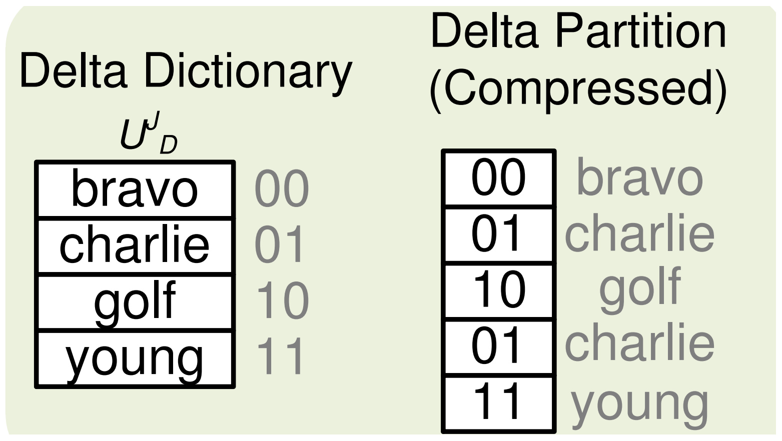
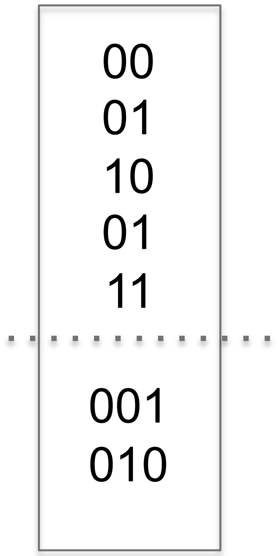
Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

