In-Memory Columnar Databases (SanssouciuDB)

Big Data Management

Knowledge objectives

1. Justify the viability of in-memory databases
2. Explain the principles of NUMA architecture
3. Enumerate 3 techniques to optimize cache usage
4. Give 4 arguments for columnar storage
5. Explain 3 classical optimization techniques in RDBMS related to column storage
6. Sketch the functional architecture of SanssouciuDB
7. Explain 4 optimizations implemented in SanssouciuDB to improve data access
8. Explain how to choose the best layout
9. Identify the difference between column-stores and NOSQL related to transactions
10. Explain 3 difficulties of pipelining in column-stores
11. Explain 3 problems to implement parallelism in in-memory column-stores
12. Explain 5 query optimization techniques specific of columnar storage

Understanding objectives

1. Given a data setting, justify the choice of either row or column storage
2. Given the data in a column, use run-length encoding with dictionary to compress it

In-Memory management

Some figures

- Hw Offers
  - Memory:

Memory per node	32Gb-100Gb
Number of nodes	20
Total	**640Gb-2Tb**

  - Cost: ~50.000US$

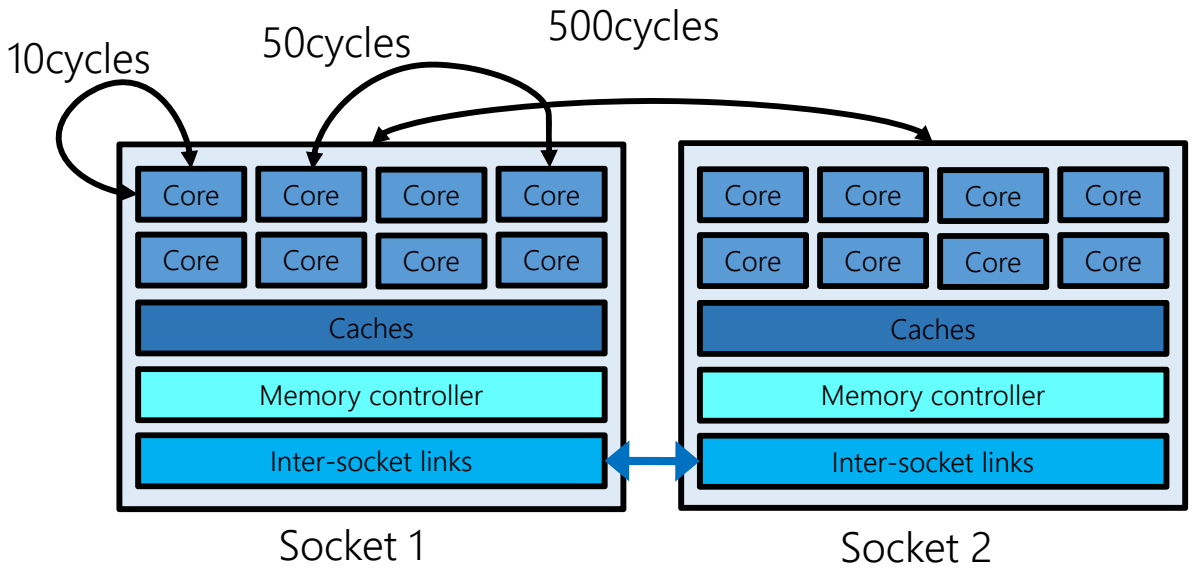
- Sw Demands
  - For TPC-C:

