Anwendung und Potentiale von spaltenbasierten Data-Warehouse-Systemen

Vortrag am {18,19,20}.12.2011

von Stefan Harinko
TU Dresden

Literatur

- Vortrag basiert auf:
- 1.Teil hauptsächlich auf Arbeiten von Daniel Abadi zu C-Store: Dissertation und Artikel ab 2005 (Query Execution in Column-Oriented Database Systems (2005), Integrating compression and execution in column-oriented database systems (2006))
- 2.Teil Wenbin Fang, Bingsheng He, and Qiong Luo: *Database compression on graphics processors* 2010

Einführung: Data-Warehouse

- Business-Intelligence: Auswertung der gespeicherten Informationen in einem Unternehmen
- hier: Speicherung in einem Data Warehouse
- Aufbau in Schichten: Präsentation - Analyse - Datenspeicherung

Einführung: Sternschema

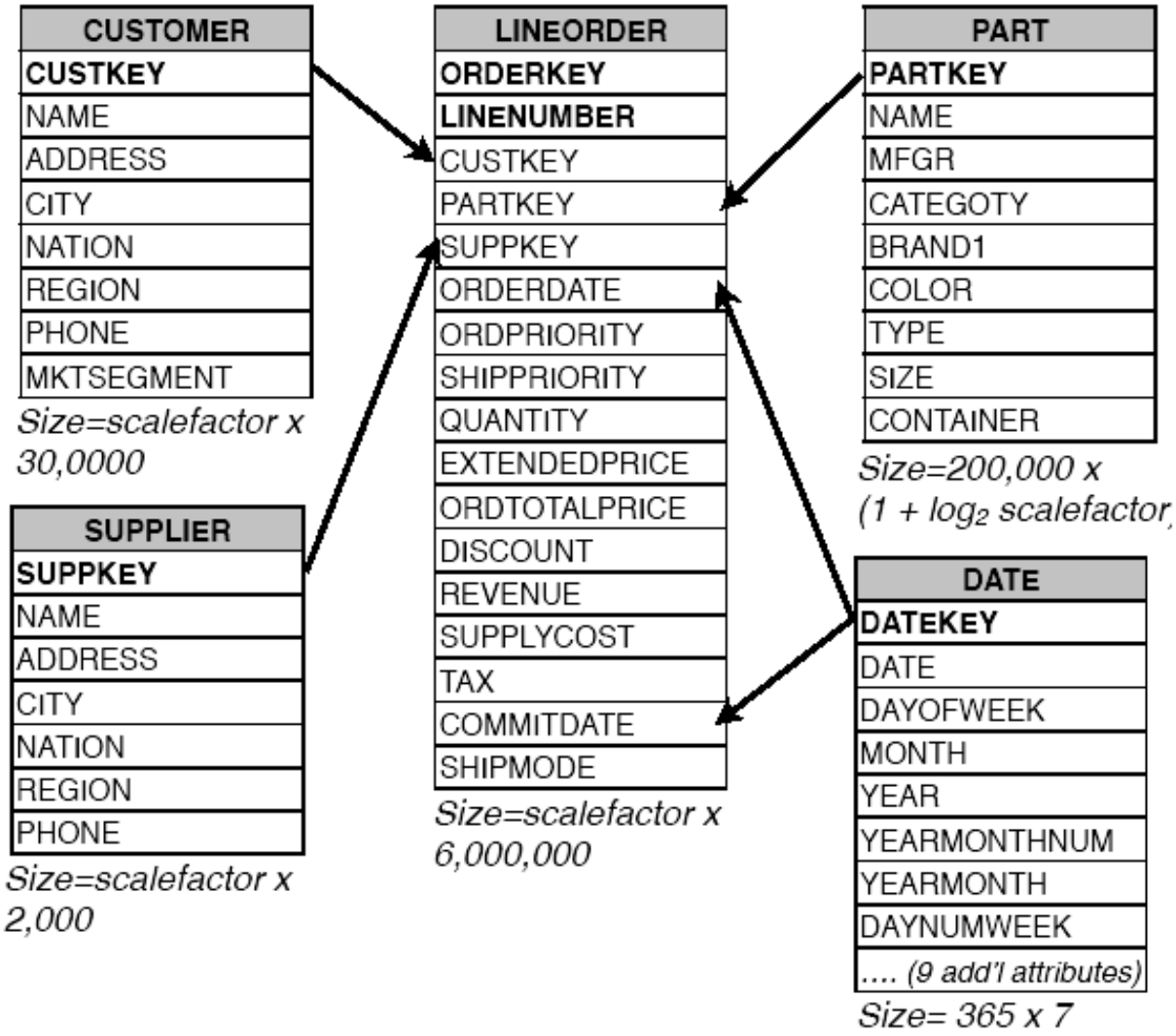

Figure: Beispiel eines Sternschemas, Quelle: Abadi (2005), S.30

Einführung: Sternschema-Abfrage

Abfrage

Abfrage: gesucht "der Umsatz aller Kunden aus ’ASIA’, die ein Produkt von einem Zulieferer aus ’ASIA’ von 1992-1997 gekauft haben"

```
SELECT c_nation, s_nation, d_year,
sum(lo_revenue)
FROM customer AS c, lineorder AS lo, supplier AS
s, date AS d
WHERE lo.custkey = c.custkey
  AND lo.suppkey = s.suppkey
  AND lo.orderdate = d.datekey
  AND c.region = 'ASIA'
  AND s.region = 'ASIA'
  AND d.year >= 1992 and d.year <= 1997
GROUP BY c.nation, s.nation, d.year
ORDER BY d.year asc, revenue desc;
```

Einführung: analytische Abfragen

- Anfragen in analytischen Anwendungen (Quelle: Abadi (2005), S.18):

- weniger vorhersagbar: "adhoc" Abfragen
- längere Laufzeit: mehr Daten werden gesammelt/ausgewertet
- mehr lese-orientiert
- Konzentration auf Attribute