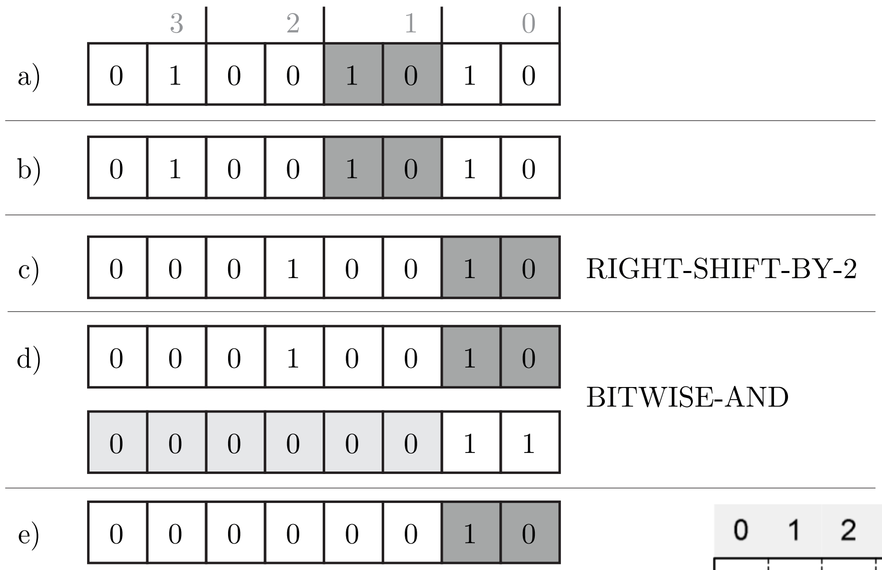
Topics

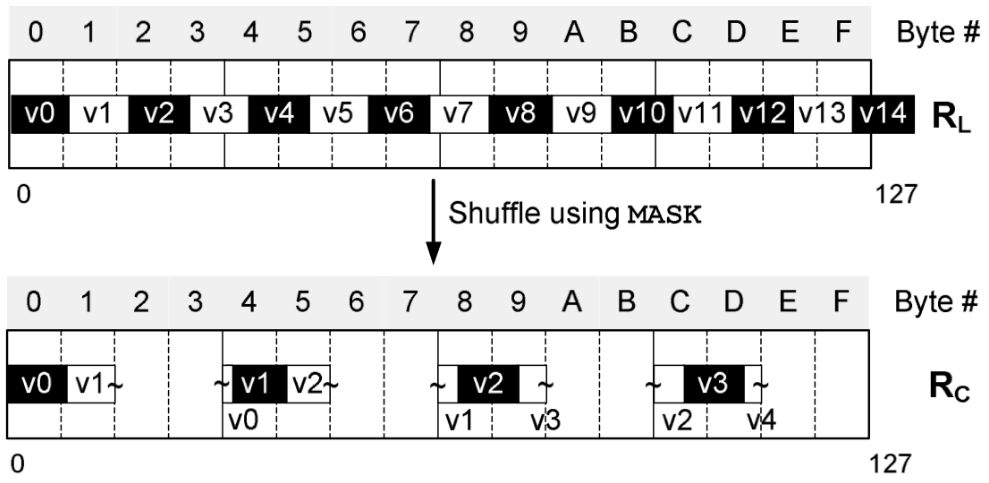
1) Partitioned Bitvector





2) Vertical Bitvector




0 1 2 3 4 5 6 7 8 9 A B C D E F Byte #
v0 v1 v2 v3 v4 v5 v6 v7 v8 v9 v10 v11 v12 v13 v14
R_L
0
127
Shuffle using MASK
0 1 2 3 4 5 6 7 8 9 A B C D E F Byte #
v0 v1 v1 v2 v0 v2 v1 v3 v3 v2 v4
R_C
0
127

3) Unsorted Dictionaries

Delta Dictionary

U^J_D

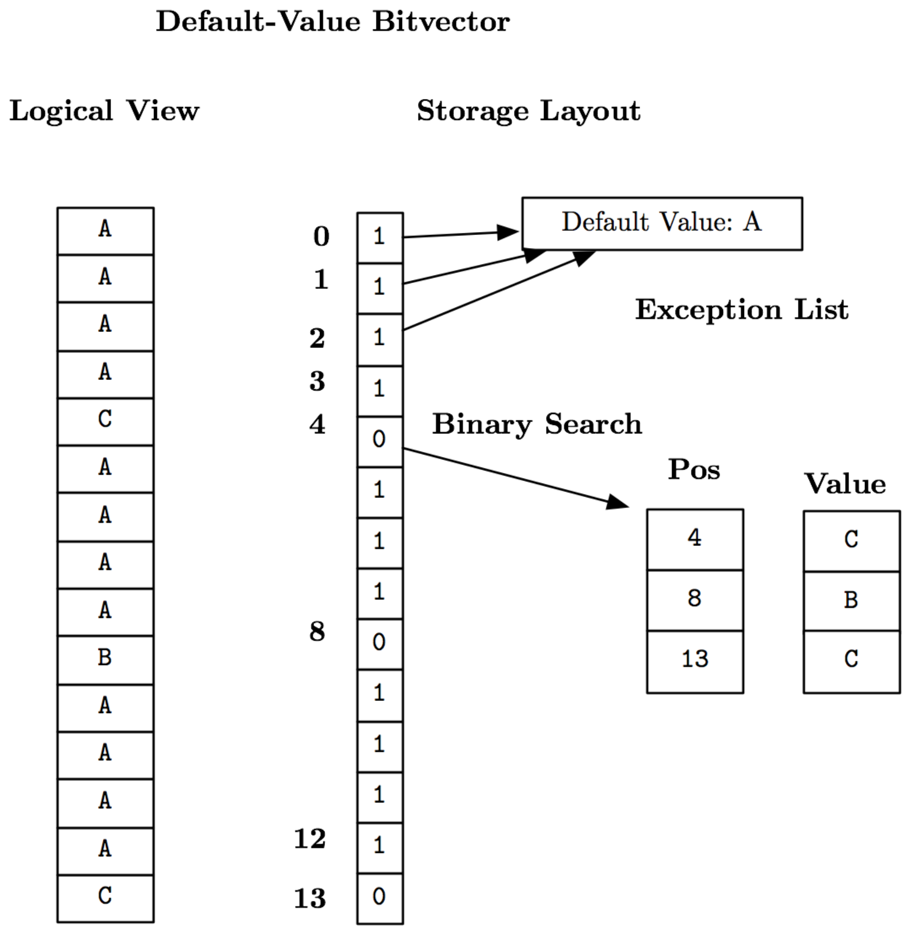
bravo	00
charlie	01
golf	10
young	11

Delta Partition (Compressed)

