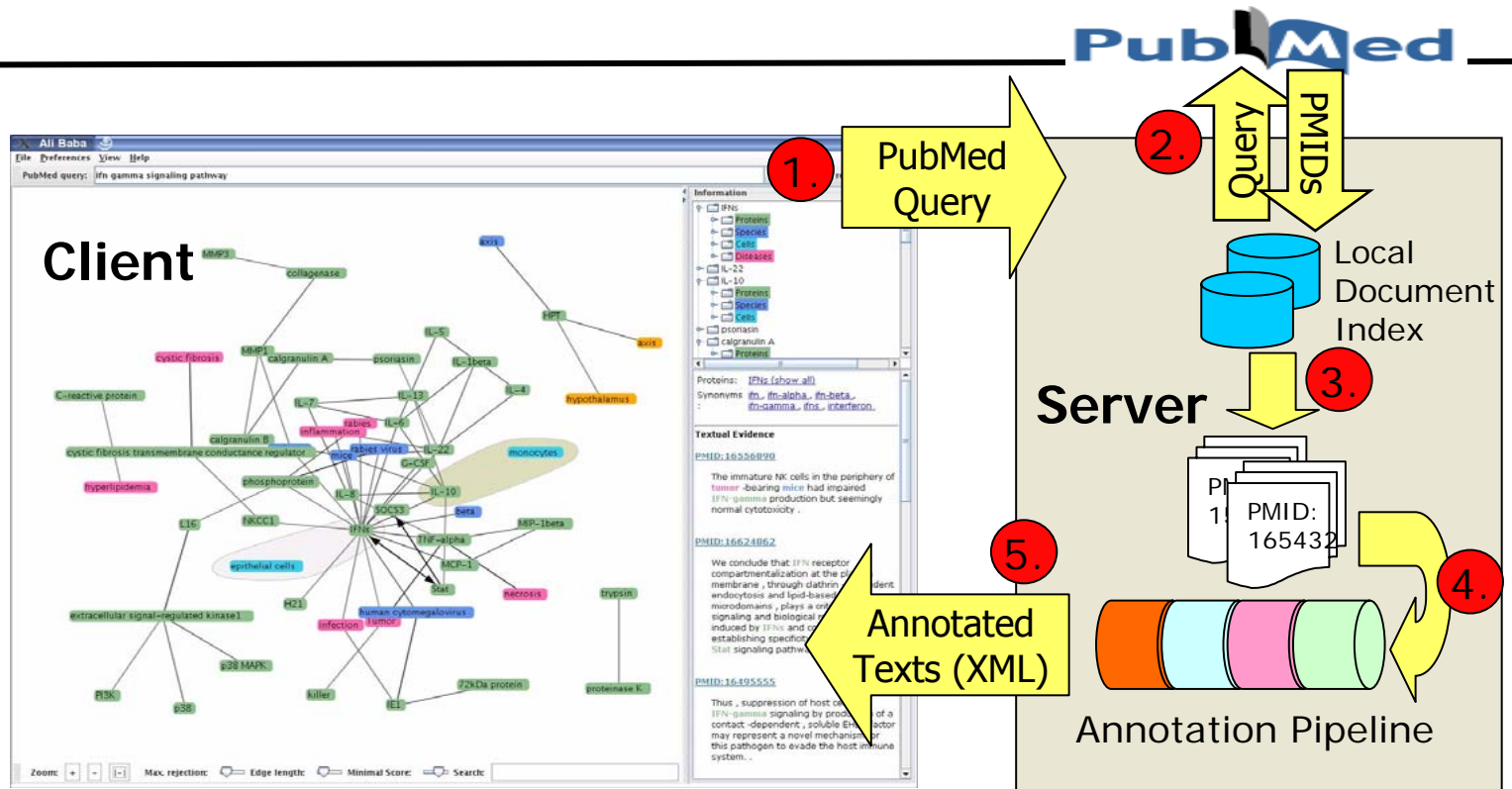
- Extraktion von Protein-Protein-Interaktionen aus biomedizinischen Texten

Disease often becomes  
Property to Therapy apy  
Cause  
Protein Protein and members  
Protein family family,  
Protein 8, Protein  
Protein examined by  
Evidence immunohistochemistry, .."