Space per warehouse	100Mb
Number of warehouses	1000
Total	**100Gb**

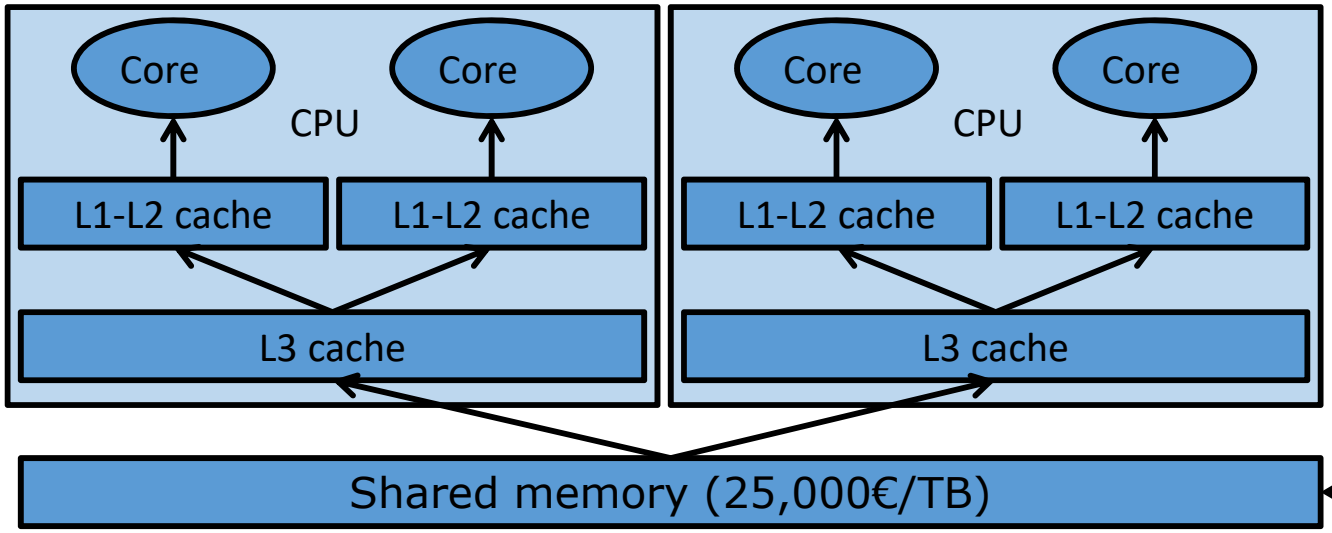
Non-Uniform Memory Architectures (NUMA)

- Exploit data locality
  - Move computation to data
    - Distance matters
    - Resource awareness

facilitate scale up

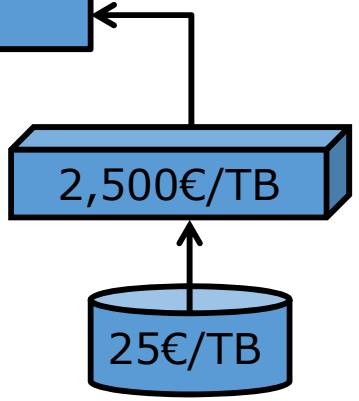


Caches hierarchy



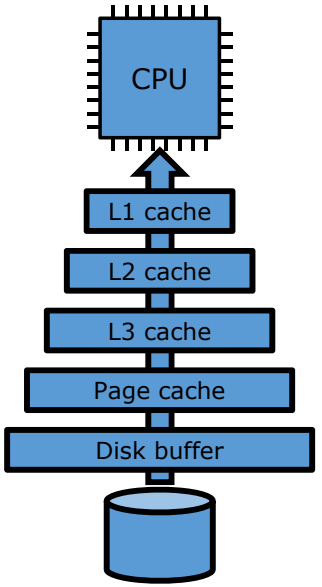
L1 low latency

	Size	Latency	Bandwidth
L1 cache	32 KB	1.5 ns	
L2 cache	256 KB	4 ns	
L3 cache (shared)	30 MB	15 ns	
Main memory (local)	256GB x 2 x 8	60ns-100ns	13-17GB/s
Main memory (neighbour blade)	4TB	500+ ns	3GB/s
Solid State memory	Up to TBs	200,000ns	0.5GB/s
Hard Disk	Up to PBs	10,000,000 ns	0.07GB/s



