NATIONAL RESEARCH UNIVERSITY

АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

Bitmaps

Рамон Антонио Родригес Залепинос
arodriges@hse.ru

Структура модуля 1

	№	Дата	Тема лекции	№	Домашние задания
Модуль № 1	1	08 сен	Введение	A1	Установить Visual Studio с поддержкой C# и C++ на все рабочие машины (стационарные, переносные, т.п.). Найти книги по структурам данных. Посетить конференции.
	2	15 сен	Асимптотика	A2	Тестовая задача (C#): знакомство с процессом
	3	22 сен	Базовые СД	1a	Задание 1a на C#
	4	29 сен	Базовые СД 2	1b	Задание 1b на C++, **перевод 1a на C++**
	5	06 окт	Bitmaps	2a	Задание 2a: Контрольное Домашнее Задание (на C#) – CW
	6	13 окт	Хэш таблицы	2b	Задание 2b на C++, **перевод 2a на C++**

Пока мы в точности следуем
нашему календарному (понедельному) плану

Требование к студентам на лекции:
слушайте внимательно!

План лекции

1. Часть I: Определение bitmaps, операции над bitmaps
2. Часть II: WAH, CONCISE, Roaring bitmaps
3. Часть III: bitmaps в действии – invisible join (реальный практический пример)
4. Резюме

Bitmap: определение

Битовая карта (bitmap) – структура данных $B[0, n-1]$, которая для любого числа $i \in [0, n)$ может принимать два значения:

$$B[i] = \begin{cases} 1 \\ 0 \end{cases}$$

Базовая реализация bitmap – массив бит:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1

Какие объекты можно представить с помощью bitmap?

- Самый популярный объект – **множество целых чисел в диапазоне $[0, n)$**
- Тогда $B[i] = 1$, если число i принадлежит множеству

Bitmap: операции

Базовая реализация bitmap – массив бит

Множество целых чисел в диапазоне $[0, n)$: $B[i] = 1$, если число i принадлежит множеству

B_1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1

B_2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	1	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0

- Мощность $|B_1|$ (сколько бит равны единице)
- Проверка принадлежности: $\in \equiv B_1[i] = 1$
- Пересечение $\cap \equiv B_1 \& B_2$ (побитовое И)
- Объединение $\cup \equiv B_1 \mid B_2$ (побитовое ИЛИ)
- Вставка $\equiv B_1[i] \leftarrow 1$
- Ранг (rank) i – сколько ненулевых элементов в диапазоне $[0, i]$
- Выбор (select) i: найти i-ый по счету ненулевой бит

- **Эти операции можно выполнить с помощью других структур – для чего нужны bitmaps?**
- **Обычный bitmap уже относительно сжатый, но можно сжать еще …**

Причина 1: современные процессоры

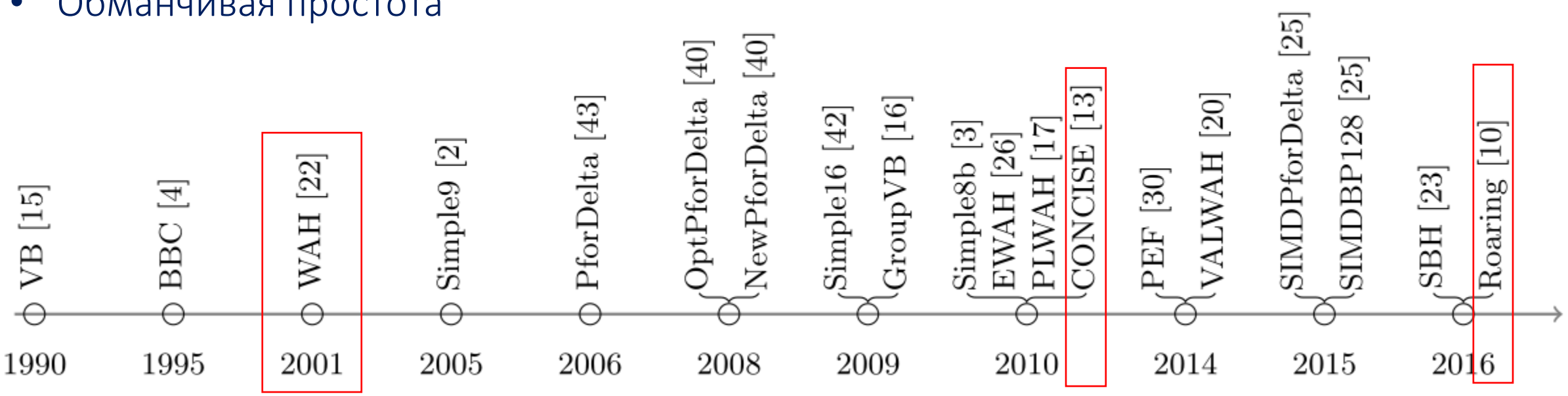
1. Часто побитовые операции ≈ 1-2 такта (зависит от числа бит)
2. Параллельное выполнение побитовых операций
3. У современных CPU достаточно объемные кэши

Причина 2: bitmaps хорошо сжимаются, меньше I/O

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

Разновидности bitmaps

- Работа ведется и пой сей день!
- Вдохновляйтесь!
- Обманчивая простота



«An Experimental Study of Bitmap Compression vs. Inverted List Compression», J. Wang et al.

WAH: Word Aligned Hybrid

Идея:

- разбить bitmap из n бит на $\left\lceil \frac{n}{w-1} \right\rceil$ «слов», каждое по $w-1$ бит
- w – «удобный» размер слова, напр. 32 бита

Два вида слов:

- Слово-заполнитель (fill word): состоит полностью из 0 либо 1 (31 бит из 0 либо 1)

30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Слово-литерал (literal word): состоит и из 0 и из 1 (31 бит из 0 и 1)

30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0

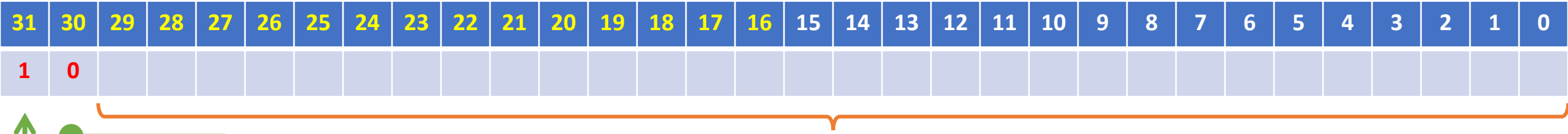
31 бит, не 32 бита

WAH: Word Aligned Hybrid

Идея:

- разбить bitmap из n бит на $\left\lceil \frac{n}{w-1} \right\rceil$ «слов», каждое по $w-1$ бит
- w – «удобный» размер слова, напр. 32 бита

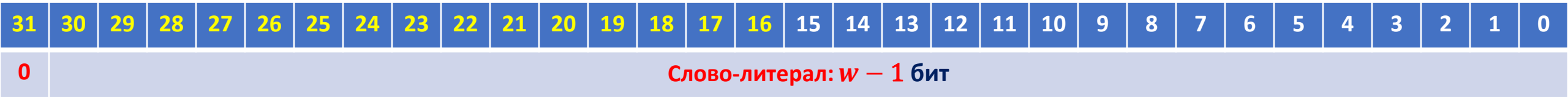
Кодируем слова-заполнители (fill words): состоят полностью из 0 либо 1 (31 бит из 0 или 1)



индикатор вида слова

тип заполнителя
0 либо 1

количество слов, которые состоят полностью из 0 либо 1
$w-2$ бита

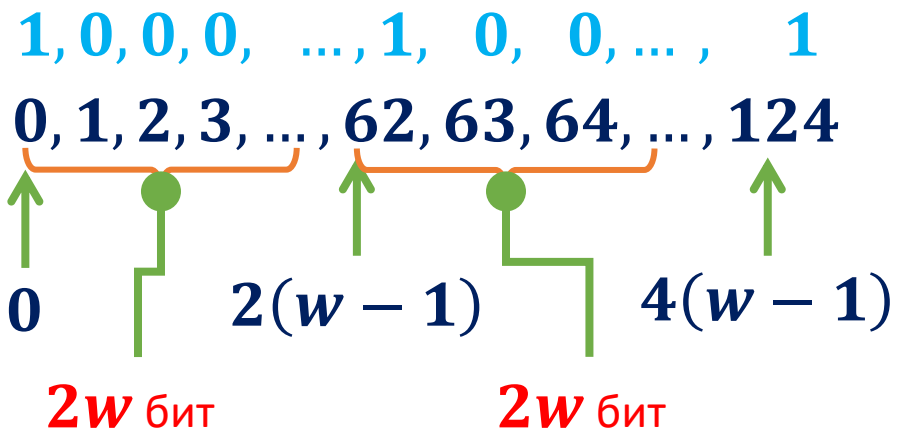


Кодируем только одно слово-литерал (literal word): состоит и из 0 и из 1 (31 бит из 0 и 1)

WAH

Пример

$$\{0, 2(w-1), 4(w-1), \dots\}$$



CONCISE: Compressed `n' Composable Integer Set

Идея:

- Аналогично WAH, но для счетчика слов r используется $w - 2 - \lceil \log_2 w \rceil$ бит вместо $w - 2$
- $\lceil \log_2 w \rceil$ бит – позиционные биты, кодирующие число $p \in [0, w)$
- Если $p = 0$, то мы закодировали $r + 1$ слов-заполнителей
- Иначе, мы закодировали r слов-заполнителей, но перед ними стоит одно слово с инвертированным битом под номером p

Пример:

$$0^{23}10^{111}1^{25}$$

6 групп по 31 бит

1. $0^{23}10^{7}$
2. 0^{31}
3. 0^{31}
4. 0^{31}

$p = 7$

$10001110^{22}011$

5. $0^{11}1^{20}$ $00^{11}1^{20}$ либо $01^{20}0^{11}$
6. 1^{5} $01^{5}0^{26}$ либо $00^{26}1^{5}$

(зависит от реализации)

