1. Changed Hardware
2. Advances in Data Processing
3. Today's Enterprise Applications
4. The In-Memory Data Management for Enterprise Applications
5. Impact on Enterprise Applications
6. Influence on Processes
All Areas have to be taken into account

Our focus

Changed Hardware

Advances in data processing (software)

Complex Enterprise Applications
Why a New Data Management?

- DBMS architecture has **not changed** over decades
- Redesign needed to handle the changes in:
  - Hardware trends (CPU/cache/memory)
  - Changed workloads
  - Data characteristics
  - Data amount
- Some academic prototypes: MonetDB, C-store, HyPer, HYRISE
- Several database vendors picked up the idea and have new databases in place (e.g., SAP, Vertica, Greenplum, Oracle)
Changes in Hardware...

... give an opportunity to re-think the assumptions of yesterday because of what is possible today.

- Multi-Core Architecture (96 cores per server)
- One blade ~$50,000 = 1 Enterprise Class Server
- Parallel scaling across blades
- 64 bit address space
- 2TB in current servers
- 25GB/s per core
- Cost-performance ratio rapidly declining
- Memory hierarchies

Main Memory becomes cheaper and larger
In the Meantime
Research as come up with...

... several advance in software for processing data

■ Column-oriented data organization
  (the column-store)
  □ **Sequential** scans allow best bandwidth utilization between CPU cores and memory
  □ **Independence** of tuples within columns allows easy partitioning and therefore parallel processing

■ Lightweight Compression
  □ Reducing data amount, while..
  □ Increasing processing speed through late materialization

■ And more, e.g., parallel scan/join/aggregation
Memory Hierarchy

Nehalem Quadcore

Core 0

Core 1

Core 2

Core 3

L3 Cache

L2

L1

TLB

QPI

Main Memory

L1 Cacheline

L2 Cacheline

L3 Cacheline

Memory Page

Nehalem Quadcore

Core 1

Core 2

Core 3

Core 0

L3 Cache

L2

L1

TLB

QPI

Main Memory
Two Different Principles of Physical Data Storage: Row- vs. Column-Store

- **Row-store:**
  - Rows are stored consecutively
  - Optimal for row-wise access (e.g. *)

- **Column-store:**
  - Columns are stored consecutively
  - Optimal for attribute focused access (e.g. SUM, GROUP BY)

- **Note:** concept is independent from storage type
  - But only in-memory implementation allows fast tuple reconstruction in case of a column store
OLTP- and OLAP-style Queries Favor Different Storage Patterns

SELECT *  
FROM Sales Orders  
WHERE Document Number = '95779216'

SELECT SUM(Order Value)  
FROM Sales Orders  
WHERE Document Date > 2009-01-20
Motivation for Compression in Databases

- Main memory access is the bottleneck
- Idea: Trade CPU time to compress and decompress data
- Lightweight Compression
  - **Lossless**
  - Reduces I/O operations to main memory
  - Leads to less cache misses due to more information on a cache line
  - Enables operations directly on compressed data
  - Allows to offset by the use of fixed-length data types
Lightweight Dictionary Encoding for Compression and Late Materialization

- Store distinct values once in separate mapping table (the dictionary)
- Associate unique mapping key (valueID) for each distinct value
- Store valueID instead of value in attribute vector
- Enables offsetting with bit-encoded fixed-length data types

**Table**

<table>
<thead>
<tr>
<th>Recld</th>
<th>Company Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAN</td>
<td>INTEL</td>
<td>€1</td>
</tr>
<tr>
<td>FEB</td>
<td>ABB</td>
<td>€2</td>
</tr>
<tr>
<td>MAR</td>
<td>HP</td>
<td>€2</td>
</tr>
<tr>
<td>APR</td>
<td>INTEL</td>
<td>€4</td>
</tr>
<tr>
<td>MAY</td>
<td>IBM</td>
<td>€5</td>
</tr>
<tr>
<td>JUN</td>
<td>IBM</td>
<td>€5</td>
</tr>
<tr>
<td>JUL</td>
<td>SIEMENS</td>
<td>€4</td>
</tr>
<tr>
<td>AUG</td>
<td>INTEL</td>
<td>€3</td>
</tr>
</tbody>
</table>

**Attribute Vector**

<table>
<thead>
<tr>
<th>Recld</th>
<th>ValueId</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

**Dictionary**

<table>
<thead>
<tr>
<th>ValueId</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABB</td>
</tr>
<tr>
<td>2</td>
<td>HP</td>
</tr>
<tr>
<td>3</td>
<td>IBM</td>
</tr>
<tr>
<td>4</td>
<td>Intel</td>
</tr>
<tr>
<td>5</td>
<td>SIEMENS</td>
</tr>
</tbody>
</table>

**Inverted Index**

<table>
<thead>
<tr>
<th>ValueId</th>
<th>RecldList</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1,4,8</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>

**Typical compression factor for enterprise software 10**
**In financial applications up to 50**
Data Modifications in a Compressed Store

- Differential Store: two separate in-memory partitions
  - Read-optimized main partition (ROS)
  - Write-optimized delta partition (WOS)
- Both represent the current state of the data
- WOS/Delta as an intermediate storage for several modifications
- Re-compression costs are shared among all recent modifications (merge process)

Insert/Update

Select (union)

WOS

Merge Process (asynchronously)

ROS
■ Enterprise applications have evolved: not just OLTP vs. OLAP
  □ Demand for real-time analytics on transactional data
  □ High throughput analytics ➔ completely in memory

■ Examples
  □ **Available-To-Promise Check** – Perform real-time ATP check directly on transactional data during order entry, without materialized aggregates of available stocks.
  □ **Dunning** – Search for open invoices interactively instead of scheduled batch runs.
  □ **Operational Analytics** – Instant customer sales analytics with always up-to-date data.