Cache Optimization Techniques

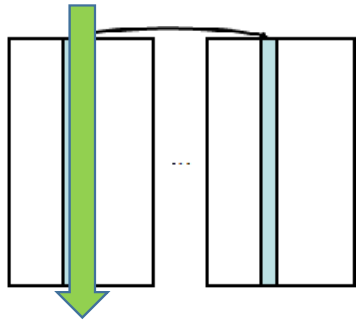
- Locality Awareness
  - Reduce the number of CPU stalls while waiting for memory
  - Two different kinds
    - Spatial
      - Use pre-fetching
        - Promote sequential access
    - Temporal
      - Eviction/Replacement policies (LRU)
- Flexible Caching
  - Bring/Keep relevant data
  - Configure associativity (typically 8-way)
    - Possibilities
      - Direct mapped
      - N-way
      - Fully
    - Low associativity facilitates searching
    - High associativity facilitates replacement policies
- Cache-conscious Programming
  - Use only aligned memory
    - Allocate memory blocks that are aligned to the width of a cache line
      - Padding if necessary
  - Store many fixed size elements consecutively
    - Avoid indirections to find contents (i.e., "next" pointer)



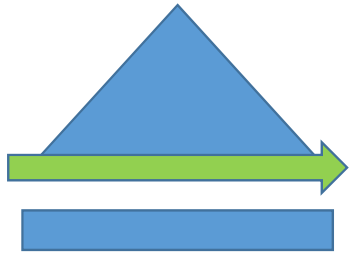
Columnar storage

Typical RDBMSs optimizations

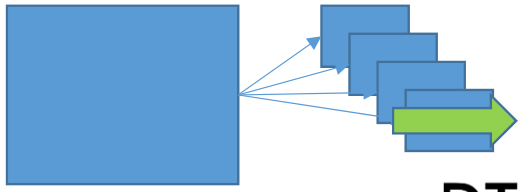
- Vertical partitioning
  - Each table splits in a set of two-columned partitions (key, attributes)
    - Improves useful read ratio



- Use index-only query plans
  - Create a collection of indexes that cover all columns used in a query
    - No table access is needed



- Use a collection of materialized views such that there is a view with exactly the columns needed to answer each query

