Обработка и исполнение запросов в СУБД (Лекция 6)

Колоночные СУБД: введение и диск-ориентированные системы

v6

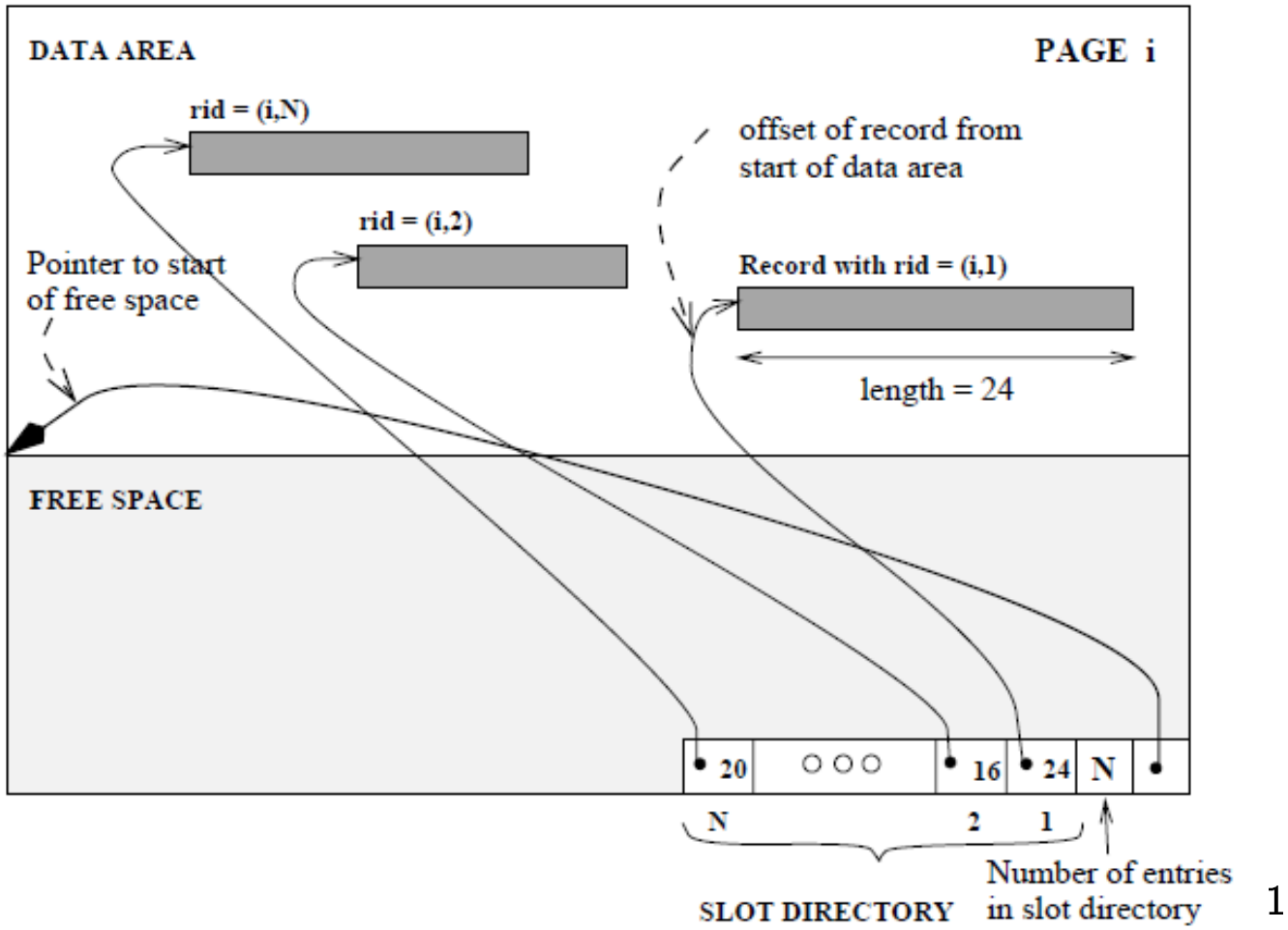
Георгий Чернышев

Высшая Школа Экономики

chernishev@gmail.com

7 октября 2020 г.

Колоночная СУБД: идея в первом приближении



[1]

- Надо считывать с диска меньше данных!

[1] Изображение взято из [Ramakrishnan and Gehrke, 2000]

Краткая история колоночного подхода

В основновном по [Harizopoulos et al., 2009]:

<1985 Типы:

- TOD, Cantor;
- индустриальные RAPID, TAXIR;
- иногда к ним относят и KDB хотя там не СУБД;
- тогда отнесем и элементы компьютеров пятого поколения;

1985 Две работы Copeland и соавторов: DSM vs. NSM;

1990 Коммерциализация подхода через Sybase IQ;

<2000 Hardware-ориентированные работы про поколоночную обработку;

~2005 Расцвет исследований, появление стартапов;

~2010 Принятие большими игроками, выпуск своих продуктов;

Предпосылки к популяризации

Один из важнейших вопросов области баз данных: “One size fits all?”.

- Новые запросы, деление на OLAP и OLTP, повышение роли OLAP систем, запрос на них со стороны индустрии [Чернышев, 2013];
- Изменения в оборудовании (разберем на следующей лекции).

Актуализация OLAP:

- рынок OLAP систем составлял в 2006 году около 6 миллиардов долларов, тогда как в 98 году он составлял всего 2 миллиарда;
- По информации TWDI (The Data Warehousing Institute) за прошедшее десятилетие* на порядок выросли и объемы обрабатываемых данных;
- Дополнительно можно отметить появление собственно термина OLAP (1993), создание OLAP Council (1995).

OLAP и OLTP нагрузки

OLAP	OLTP
Запросы заранее неизвестны	Шаблоны запросов известны заранее, например транзакция по оплате услуги или по добавлению пользователя
Запросы могут затрагивать десятки таблиц	Запросы затрагивают малое количество таблиц
Пакетное обновление или полное отсутствие обновления данных	Запросы обновляют данные, при этом делают это на лету
Сложные запросы, требующие оптимизации	Запросы простые — не содержат сложных операций, таких как соединение и агрегирование, а значит и не сильно зависят от оптимизации
Большой результат запроса — тысячи и миллионы записей	Результат запроса мал — десятки или сотни записей
Низкоселективные запросы	Высокоселективные запросы — результатом будет малая часть исходного отношения
Запрос затрагивает отдельные атрибуты из сотен или даже тысяч атрибутов таблицы	Запрос затрагивает все атрибуты таблицы
Малое количество клиентов — десятки или сотни, меньшая важность параллелизма	Запросы поступают параллельно, от большого числа клиентов

Предшественники колоночных систем

Работы Copeland et al:

- N-ary Storage Model (NSM) — стандартный подход;
- Decomposition Storage Model (DSM) — каждый атрибут в отдельном файле, нужно много соединений.

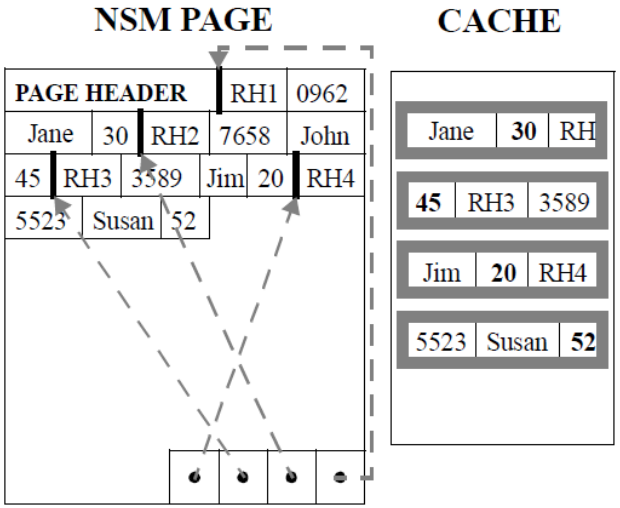


FIGURE 1: *The cache behavior of NSM.*

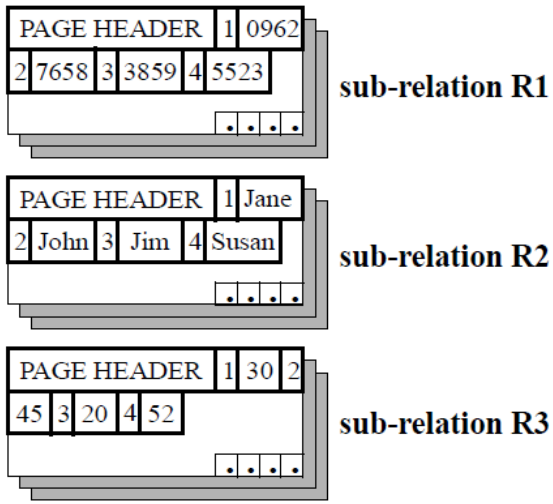
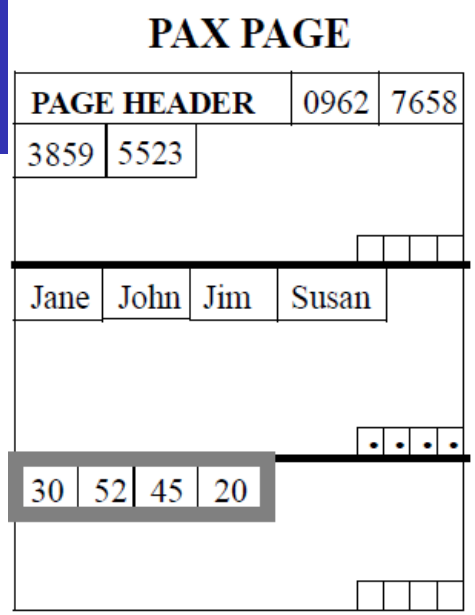


FIGURE 2: *The Decomposition Storage Model (DSM).*

Предшественники колоночных систем



Несколько работ из начала нулевых, buffer pool:

- PAX — статика, в памяти
- Data Morphing — динамика, в памяти;
- Clotho — диск, MEMS, динамика;
- Fractured Mirrors — диск, две копии NSM и DSM.