■ Data integration as big challenge (e.g. POS data)
Enterprise Workloads are Read-Mostly

- It is a **myth** that OLTP is write-oriented, and OLAP is read-oriented
- Real world is more complicated than single tuple access, lots of range queries
Enterprise Data is Typically Sparse

- Enterprise data is **wide** and **sparse**
- Most columns are **empty** or have a **low** cardinality of distinct values
- Sparse distribution facilitates high compression

![Bar Chart]

- **% of Columns**
  - 1 - 32: 64%
  - 33 - 1023: 12%
  - 1024 - 10000000: 24%

Legend:
- **Inventory Management**
- **Financial Accounting**
Challenge 1 for Enterprises: Dealing with all Sorts of Data

Transactional Data Entry

Sources: Machines, Transactional Apps, User Interaction, etc.

Event Processing, Stream Data

Sources: machines, sensors, high volume systems

Real-time Analytics, Structured Data

Sources: Reporting, Classical Analytics, planning, simulation

Text Analytics, Unstructured Data

Sources: web, social, logs, support systems, etc.

Data Management

CPUs (multi-Core + Cache + Memory)
... create different application-specific silos with redundant data that reduce real-time behavior & increase complexity.
Historically, OLTP and OLAP system are separated because of resource contention and hardware limitations.

But, this separation has several disadvantages:

- OLAP system does not have the latest data
- OLAP system does only have predefined subset of the data
- Cost-intensive ETL process has to keep both systems in synch
- There is a lot of redundancy
- Different data schemas introduce complexity for applications combining sources
Approach

- Change overall data management system assumption
  - In-Memory only
  - Vertically partitioned (column store)
  - CPU-cache optimized
  - Only one optimization objective – main memory access

- Rethink how enterprise application persistence is build
  - Single data management system
  - No redundant data, no materialized views, cubes
  - Computational application logic closer to the database
    (i.e. complex queries, stored procedures)
Hardware advances
- More computing power through multi-core CPU’s
- Larger and cheaper main memory
- Algorithms need to be aware of the “memory wall”

Software advances
- Column Stores superior for analytic style queries
- Light-weight compression schemes utilize modern hardware

Enterprise applications
- Need to execute complex queries in real-time
- One single source of truth is needed
How does it all come together?

1. Mixed Workload combining OLTP and analytic-style queries
   - **Column Stores** are best suited for analytic-style queries
   - **In-memory** database enables fast tuple re-construction
   - In-memory column store allows aggregation on the fly

2. Sparse enterprise data
   - Lightweight **compression** schemes are optimal
   - Increases query execution
   - Improves feasibility of in-memory database

3. Mostly read workload
   - Read-optimized stores provide best throughput
     - i.e. compressed in-memory column-store
   - Write-optimized store as delta partition to handle data changes is sufficient
In-Memory Database (IMDB)

- Data resides **permanently** in main memory
- Main Memory is the **primary** “persistence”
- Still: logging to **disk**/recovery from **disk**
- Main memory access is the new **bottleneck**
- Cache-conscious algorithms/data structures are **crucial** (locality is king)
## Impact on Application Development

<table>
<thead>
<tr>
<th></th>
<th>Traditional</th>
<th>In-Memory Column-Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application cache</td>
<td></td>
<td>✗</td>
</tr>
<tr>
<td>Materialized views</td>
<td></td>
<td>✗</td>
</tr>
<tr>
<td>Prebuilt aggregates</td>
<td></td>
<td>✗</td>
</tr>
<tr>
<td>Raw data</td>
<td></td>
<td>✗</td>
</tr>
</tbody>
</table>

- Less caches needed
- No redundant objects
- No maintenance of materialized views or aggregates
- Minimized index maintenance
- Data movements are minimized
Impact on Enterprise Applications: Simplified Financials on In-Memory DB

- Only base tables, algorithms, and some indexes
- Reduces complexity
- Lowers TCO
- While adding more flexibility, integration, and functionality
In-memory column stores are better suited as database management system (DBMS) for enterprise applications than conventional DBMS

- In-memory column stores utilizes modern hardware optimally
- Several data processing techniques leverage in-memory only data processing

Enterprise applications show specific characteristics:

- Sparsely filled data tables
- Complex read-mostly workload

Real-world experiences have proven the feasibility of the in-memory column-store
Challenge the status quo

- Don’t ask: Where do we currently have performance problems
- Systems are productive with majority of performance issues addressed through user training, process adaptations, workarounds ...
- People are used to situations and have adapted

- Don’t only think incremental
- Only making things faster will not create the full potential
- Design new processes around the assumption of ‘available information’
Ask indirectly (1/2)

- Where did we define processes around performance
- Where did we train people
- What did we forbid/disable to do
- Where do we have workaround processes
- Where do we have exceptions driven processes deriving from lack of information
- Where do we have the most change requests in reporting
- Which batch processes run multiple times a day
- Where would I love to drill down from reporting into transactions
Where do my users work with the pattern data gathering, intermediate storage in Excel

Where have you simplified the data model

Automation of information worker tasks

Which processes include massive re-work due to data latency

Which processes are too slow

Which processes have not been implemented due to performance requirements & data quantity

Which processes require variations of existing aggregates
Questions?

Martin Faust
Hasso Plattner Institute
Martin.Faust@hpi.uni-potsdam.de