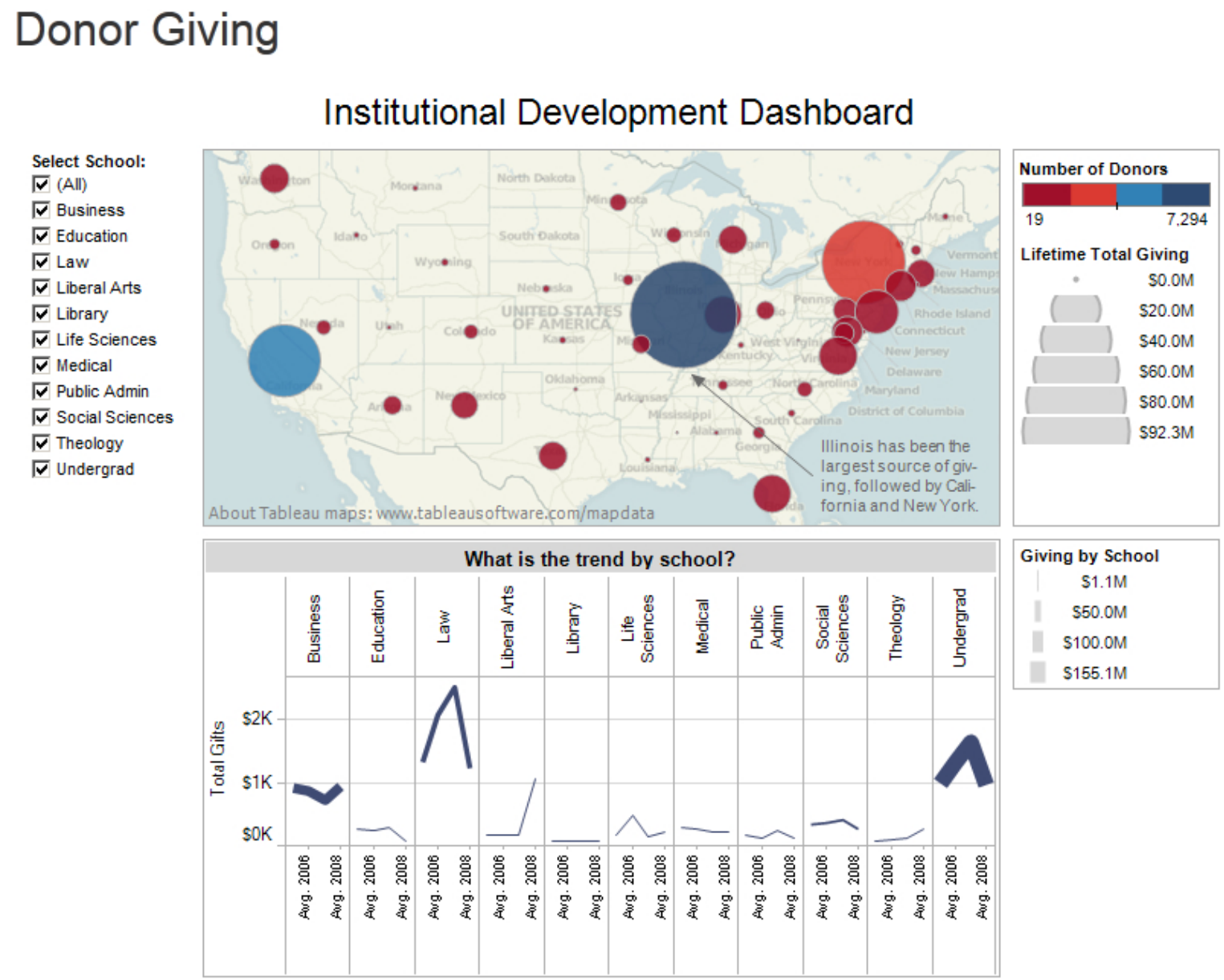

Figure: Beispiel von adhoc-Abfragen mit "tableau"-Software (Quelle: http://www.tableausoftware.com/donor-dashboard, Zugriff: 17.12.2011, 21:23 Uhr)

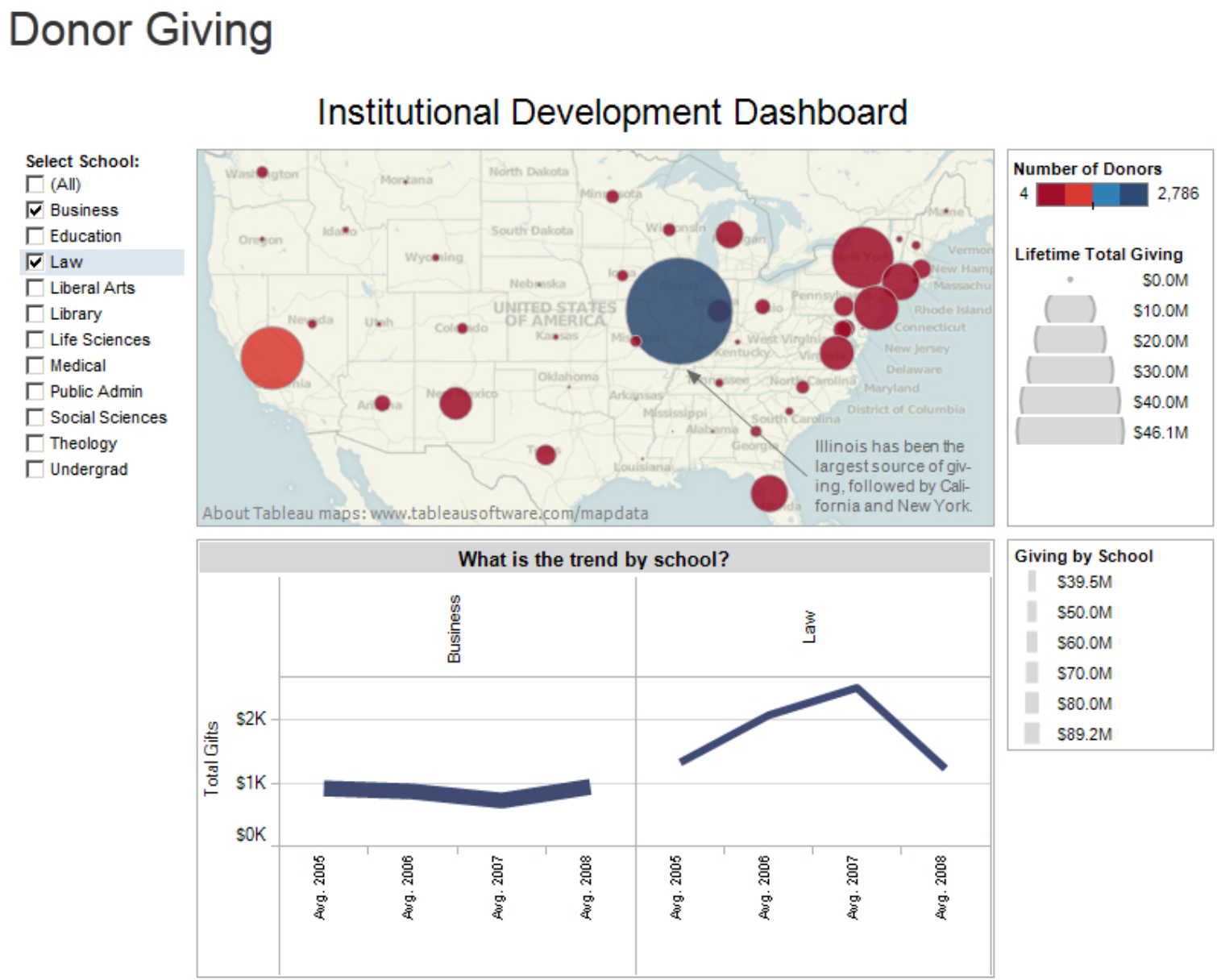

Figure: Beispiel von adhoc-Abfragen mit "tableau"-Software (Quelle: http://www.tableausoftware.com/donor-dashboard, Zugriff: 17.12.2011, 21:26 Uhr)

Aufbau

- Motivation der Spaltenorientierung
- Besonderheiten der Spaltendatenbank-Operationen
- Benchmark-Ergebnisse des "Stern-Schema-Benchmarks" (ähnlich TPC-H)
- Neue Entwicklungsrichtungen, insbesondere GPU-Beschleunigung

Motivation

CUSTKEY	NAME	ADDRESS	CITY	NATION	PHONE
1	A	Astreet 1	Acity	Anation	123
2	B	Bstreet 1	Bcity	Bnation	456
3	C	Cstreet 1	Bcity	Bnation	789

- zwei-dimensionale Tabelle muss in ein-dimensionaler Form gespeichert werden: zeilen- oder spaltenweise?
- Entscheidung mithilfe Benchmarks
  - Zeilen-Datenbank (Row-Store)
  - Spaltenorientierung simuliert in Row-Store
  - Spaltenspeicherung und Zusammensetzen vor Bearbeitung
  - Spaltenspeicherung und direktes arbeiten mit Spalten wie in C-Store

SELECT c1, c4, c6 FROM table WHERE c4 < ?