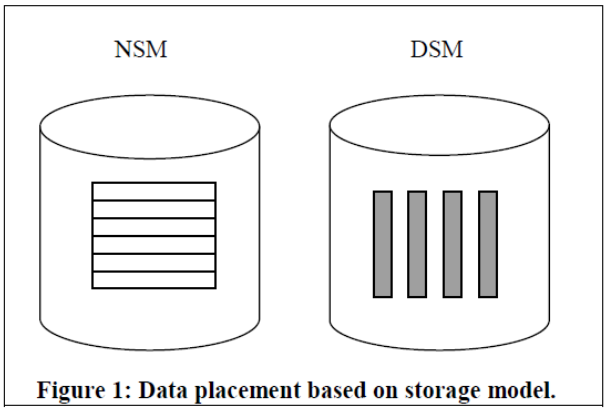
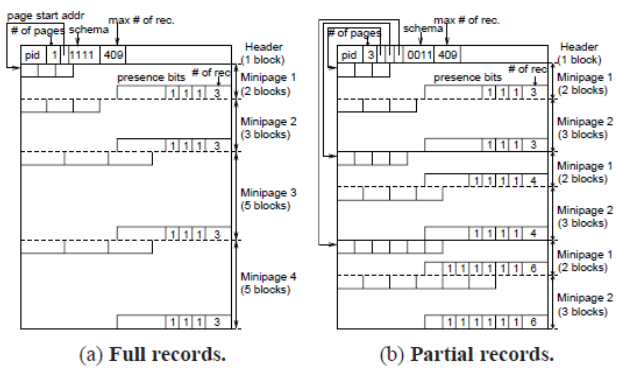


Figure 1: Data placement based on storage model.



(a) Full records. (b) Partial records.

Figure 3: C-page layout.

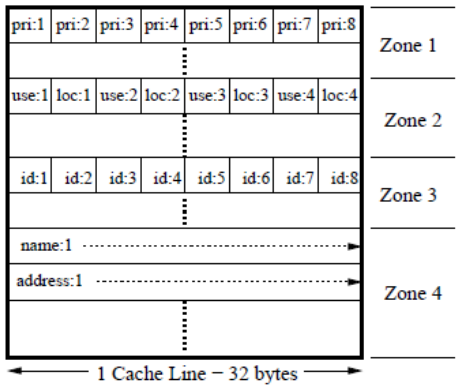
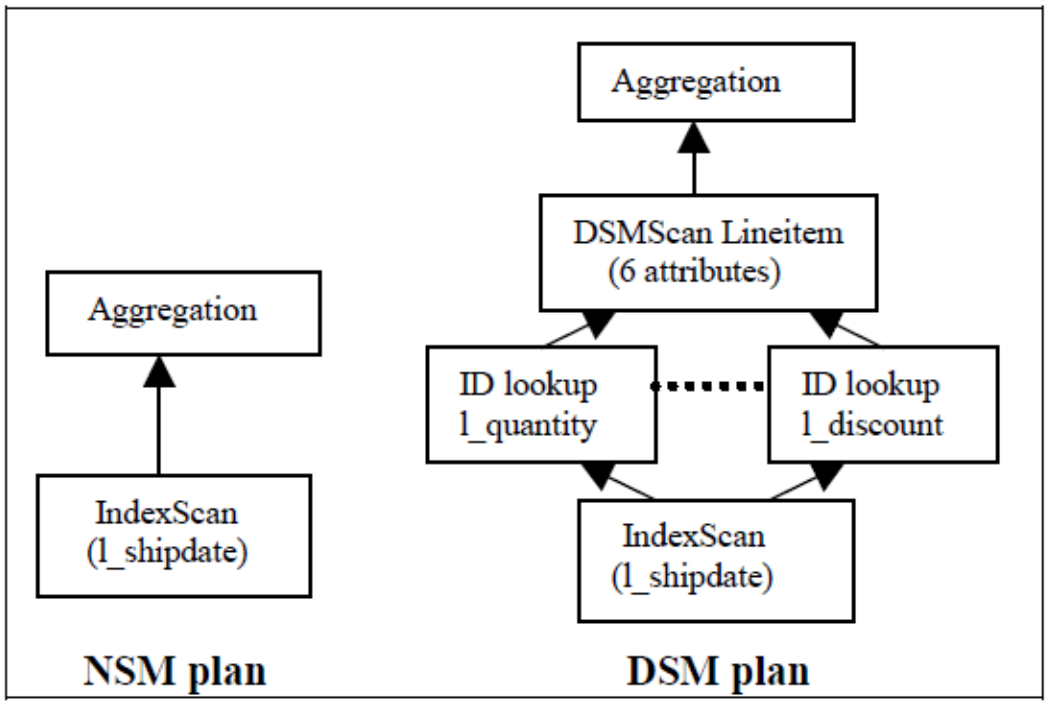


Figure 4: Attribute Grouping

Обработка запроса во fractured mirrors (идея)

Там была не только переработка buffer pool, а настоящая поколоночная обработка!

Q1: select ... from lineitem where l_shipdate => date '1998-12-01' - interval '[DELTA]' day groupby ... orderby ...



Два подхода к колоночным СУБД[2]

- C-Store — дисковая, опирается на порядки сортировки, сжатие, раннюю и позднюю материализацию, новые операторы соединения и оперирование над сжатыми данными;
- MonetDB — в оперативной памяти, упор на эффективное использование железа: минимизация промахов в кешах кешей, использование хардварного параллелизма (SIMD). Специальная колоночная алгебра (BAT-алгебра), адаптивное индексирование, операторы для максимально эффективного использования железа. Тема 8 лекции.

[2] Есть немного на русском здесь: http://citforum.ru/database/articles/column_vs_row_store/

Общая архитектура дисковой колоночной системы

- Каждая колонка хранится в отдельном файле;
  - сжата, возможно своим алгоритмом;
  - упорядочена согласно какомой-либо колонке (набору);
- Наборов колонок может быть много (репликация);
- Операторы работают над одной колонкой или “сборкой” из нескольких;
- Между операторами ходят не только данные, но и ID записей;
- Когда “менять” ID на данные? Вопрос материализации.

Основная публикация: D. Abadi et al. Materialization Strategies in a Column-Oriented DBMS, 2007 IEEE 23rd International Conference on Data Engineering, Istanbul, 2007, pp. 466-475.

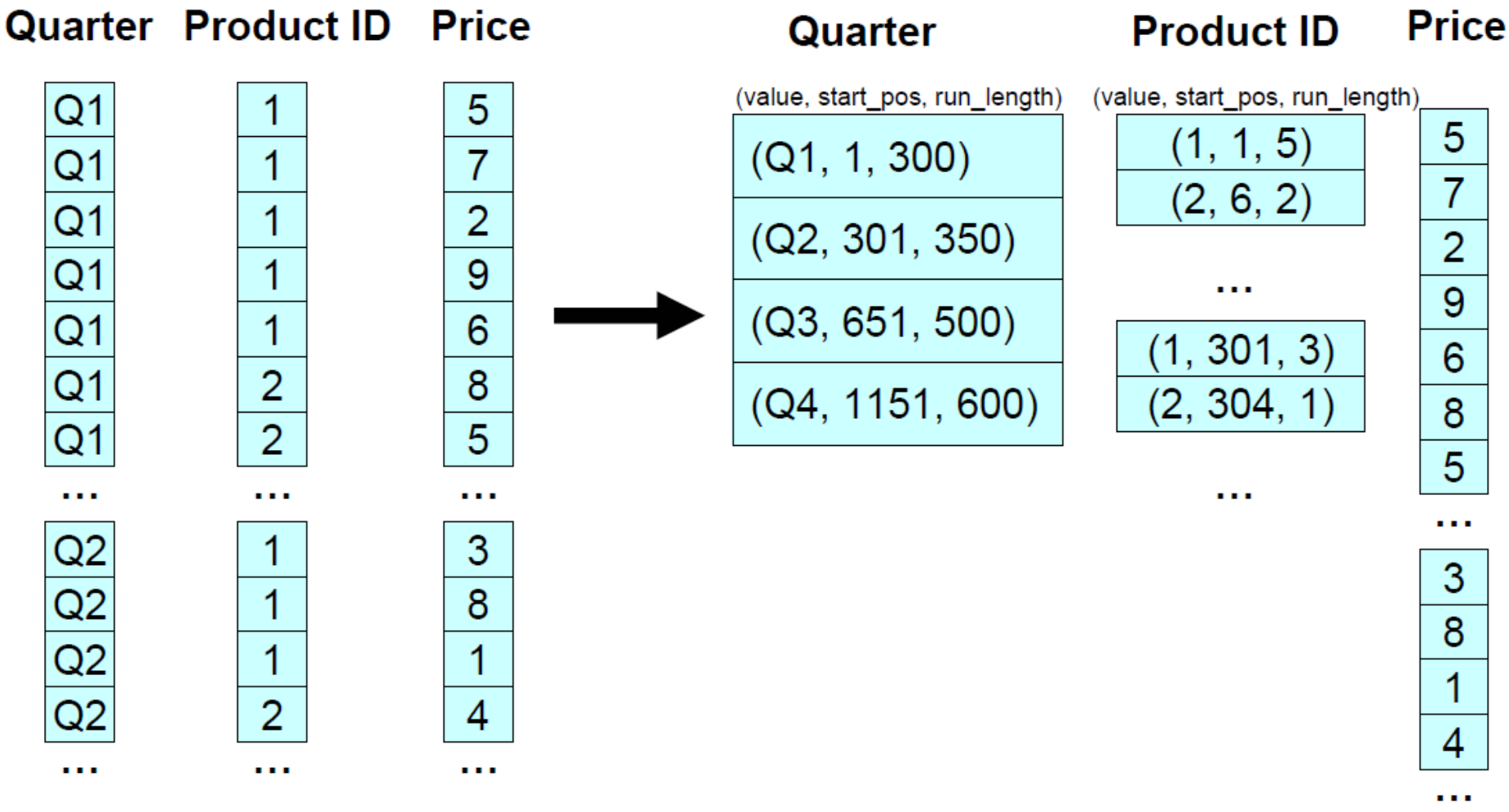
Сжатие

Один из столпов колоночной технологии. “Обмениваем” CPU на диск!

- До колоночной эры сжатие было, но другое:
  - сжимали гетерогенные данные $\longrightarrow$ в 2-3 раза;
  - более тяжеловесные методы сжатия;
  - не было операторов работающих со сжатыми данными.