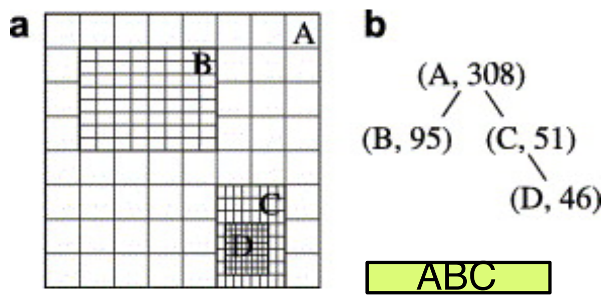
00	bravo
01	charlie
10	golf
01	charlie
11	young

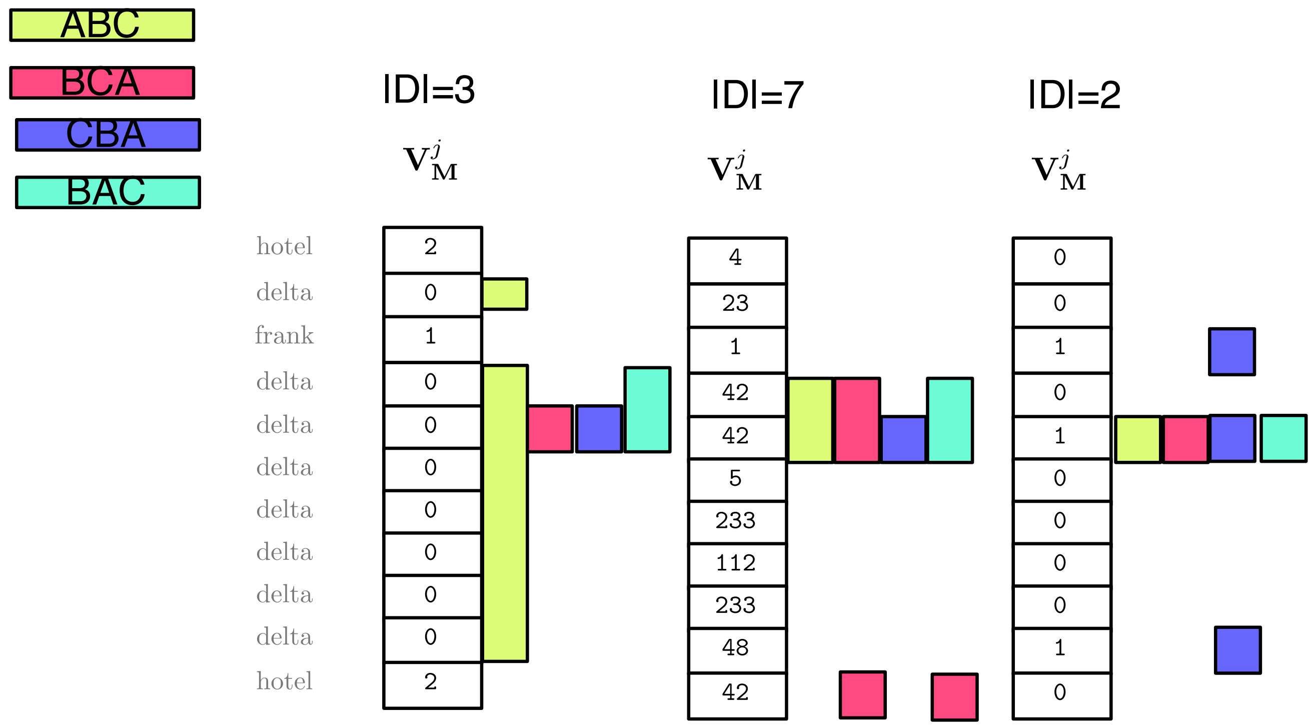
SELECT name FROM table WHERE name > ‘charlie’