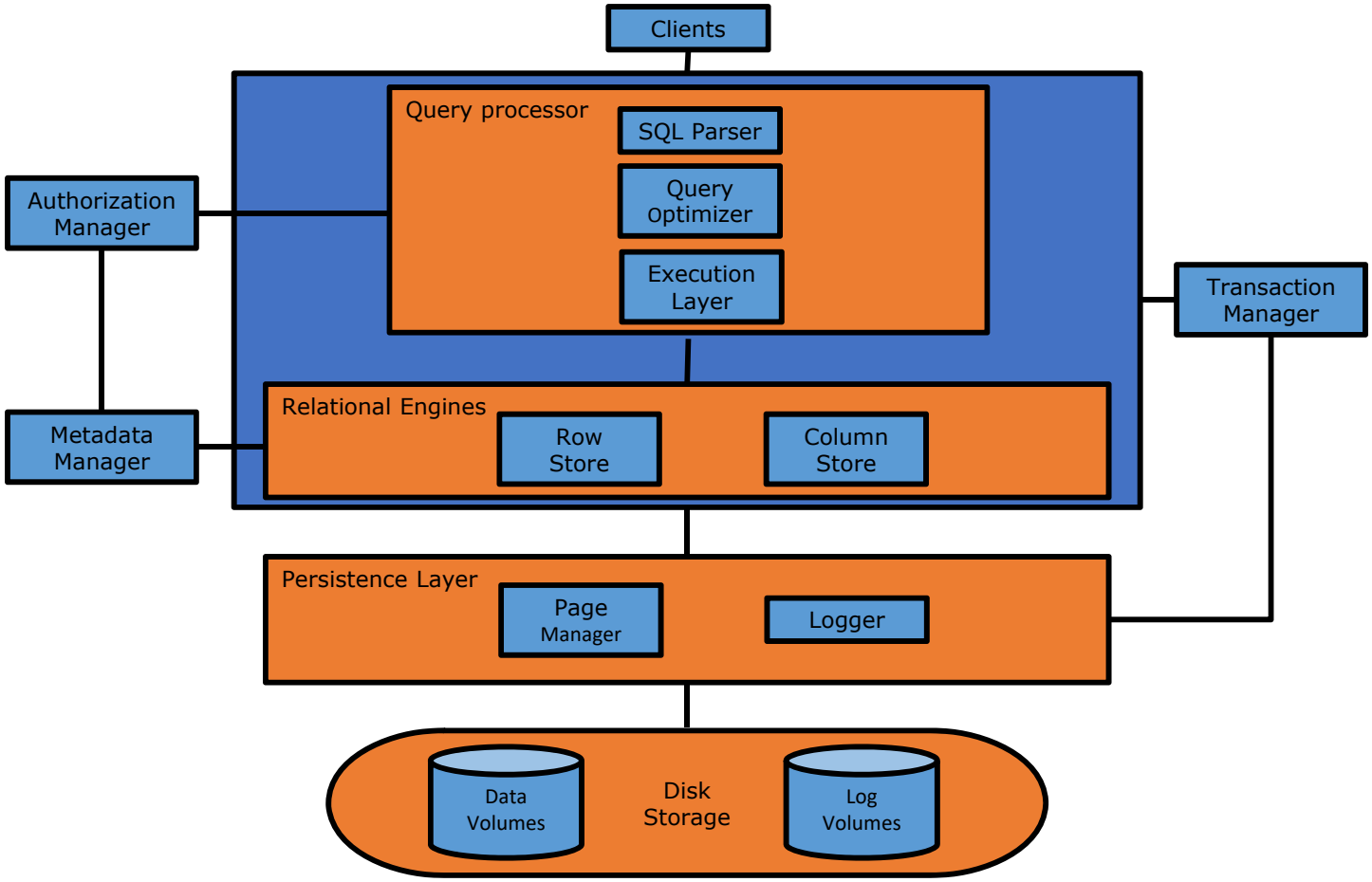


Advantages of columnar storage

- Higher data compression ratios
  - Compressed data can be loaded into CPU cache more quickly
  - With dictionary encoding, the columns are stored as sequences of bit-coded integers
  - Compression can speed up operations such as scans and aggregations if the operator is aware of the compression
- Higher performance for column operations
- Parallelization
- Elimination of additional indexes

3. They do not require managing materialized views nor indexes.
4. ETL is not needed, because both transactional and decisional queries can be served from the same database.

SanssouciuDB architecture



Data access optimizations

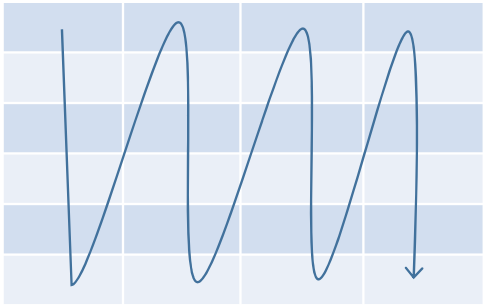
- Use stored procedures
- Data aging by dynamic horizontal partitioning depending on the lifecycle of objects
  - By default only active data is incorporated into query processing
  - The definition of active data is given by the application
- Modifications are performed on a differential buffer
  - Merge process is carried out per table
    - Implies decompressing the table and compressing everything back
    - It is done on-line
- Append-only tables
  - Point representation (i.e., timestamp of the change) for OLTP
  - Interval representation (i.e., valid time of the tuple version) for OLAP

Data Design

Challenge I

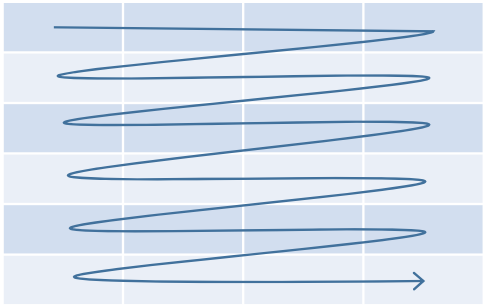
Columnar storage conditions

- Calculations are executed on a single column or a few columns only
- The table is searched based on the values of a few columns
- The table has a large number of columns
- The table has a large number of rows, and columnar operations are required (aggregate, scan, and so on)
- The majority of columns contain only a few distinct values (compared to the number of rows), resulting in higher compression ratios