- Легковесное, примененное к одной колонке:
  - Экономит место на диске до 10 раз;
  - Типы:
    - RLE
    - Bit-vector
    - Frame of Reference (FoR)
    - Differential
    - ...
  - Сжатие/разжатие можно o-SIMD-ить:
    - J. Wang et al., An Experimental Study of Bitmap Compression vs. Inverted List Compression, SIGMOD 2017
    - P. Damme et al., Insights into the Comparative Evaluation of Lightweight Data Compression Algorithms, EDBT 2017

Сжатие: RLE

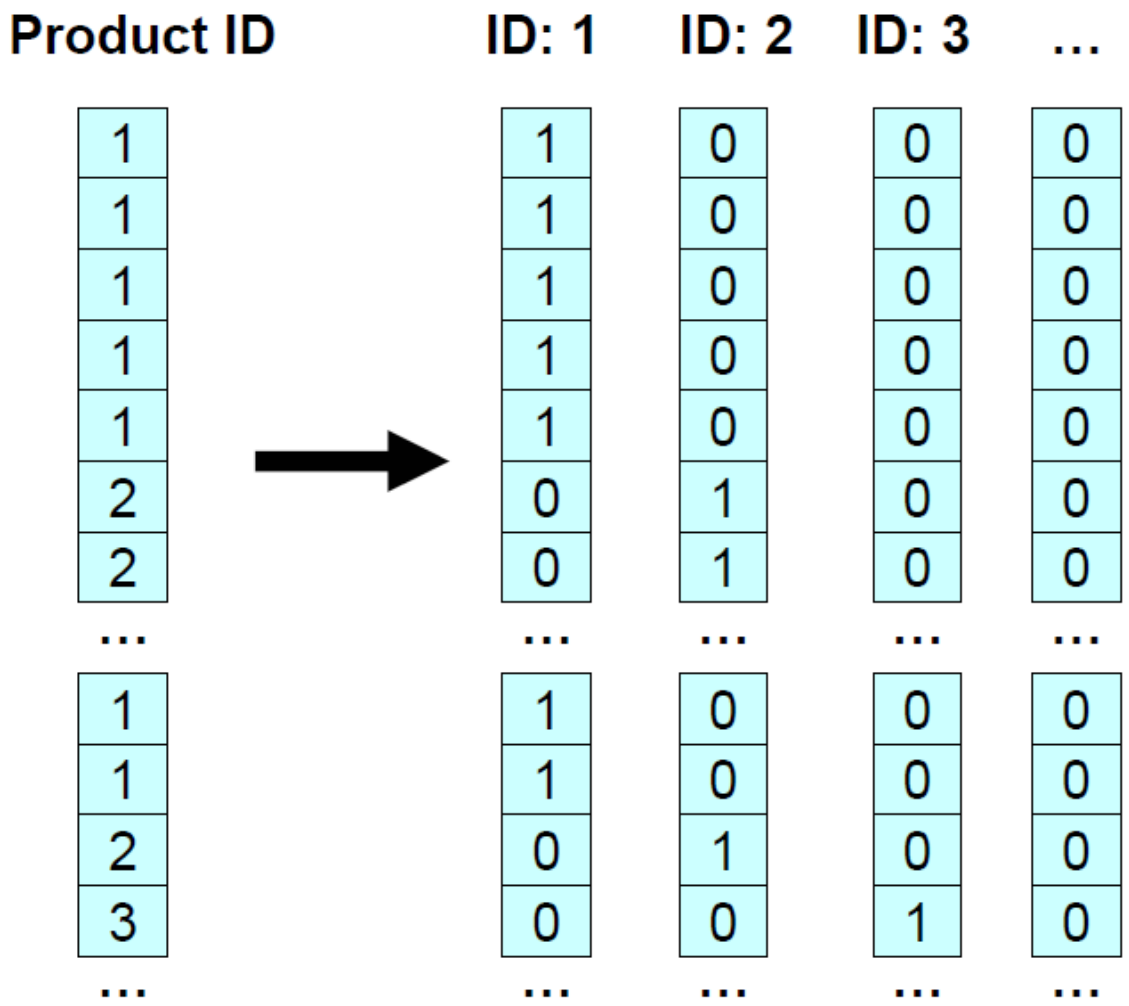


3

[3] Изображение взято из [Harizopoulos et al., 2009]

Сжатие: bit-vector

- For each unique value, v, in column c, create bit-vector b
  - b[i] = 1 if c[i] = v
- Good for columns with few unique values
- Each bit-vector can be further compressed if sparse

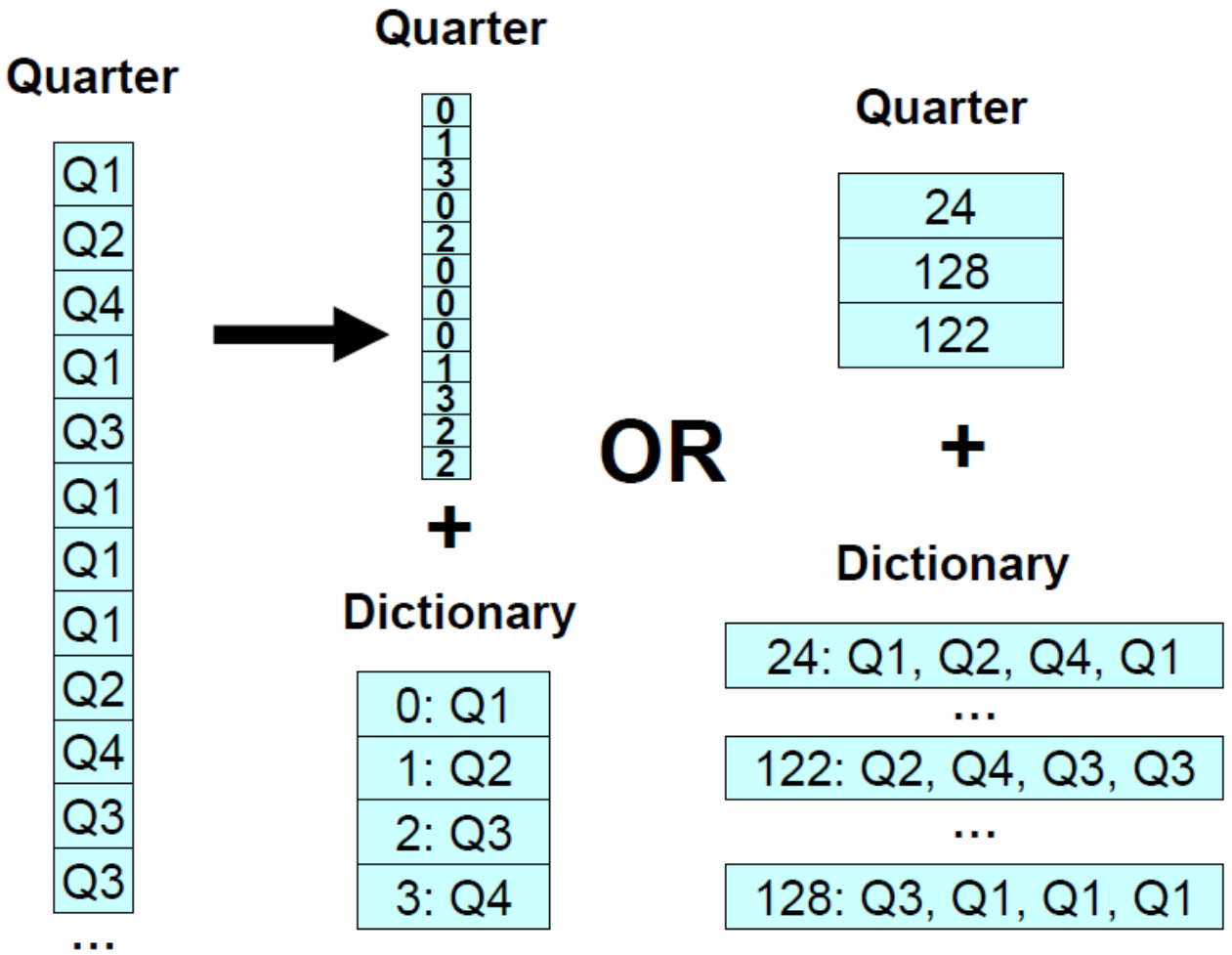


4

[4] Изображение взято из [Harizopoulos et al., 2009]

Сжатие: dictionary

1. For each unique value create dictionary entry
1. Dictionary can be per-block or per-column
1. Column-stores have the advantage that dictionary entries may encode multiple values at once



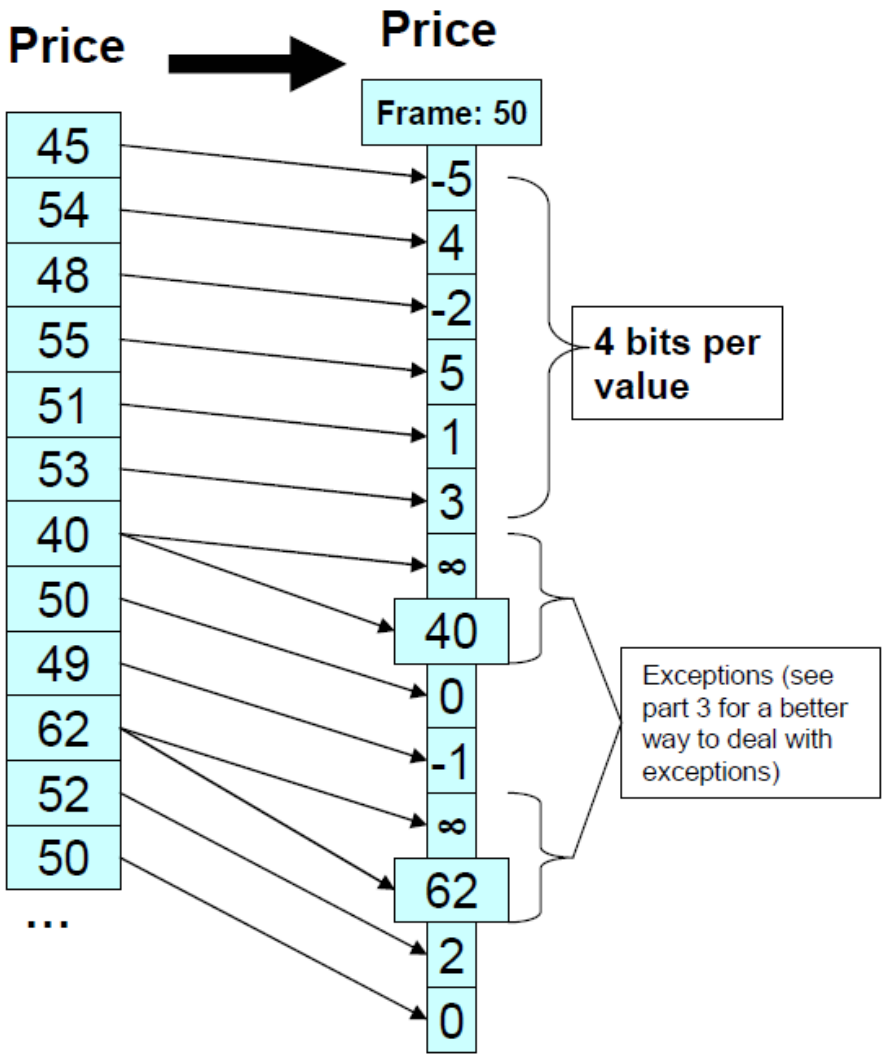
5

[5] Изображение взято из [Harizopoulos et al., 2009]