4) Default Value Index



5) Multi Column Indices





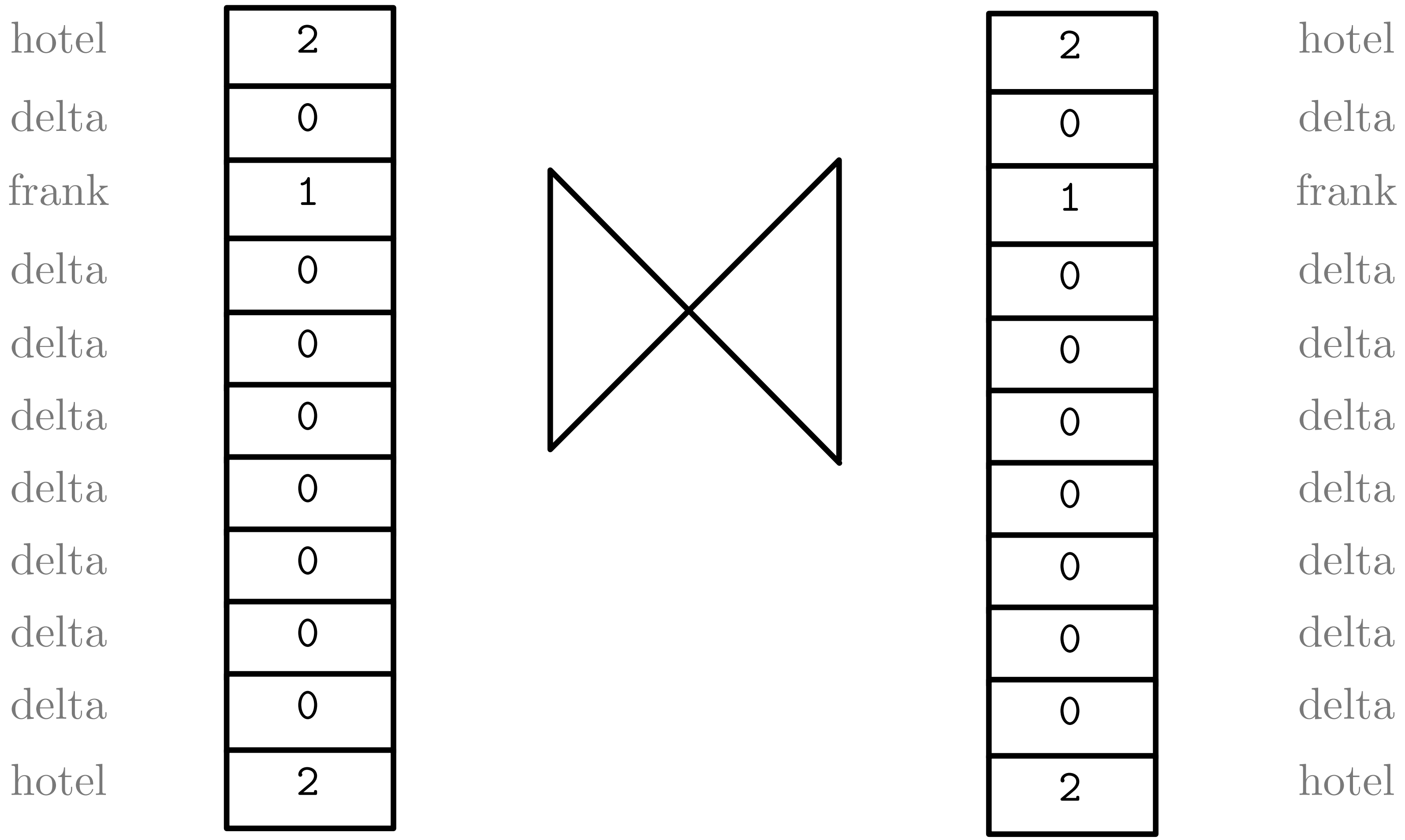
Select * from T where colA='delta' and colB='42' and colC=1

6) Multi Column Join Algorithms

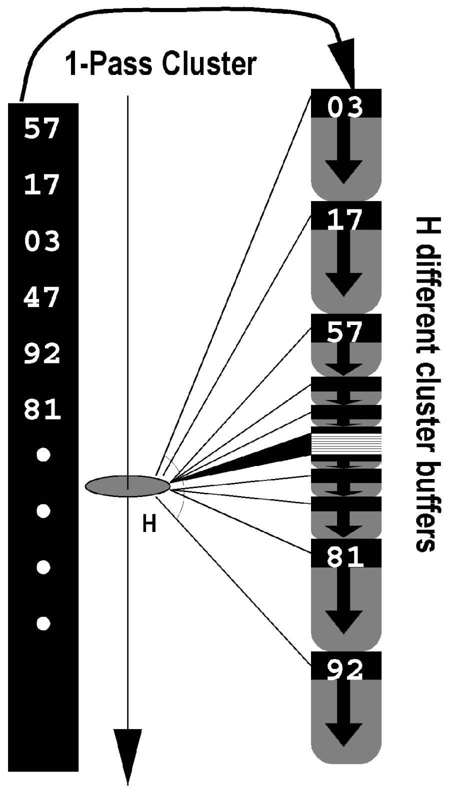
When joining tables with multiple key columns, column stores need to materialize all keys or need an uncompressed tree structure with actual values not value-ids.