Row storage conditions

- The table has a small number of rows, such as configuration tables
- The application needs to process only a single record at a time (many selects or updates of single records)
- The application typically needs to access the complete record
- The columns contain mainly distinct values so the compression ratio would be low
- Aggregations and fast searching are not required



Catalog Management

Challenge II

Classical Relational Catalog

- Similar to any Relational system
- Stored row-wise

Transaction Management

Challenge III

Replication management avoided

- Full support for ACID transactions
  - Do not materialize aggregates (only atomic data)
    - No synchronization required
      - Queries always offer the latest updates
  - Exceptional eager distributed replication
    - Only small, static tables exceptionally replicated
      - E.g., Dimension tables in OLAP processing

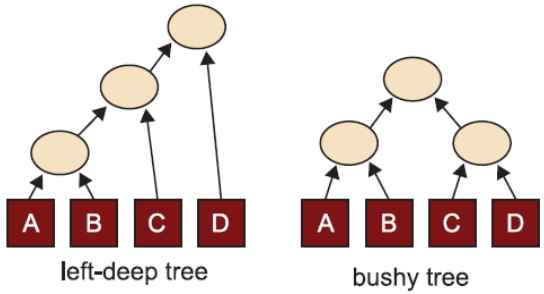
They implement an Eager/Secondary-copy synchronization mechanism.

Query processing

Challenge IV

Parallelism

- Techniques
  - Fragmenting (hybrid)
    - Typical operations benefitting
      - Table scan
      - Aggregation
      - Join
  - Pipelining
    - Difficulties:
      - Short process trees
      - Some operators need all input data at once
      - Skewed cost of operations
- Problems
  - High startup cost
    - One process per core
  - Contention (at Hw level)
    - Use multi-channel memory controllers
  - Skewness
    - Define too fast operations



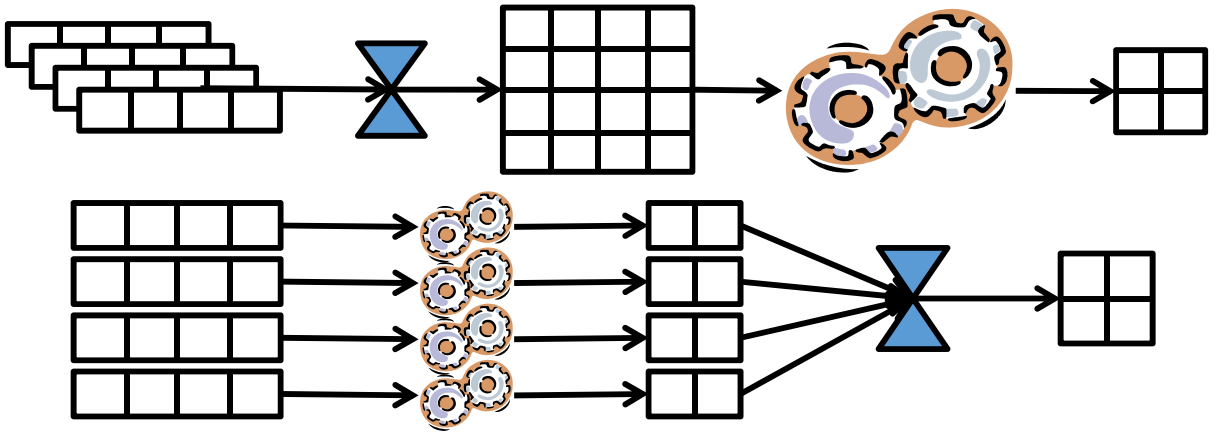
1. Short process trees
2. Some operations need all input data at once Ex: Sort
3. Skewed cost of operations

Column-Oriented Specific Optimizations

- Late materialization
- Tuples are identified by their position
  - PK does not need to be replicated with each column
- Column-specific compression techniques
  - Multiple sorting of data (replication)
    - Not in SanssouciuDB
- Block iteration
  - Known as *vectorized query processing*, when combined with late materialization
- Specific join algorithms