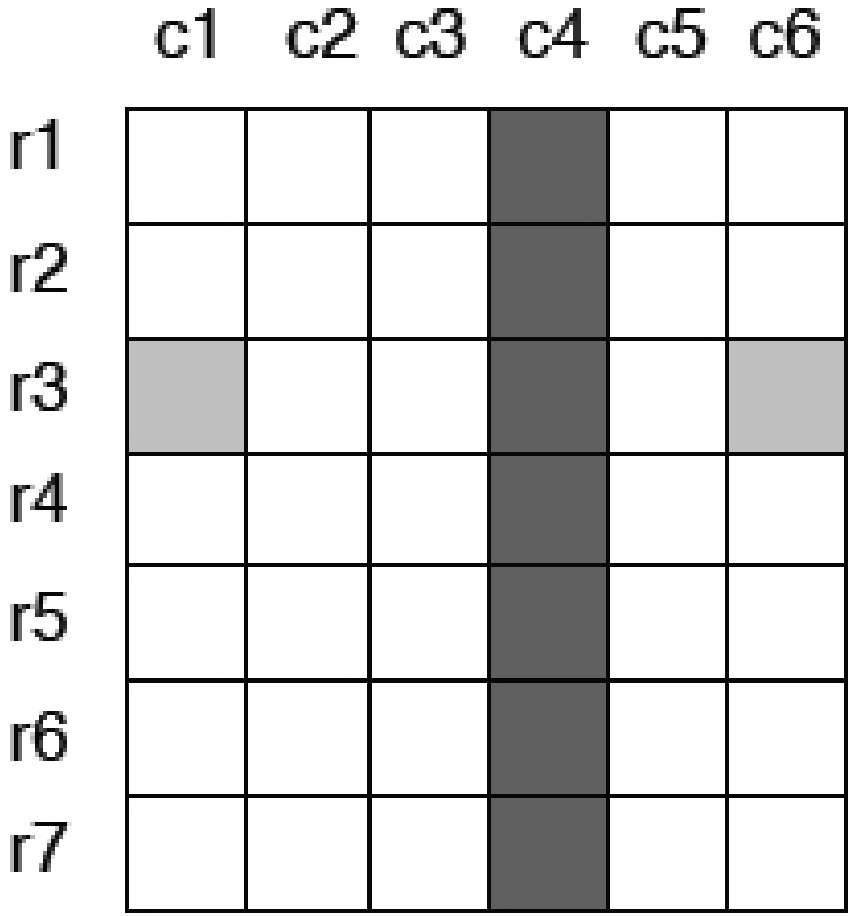

Figure: Beispielabfrage aus Plattner (2009)

Spalten vs. Zeilenoperationen

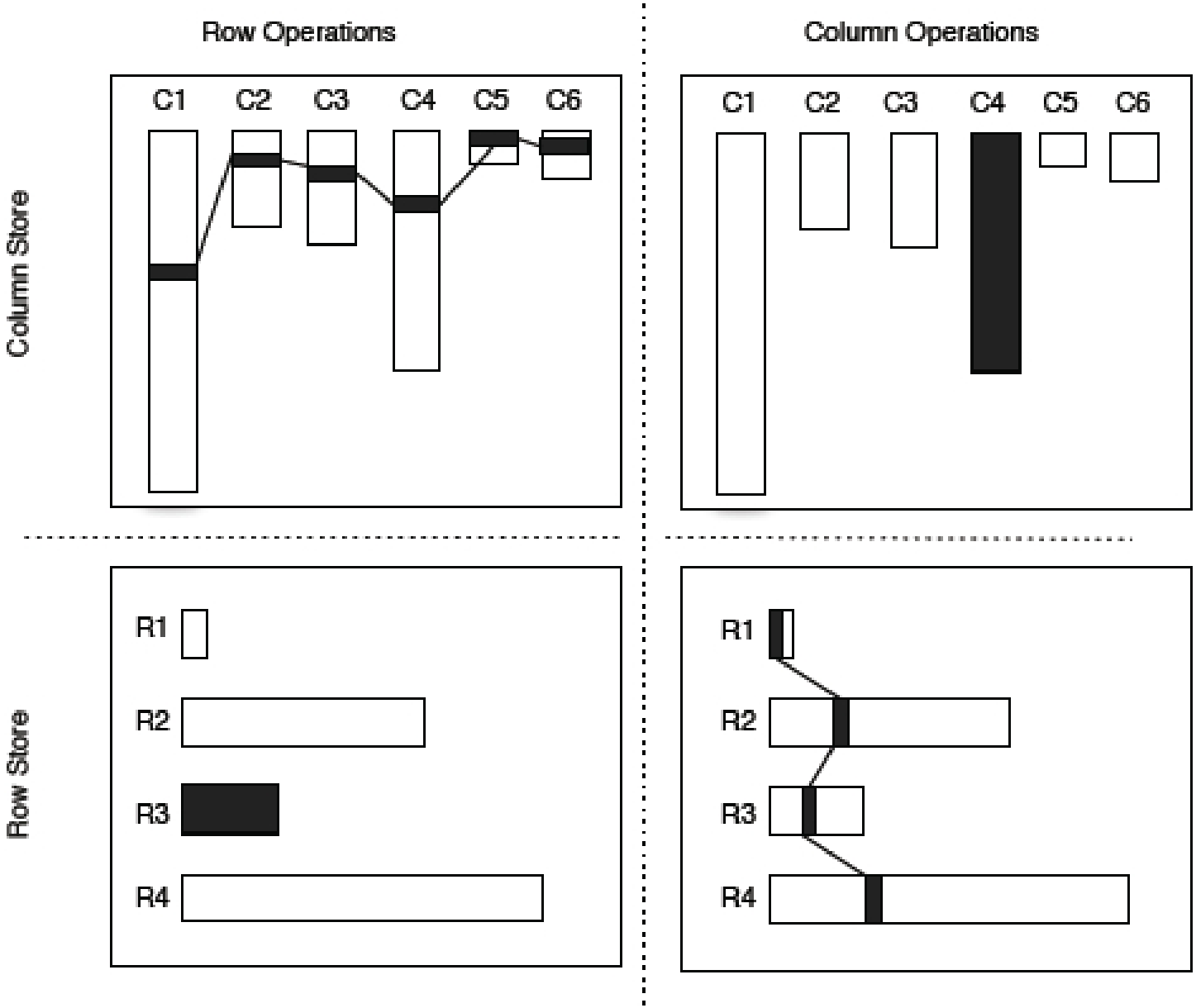

Figure: Zugriff auf Spalten/Zeilen aus Plattner (2009)

einige Besonderheiten spaltenorientierter Datenbanken

- Query-Executor
- Kompression
- Materialiserungs-Strategien
- Invisible Join: geeignet bei Sternschemata