Проблемы WAH и CONCISE

- Нет быстрого random access
- Нельзя быстро проверить принадлежность числа множеству

В обоих случаях надо раскодировать всю последовательность

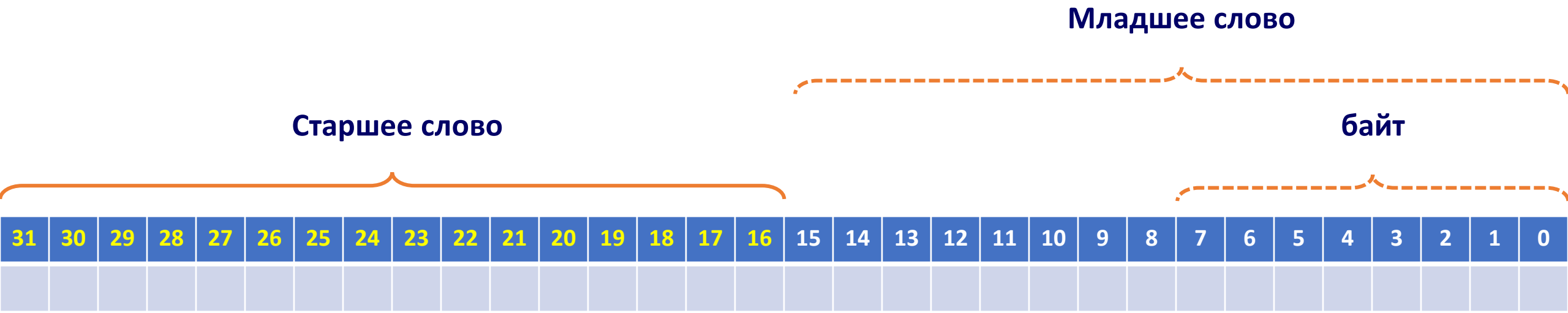
- Нельзя легко и быстро пропускать части последовательности

Например, при длинных последовательностях нулей при выполнении операции И/ИЛИ

Представление целых чисел

Нам для следующего способа (roaring bitmaps) потребуется:

- **32-битный**
- **беззнаковый (unsigned)**
- **целочисленный тип**



Важные свойства:

- **Диапазон значений** $[0, 2^{32} - 1]$

Roaring bitmaps

Предложены примерно в 2016 году

HEX	FFFF
DEC	65 535
OCT	177 777
BIN	1111 1111 1111 1111

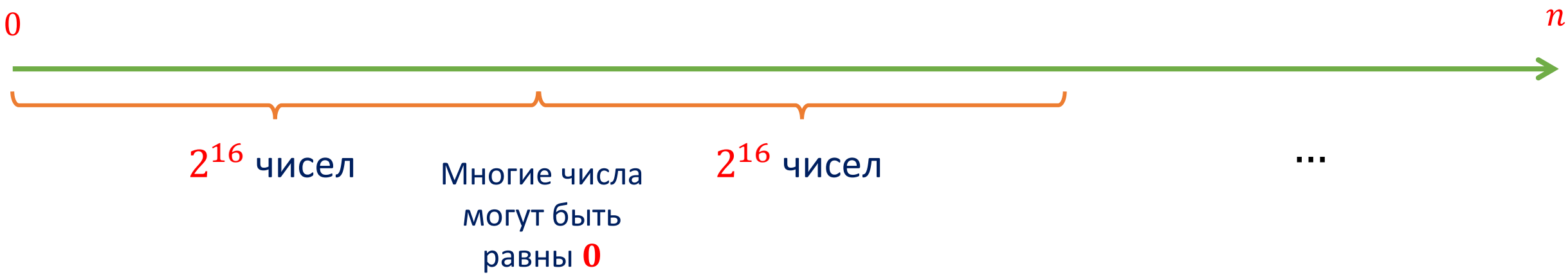
Старшее слово – общее для чисел $[0, 2^{16})$

Идея:

- делим всю последовательность 32 битных чисел $[0, n)$ на порции (**chunks**) по 2^{16} чисел
- поскольку числа идут подряд, у каждой порции общие старшие $\mathbf{16}$ бит
- кодируем каждую порцию (chunk) отдельно

Roaring bitmaps

Кодирование порций: 2 вида контейнеров



Разреженная порция (sparse) – не более 4096 чисел, не равных 0

- список этих чисел, отсортированный
- каждое число из 2^{16} бит

Плотная порция (dense) – более 4096 чисел, не равных 0

- bitmap (не сжатый)
- размер bitmap - 2^{16} бит

Roaring bitmaps: пример

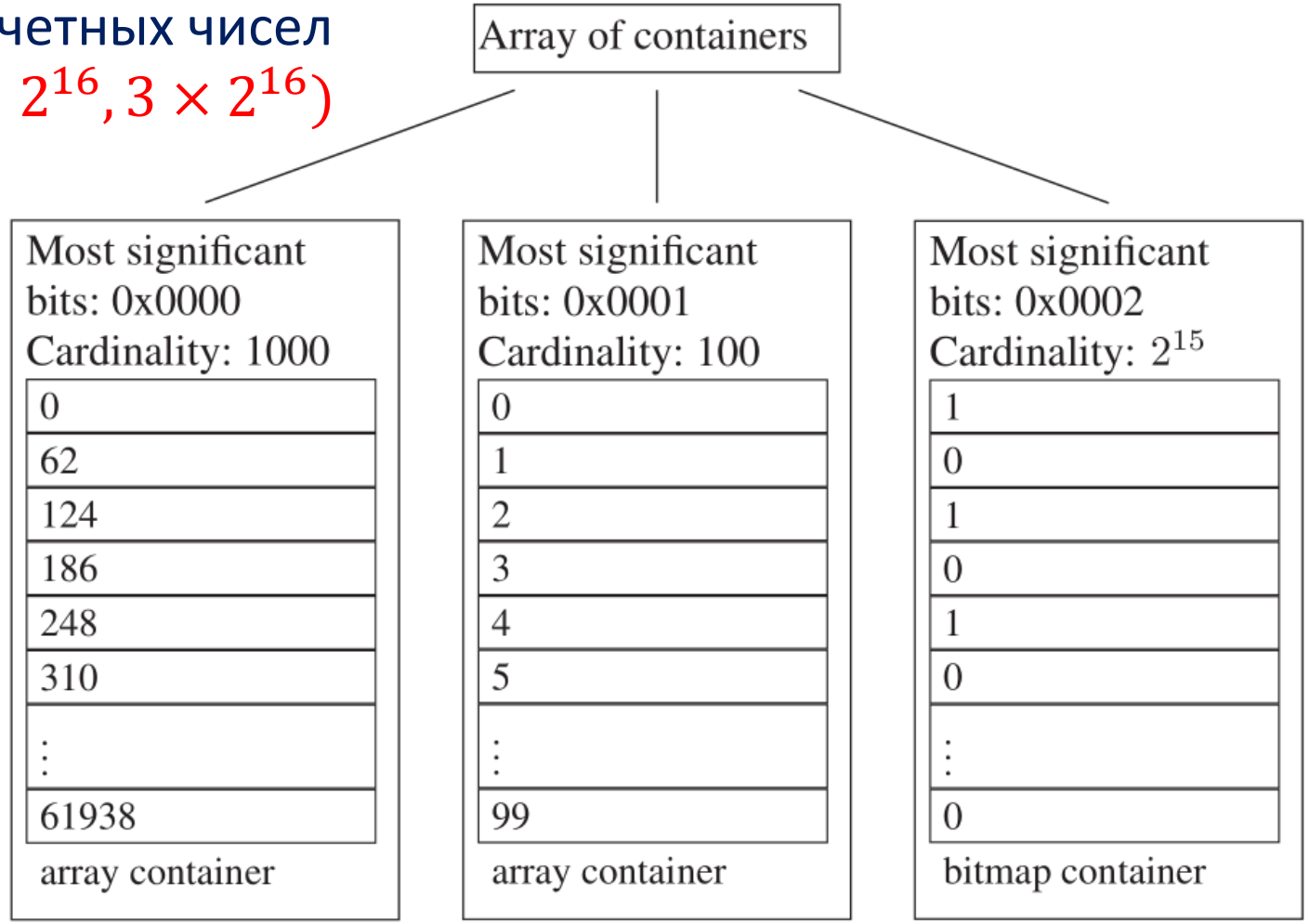
Кодируем 3 порции

1000 чисел
$i \times 62$

100 чисел
$[2^{16}, 2^{16} + 100)$

2^{15} четных чисел
$[2 \times 2^{16}, 3 \times 2^{16})$

- 4096 – эвристическое число
- данные любого контейнера занимают $\leq$ 8kB
  - 4096*2 / 1024 = 8
  - $2^{16}/2^3$ / 1024 $= 2^{13}/2^{10} = 8$
- “array of containers” – массив максимум из 2^{16} элементов с указателями на контейнеры
  - это динамический массив
  - обычно небольшой: для $n = 1$M макс. 16 элементов
  - может находится в кэше CPU
- каждый контейнер хранит кол-во элементов
  - расчет мощности прост: макс. $\lceil n/2^{16} \rceil$ операций
  - ранг (rank): число 1 в $[0, i]$
  - выбор (select): i-ый ненулевой бит



«Better Performance with Roaring Bitmaps», Samy Chambi et al.