- Todo: Erkennen und Klassifizieren von Objektnamen, Erkennen von Beziehungen, Extraktion von Annotationen
- Themengebiete: Machine Learning, Linguistik, Java, Pattern Matching
- Umfeld: Team von 2-4 Personen, Kooperation mit MPI-MG und LFE Scheffer

# AliBaba



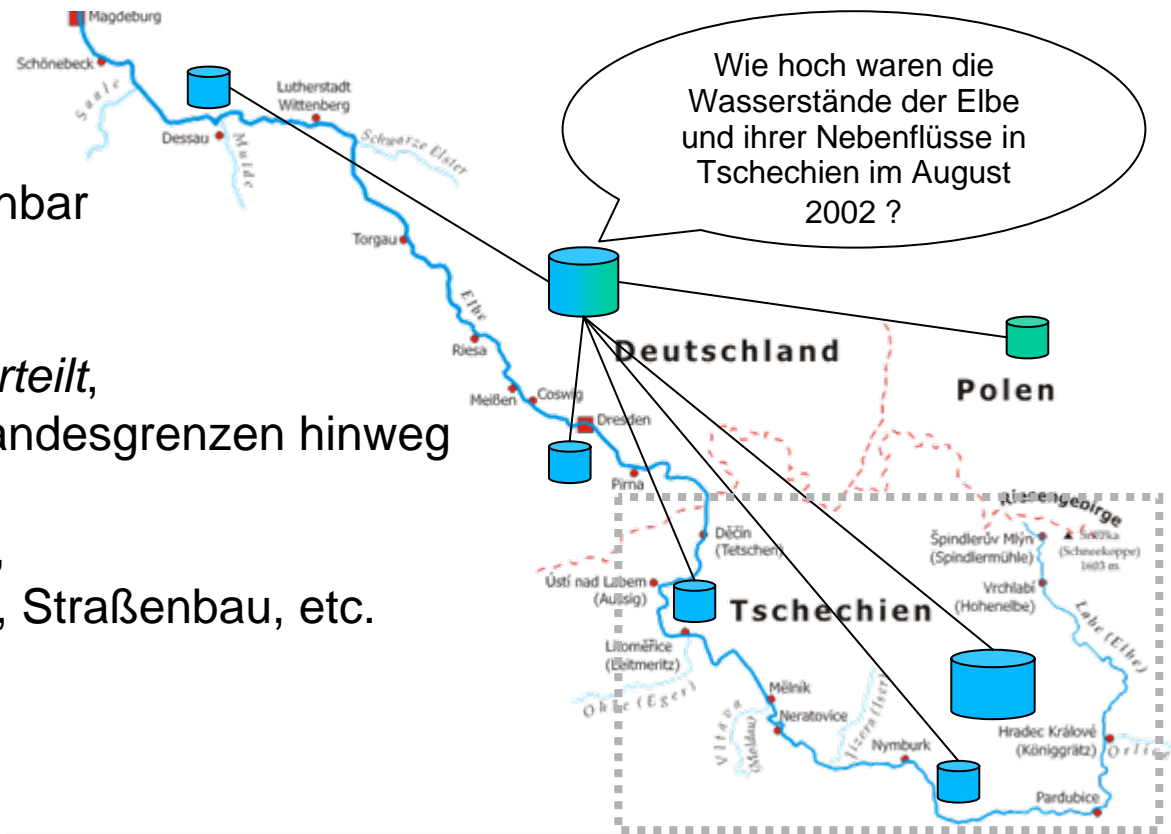
- Extraktion und Visualisierung von Beziehungen zwischen biomedizinischen Objekten
- Warum nur PubMed (Google, OMIM, UniProt, ...)
- Integration von Webquellen
- Server verbessern

# Katastrophenmanagement

*Zeitnahe Versorgung* mit Informationen *ist* entscheidend für *schnelles* und *effektives* Handeln in Fall einer Katastrophe.