- How can column stores perform fast and optimized joins over multiple columns
- Avoid complete materialization, leverage dictionary compression
- Special index structures might be used

7) Join Algorithms on shared Dictionaries



8) Cache-Optimized Parallel Join



traightforward clustering algorithm.

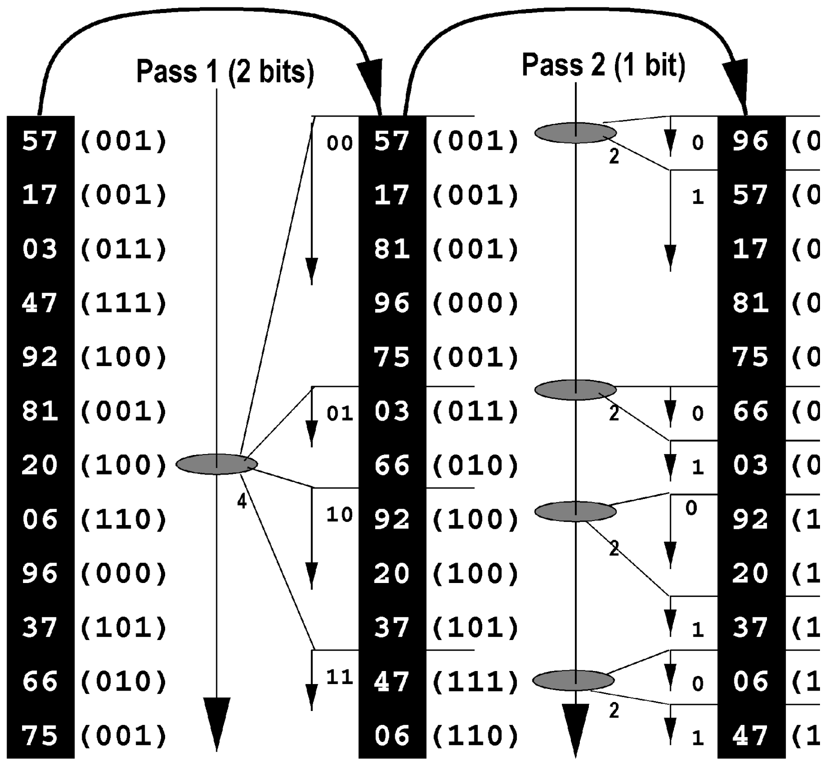
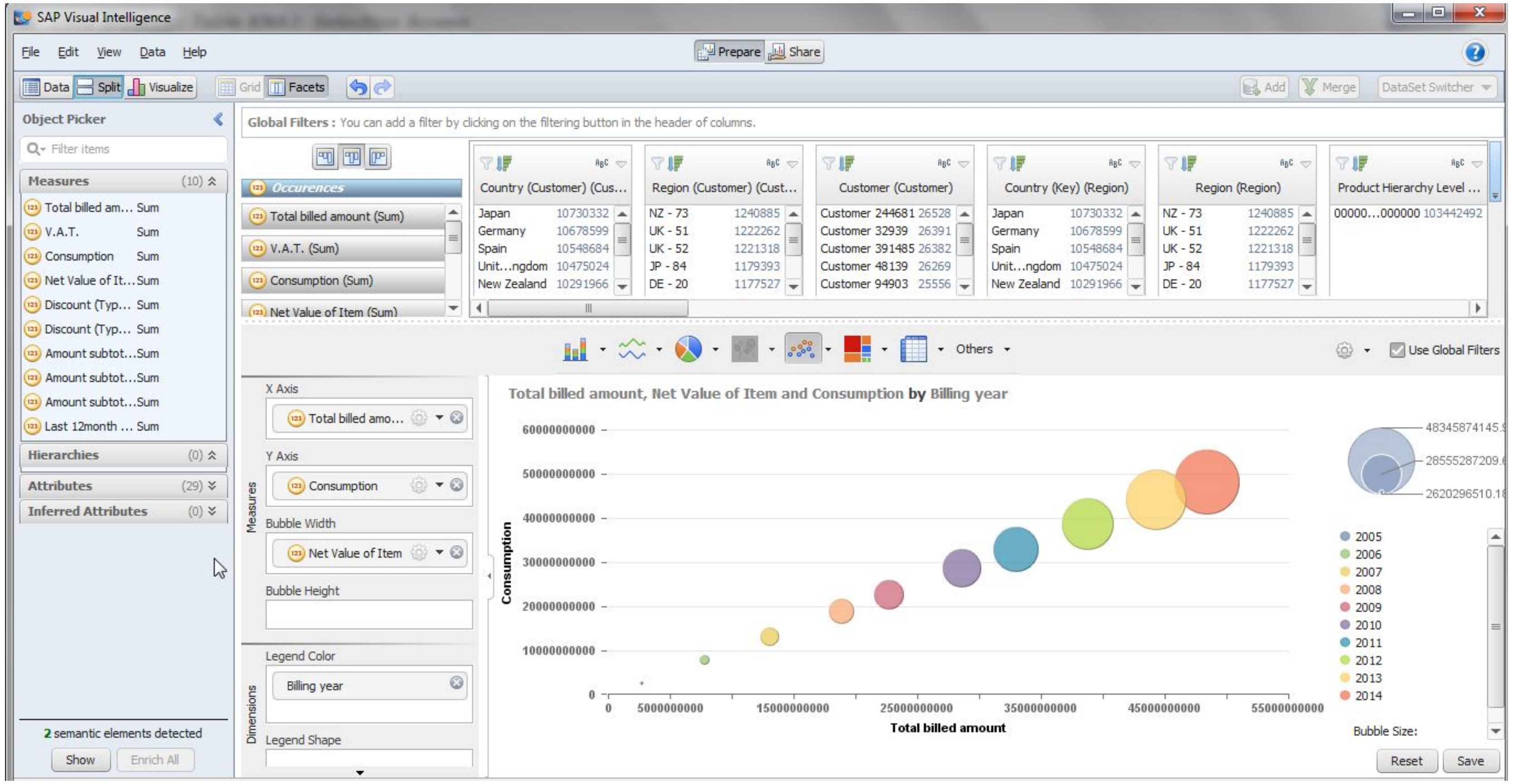