Сжатие: FoR

- Encodes values as b bit offset from chosen frame of reference
- Special escape code (e.g. all bits set to 1) indicates a difference larger than can be stored in b bits
  - After escape code, original (uncompressed) value is written

"Compressing Relations and Indexes" Goldstein, Ramakrishnan, Shaft, ICDE'98

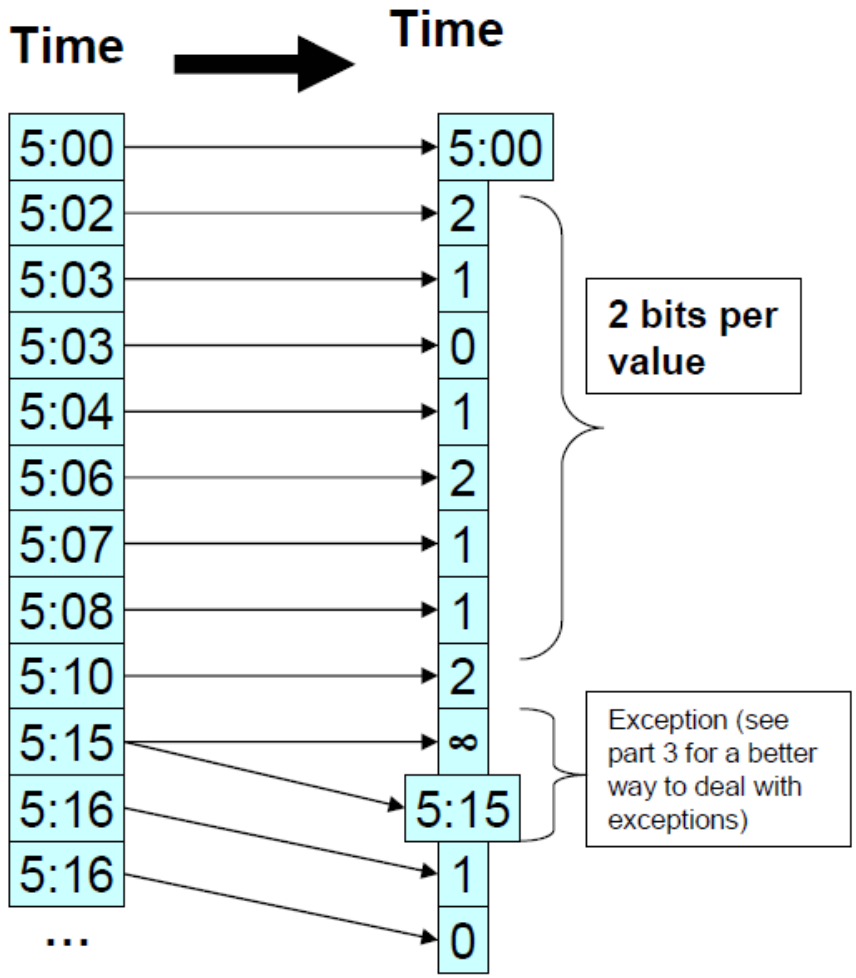


6

Сжатие: differential

- Encodes values as b bit offset from previous value
- Special escape code (just like frame of reference encoding) indicates a difference larger than can be stored in b bits
  - After escape code, original (uncompressed) value is written
- Performs well on columns containing increasing/decreasing sequences
  - inverted lists
  - timestamps
  - object IDs
  - sorted / clustered columns

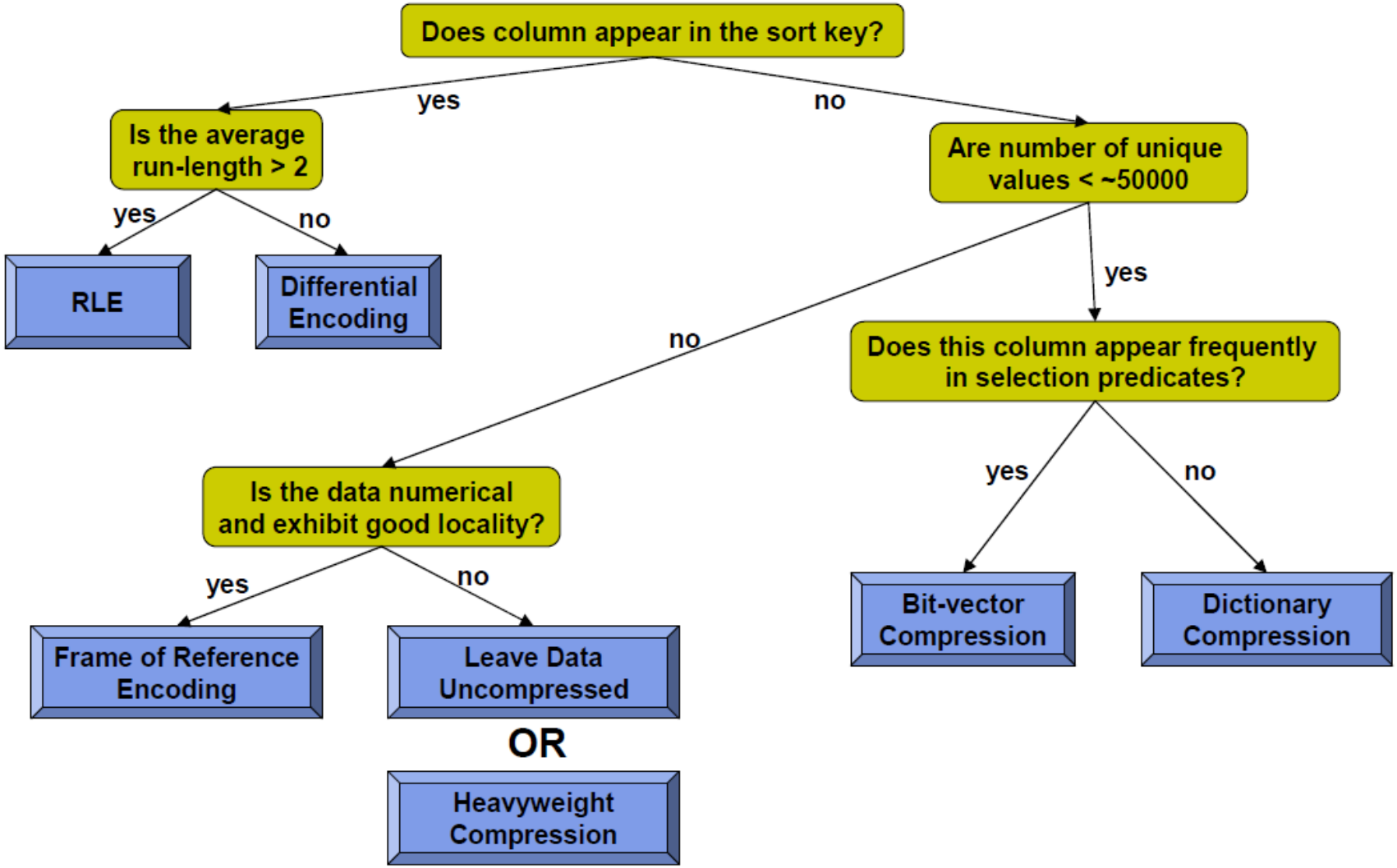
"Improved Word-Aligned Binary Compression for Text Indexing" Ahn, Moffat, TKDE'06



7

[7] Изображение взято из [Harizopoulos et al., 2009]

Сжатие: как выбирать алгоритм?



8

[8] Изображение взято из [Harizopoulos et al., 2009]

Почему легковесные схемы?

Heavy-Weight Compression Schemes

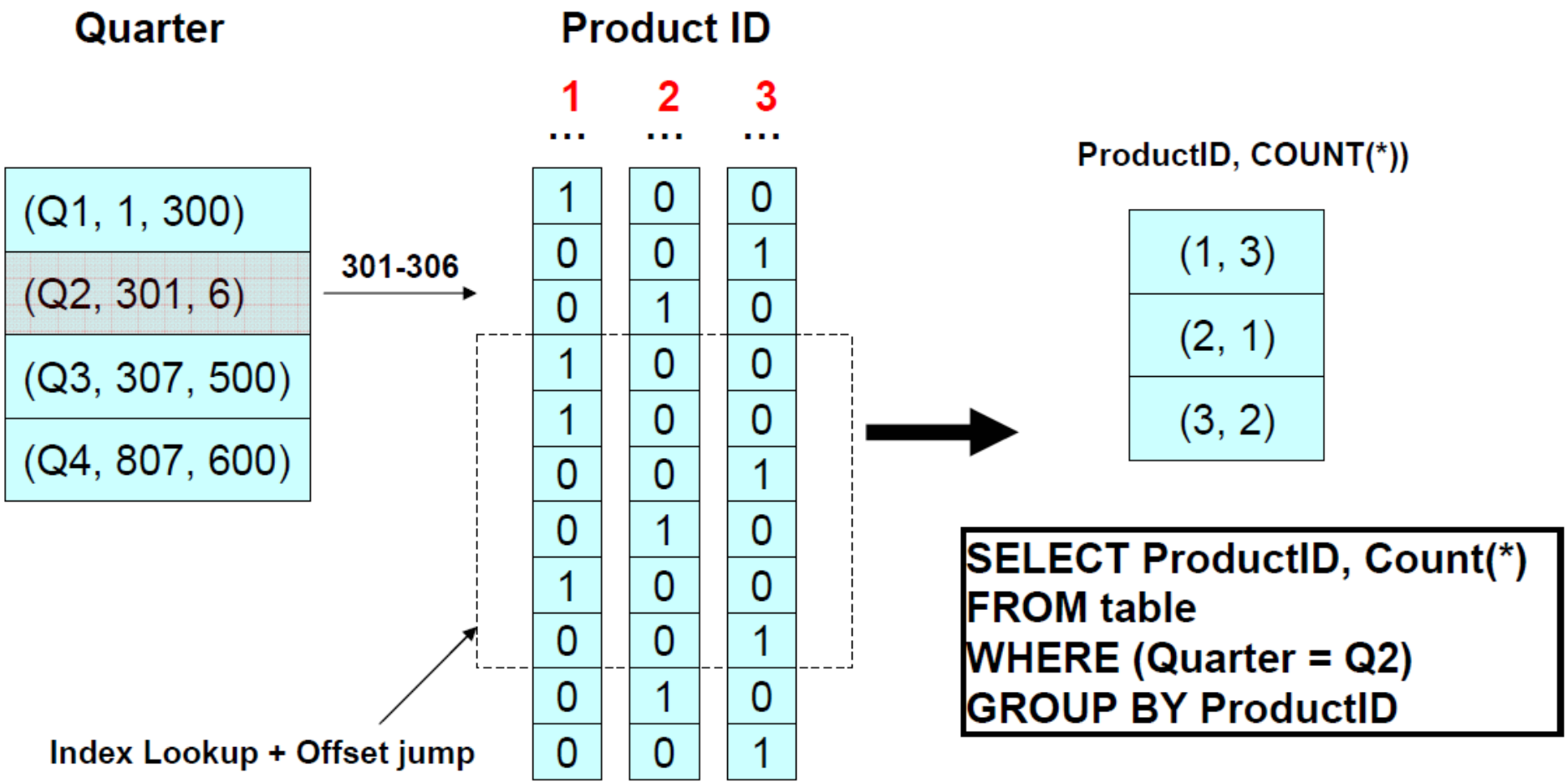
Algorithm	Decompression Bandwidth
BZIP	10 MB/s
ZLIB	80 MB/s
LZO	300 MB/s