Query-Executor: Abfrage-Pläne

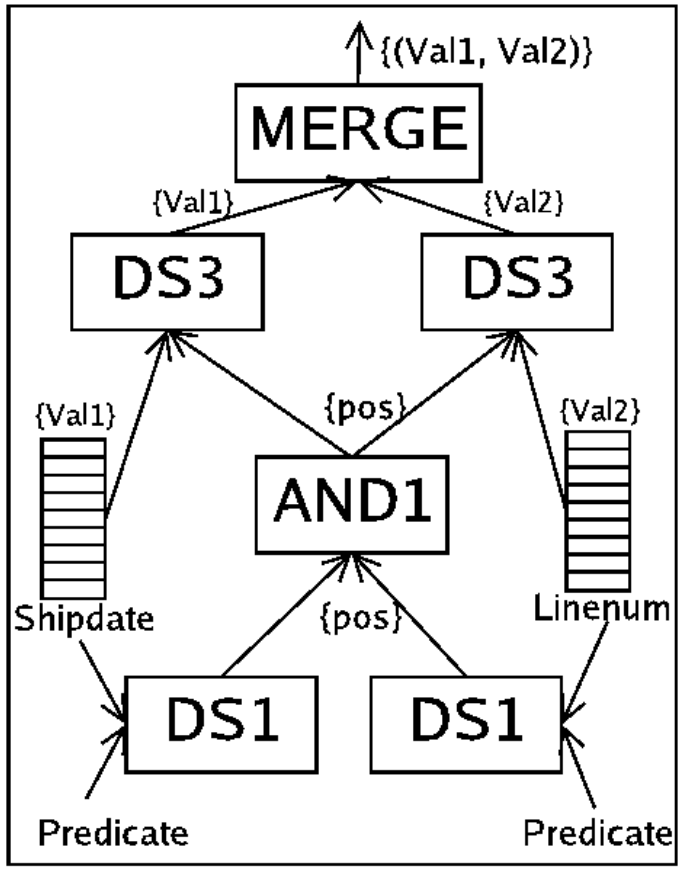

Figure: Abfrageplan (Quelle: Abadi (2005), S.43)

- keine Baumstruktur: synchronisierte Abfrage der Kindknoten
- Operatoren auf Teilen der Spalten: auf sogenannten *Vektoren*

Kompression

- Spaltenwerte sind sich sehr ähnlich und daher gut komprimierbar
- eingesparte I/O-Last vs. erhöhte CPU-Last
- Operationen direkt mit komprimierten Spalten

- Run Length Encoding z.B. ab Position 66 kommt "grün" 500 Mal vor:
- RLE: ("grün", 66, 500)
- Bitmap-Encoding Auftreten eines Werts durch 0/1 kodiert
- z.B. (1,1,2,3,3,2,1)
- Zahl 1: 1100001
- Zahl 2: 0010010
- Zahl 3: 0001100

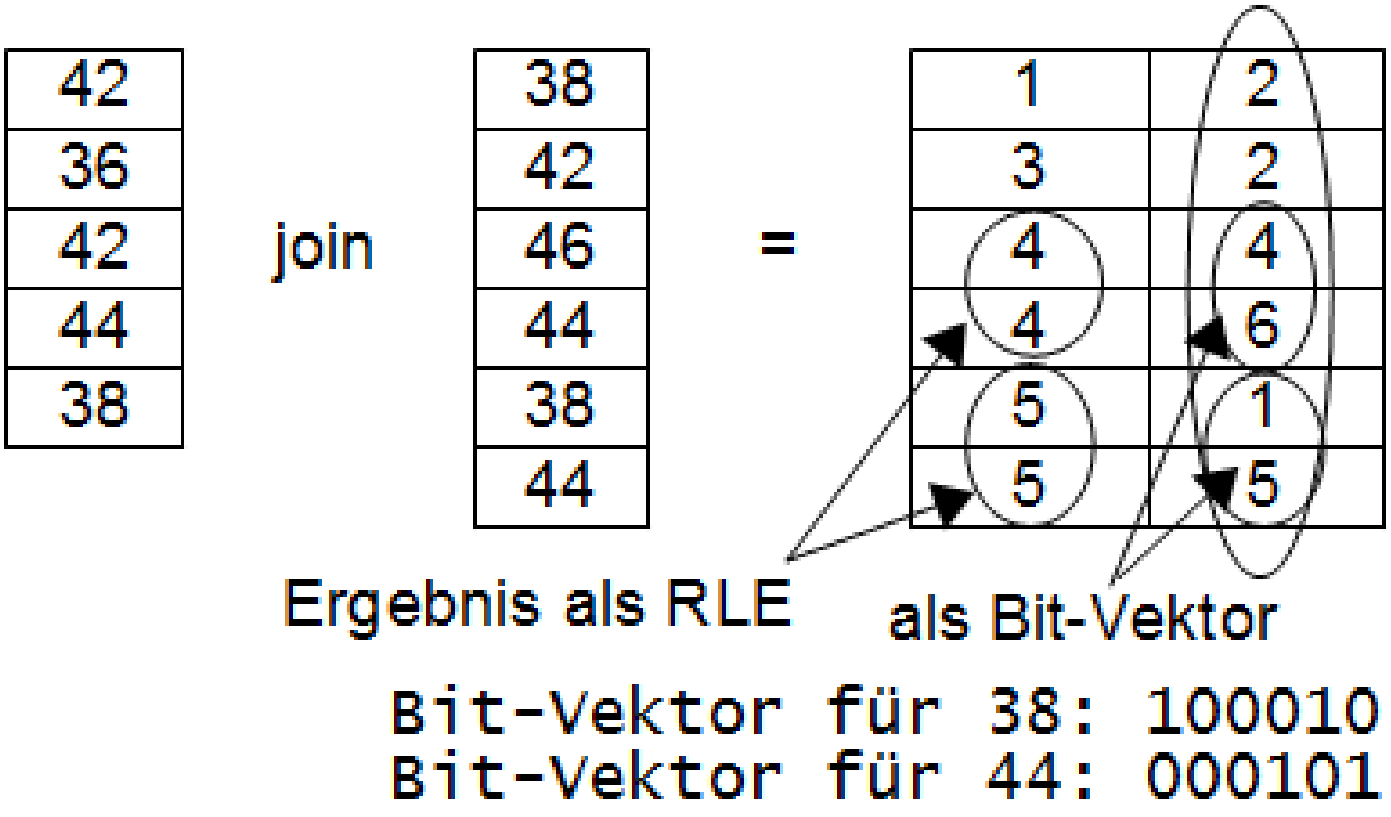

Figure: Join einer unkomprimierten mit Bit-Vektor komprimierten Spalte, (vgl. Pseudocode von Abadi (2005), S.54)

Ergebnis z.B. für 5&6 Zeile der Ergebnistabelle: [(38,5,2,); 100010]