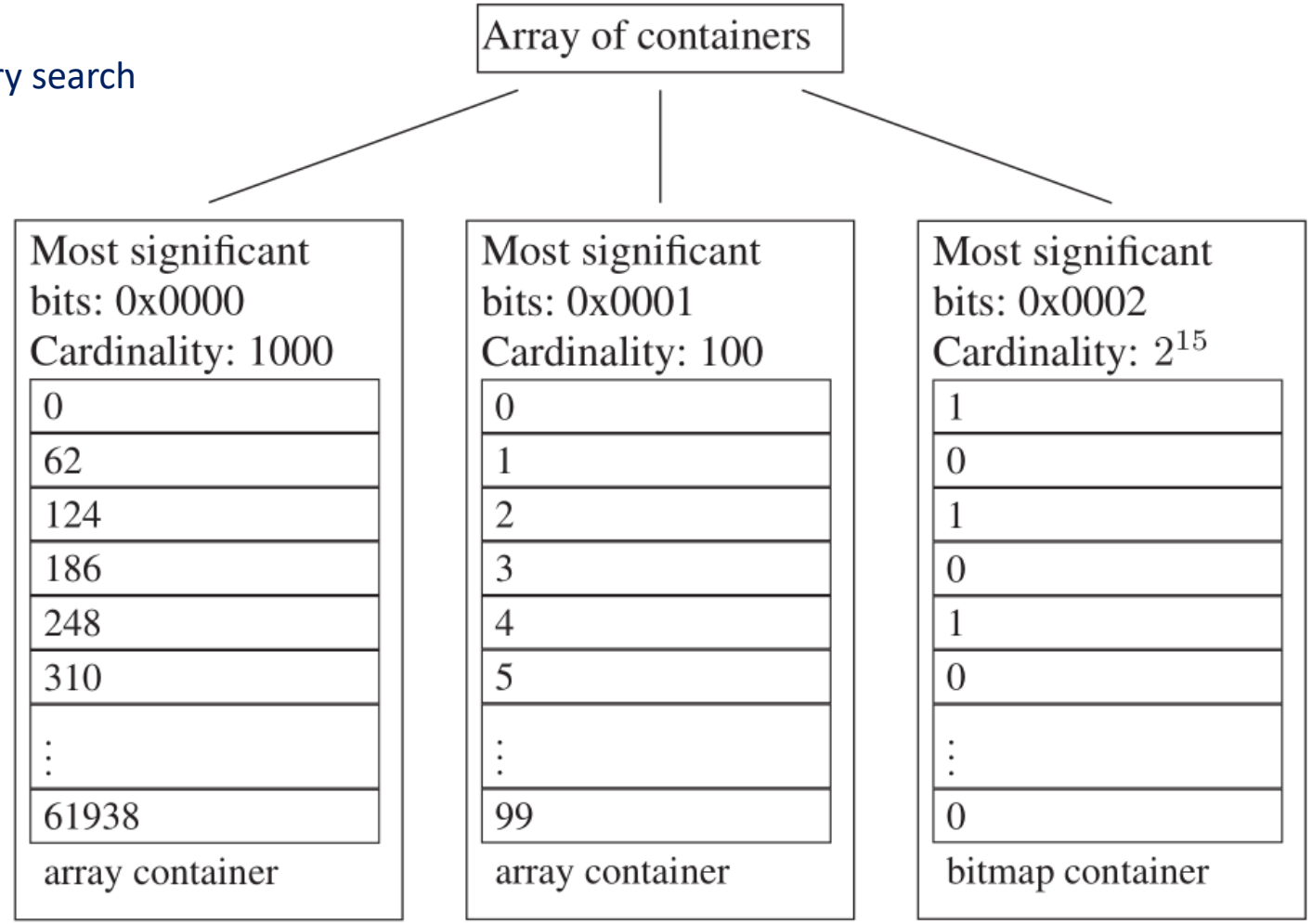
Roaring bitmaps: операции доступа

Принадлежность (S – множество) $x \in S$

1. Найти контейнер с индексом $x/2^{16}$ в “array of containers” (binary search)
2. Если таковой существует
   a) Это bitmap контейнер – найти бит $(x \ mod \ 2^{16})$
   b) Это array контейнер – найти число с помощью binary search

Удаление/добавление

1. Аналогично проверке принадлежности
2. Если таковой существует
   a) Это bitmap контейнер – установить/обнулить бит $(x \ mod \ 2^{16})$
   b) Это array контейнер – вставить/удалить число со сдвигом массива
3. При удалении: если мощность bitmap контейнера стала < 4096, превратить его в array контейнер
4. При вставке: если мощность array контейнера стала > 4096, превратить его в bitmap контейнер



«Better Performance with Roaring Bitmaps», Samy Chambi et al.

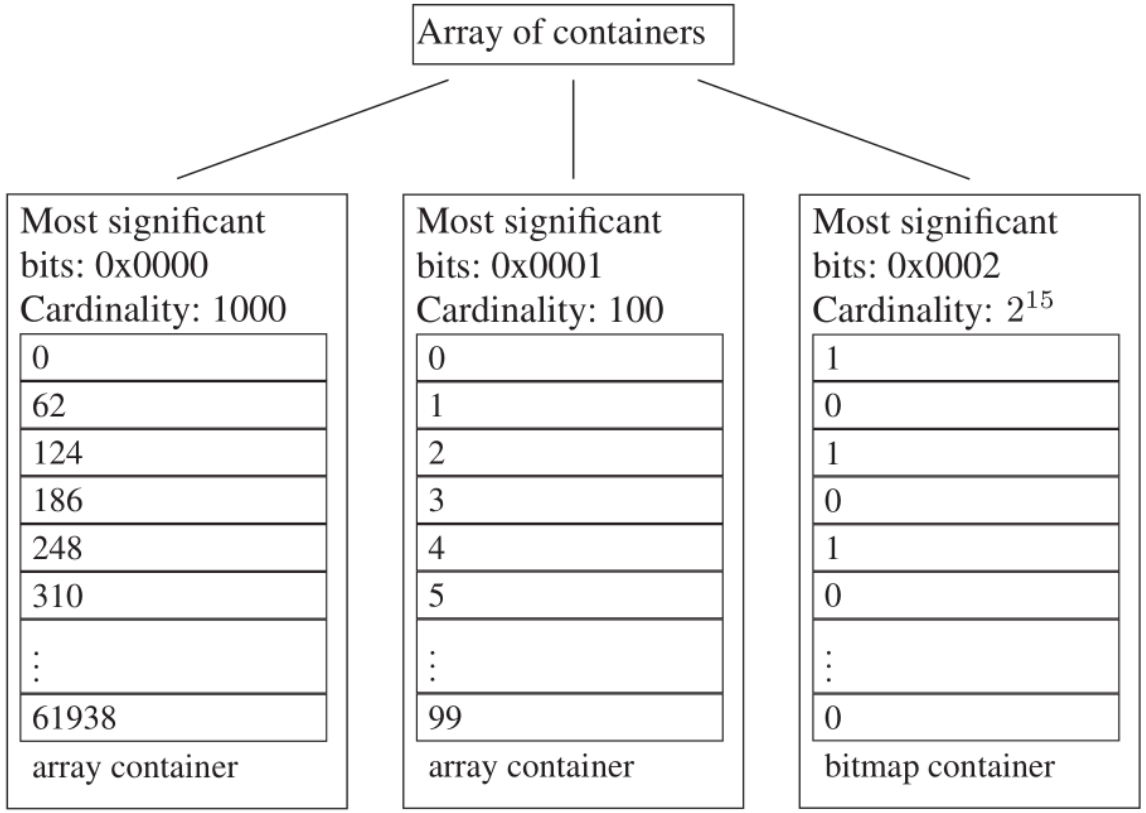
Roaring bitmaps: логические операции, побитовое ИЛИ

Algorithm 1 Routine to compute the union of two bitmap containers.

1: **input**: two bitmaps A and B indexed as arrays of 1024 64-bit integers
2: **output**: a bitmap C representing the union of A and B, and its cardinality c
3: $c \leftarrow 0$
4: Let C be indexed as an array of 1024 64-bit integers
5: **for** $i \in \{1, 2, \ldots, 1024\}$ **do**
6: $\quad C_i \leftarrow A_i$ OR B_i
7: $\quad c \leftarrow c + \text{bitCount}(C_i)$
8: **return** C and c

Побитовое ИЛИ

- Итерируем по «array of containers» обоих roaring bitmaps
- Три случая
  - OR для двух bitmap контейнеров (Алгоритм 1)
  - OR для bitmap и array контейнеров
  - OR для двух array контейнеров
- bitCount – быстрая операция (popcnt – 1 такт CPU)
- Long.bitCount – Java транслируется в popcnt
- Суперскалярность – OR, bitCount, store – параллельно



«Better Performance with Roaring Bitmaps», Samy Chambi et al.

Roaring bitmaps: логические операции, побитовое И

Побитовое И

(1) вычислить мощность,
(2) создать bitmap контейнер, либо
(3) создать array контейнер

1: **input**: two bitmaps A and B indexed as arrays of 1024 64-bit integers
2: **output**: a bitmap C representing the intersection of A and B, and its cardinality c if $c > 4096$ or an equivalent array of integers otherwise
3: $c \leftarrow 0$
4: **for** $i \in \{1, 2, \ldots, 1024\}$ **do**
5: $\quad c \leftarrow c + \text{bitCount}(A_i \text{ AND } B_i)$
6: **if** $c > 4096$ **then**
7: $\quad$ Let C be indexed as an array of 1024 64-bit integers
8: $\quad$ **for** $i \in \{1, 2, \ldots, 1024\}$ **do**
9: $\quad\quad C_i \leftarrow A_i \text{ AND } B_i$
10: $\quad$ **return** C and c
11: **else**
12: $\quad$ Let D be an array of integers, initially empty
13: $\quad$ **for** $i \in \{1, 2, \ldots, 1024\}$ **do**
14: $\quad\quad$ append the set bits in A_i AND B_i to D using Algorithm
15: $\quad$ **return** D

(1)

(2)

(3)

"Алгоритмические трюки для программистов" (книга)

Let S be an initially empty list
while $w \neq 0$ **do**
$\quad t \leftarrow w \text{ AND } -w$
$\quad$ append bitCount$(t-1)$ to S
$\quad w \leftarrow w \text{ AND } (w-1)$
return S

«Better Performance with Roaring Bitmaps», Samy Chambi et al.

Roaring bitmaps: логические операции

bitmap контейнер и array контейнер

Побитовое И

(1) Пройтись по **array контейнер** и проверить, есть ли соответствующее значение в **bitmap контейнер**
(2) Результат записать в **array контейнер** – *Q*: почему?

