In-Memory Databases

Knowledge objectives

1. Justify the viability of in-memory databases
2. Sketch the functional architecture of SAP HANA
3. Explain three techniques to improve memory usage
4. Explain two techniques to implement parallelism
5. Explain three problems to implement parallelism
6. Explain three typical optimization techniques in RDBMS related to column storage
7. Explain five optimization techniques specific of columnar storage
8. Explain how SAP HANA chooses the best layout
9. Explain four optimizations implemented in SAP HANA to improve data access

Understanding Objectives

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

Some figures

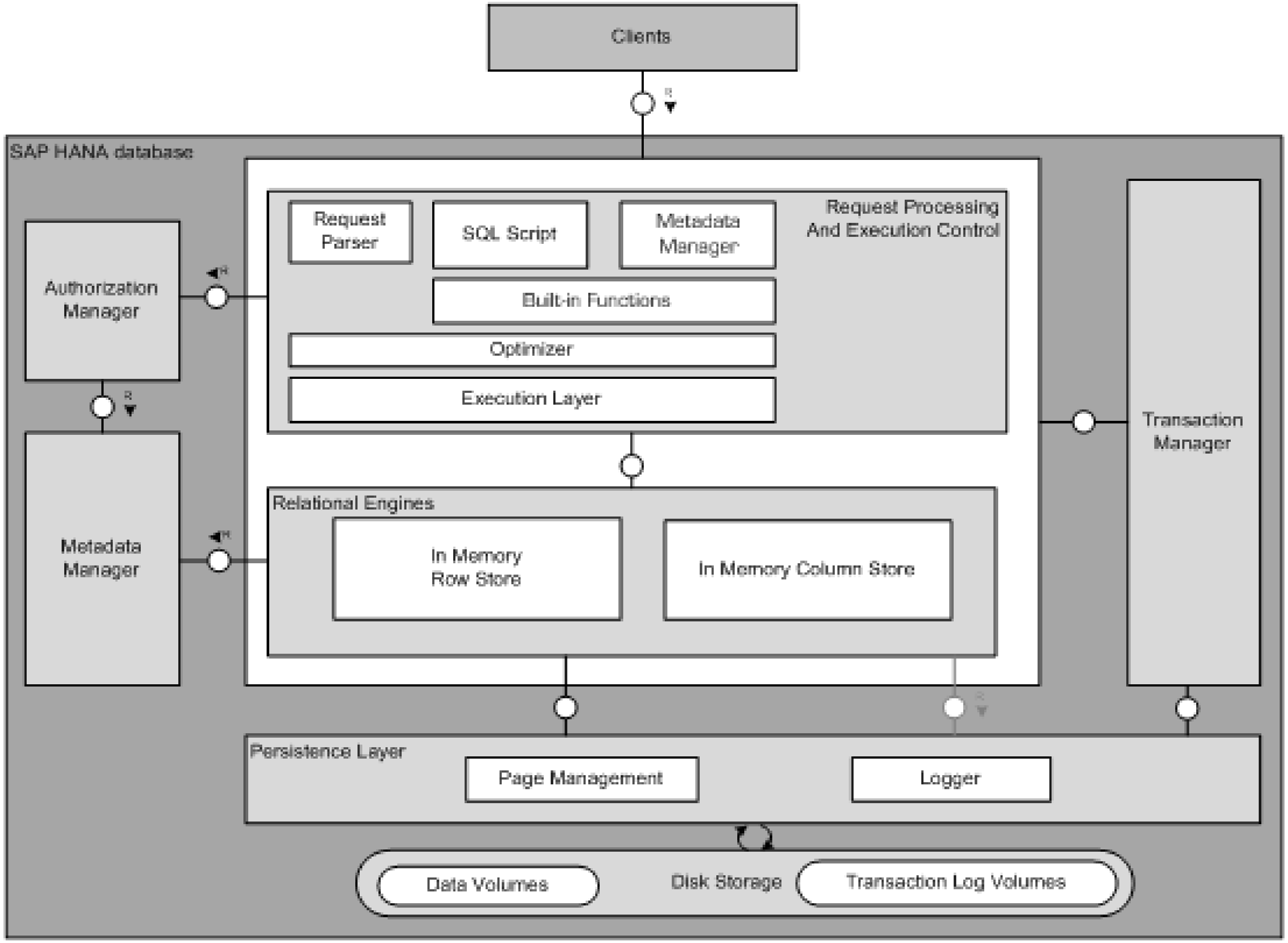
- Hw Offers
  - Memory:

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

  - Cost: Less than 50.000US$
- Sw Demands
  - For TPC-C:

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

SAP HANA architecture



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

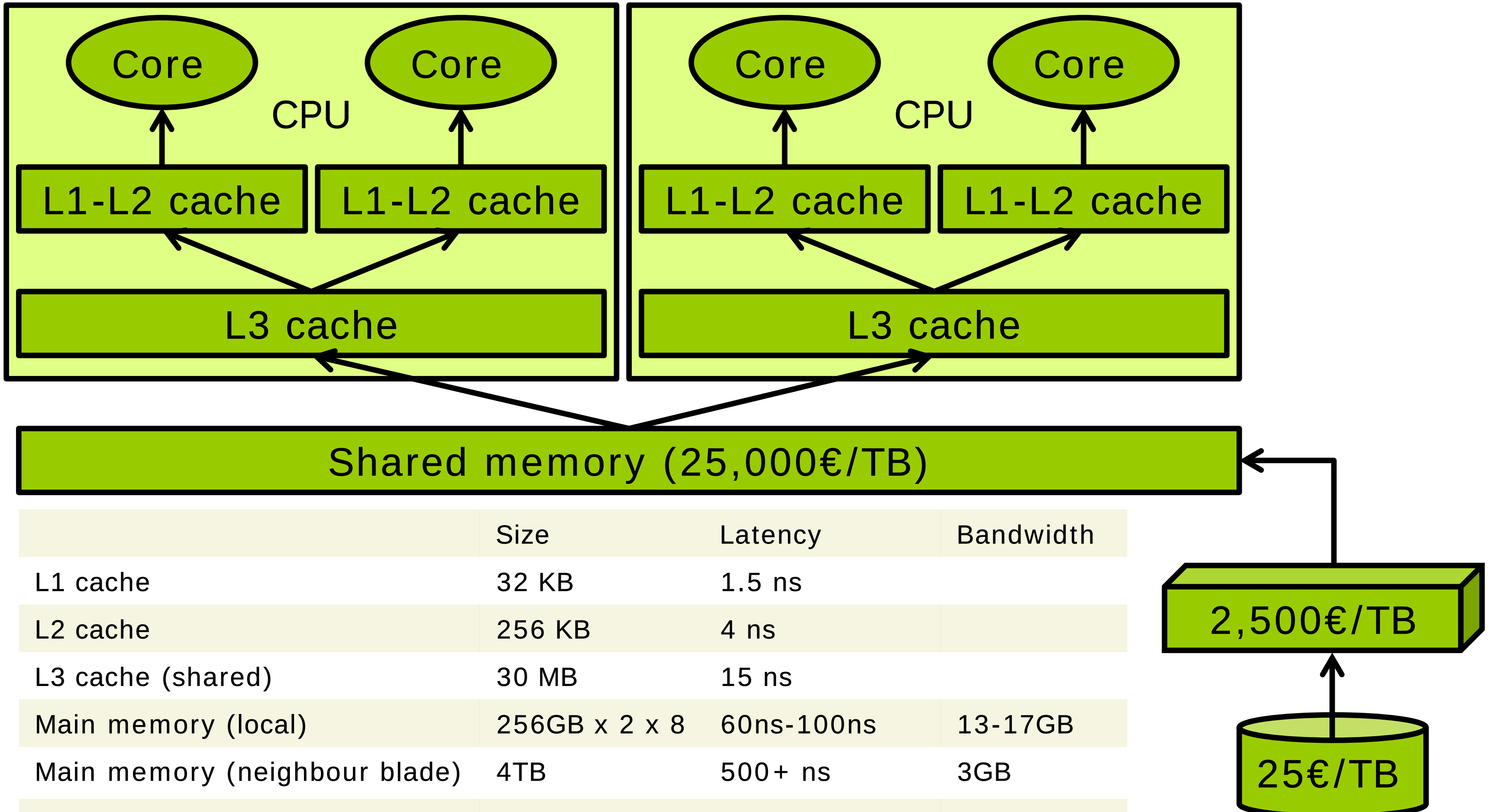
Materialized aggregates are not necessary