Kompressions-Benchmarks

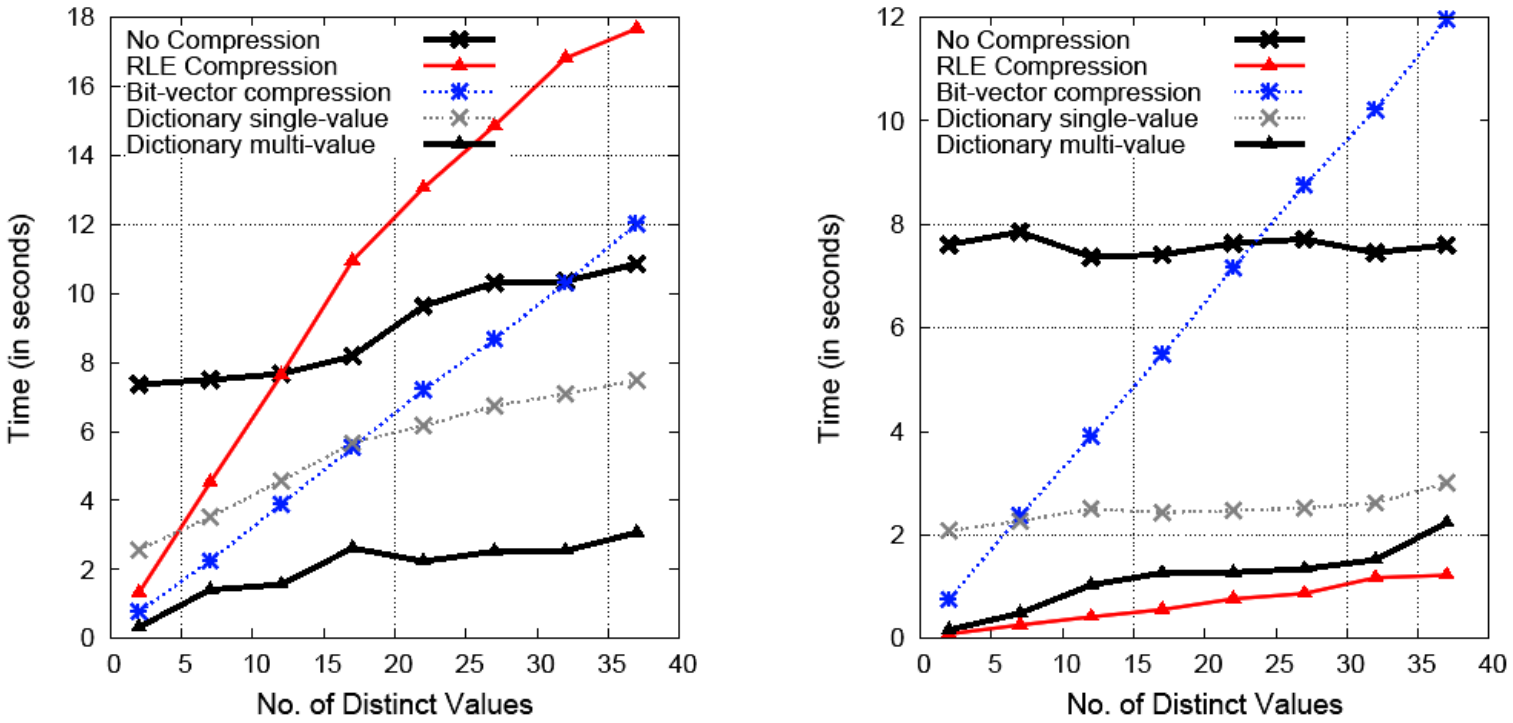

Figure: select sum(c) from table group by c, links: Runlength: 50, rechts: Runlength: 1000, (Quelle: Abadi (2006), Figure 4,5)

Materialisierungs-Strategien

- Materialisierung: wenn Werte aus Spalten gelesen und zu Tupel zusammengesetzt werden
- früh oder möglichst spät?
- Beispiel Abfrage:
  SELECT X FROM TABLE WHERE Y < CONST
- in Spalten-DB: zuerst mit Y < CONST Positionen bestimmen, dann Werte aus X lesen
- vgl. dazu (X,Y) Zusammensetzen und dann viele Tupel wieder verwerfen

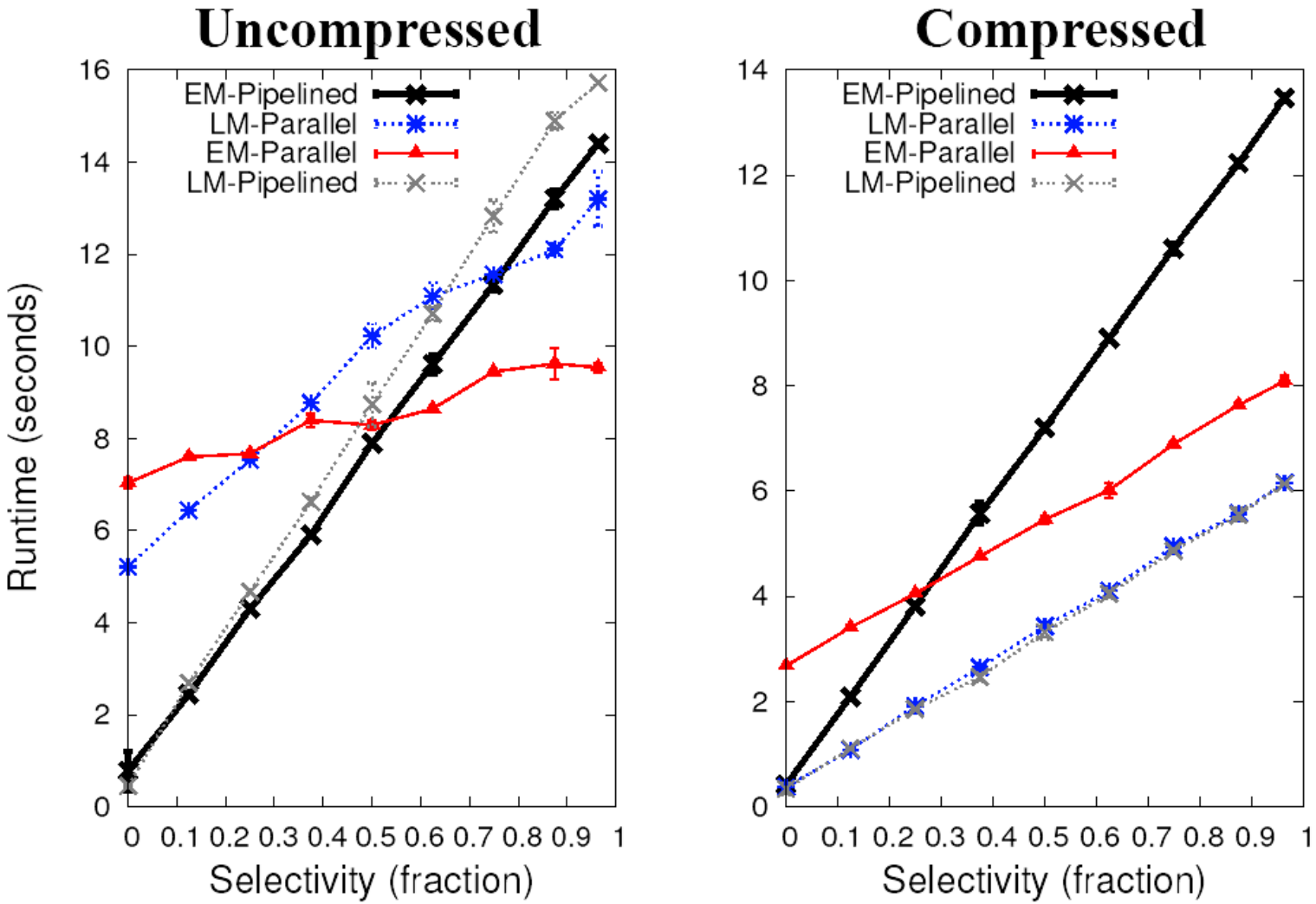

Figure: Materialisierungs-Benchmarks: Early und Late-Materialiserung jeweils pipelined und parallel (Quelle: Abadi et al. (2007), Figure 10)