Побитовое ИЛИ

(1) Создать копию **bitmap контейнера и поместить результат сюда** – *Q*: почему?
(2) Пройтись по **array контейнер** и установить биты в новом **bitmap контейнере**

array контейнер и array контейнер

Побитовое И

(1) Результат записываем **array контейнер** – *Q*: почему?
(2) Итерируем одновременно по двум **array контейнерам** как при слиянии массивов – *здесь упрощение по сравнению с оригинальной статьей для облегчения понимания и программирования*

Побитовое ИЛИ

(1) Если результат не более 4096, слить массивы
(2) Иначе создать bitmap контейнер и поместить туда результат

Roaring bitmaps: производительность

На разных datasets

Table II. Results on real data.

	CENSUS1881	CENSUSINCOME	WIKILEAKS	WEATHER
(a) Size expansion if Roaring is replaced with other schemes.				
Concise	2.21	1.38	0.79	1.38
WAH	2.43	1.63	0.79	1.51
BitSet	41.50	2.89	55.45	3.49
(b) Time increase, for AND, if Roaring is replaced with other schemes				
Concise	921.81	6.58	8.30	6.26
WAH	841.08	5.89	8.16	5.40
BitSet	733.85	0.42	27.91	0.64
(c) Time increases, for OR, if Roaring is replaced with other schemes.				
Concise	33.80	5.41	2.14	3.87
WAH	30.58	4.85	2.06	3.39
BitSet	28.73	0.43	6.72	0.48

«Better Performance with Roaring Bitmaps», Samy Chambi et al.

Roaring bitmaps: где применяется

- **После 2016 года были улучшения**
- **Это один из самых современных и широко используемых bitmaps**

- Google Procella: YouTube's SQL Engine,
- Apache Lucene and derivative systems such as Solr and Elasticsearch,
- Apache Druid,
- Apache Spark,
- Apache Hive,
- Apache Tez,
- Apache Zeppelin,
- Apache Doris,
- Apache CarbonData,
- Yandex ClickHouse,
- Netflix Atlas,
- LinkedIn Pinot,
- OpenSearchServer,
- Cloud Torrent,
- Whoosh,
- InfluxDB,
- Pilosa,
- Bleve,
- Microsoft Visual Studio Team Services (VSTS),
- Intel's Optimized Analytics Package (OAP),
- Tablesaw,
- Jive Miru,
- Gaffer,
- Apache Hivemall,
- lindb,
- Elasticell,
- SourceGraph,
- M3,
- trident,
- eBay's Apache Kylin

https://roaringbitmap.org/

Таблицы и базы данных (СУБД) – «хлеб» программиста

Сеть продуктовых магазинов ведет базу данных транзакций покупок
(транзакция содержит информацию о покупке – дата, продукт, в каком магазине произошла покупка, …)

Список транзакций покупок в сети магазинов

ID: целое, **дата**: datatime, **название товара**: string, **адрес магазина**: string

1, 06.10.2020 14:20, молоко, Покровский бульвар
2, 06.10.2020 14:25, молоко, Покровский бульвар
3, 06.10.2020 14:26, хлеб, Покровка ул.
4, 06.10.2020 14:29, молоко, Покровка ул.

дублирование данных

ID	дата	название товара	адрес магазина
1	06.10.2020 14:20	молоко	Покровский бульвар
2	06.10.2020 14:25	молоко	Покровский бульвар
3	06.10.2020 14:26	хлеб	Покровка ул.
4	06.10.2020 14:29	молоко	Покровка ул.

Таблицы фактов и измерений

Таблица фактов

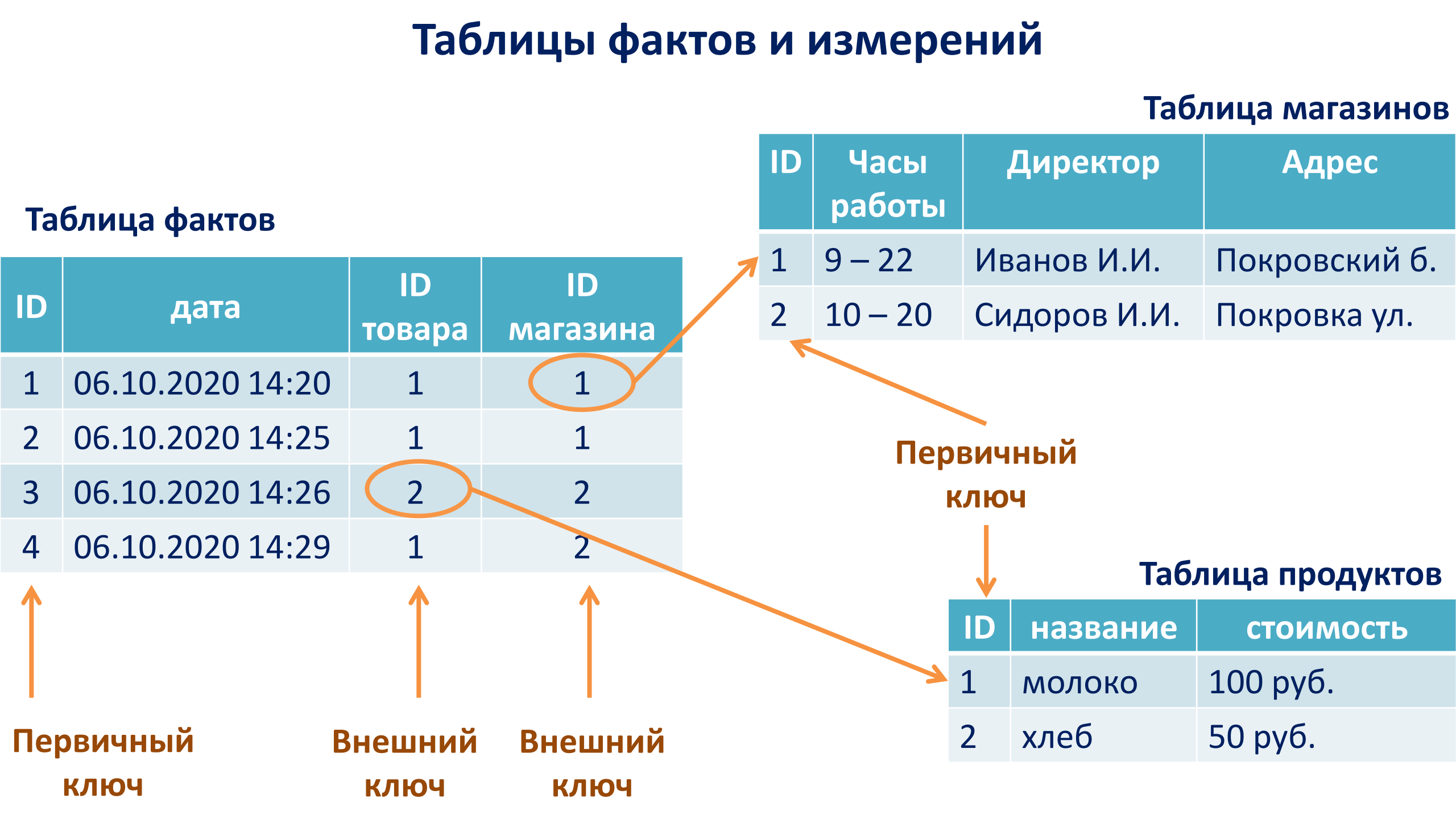
ID	дата	ID товара	ID магазина
1	06.10.2020 14:20	1	1
2	06.10.2020 14:25	1	1
3	06.10.2020 14:26	2	2
4	06.10.2020 14:29	1	2

Таблица магазинов

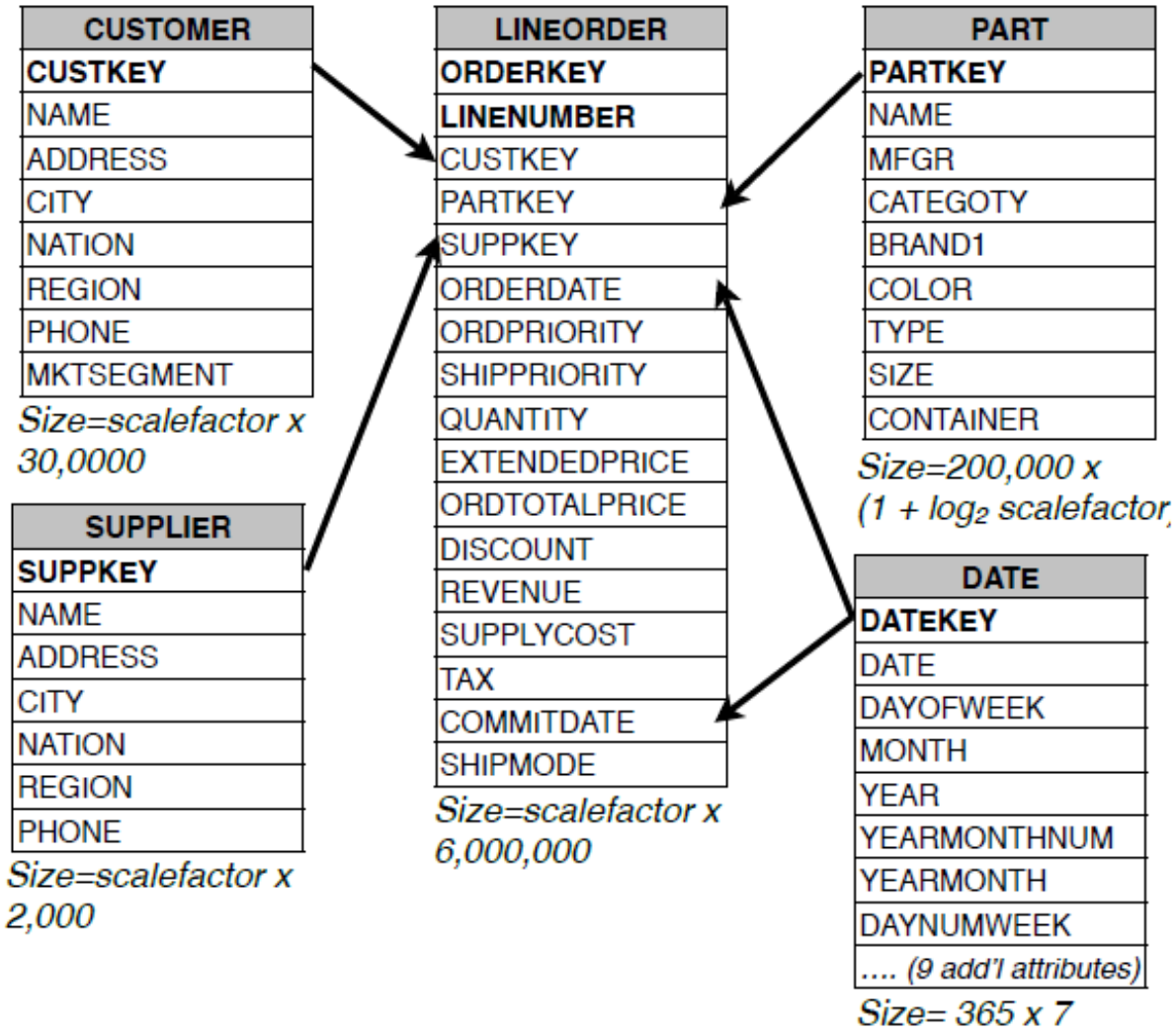
ID	Часы работы	Директор	Адрес
1	9 – 22	Иванов И.И.	Покровский б.
2	10 – 20	Сидоров И.И.	Покровка ул.