Herausforderungen:

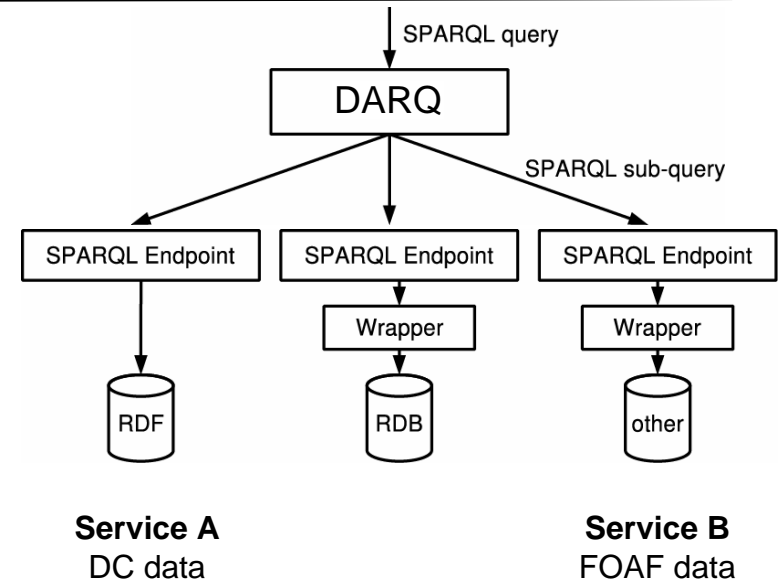
- Katastrophen sind nicht planbar  
→ *on-the-fly Integration*
- Quellen sind *hochgradig verteilt*,  
über Organisationen und Landesgrenzen hinweg
- Daten aus *vielen Domänen*,  
Wasserbau, Forstwirtschaft, Straßenbau, etc.
- *Datenqualität* ist wichtig