- Modern disk arrays can achieve > 1GB/s
- 1/3 CPU for decompression Ł 3GB/s needed

- **Lightweight compression schemes are better**
- **Even better: operate directly on compressed data** [9]

[9] Изображение взято из [Harizopoulos et al., 2009]

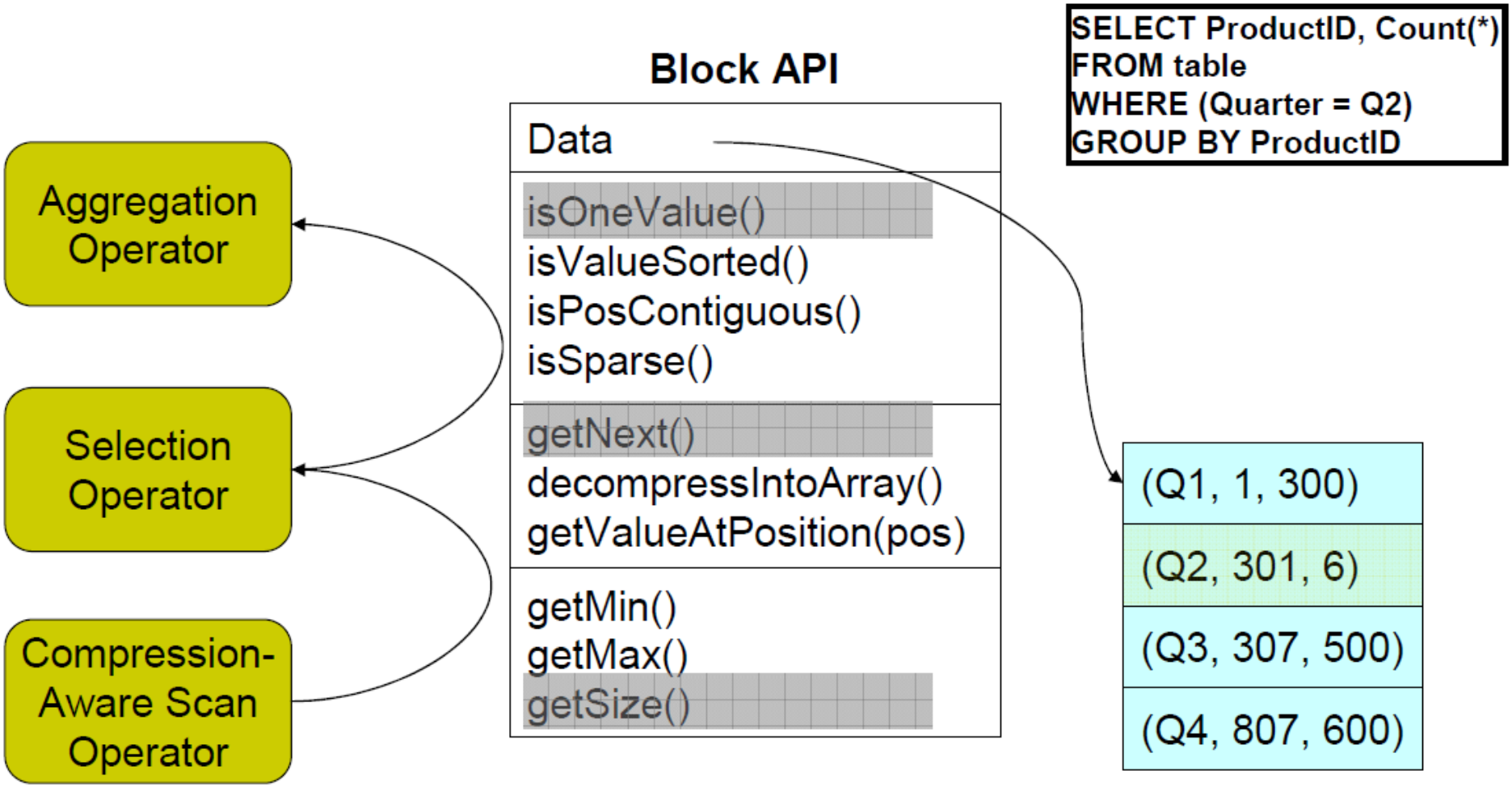
Операции над сжатыми данными I



10

[10] Изображение взято из [Harizopoulos et al., 2009]

Операции над сжатыми данными II



11

[11] Изображение взято из [Harizopoulos et al., 2009]

Исполнение запросов: материализация vs проекция

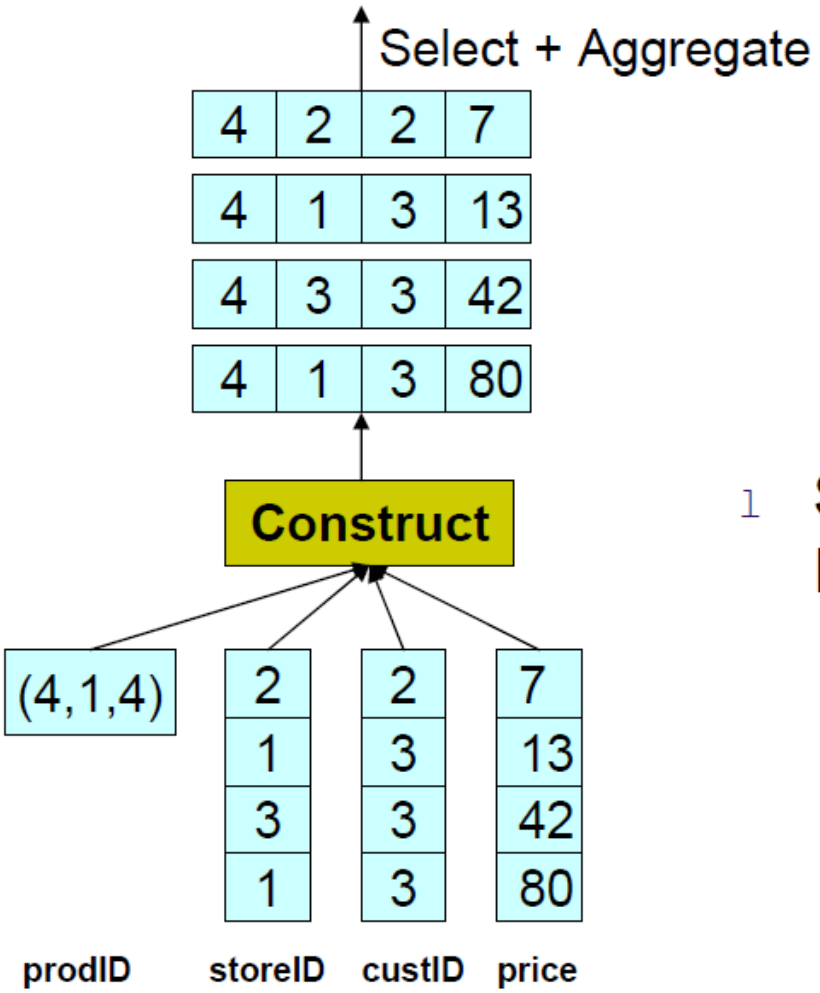
Кажется что материализация это примерно проекция, однако...

Проекция:

- В классических системах: выгодно как можно раньше;
- В колоночных системах: зависит!
  - Материализация: считать нужные колонки и достать значения из нужных для конструирования ответа;
  - Ранняя: в самом начале получить все нужные значения и выдавать наверх строки;
  - Поздняя: максимально отложить этот процесс.

Ранняя: проста в понимании и реализации, дешева с вычислительной точки зрения и точно всегда дает выигрыш за счет эффективной работы с диском (сжатие, SIMD, игнорирование ненужных колонок).
Поздняя: гораздо сложнее делать, но может быть дополнительный выигрыш.