Late Materialization

- *Tuple construction* can be done at the beginning or at the end of the query



- Advantages
  - Some tuples do not need to be constructed (because of selections and projections)
  - Some columns remain compressed more time
  - Cache performance is improved (kept at column level)
  - Helps block iteration for values of fixed length columns

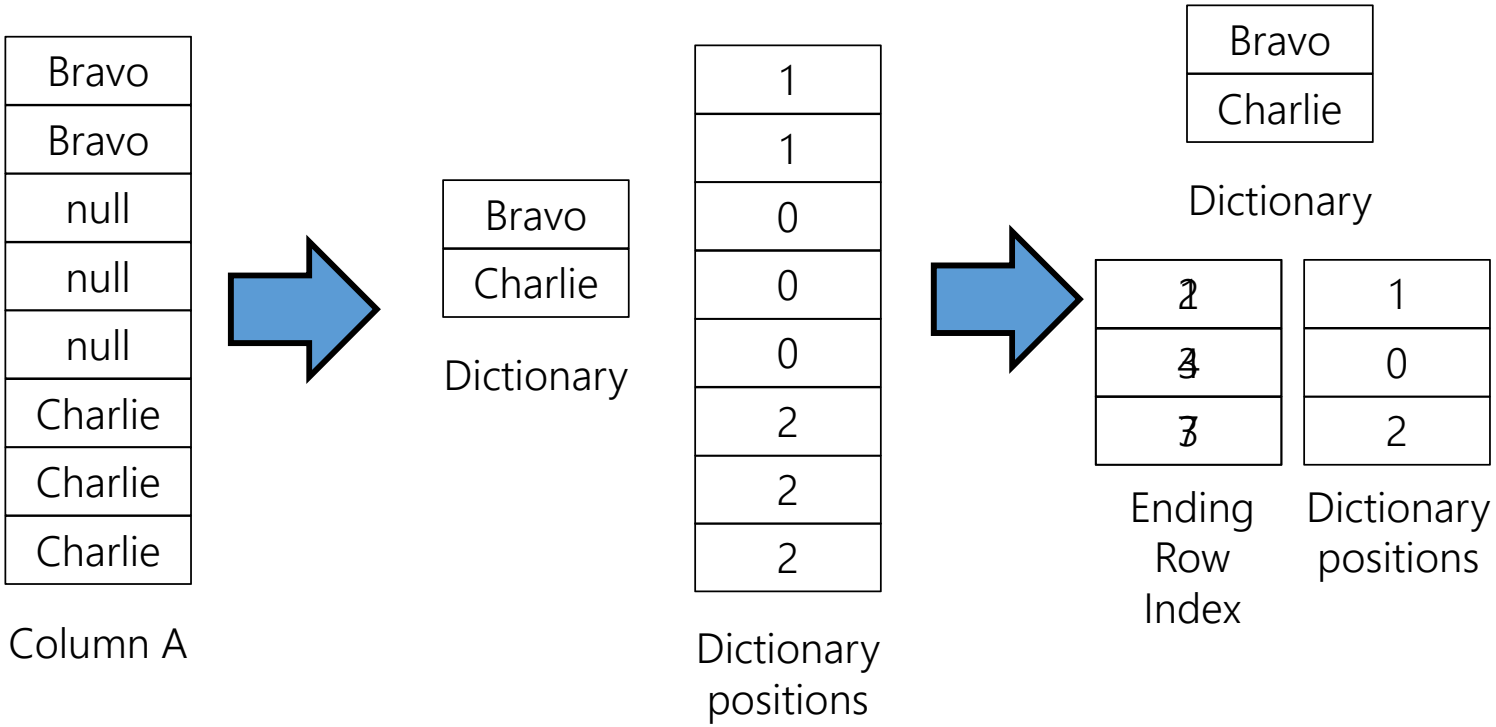
Compression

- Data stored in columns is more compressible than data stored in rows
  - High data value locality (less value entropy)
  - Benefits from sorting
- Main objective is not reducing data space but reducing I/O
- Two main trends
  - Heavyweight compression (e.g., Lempel-Ziv)
    - In general, not that useful
      - It might be if there is a (huge) gap between memory bandwidth and CPU performance
  - Lightweight compression (e.g., Run-Length Encoding)
    - Improves performance by reducing I/O cost
    - May allow the query optimizer work directly on compressed data
      - Decompression is not needed in front of bitwise AND / OR