Таблица продуктов

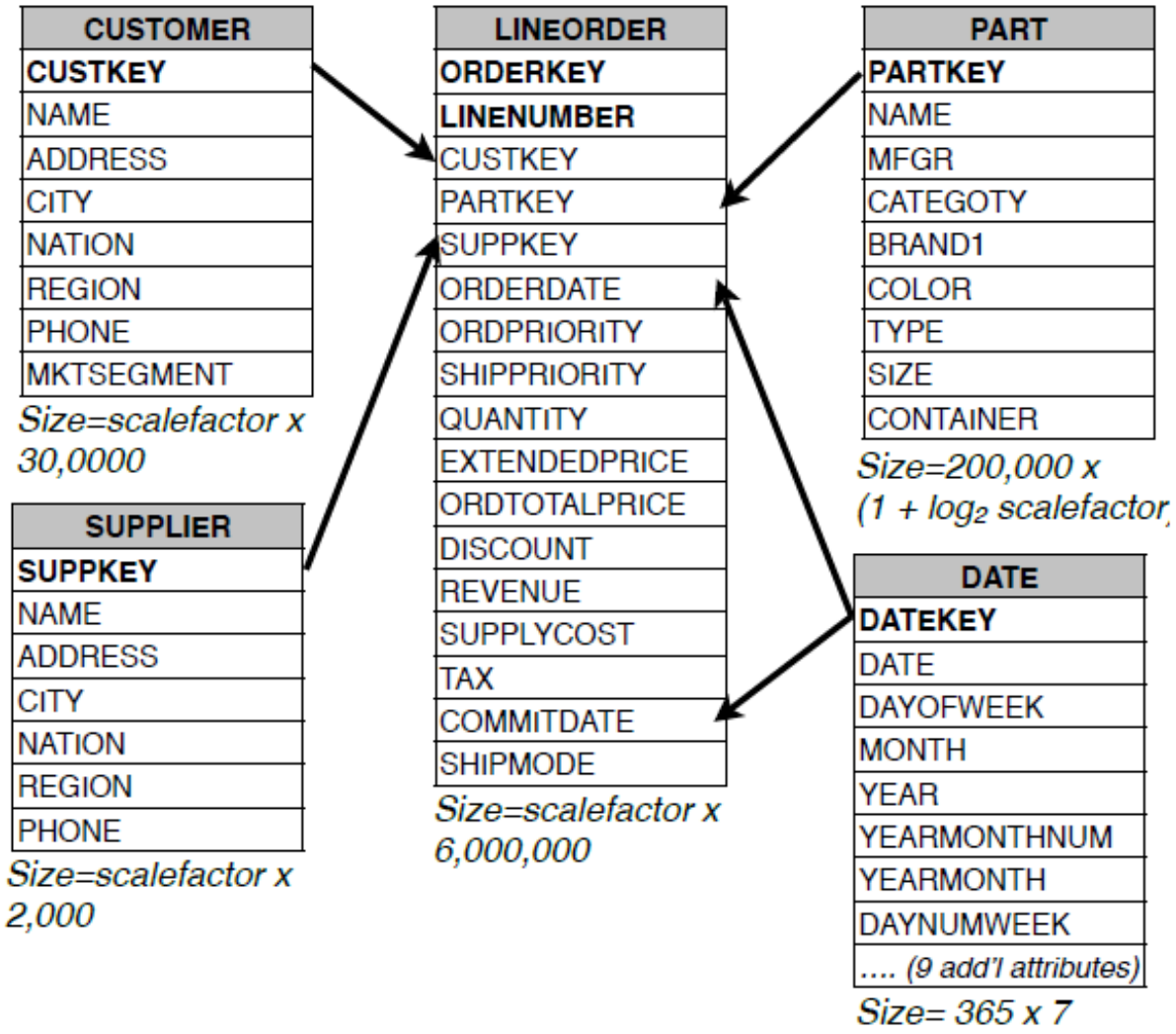
ID	название	стоимость
1	молоко	100 руб.
2	хлеб	50 руб.



SSBM (Star Shema Benchmark) – схема «звездочка»



СУБД – column stores: хранят таблицы поколоночно



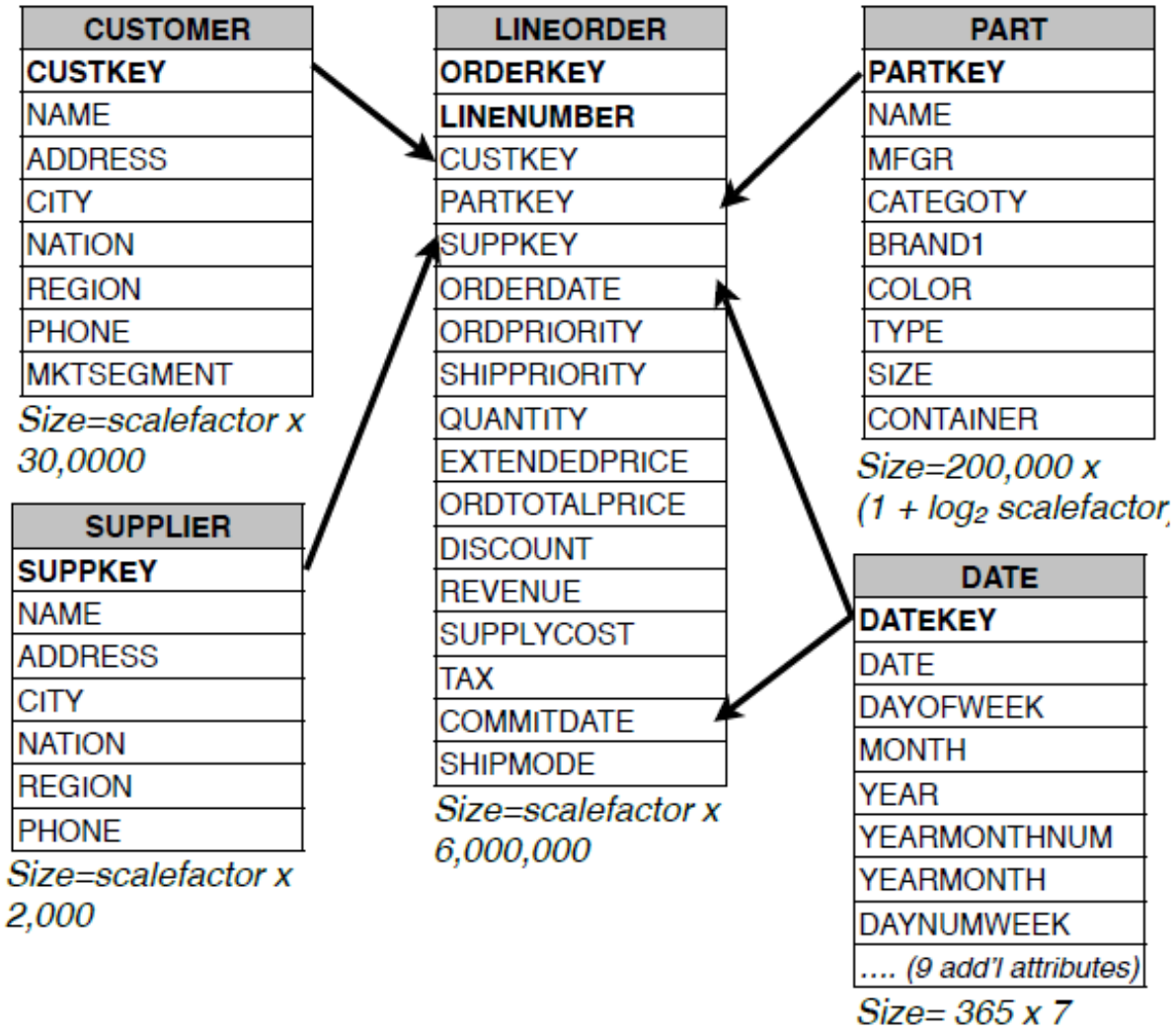
Data Warehouses:

- Mostly read-only queries
- Mostly analytical queries

Column-stores

- Store each column separately
- Respective optimizations

СУБД – column stores: хранят таблицы поколоночно

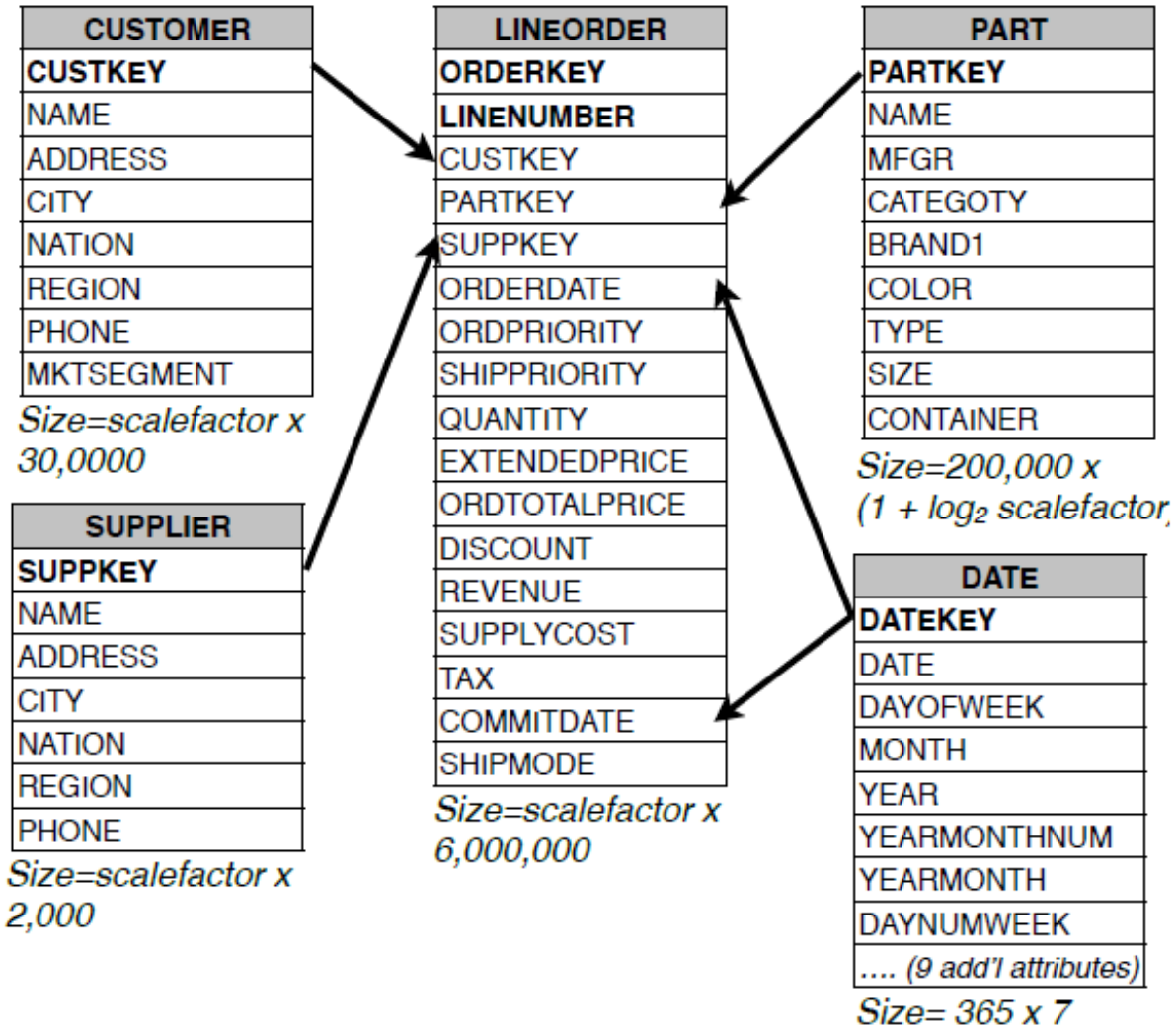


Задача:

Найти заказы, у которых

1. Customer из Asia
2. Supplier из Asia
3. Год [1992, … , 1997]

СУБД – column stores: невидимое соединение **«invisible join»**



Алгоритм соединения таблиц, которые спроектированы по “star schema” (схема «звездочка»)

Разработан около 2008 года

Invisible join: фаза 1

Apply “region = ‘Asia’” On Customer Table

custkey	region	nation	...
1	ASIA	CHINA	...
2	EUROPE	FRANCE	...
3	ASIA	INDIA	...

→ **set** Containing Keys 1 and 3

Apply “region = ‘Asia’” On Supplier Table

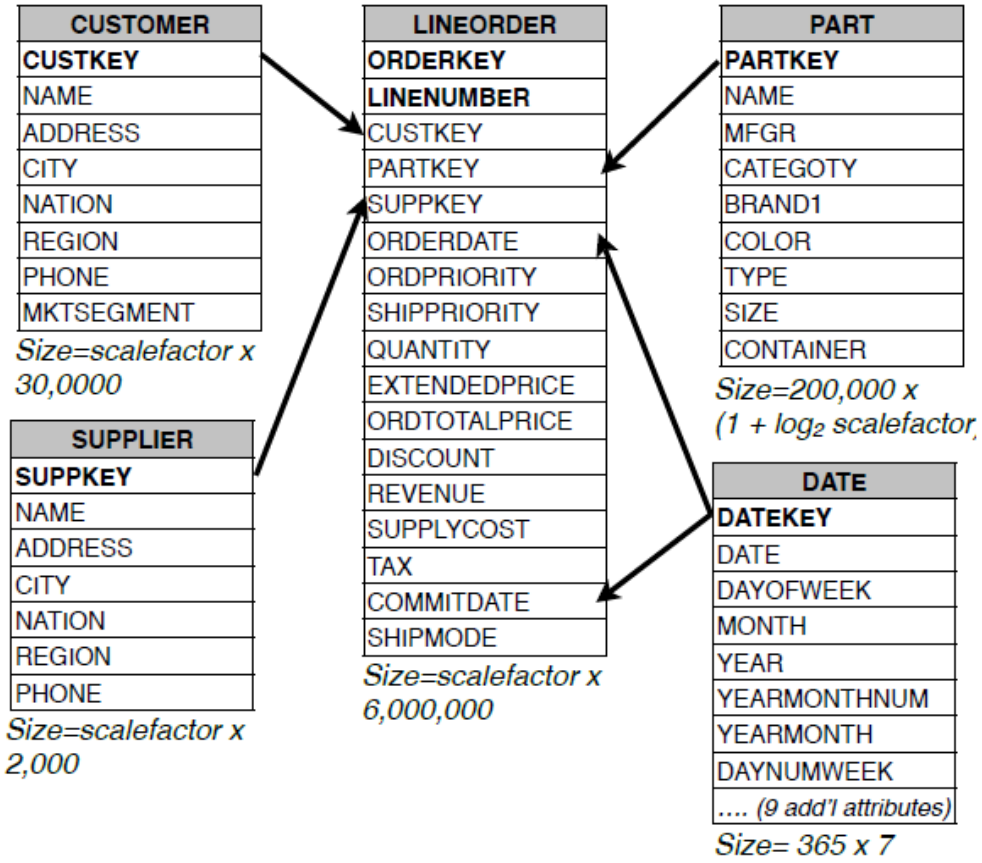
suppkey	region	nation	...
1	ASIA	RUSSIA	...
2	EUROPE	SPAIN	...

→ **set** Containing Key 1

Apply “year in [1992,1997]” On Date Table

dateid	year	...
01011997	1997	...
01021997	1997	...
01031997	1997	...

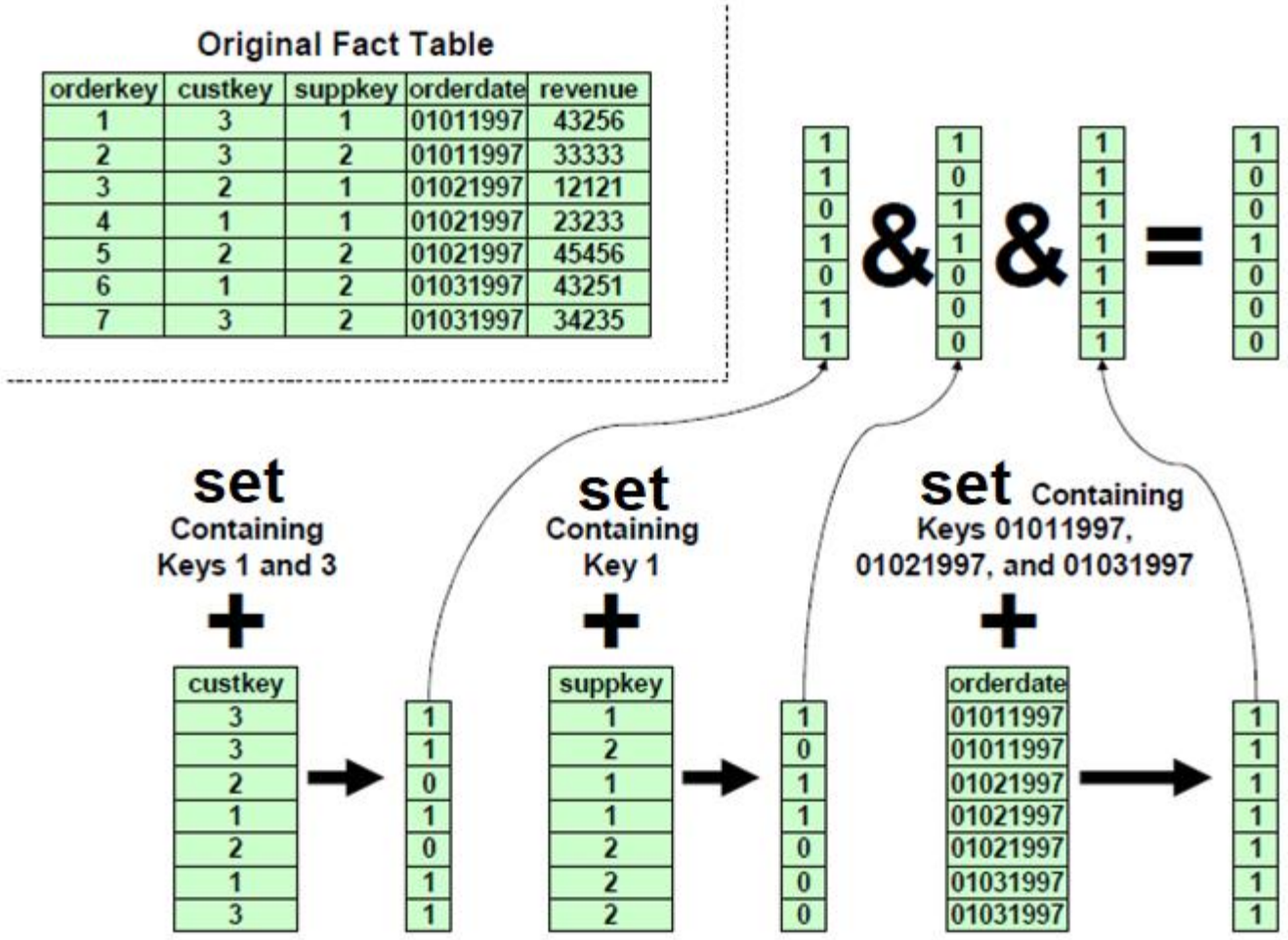
→ **set** Containing Keys 01011997, 01021997, and 01031997