Fig. 9. Two-pass/3-bit radix cluster (lower bits indicated b
parentheses).

From [1] S. Manegold, P. Boncz, M. Kersten, S. Manegold, P. Boncz, and M. Kersten, “Optimizing main-memory join on modern hardware,” IEEE Transactions on Knowledge and Data Eng, 2002.

9) Analytical User Interface for HYRISE



Global Filters : You can add a filter by clicking on the filtering button in the header of columns.

Measures	(10)
Total billed am…	Sum
V.A.T.	Sum
Consumption	Sum
Net Value of It…	Sum
Discount (Typ…	Sum
Discount (Typ…	Sum
Amount subtot…	Sum
Amount subtot…	Sum
Amount subtot…	Sum
Last 12month …	Sum

Hierarchies (0)
Attributes (29)
Inferred Attributes (0)

2 semantic elements detected
Show Enrich All

Total billed amount (Sum)
V.A.T. (Sum)
Consumption (Sum)
Net Value of Item (Sum)

Country (Customer) (Cus…		Region (Customer) (Cust…		Customer (Customer)		Country (Key) (Region)		Region (Region)		Product Hierarchy Level …	
Japan	10730332	NZ - 73	1240885	Customer 244681	26528	Japan	10730332	NZ - 73	1240885	00000…000000	103442492
Germany	10678599	UK - 51	1222262	Customer 32939	26391	Germany	10678599	UK - 51	1222262		
Spain	10548684	UK - 52	1221318	Customer 391485	26382	Spain	10548684	UK - 52	1221318		
Unit…ngdom	10475024	JP - 84	1179393	Customer 48139	26269	Unit…ngdom	10475024	JP - 84	1179393		
New Zealand	10291966	DE - 20	1177527	Customer 94903	25556	New Zealand	10291966	DE - 20	1177527		