Invisible Join

- Invisible Join als Prädikat um ungeordnetes Lesen aus Tabelle zu vermindern
- zuerst Prädikate auf Dimensionstabelle um Werte der Primärschlüssel zu speichern
- damit aus Faktentabelle Positionen, die alle Prädikate erfüllen
- zuletzt werden dann Werte aus den Dimensionstabellen gelesen

Benchmark: Star Schema Benchmark

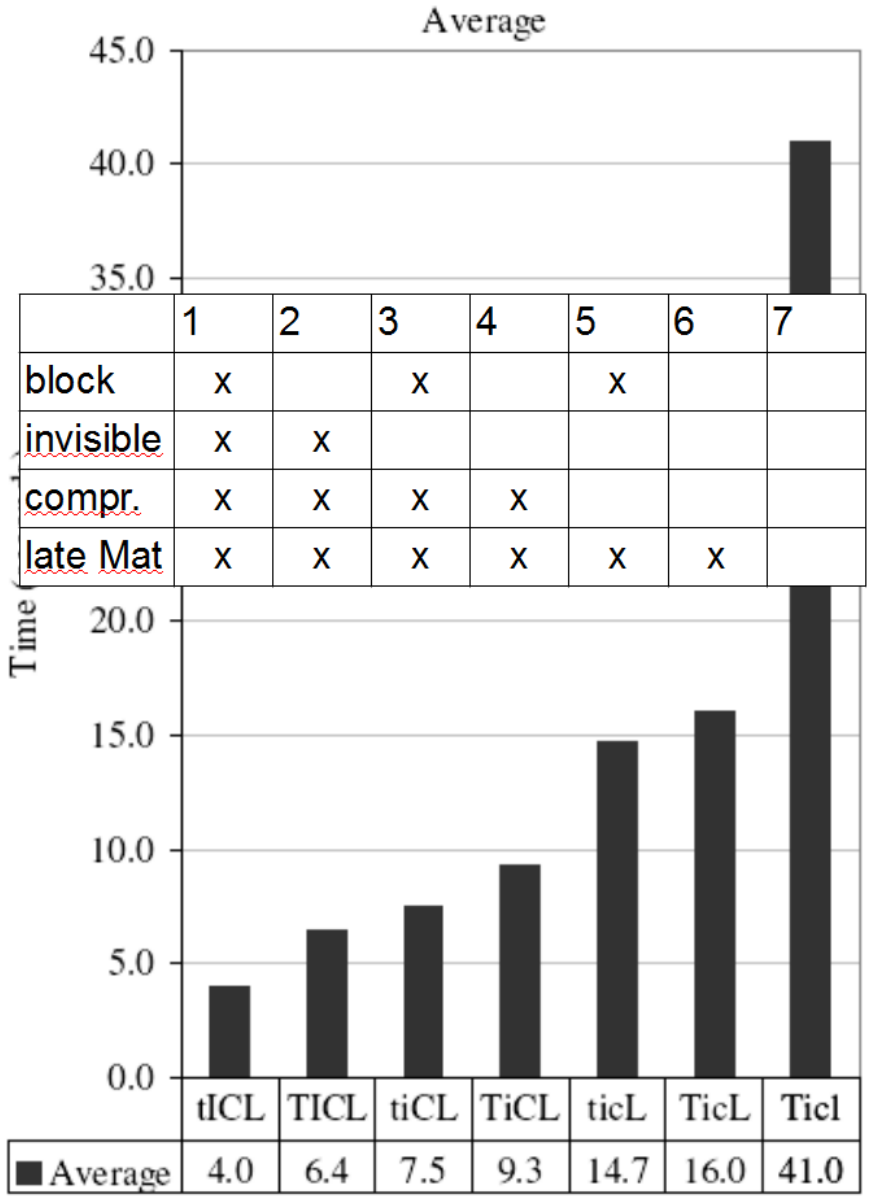

Figure: Star Schema Benchmark: Durchschnittswerte (Quelle: Abadi et al. (2008), Figure 7)