Исполнение запросов: EM пример



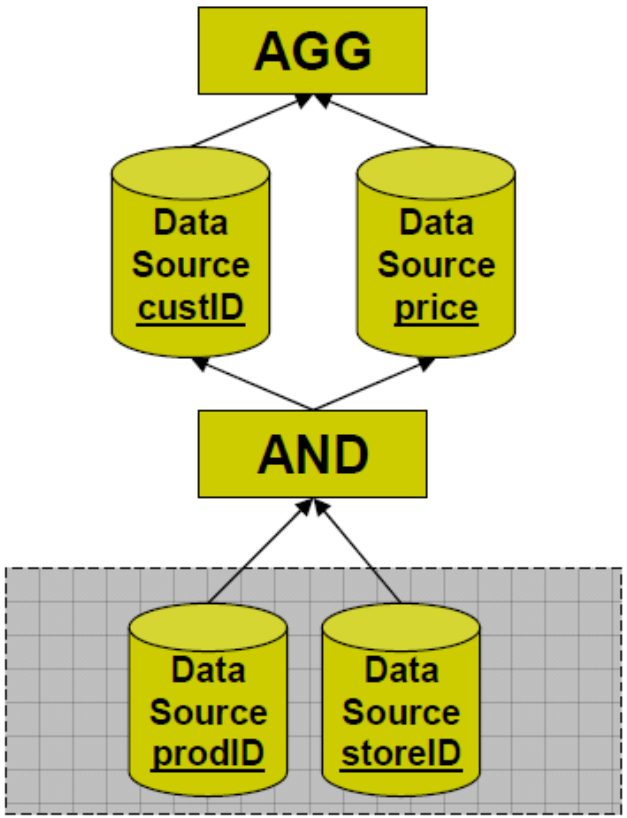

```
QUERY:

SELECT custID,SUM(price)
FROM table
WHERE (prodID = 4) AND
      (storeID = 1) AND
GROUP BY custID
```

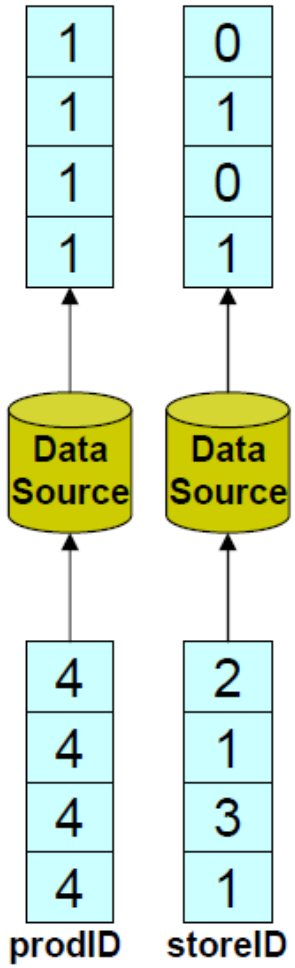
1. **Solution 1: Create rows first (EM). But:**
   1. **Need to construct ALL tuples**
   1. **Need to decompress data**
   1. **Poor memory bandwidth utilization**

12

[12] Изображение взято из [Harizopoulos et al., 2009]

Исполнение запросов: LM пример I





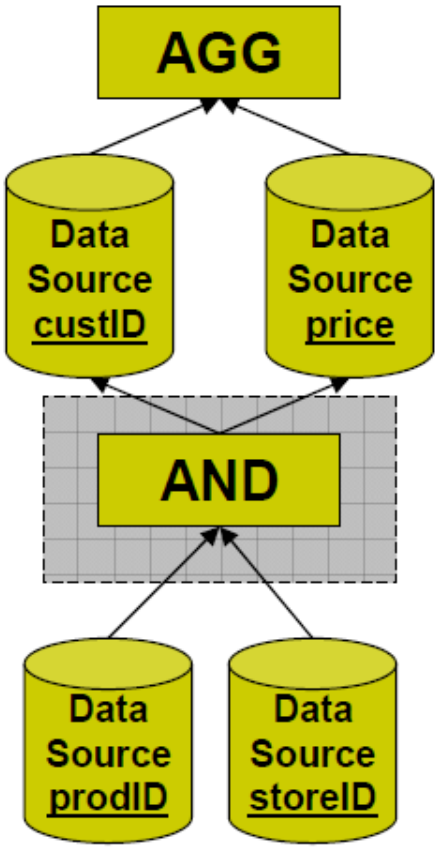

```
QUERY:

SELECT custID,SUM(price)
FROM table
WHERE (prodID = 4) AND
          (storeID = 1) AND
GROUP BY custID
```

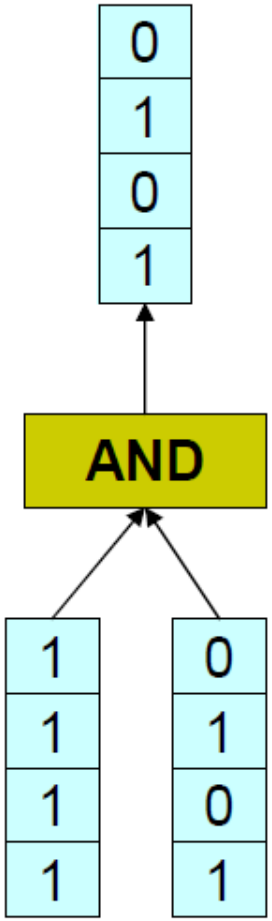
prodID	storeID	custID	price
4	2	2	7
4	1	3	13
4	3	3	42
4	1	3	80

13

[13] Изображение взято из [Harizopoulos et al., 2009]

Исполнение запросов: LM пример II





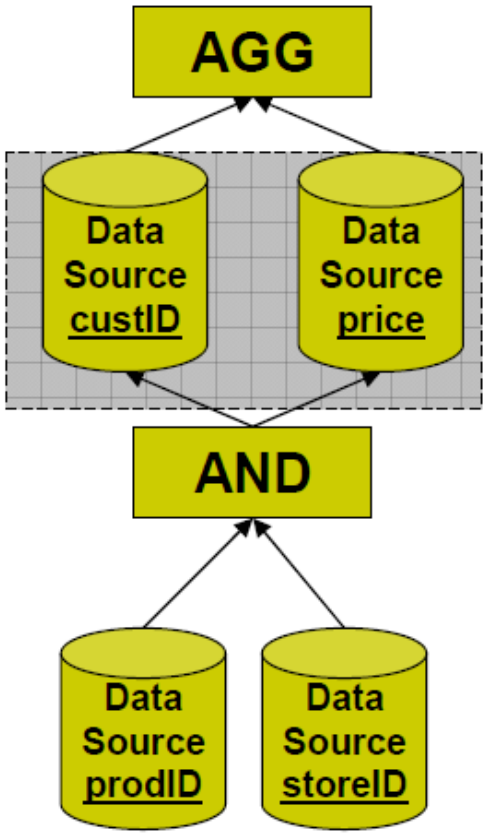

```
QUERY:

SELECT custID,SUM(price)
FROM table
WHERE (prodID = 4) AND
      (storeID = 1) AND
GROUP BY custID
```

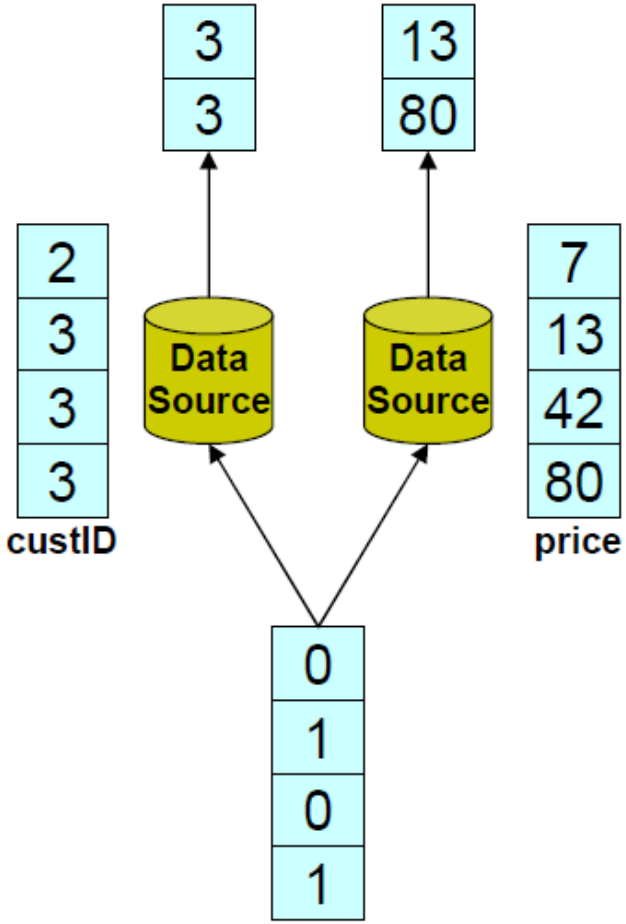
prodID	storeID	custID	price
4	2	2	7
4	1	3	13
4	3	3	42
4	1	3	80

14

[14] Изображение взято из [Harizopoulos et al., 2009]

Исполнение запросов: LM пример III





QUERY:

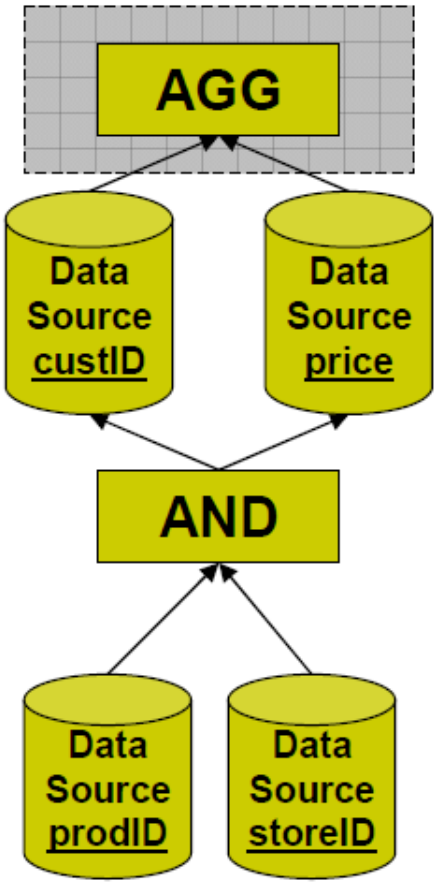
```
SELECT custID,SUM(price)
FROM table
WHERE (prodID = 4) AND
      (storeID = 1) AND
GROUP BY custID
```

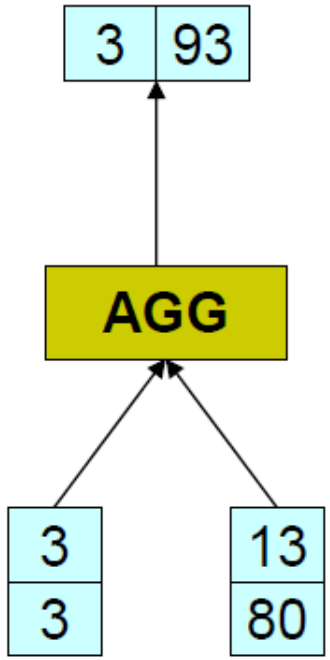
prodID	storeID	custID	price
4	2	2	7
4	1	3	13
4	3	3	42
4	1	3	80