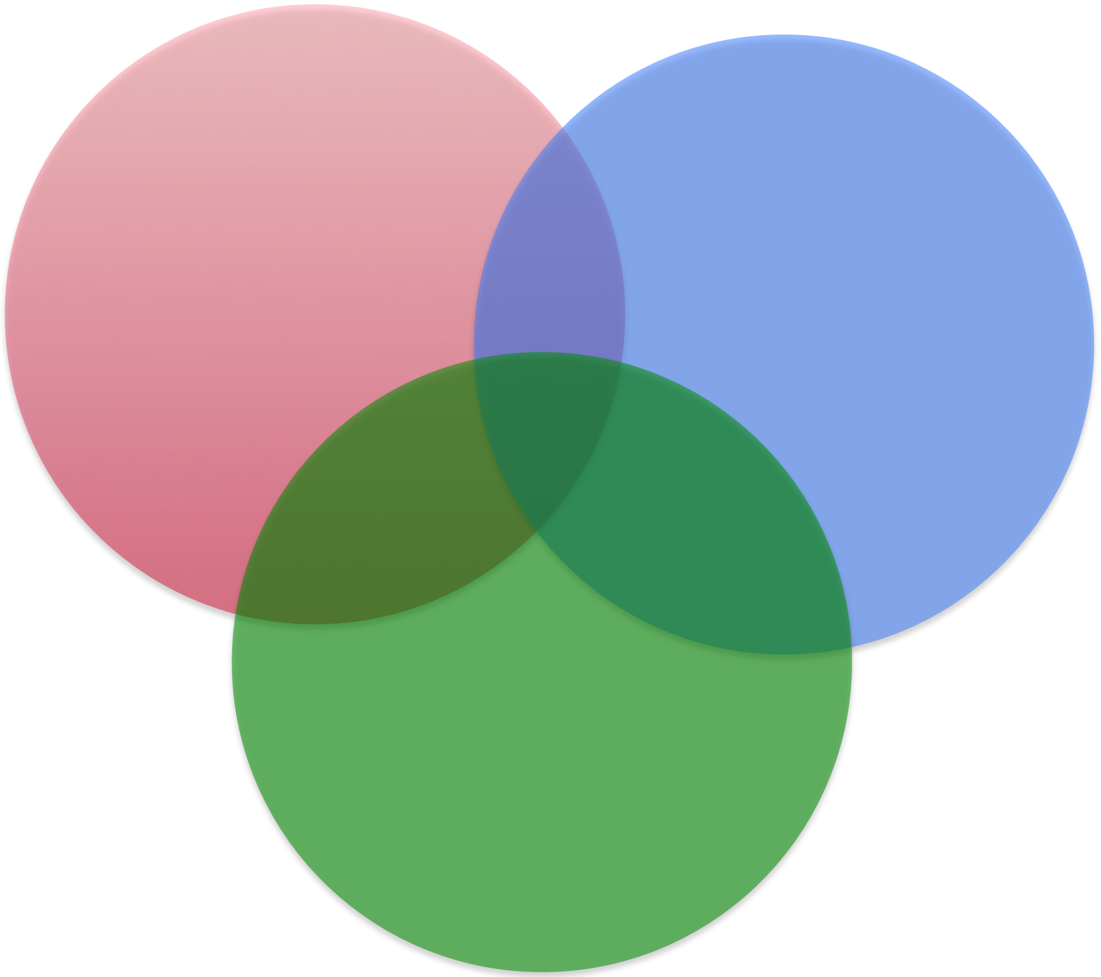
Y axis (Consumption): 0, 10000000000, 20000000000, 30000000000, 40000000000, 50000000000, 60000000000

X axis (Total billed amount): 0, 5000000000, 15000000000, 25000000000, 35000000000, 45000000000, 55000000000