- Simplified data model
- Simplified application logic
- Higher level of concurrency
- Contemporaneousness of aggregated values

Technical foundations

- Optimizing the usage of memory hierarchies
- Using parallelism
- Optimizing the data layout
- Using compression
- Virtualizing
  - Limited overhead
    - When reading memory pages, the hypervisor is often not required to be called

Caches hierarchy



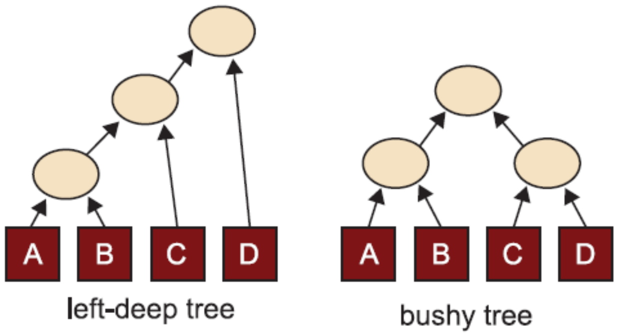
	Size	Latency	Bandwidth
L1 cache	32 KB	1.5 ns	
L2 cache	256 KB	4 ns	
L3 cache (shared)	30 MB	15 ns	
Main memory (local)	256GB x 2 x 8	60ns-100ns	13-17GB
Main memory (neighbour blade)	4TB	500+ ns	3GB
Solid State memory	Up to TBs	200,000ns	0.5GB
Hard Disk	Up to PBs	10,000,000 ns	0.07GB

Memory usage

- Avoid cache misses
  - Bring only relevant data
  - Use associativity (typically 8-way)
    - Possibilities
      - Direct mapped
      - N-way
      - Fully
    - Low associativity facilitates searching
    - High associativity facilitates replacement policies
- Use locality
  - Two different kinds
    - Spatial
      - Use pre-fetching
        - Promote sequential access
    - Temporal
      - Replacement policies (LRU)
  - Reduces the number of CPU stalls while waiting for memory
- Create a cache-conscious design
  - 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)

Parallelism

- Kinds
  - Inter-transaction
  - Intra-transaction
    - Inter-query
    - Intra-query
      - Inter-operation
      - Intra-operation
- Techniques
  - Pipelining
    - Difficulties:
      - Short process trees
      - Some operators need all input data at once
      - Skewed cost of operations
  - Partitioning
    - Typical operations benefitting
      - Table scan
      - Aggregation
      - Join
- Problems
  - High startup cost
    - One process per core
  - Contention (at Hw level)
    - Use multi-channel memory controllers
  - Skew
    - Define fast operations



Column-Oriented Specific Optimizations

- Tuples are identified by their position
  - No PK needed to be replicated with each column
- Specific join algorithms
- Column-specific compression techniques
  - Multiple sorting of data (replication)
    - Not in SAP HANA
- Block iteration
  - When combined with late materialization it is known as *vectorized query processing*
- Late materialization