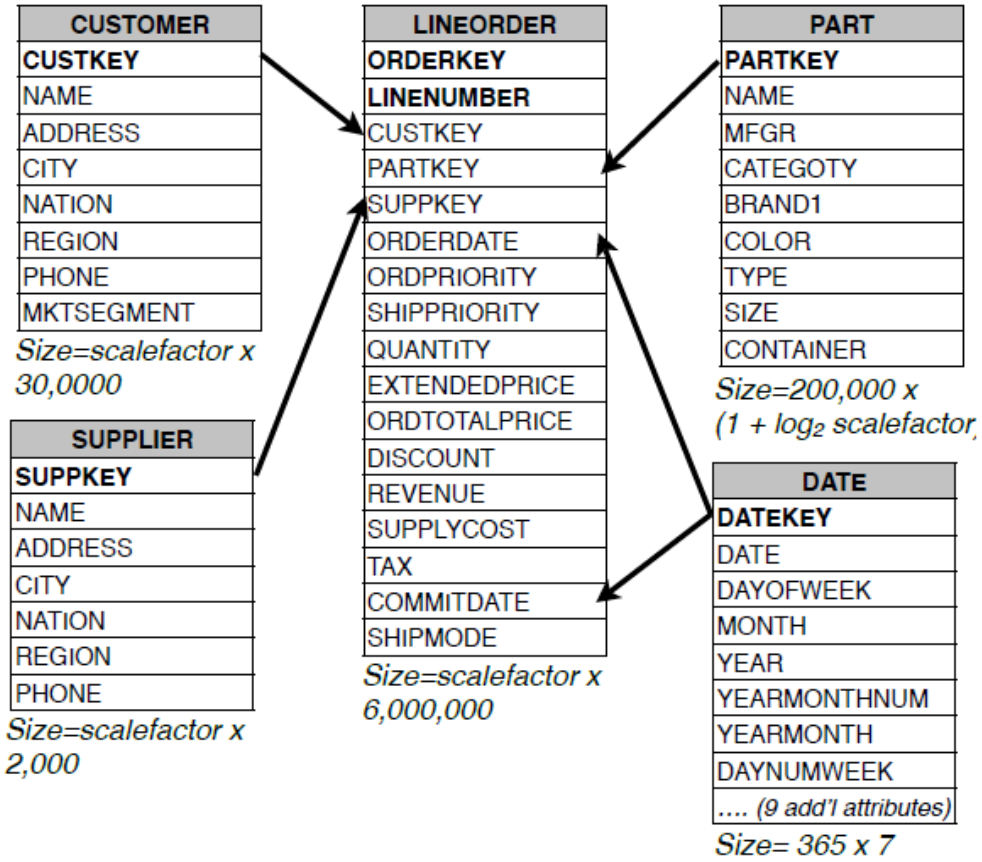


Предикаты применяются к соответствующей таблице измерений для извлечения **множества** первичных ключей, который удовлетворяют предикату

Query Execution in Column-Oriented Database Systems», Daniel J. Abadi

Invisible join: фаза 2

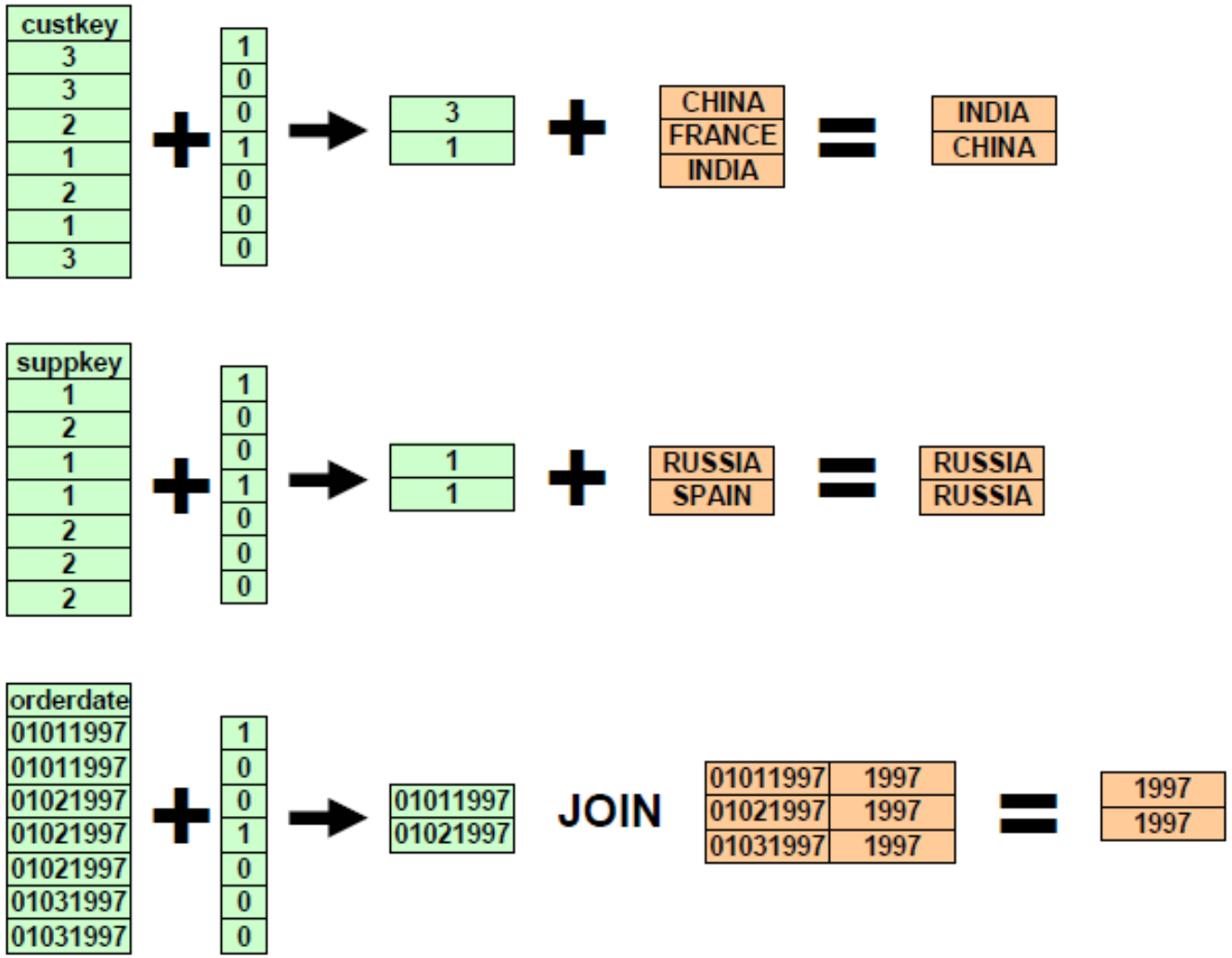


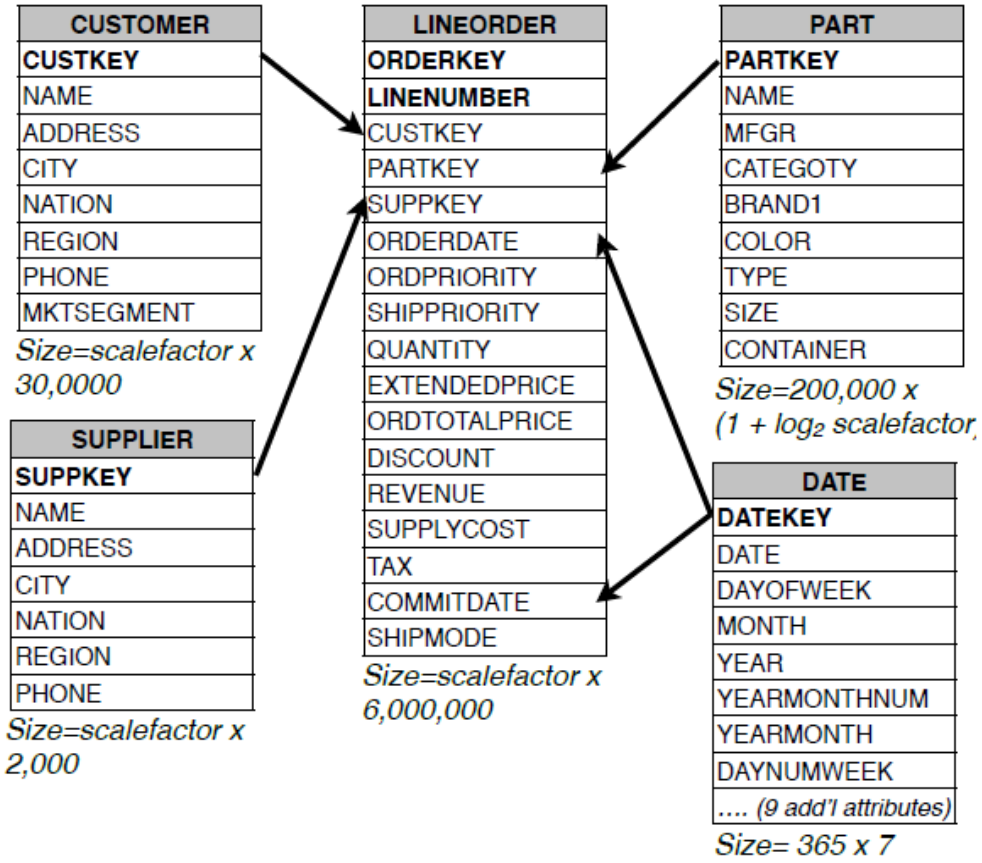


Каждый внешний ключ в таблице фактов в каждой строке проверяется на принадлежность множеству, строится bitmap