Karte: de.wikipedia.org

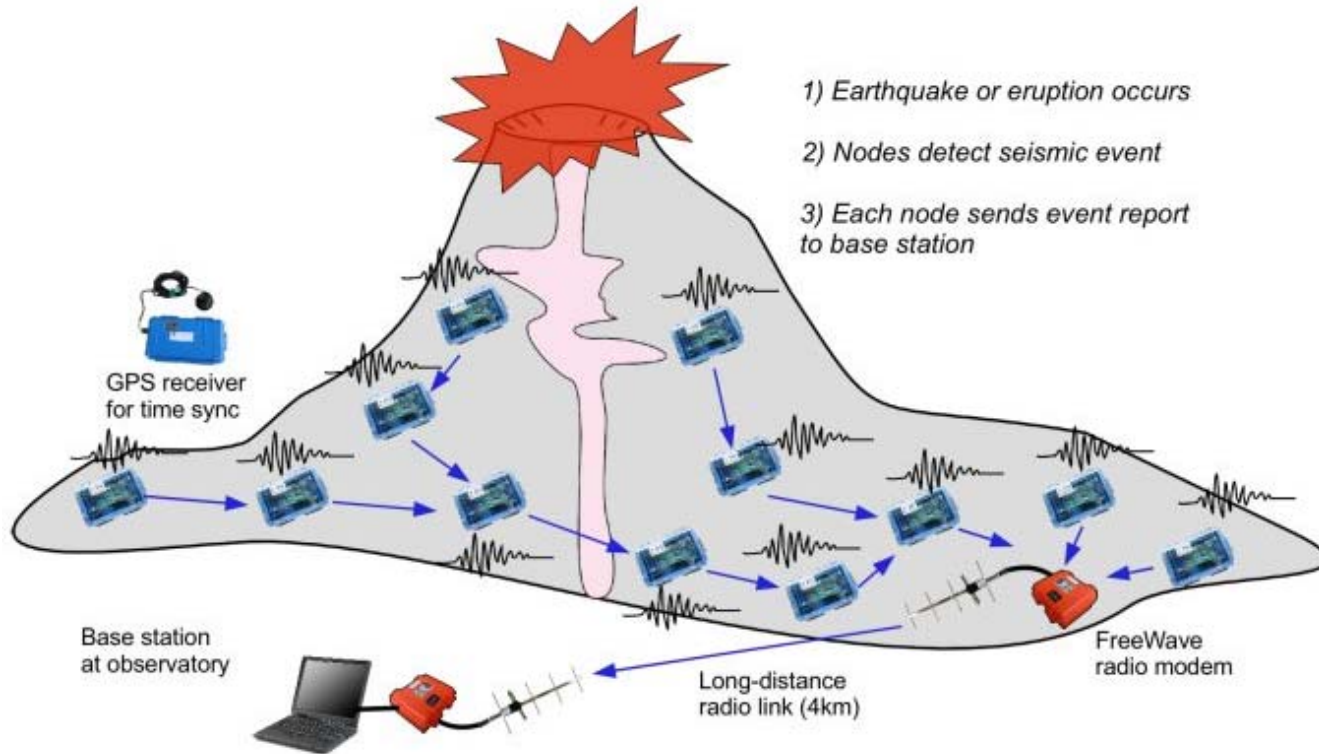
# Verteilte SPARQL Anfragen

- Mit DARQ existiert ein System für föderierte SPARQL Anfragen an verteilte Quellen
- Einfache Optimierung ist umgesetzt, aber bei vielen sich überlappenden Quellen nicht performant



- Möglichkeit 1: Verbesserung der Optimierung:
  - Evaluierung der Übertragbarkeit von existierenden Optimierungsmethoden und prototypische Implementierung, z.B. Qualitätsbasierte Optimierung
- Möglichkeit 2: Erweiterung der Anfrageplanung
  - Verschiedene Quellen nutzen unterschiedliches Vokabular
  - Implementierung eines System, dass Mappings zwischen verschiedenen Vokabularen bei der Anfrageplanung berücksichtigt

# Sensornetzwerke



# Datenmanagement in WSN

---

- Drahtlose Sensornetze: billig, flexibel, genau – „Smart Dust“
- Neue Probleme
  - Ressourcenbeschränktheit (Strom!)
  - Fehleranfällige Kommunikation
  - Dynamische Netzwerktopologien
- Studienarbeiten
  - Anbindung eines Sensornetzwerks ans Internet
  - Evaluierung von bestehenden Ansätzen zum Datenmanagement bezüglich Skalierbarkeit
- Diplomarbeiten
  - Motes und High-Level Programming Languages?
  - Simulationsumgebungen für Erdbebenfrühwarnung