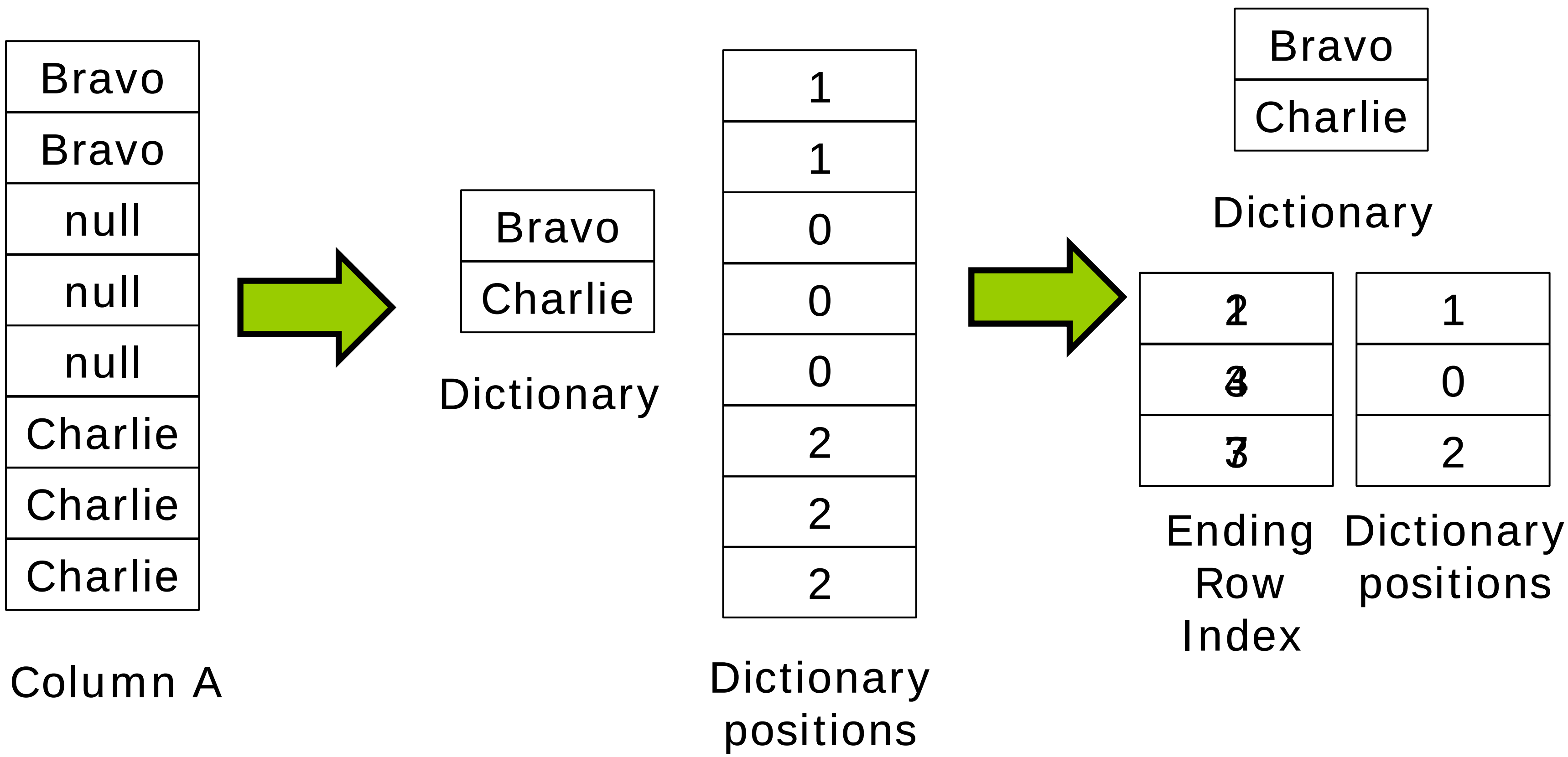
Compression

- Main objective is not reducing data space but reducing I/Os
- Data stored in columns is more compressible than data stored in rows
  - High data value locality (less value entropy)
  - Benefits from sorting
- Two main trends
  - Heavy weight compression (e.g., Lempel-Ziv)
    - In general, not that useful but 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
    - Sparse coding
  - Cluster coding
  - Run-length encoding
- Memory usage optimization
  - Bit compression
  - Variable byte coding