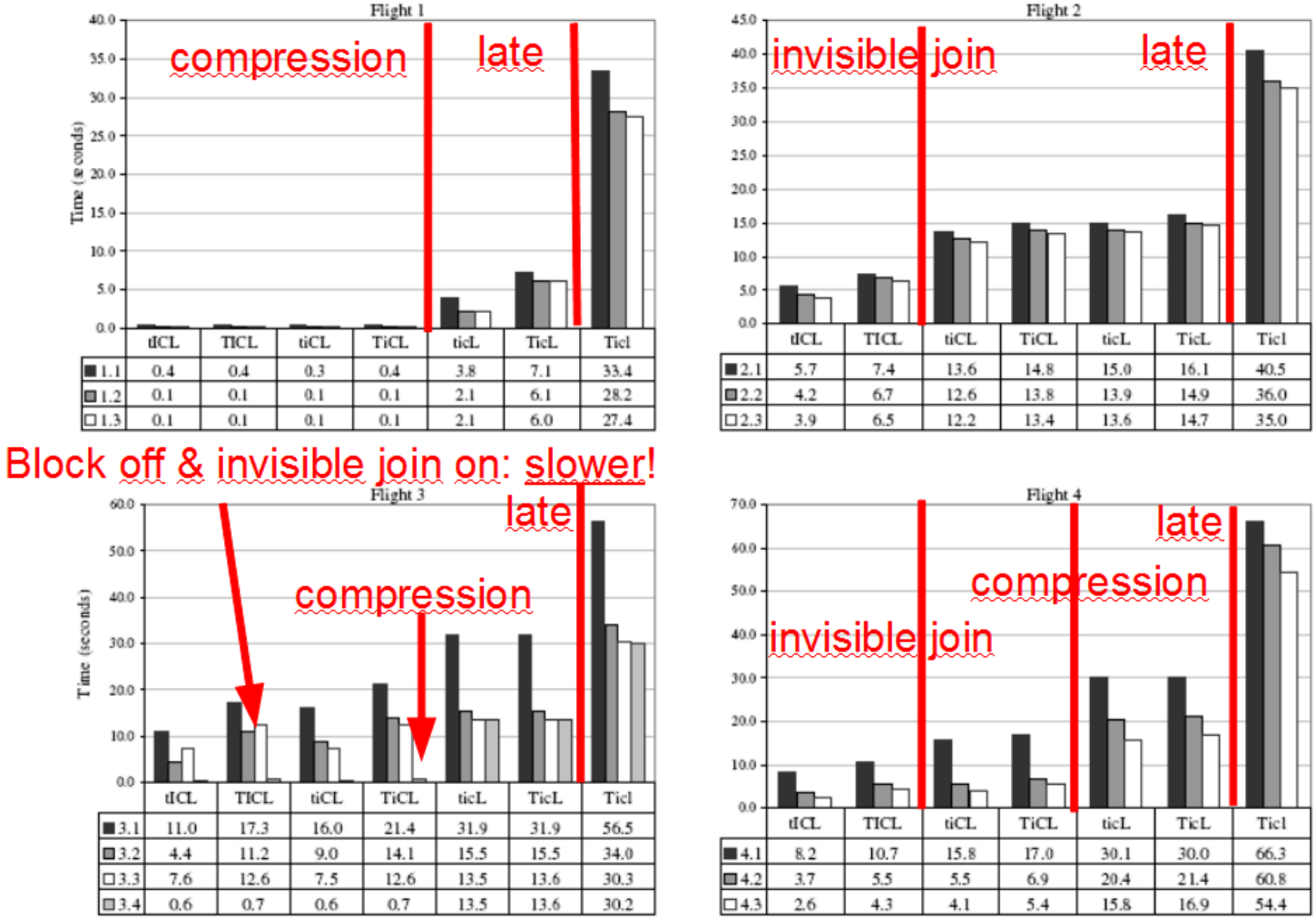

Figure: Star Schema Benchmark: Detailwerte (Quelle: Abadi et al. (2008), Figure 7)

Neuere Entwicklungen

- Kombination von OLAP (analytisch) und OLTP (transaktionsbasiert) bei "in-memory"-Datenbanken
- Cache-Optimierung: von Daten- und Instruktionscache u.a. bei MonetDB (Forschungsdatenbankprojekt aus den Niederlanden)
- Cooperative Scans
- grafikkartenbasierte Beschleunigung: "GPU-Co-Prozessor"
- Spaltenorientierte Datenbank als Basis von verteilten Datenbanksystemen

Motivation der GPU-Beschleunigung: Präfix-Summe für GPU optimiert

up-sweep Phase

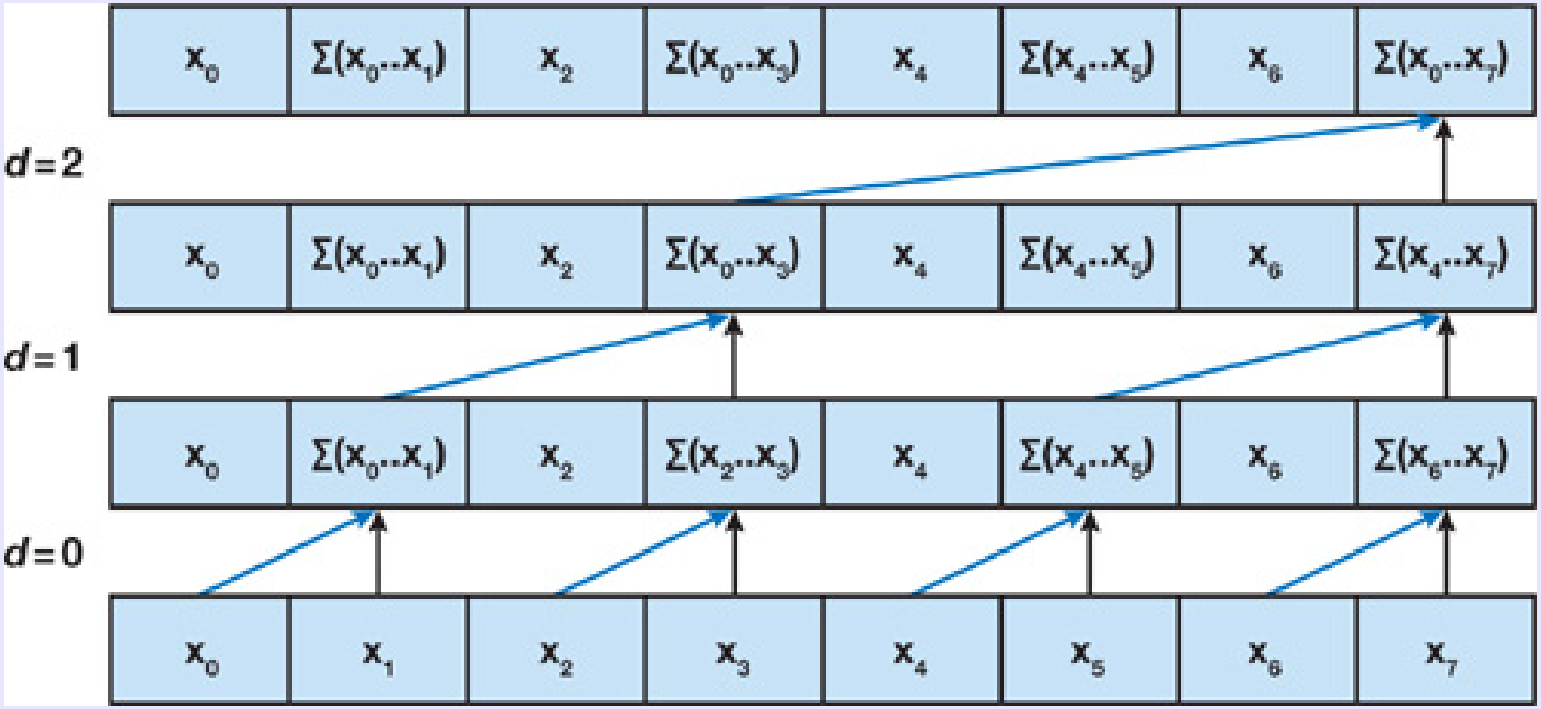

Figure: Präfix-Summe: up-sweep (Quelle: http http.developer.nvidia.comGPUGems3gpugems3_ch39.html, verfügbar am 30.11.2011, 20:05 Uhr)