# Optimale Substitutionsmatrizen

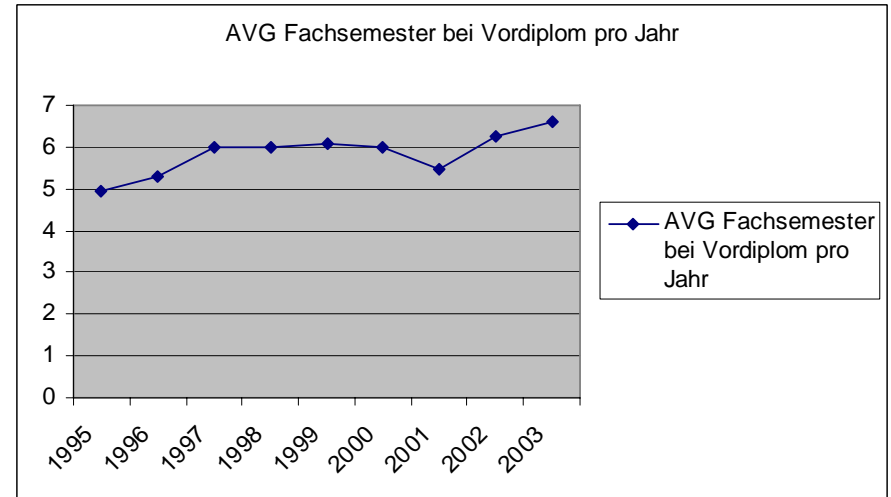
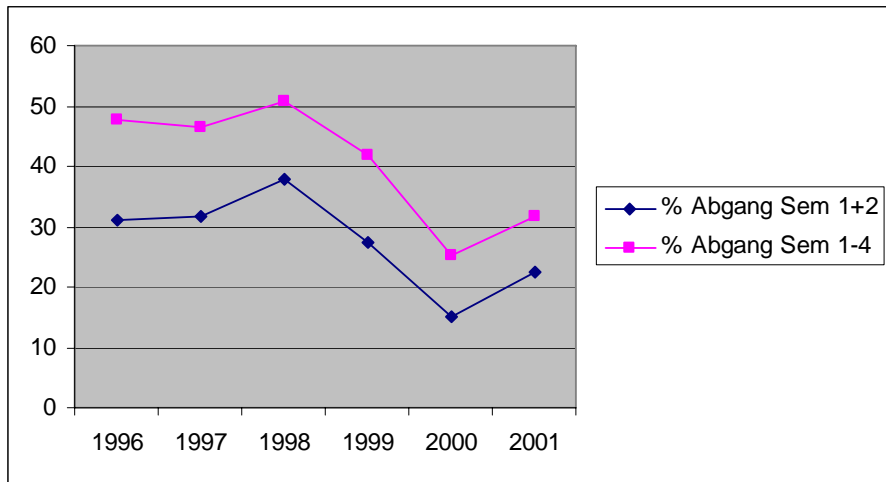
|                       |      |             |           |              |
|-----------------------|------|-------------|-----------|--------------|
| <b>Original token</b> | FADD | immediately | activates | procaspase-8 |
| <b>Class / POS</b>    | PTN  | ADV         | VRB       | PTN          |
| <b>Word stem</b>      | PTN  | immediat    | activat   | PTN          |

|      |   |    |     |    |     |     |
|------|---|----|-----|----|-----|-----|
|      |   | NN | VBZ | DT | PTN | CC  |
|      | 0 | 0  | 0   | 0  | 0   | 0   |
| PTN  | 0 | 0  | 0   | 0  | 4   | 0   |
| CC   | 0 | 0  | 0   | 0  | 0   | 5.6 |
| PTN  | 0 | 0  | 0   | 0  | 4   | 0   |
| IVBD | 0 | 0  | 0   | 0  | 0   | 0   |
| PTN  | 0 | 0  | 0   | 0  | 4   | 0   |

|      |          |     |    |     |     |      |      |     |     |
|------|----------|-----|----|-----|-----|------|------|-----|-----|
|      | -        | PTN | NN | VBZ | VBD | IVBZ | IVBD | DT  | CC  |
| -    | $\infty$ | -10 | -6 | -7  | -7  | -10  | -10  | -1  | -10 |
| PTN  | -10      | 4   | -8 | -10 | -10 | -10  | -10  | -10 | -10 |
| NN   | -6       | -8  | 2  | -3  | -3  | -5   | -5   | -2  | -5  |
| VBZ  | -7       | -10 | -3 | 2   | 1   | -1   | -2   | -2  | -2  |
| VBD  | -7       | -10 | -3 | 1   | 2   | -2   | -1   | -2  | -2  |
| IVBZ | -10      | -10 | -5 | -1  | -2  | 3    | 2    | -10 | -10 |
| IVBD | -10      | -10 | -5 | -2  | -1  | 2    | 3    | -10 | -10 |
| DT   | -1       | -10 | -2 | -2  | -2  | -10  | -10  | 2   | -2  |
| CC   | -10      | -10 | -5 | -2  | -2  | -10  | -10  | -2  | 2   |

- Optimale Substitutionsmatrix
- Komplexen Algorithmus implementieren (lineare Optimierung)
- Anwendung im Projekt Ali Baba

# Open Source BI Tools



- Auswahl Evaluation von Open Source Business Intelligence Tools
  - Clover ETL, Enhydra, octopus, Mondrian, Birt, Data Vision, Marvelite Dash Portal, ...
- Anhand konkreter Aufgabe: Studentenstatistiken des Instituts
- Erstellen einer Webapplikation mit Zugriff auf Oracle