Run-length Encoding with dictionary

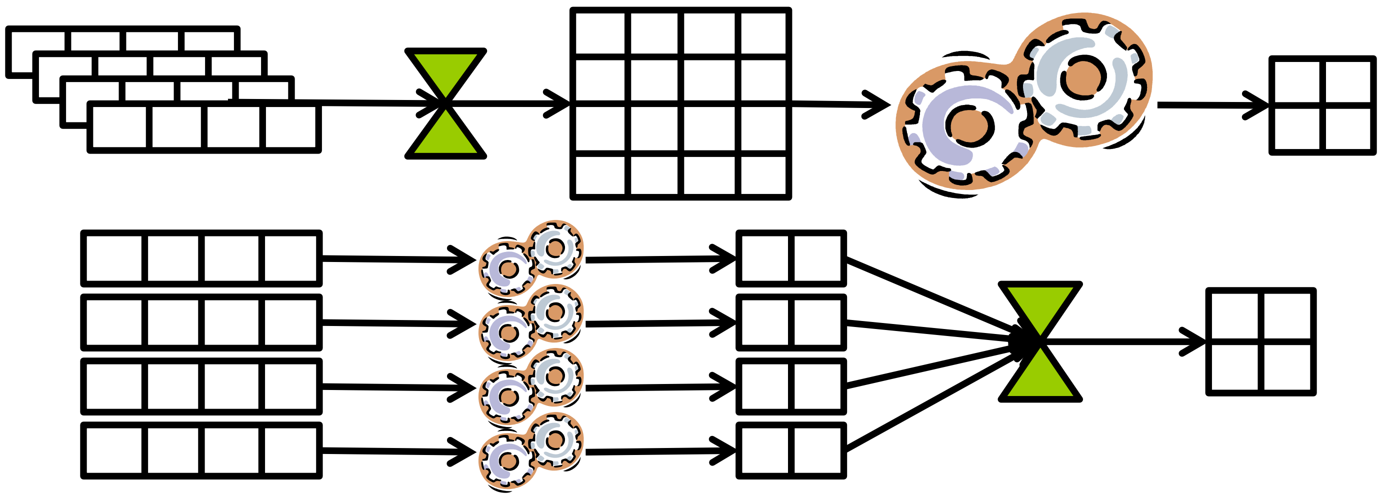


Block Iteration

- 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

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

Advantages of columnar tables

- Higher performance for column operations
- Higher data compression rates
  - Compressed data can be loaded into CPU cache more quickly
  - With dictionary coding, 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
- Elimination of additional indexes
- Parallelization

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 rate would be low
- Aggregations and fast searching are not required

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 rates

Finding the best layout

- Consider hybrid partitioning per table
  - Computational cost is NP-hard
- Needed information
  - Workload
    - Frequency of each query
    - Access plan and cost of each query
      - Take intermediate results and repetitive access into account
  - Value distribution and selectivity of predicates
- Work in three phases
  1. Determine primary partitions (i.e., subsets of attributes always accessed together)
  2. Inspect permutations of primary partitions
  3. Inspect all combinations generated in the previous phase
     1. Generate a disjoint and covering combination
     2. Evaluate its cost

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

Activity

- *Objective: Understand the contribution of in-memory databases*
- *Tasks:*
  1. *(3') Read one use case*
  2. *(5') Explain the use case to the others*
  3. *(5')* Find the main contribution of SAP HANA in all the cases
  4. *Hand in a brief explanation of the contribution*
- *Roles for the team-mates during task 2:*
  a) *Explains his/her material*
  b) *Asks for clarification of blur concepts*
  c) *Mediates and* ***controls time***

Summary

- Technical foundations of SAP HANA
  - Optimizing the usage of memory hierarchies
  - Using parallelism
  - Optimizing the data layout
    - Row storage
    - Column storage
    - Hybrid
  - Using compression
  - Virtualizing
- Data access optimizations in SAP HANA

Bibliography

- 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