Examples of light-weight compression

- Values coding
  - Dictionary encoding
- Repetitions coding
  - Common value suppression It is good when data is sparse
    - Sparse coding
  - Cluster coding
  - Run-length encoding
- Memory usage optimization
  - Bit compression
  - Variable byte coding

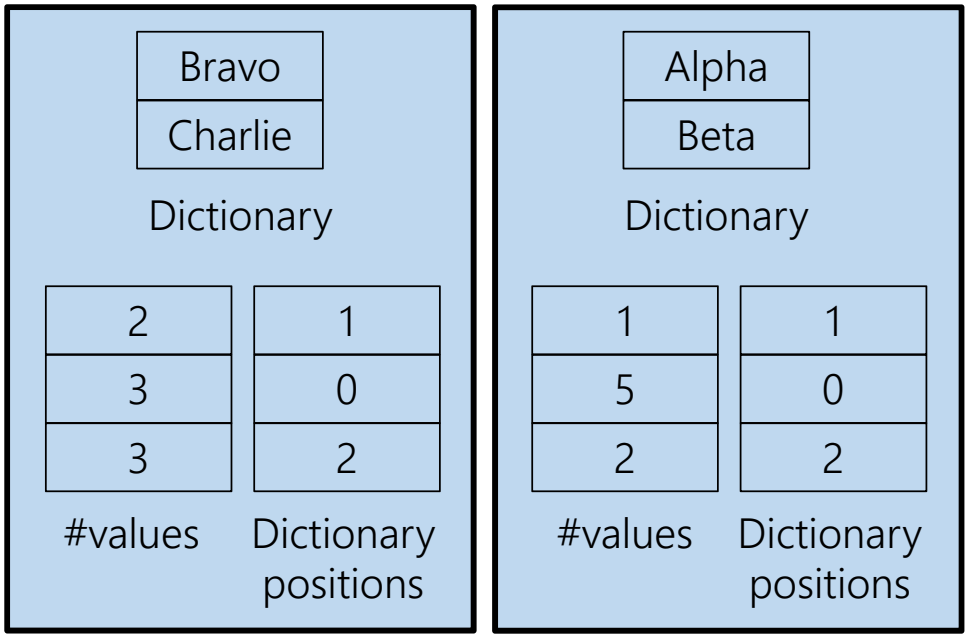
Run-length encoding with dictionary



H. Plattner and A. Zeier

Operating on encoded bit-vectors

- We need to decode
  - Not necessarily the whole vector at once
    - One run at a time



Bravo OR Alpha

Charlie AND NULL

Block Iteration

- Blocks of values of the same column are passed to the next operation in a single function call
- Values inside the block can be:
  - Iterated as in an array (fixed-width)
  - Remain codified (compressed) together
    - Not necessarily using multiples of 8 bits
    - I can count or even identify the tuples for which the predicate is true
  - Exploits parallelism / pipelining

Closing

Summary

- Technical foundations of in-memory column stores
  - Optimizing the usage of memory hierarchies
    - Non-Uniform Memory Architecture (NUMA)
  - Optimizing the data layout
    - Row storage
    - Column storage
    - Hybrid
  - Using parallelism
  - Using compression

References

- H. Plattner and A. Zeier. *In-Memory Data Management*. Springer , 2011
- SAP HANA. *Database for Next-Generation Business Applications and Real-Time Analytics*. White paper, 2012
- D. Abadi, et al. *Column-stores vs. row-stores: how different are they really?* SIGMOD Conference, 2008
- M. Stonebraker et al. C-Store: A Column-oriented DBMS. VLDB, 2005
- G. Copeland and S. Khoshafian. *A Decomposition Storage Model*. SIGMOD Conference, 1985

Resources

- http://developers.sap.com
- http://www.vertica.com
- https://www.monetdb.org
- http://ibmbluhub.com
- http://www.oracle.com/us/corporate/features/database-in-memory-option/index.html