15

[15] Изображение взято из [Harizopoulos et al., 2009]

Исполнение запросов: LM пример IV





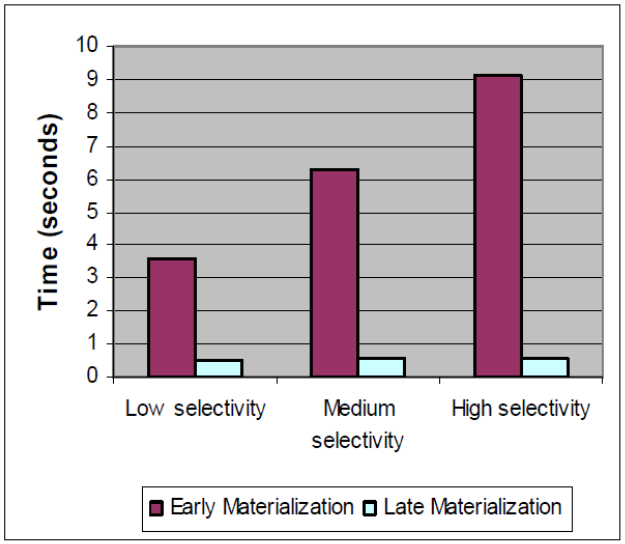

```
QUERY:

SELECT custID,SUM(price)
FROM table
WHERE (prodID = 4) AND
      (storeID = 1) AND
GROUP BY custID
```

prodID	storeID	custID	price
4	2	2	7
4	1	3	13
4	3	3	42
4	1	3	80

16

[16] Изображение взято из [Harizopoulos et al., 2009]

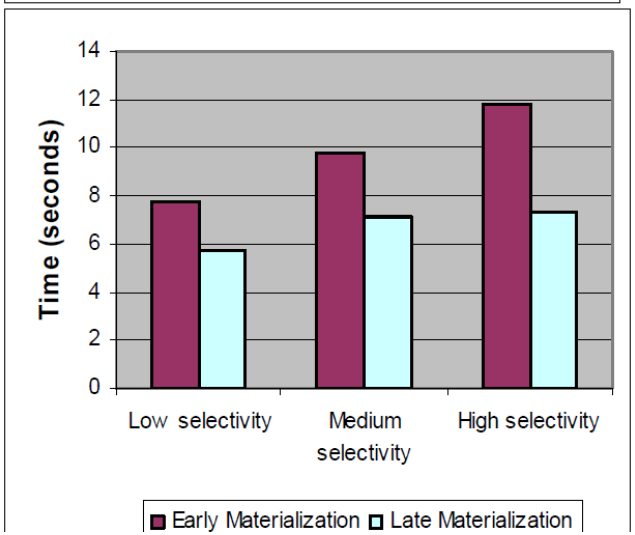
Что лучше, если нет соединений?



QUERY:

```
SELECT C1, SUM(C2)
FROM table
WHERE (C1 < CONST) AND
      (C2 < CONST)
GROUP BY C1
```

- **Ran on 2 compressed columns from TPC-H scale 10 data**



QUERY:

```
SELECT C1, SUM(C2)
FROM table
WHERE (C1 < CONST) AND
      (C2 < CONST)
GROUP BY C1
```

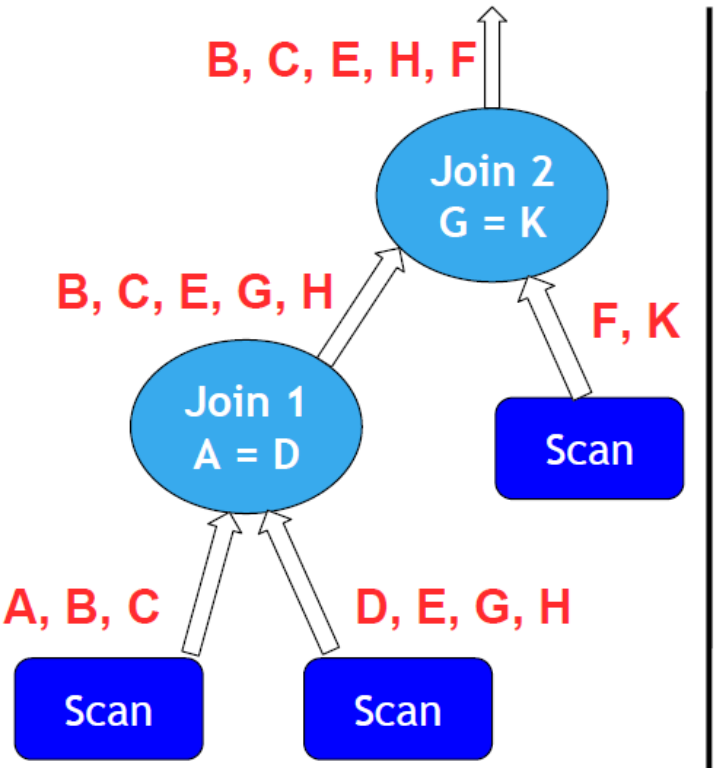
- **Materializing late still works best**

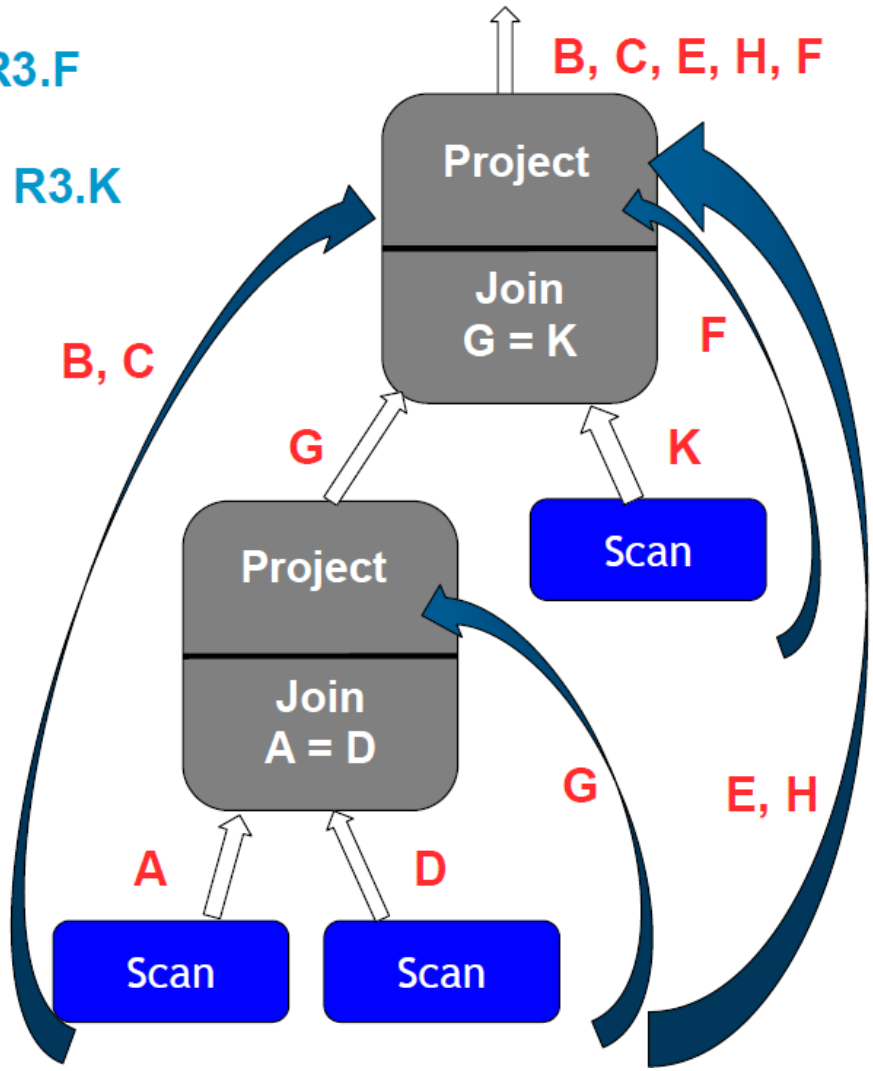
17

[17] Изображение взято из [Harizopoulos et al., 2009]

Соединения

```
Select R1.B, R1.C, R2.E, R2.H, R3.F
From R1, R2, R3
Where R1.A = R2.D AND R2.G = R3.K
```