Query Execution in Column-Oriented Database Systems», Daniel J. Abadi

Invisible join: фаза 3

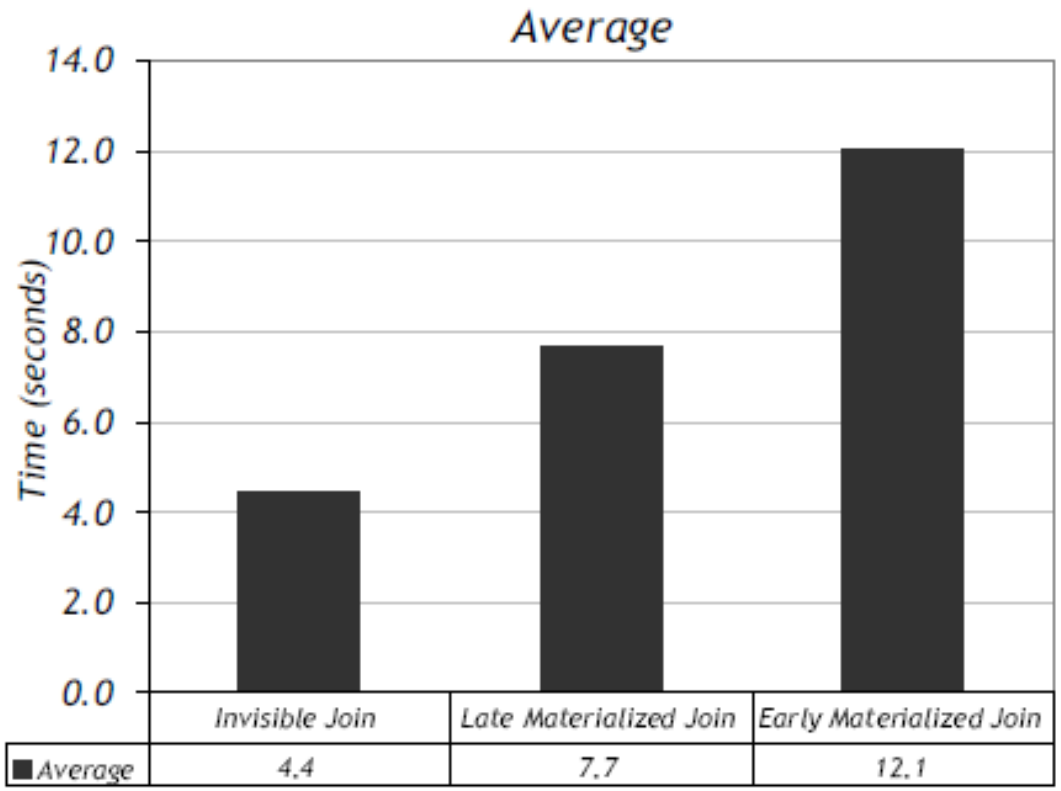


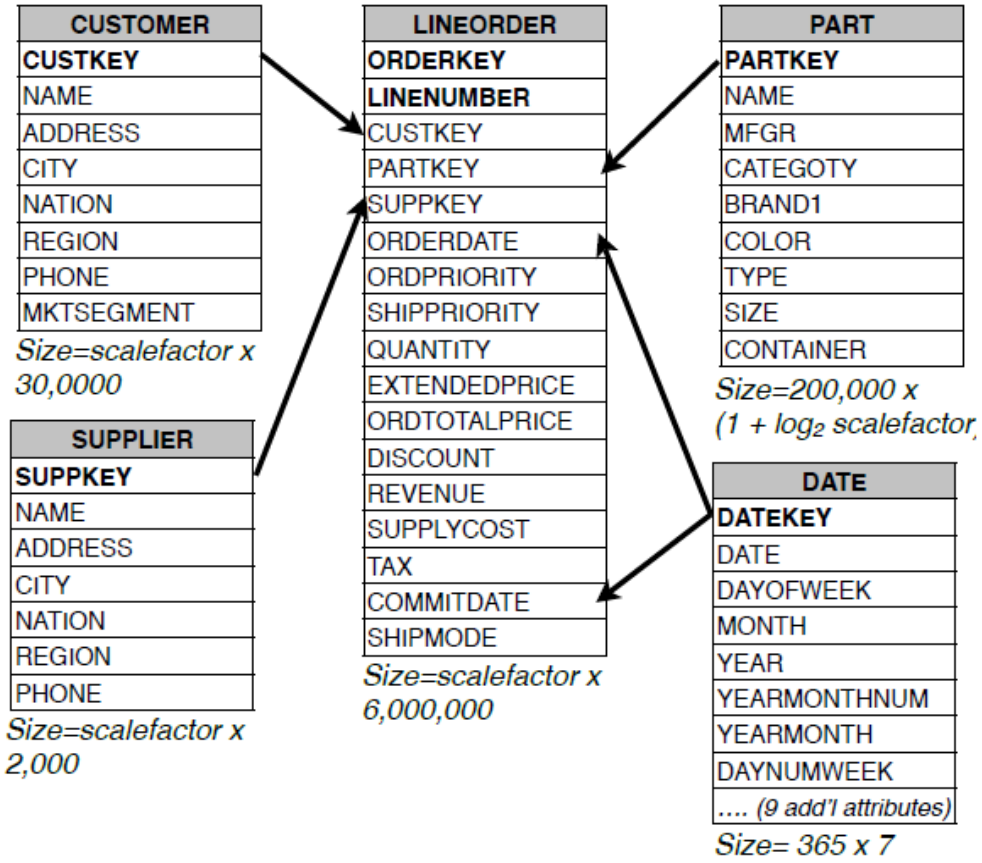


Извлекаем значения из таблиц-измерений

Query Execution in Column-Oriented Database Systems», Daniel J. Abadi

Invisible join: performance





Усредненные значения по всем запросам SSBM

Query Execution in Column-Oriented Database Systems», Daniel J. Abadi

Invisible Join впервые реализован в C-Store

- Research: C-Store
- Commercial: Vertica
  Инновационное предпринимательство

C-Store

Тип	СУБД
Автор	Майкл Стоунбрейкер и коллеги
Написана на	C++
Первый выпуск	2005
Аппаратная платформа	Кроссплатформенное программное обеспечение
Последняя версия	0.2 (2006)
Состояние	Коммерциализирована как продукт компании Vertica
Лицензия	BSD
Сайт	db.lcs.mit.edu/projects/cstore/

HP Vertica (E-Bay, кластер из 100 узлов, 7.5PB, 2007 г.)

Michael Stonebraker

IEEE John von Neumann Medal (2005)
ACM Turing Award (2014)

PostgreSQL, Vertica, Streambase, Illustra, VoltDB, SciDB

https://db-engines.com/en/ranking

Vertica → Hewlett Packard, 2011
HP → Micro Focus, 2017

VERTICA

Industry	Enterprise Software & Database Management & Data Warehousing
Founded	2005
Founder	Andrew Palmer and Michael Stonebraker
Headquarters	Cambridge, MA
Key people	Colin Mahony (SVP and General Manager) Joy King (VP of Product Marketing & Product Management) Misha Davidson (Director of Engineering)
Products	Vertica Analytics Platform Enterprise Edition, Vertica SQL on Hadoop, Vertica Analytics Platform Community Edition
Parent	Micro Focus
Website	www.vertica.com

ChronosDB – инновационная растровая СУБД

R.A. Rodriges Zalipynis.
ChronosDB: Distributed, File Based, Geospatial Array DBMS
VLDB, Рио-де-Жанейро, Бразилия
27 – 31 Августа 2018 г.

Единственный устный доклад от РФ за последние 10 лет

Один из самых известных ученых в области баз данных

Michael Stonebraker

SciDB = Scientific DB

Единственная на сегодняшний день распределенная растровая СУБД в открытом доступе

ChronosDB в среднем в 75 раз быстрее SciDB

IEEE John von Neumann Medal (2005)
ACM Turing Award (2014)

PostgreSQL, Vertica, Streambase, Illustra, VoltDB, SciDB

Резюме

1. Часть I: Определение bitmaps, операции над bitmaps
2. Часть II: WAH, CONCISE, Roaring bitmaps
3. Часть III: bitmaps в действии – invisible join (реальный практический пример)

Литература

Диссертация:

- Abadi, D. J. (2008). Query execution in column-oriented database systems (Doctoral dissertation, Massachusetts Institute of Technology). (**invisible join**)

Статьи:

- Chambi, S., Lemire, D., Kaser, O., & Godin, R. (2016). Better bitmap performance with roaring bitmaps. Software: practice and experience, 46(5), 709-719. (**Roaring bitmaps**)
- Wu, K., Otoo, E. J., & Shoshani, A. (2006). Optimizing bitmap indices with efficient compression. ACM Transactions on Database Systems (TODS), 31(1), 1-38. (**WAH**)
- Wang, J., Lin, C., Papakonstantinou, Y., & Swanson, S. (2017, May). An experimental study of bitmap compression vs. inverted list compression. In Proceedings of the 2017 ACM International Conference on Management of Data (pp. 993-1008). (**survey of bitmaps**)

NATIONAL RESEARCH UNIVERSITY

Благодарю за внимание!

Рамон Антонио Родригес Залепинос
arodriges@hse.ru