down-sweep Phase

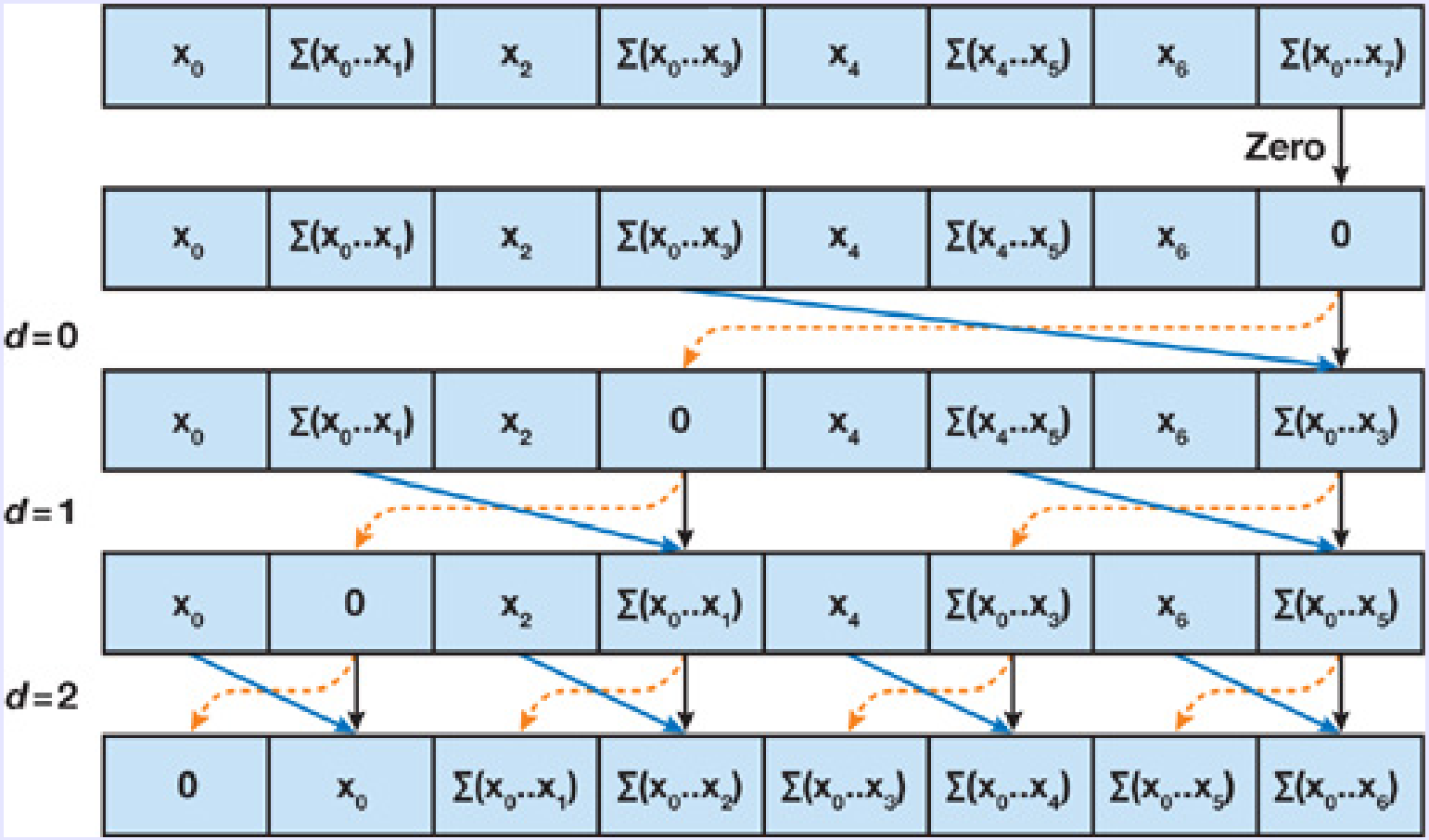

Figure: Präfix-Summe: down-sweep (Quelle: http http.developer.nvidia.comGPUGems3gpugems3_ch39.html, verfügbar am 30.11.2011, 20:05 Uhr)

GPU Kompression und Query Co-Processing

- auf GPU sind mehrere einfache Kompressionsalgorithmen hintereinander möglich bzw. sogar schwergewichtigere wie GZIP
- auch komplexere Operationen wie Joins können durch GPU beschleunigt werden
- am besten ist geschickte Aufteilung auf GPU und CPU, da PCI-E Bus den Flaschenhals darstellt

GPU Benchmarks

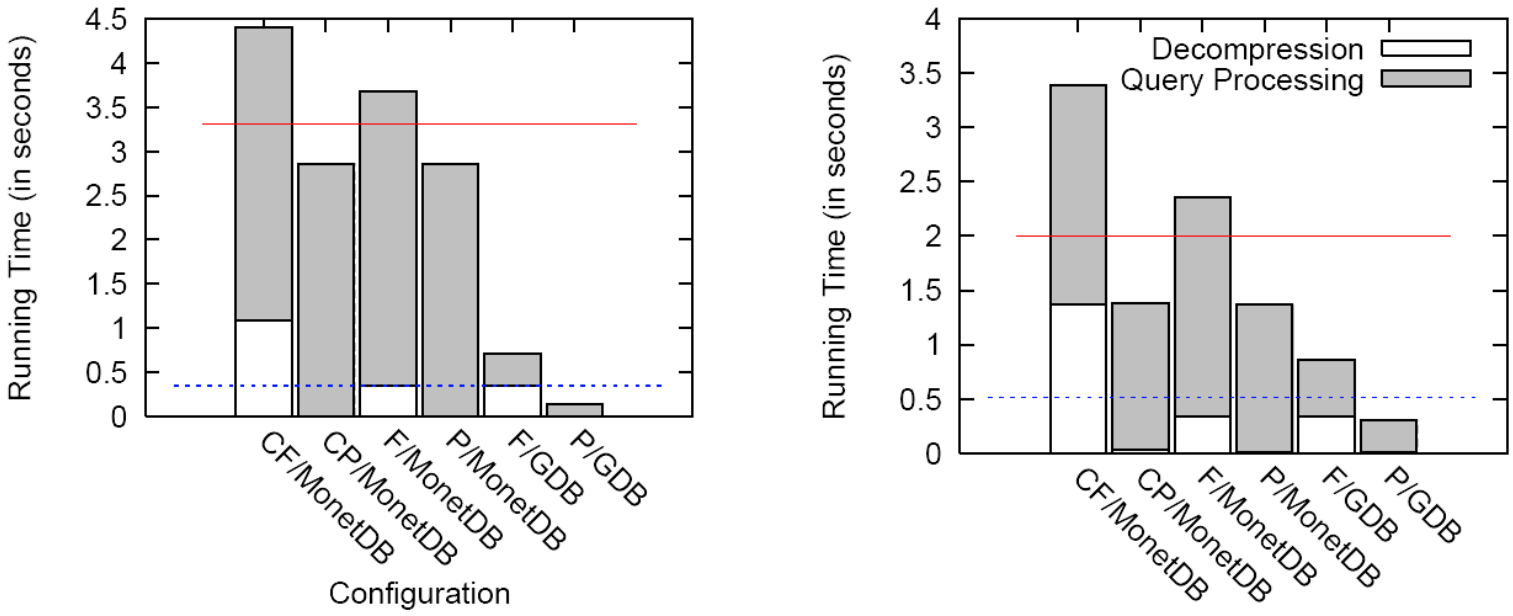

Figure: Partielle vs. Volle Dekomprimierung bei zwei Queries aus TPC-H Benchmark (Quelle: Fang et al. (2010), Figure 4 oben, Figure 6 unten)

Fazit

- Column-Stores sind gut für ideale Sternschemata
- viele Besonderheiten = viele Optimierungsmöglichkeiten
- beliebt in kommerziellen Anwendungen (Vectorwise, Vertica, ParStream, ...)

- Abadi et al. (2005), Query Execution in Column-Oriented Database Systems
- Abadi et al. (2006), Integrating compression and execution in column-oriented database systems
- Abadi et al. (2007), Materialization Strategies in a Column-Oriented DBMS
- Abadi et al. (2008), Column-stores vs. row-stores: how different are they really?
- Fang et al. (2010), Database compression on graphics processors
- He et al. (2008), Relational joins on graphics processors
- He et al. (2009), Relational query co-processing on graphics processors
- Plattner (2009), A Common Database Approach for OLTP and OLAP Using an In-Memory Column Database