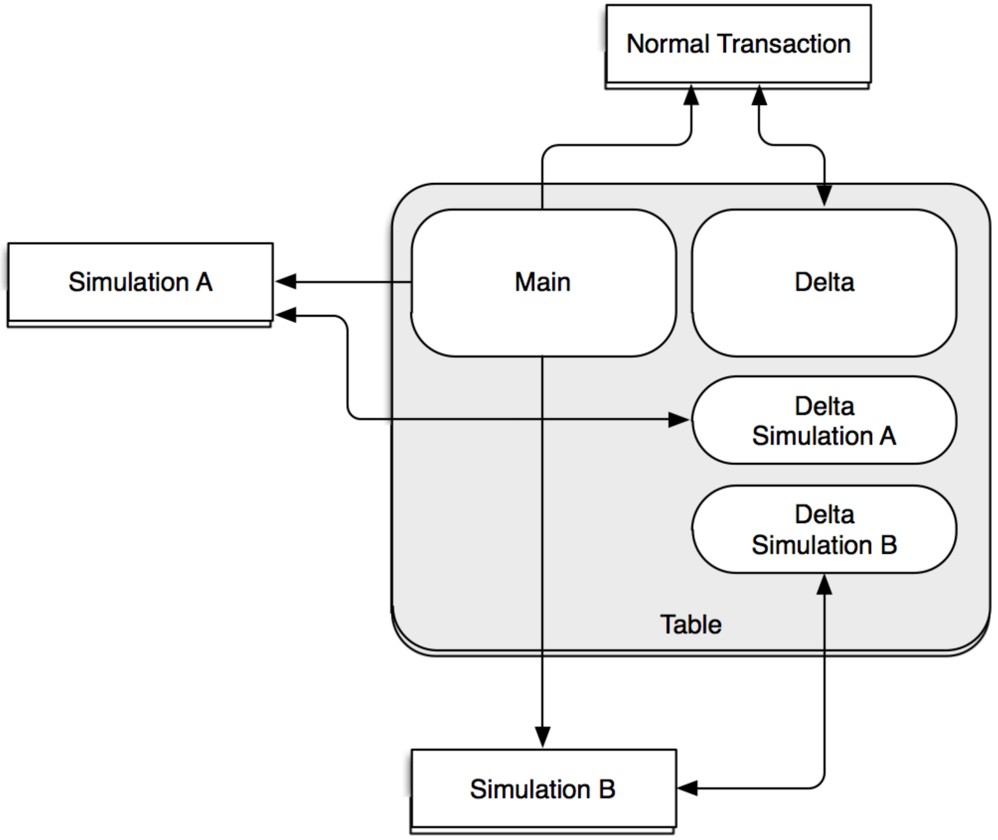
Bubble size: 48345874145.9, 28555287209.6, 2620296510.18

Legend: ● 2005 ● 2006 ● 2007 ● 2008 ● 2009 ● 2010 ● 2011 ● 2012 ● 2013 ● 2014

10) Set-based Query Language



11) Branching Deltas (Simulation)



12) Result Set Compression

Currently, result sets are transferred “as is” to the client, where it is simply read row-by-row.

- In scenarios where the connection between database and applications is limited (mobile, cloud), compressing the result set might lead to significant performance gains
- Standard compression algorithms (like gzip)
- Late materialization on client side

13) Memory Compression Algorithms in Hardware (PPC)

Modern PowerPC processors have hardware supported main memory compression.

- Main memory is separated into two areas – compressed and uncompressed
- When accessing memory area that is compressed, decompression is performed transparently
- Memory compression double available memory size

A first evaluation could investigate how well it compresses enterprise data, how performance is affected, and how a row store compares to a column store.

14) Query runtime prediction based on optimizer results

The idea is to used a cost-based query optimizer to predict query execution time:

- In the seminar, the student should validate whether the HANA query optimizer can be used to roughly predict the query execution time without actually running it
- The objective is to support developers to estimate the runtime of queries during development

Envisioned steps:

- Get to know the query optimizer of HANA
- Experimentally validate the correlation of estimated cost and actual run-time
- Develop a wrapper that estimates runtime for a given query

Topics (vii) - Database Co-Processors

15) Execution model for co-processors

Based on the memory bandwidth limitation of database co-processors, multiple execution models are possible. Evaluation of the following different models: Co-Process, Procedural Language Architecture, GPU hosted data.

16) Co-processor integration into databases

Evaluation how a co-processor can be integrated with a database for application specific logic with SAP HANA (AFL) or HYRISE.

17) Automatic execution unit detection in a heterogeneous system

In a heterogeneous system, the capabilities of database (co-)processors differ with respects to bandwidth, parallelization, clock frequency. To determine the best suited execution unit an execution model needs to be created based on device benchmarks and a programs current workflow.

Topics (viii) - Database Co-Processors

18) Optimized co-processor data structures

Database co-processors used for application specific tasks, such as prediction algorithms or scientific calculations, require the to have a different layout than database tables. The student working on this topic will evaluate which data structures are favorable for different co-processors and algorithms.

19) Automatic code optimization for co-processors

As a heterogeneous programming framework, OpenCL enables the developer to write a single program that can be executed on many different hardware platforms. Nevertheless, the code needs to be optimized for each device to run optimal. The goal will be to optimize the code written in OpenCL C for different devices automaticaly based on micro benchmarks.