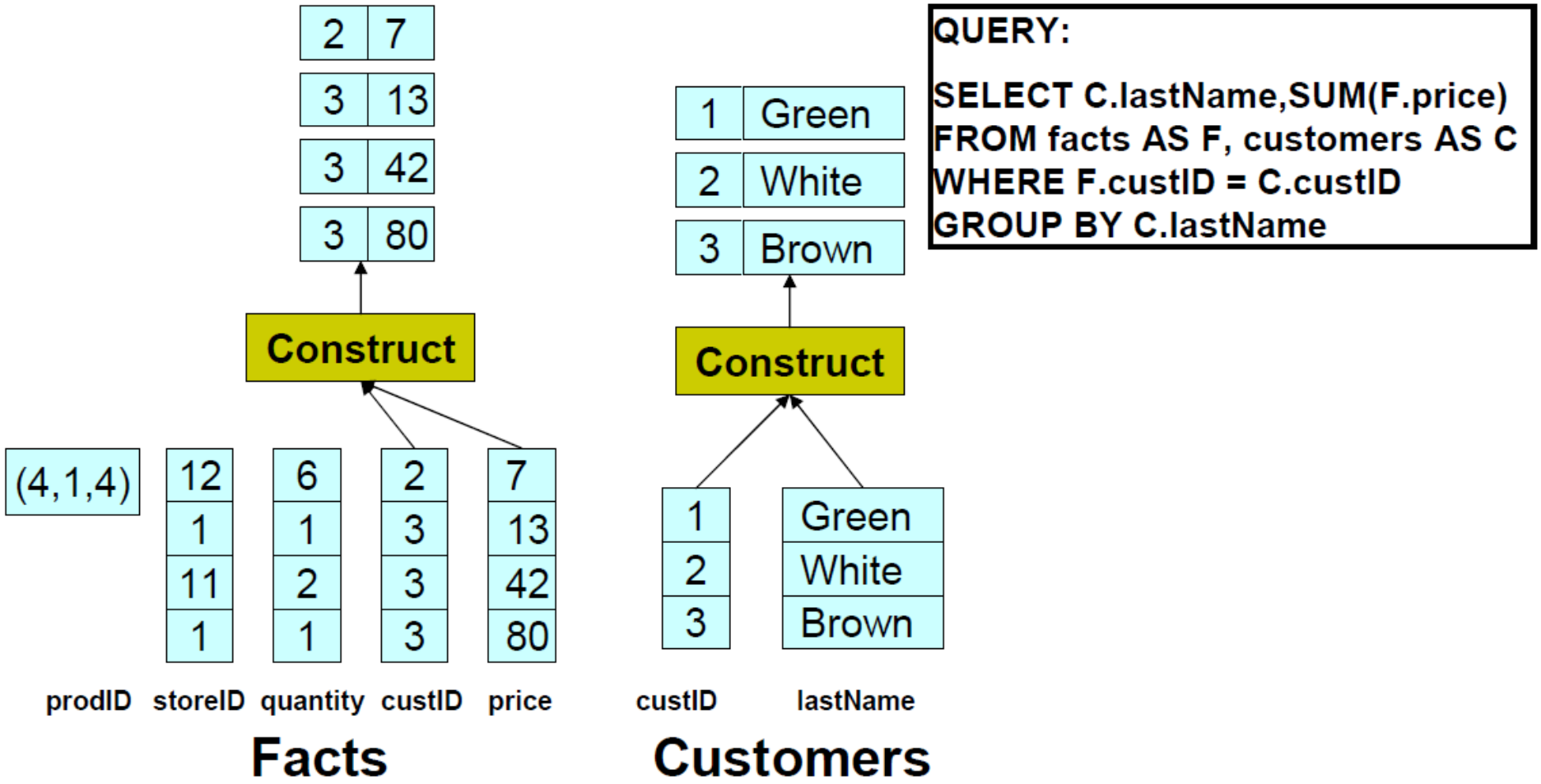




18

[18] Изображение взято из [Harizopoulos et al., 2009]

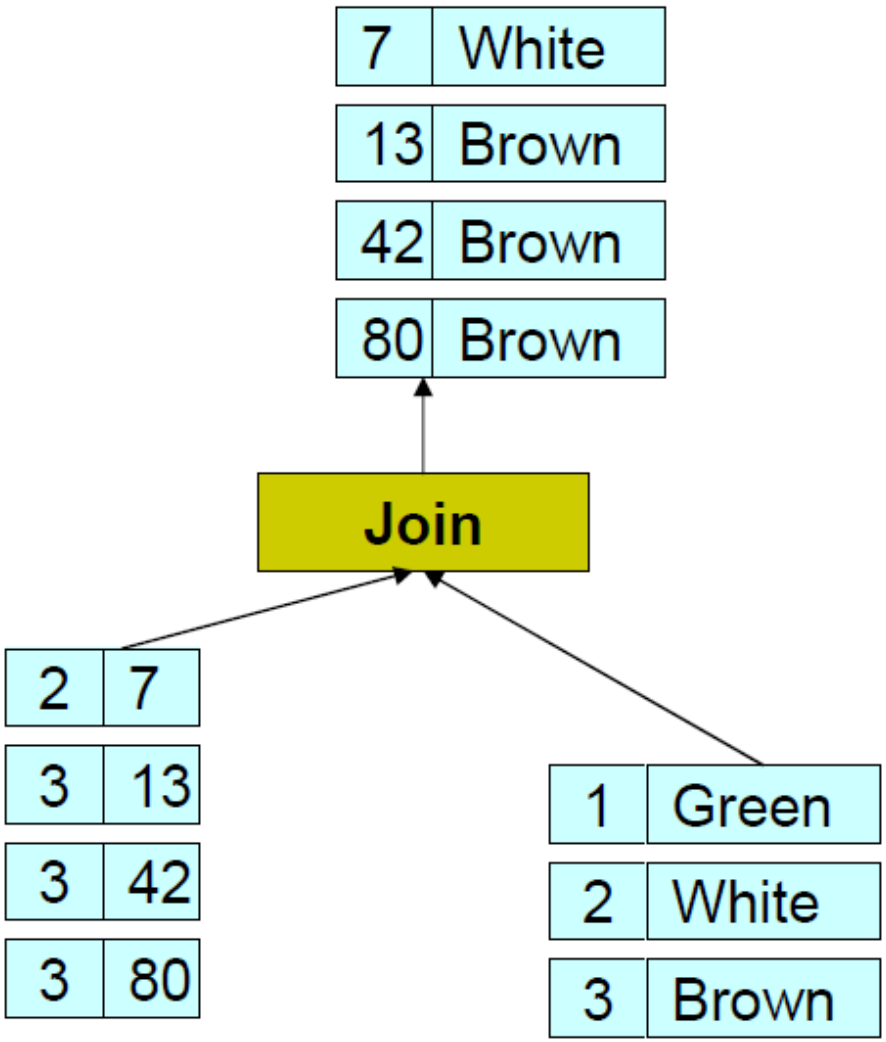
Соединения и EM I



19

[19] Изображение взято из [Harizopoulos et al., 2009]

Соединения и EM II



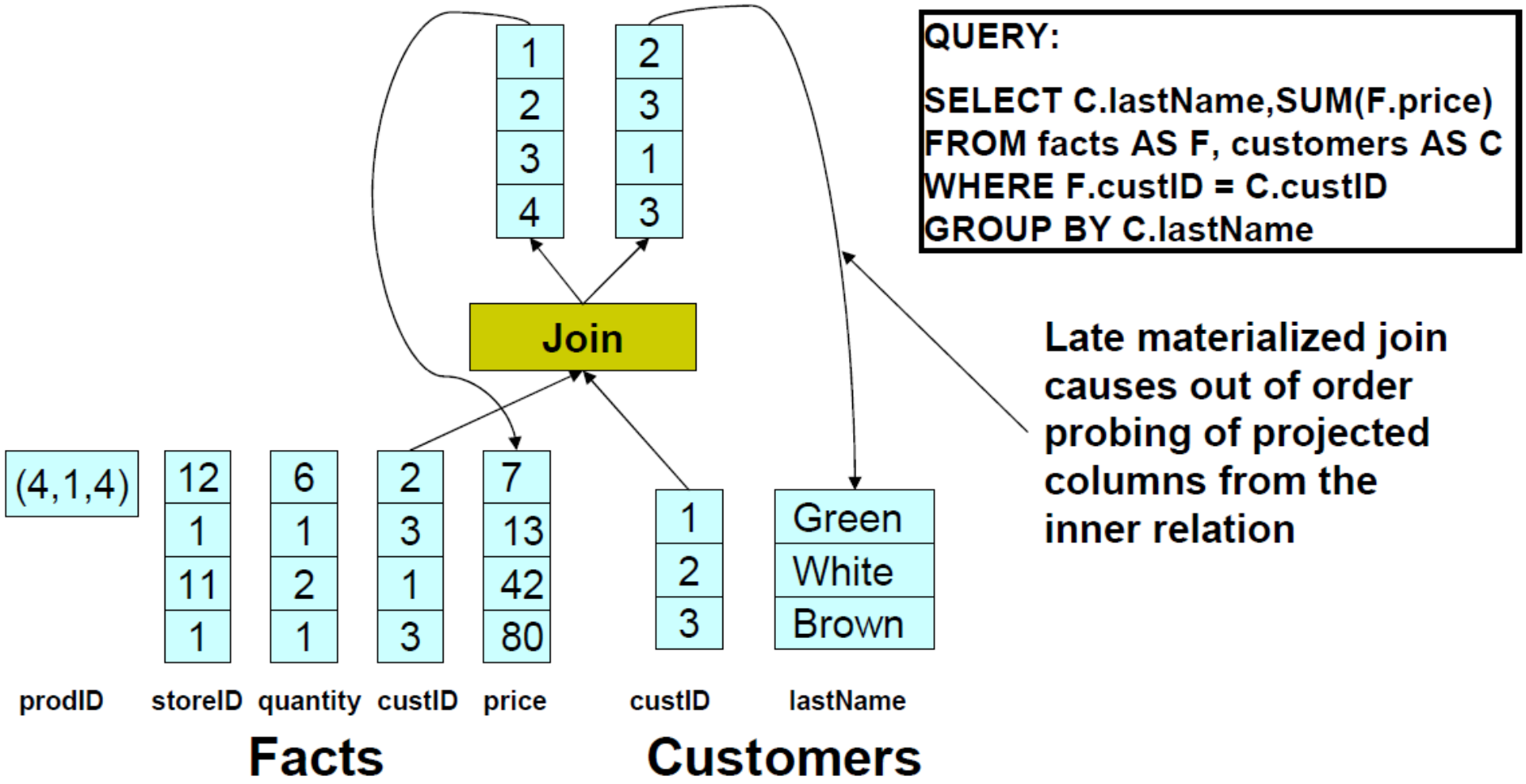
QUERY:

```
SELECT C.lastName,SUM(F.price)
FROM facts AS F, customers AS C
WHERE F.custID = C.custID
GROUP BY C.lastName
```

20

[20] Изображение взято из [Harizopoulos et al., 2009]

Соединения и LM



[21] Изображение взято из [Harizopoulos et al., 2009]

Промежуточный итог

- Наивный LM в сочетании с соединением хуже в 2 раза (зависит от многих факторов, селективностей, объема памяти...) нежели EM;
- Ответ: новые алгоритмы соединения — на следующей лекции.

Изображения

Большинство изображений взяты из оригиналов статей или прямо из слайдов [Harizopoulos et al., 2009].

Ссылки I


Daniel Abadi, Peter Boncz, Stavros Harizopoulos. The Design and Implementation of Modern Column-Oriented Database Systems. Foundations and Trends(R) in Databases Vol. 5, No. 3 (2012) 197–280


Stavros Harizopoulos, Daniel Abadi, Peter Boncz. Column-Oriented Database Systems. VLDB 2009 Tutorial (slides).


Г. А. Чернышев, «Организация физического уровня колоночных СУБД», Тр. СПИИРАН, 30 (2013), 204–222


Кузнецов С.Д., « MapReduce: внутри, снаружи или сбоку от параллельных СУБД?», Труды Института системного программирования РАН, 19 (2010), 35–70


Raghu Ramakrishnan and Johannes Gehrke. 2000. Database Management Systems (2nd ed.). Osborne/McGraw-Hill, Berkeley, CA, USA.