IN-MEMORY DATABASE SYSTEMS
Motivation

• Analytical Processing Today
  – Separation of OLTP and OLAP
Analytical Processing Today

• Separation of OLTP and OLAP: Status Quo
  – No real-time analytics
  – Special materialized data structures (cubes) ⇒ only a particular set of reports can be served by one cube
  – Bad response times
  – No ad-hoc reports
Vision for Analytical Processing

Information in Real Time: Anything, Anytime, Anywhere

• Real-Time Analytics
• Computation on the Fly
• Response time at the Speed of Thought

Online Transaction Processing (OLTP) + Online Analytical Processing (OLAP)
In-Memory Database Systems

• Preface

• Impact of Recent Hardware Trends

• Architecture and Implementation

• Application Architecture

• Trends
In-Memory Database Systems: Preface

• **History**
  – First approaches (called **main memory databases**) in the early 80\textsuperscript{th}
  – Always topic within the database research community
  – Search for “main memory database” on DBLP (http://dblp.uni-trier.de/)
    – back to the early 80\textsuperscript{th} (SIGMOD 1984, VLDB 1986), e.g.
      • DeWitt, Katz, Olken, Shapiro, Stonebraker, Wood: *Implementation techniques for main memory database systems*. SIGMOD 1984
• Literature
  – Lot of research papers (SIGMOD, VLDB, BTW, etc.) for special topics (column store, data compression, operator parallelization, recovery, data layout etc.)
In-Memory Database Systems

• Preface

• Impact of Recent Hardware Trends

• Architecture and Implementation

• Application Architecture

• Trends
Trends in Memory Hierarchies

- Cost/size relation for disk as well as main memory has decreased exponentially
- Jim Gray (2006): “Tape is Dead, Disk is Tape, Flash is Disk, RAM Locality is King”

• In-Memory DBMS = Disk-Based DBMS + Tons of Memory?
Trends in Memory Hierarchies (Cont.)

• Flash as new persistent layer?
  – No efficient update in-place
  – Asymmetric read/write performance, especially bad performance for random writes
    ⇒ Design of new flash-aware algorithms

• Performance bottleneck in the past: Disk I/O Performance
• Performance bottleneck today: CPU waiting for data to be loaded from memory into cache
  ⇒ Padding, (correct) Prefetching, ...
From Maximizing CPU Speed to Multi-Core Processors

- Clock speed, FSB speed, and transistor development

⇒ The Free Lunch Is Over!

Source: Plattner/Zeiger:2012
• Development of number of cores

⇒ Parallel data processing using multi-core necessary
   – Inter-Operator parallelism
   – Intra-Operator parallelism

Source: Plattner/Zeiger:2012
Parallel Data Processing Across Servers

• Scale up
  – Shared memory
  – Shared disk
• Scale out
  – Shared nothing

• Discussion scale up vs. scale out ⇒ chapter NoSQL Database Systems

• Enterprise systems
  – Typically use high-end hardware
    • More reliable, better performance, but at a higher cost
  – Scale up before being scaled out
In-Memory Database Systems

• Preface

• Impact of Recent Hardware Trends

• Architecture and Implementation
  – In-Memory DBMS Architecture
  – Implementation Techniques

• Application Architecture

• Trends
(Traditional) Disk-based DBMS Architecture

- Query Decomposition & Optimization Manager
- Access Manager
- Storage Manager
- Buffer Manager
- Transaction Manager & Lock Manager
- Recovery Manager & Log Manager

DBMS Layer Architecture
Traditional DBMS Architecture

- Harizopoulos/Abadi/Madden/Stonebraker (SIGMOD 2008)
  - Breakdown of instruction count for various DBMS components for the New Order transaction from TPC-C
  - Measured on open source DBMS (Shore)

\[ \Rightarrow \text{M. Stonebraker: It's Time for a Complete Rewrite!} \]
In-Memory DBMS: Architecture Implications

- **Buffer Manager**
  - Not necessary anymore
- **Locking**
  - Use lock-free concurrency protocols (MVCC)
  - Eventual consistency where appropriate
- **Latching**
  - Single-threaded processing instead of multi-threaded?
- **Logging**
  - Still necessary
- **Access Paths/Btrees**
  - Redesign?
DBMS Layer Architecture

Disk-based

- Query Decomposition & Optimization Manager
  - Access Manager
  - Storage Manager
  - Buffer Manager
  - Transaction Manager & Lock Manager
  - Recovery Manager & Log Manager

In-Memory

- Query Decomposition & Optimization Manager
  - Access Manager
  - Storage Manager
  - Transaction Manager & (Lock Manager)
  - Recovery Manager & Log Manager
In-Memory DBMS: Architecture (Example)

- Architecture of SanssouciDB

Source: Plattner:2013
In-Memory DBMS: Architecture (Example)

• Architecture of SAP HANA
In-Memory Database Systems

• Preface

• Impact of Recent Hardware Trends

• Architecture and Implementation
  – In-Memory DBMS Architecture
  – Implementation Techniques
    • Data Layout
    • Differential Buffer

• Application Architecture

• Trends
Column, Row, or Hybrid Data Layout

- **OLTP**
  - Transcational Workload
  ⇒ **Row-store** data layout

- **OLAP**
  - Analytical Workload
  ⇒ **Column-store** data layout (see chapter *Column Store Database Systems*

- **OLTP + OLAP**
  - Transactional + Analytical Workload (Mixed Workload)
  - ??? data layout
Hybrid Data Layout

• First approach
  – Decision on a per table basis

• How to find an optimal hybrid data layout?
  – Based on workloads (analyze operators)
    • Aggregation queries
    • Point and range queries
    • Join queries
    • Inserts and updates
  – Analyze CPU (cache misses)
  – Build cost models to compute “optimal” layout for given workload
Hybrid Data Layout (Cont.)

• More elaborated approach:
  – Use horizontal and vertical partitioning
  – Rösch/Dannecker/Hackenbroch/Färber: A Storage Advisor for HybridStore Databases, VLDB2012
    • implemented in SAP HANA
  – Example: Horizontal Partitioning

![Diagram of data layout](source: Rösch et al:2012)
Hybrid Data Layout (Cont.)

- Use horizontal and vertical partitioning (Cont.)
  - Example: Vertical Partitioning

  
  ![Diagram](image)

  - Horizontal and vertical partitioning can be combined

- Other approach (*partially decomposed storage model*): Pirk et al. *CPU and Cache Efficient Management of Memory-Resident Databases*, ICDE 2013

Source: Rösch et al:2012
Further Implementation Techniques

- **Compression** for Speed and Memory Consumption
  
  ⇒ see chapter *Column Store Database Systems*

- **Reduce Locking Overhead**
  
  – Multiversion Concurrency Control (MVCC)
  
  – Optimistic Concurrency Control

- **Differential Buffer**
  
  ⇒ see *next slides*

- **Insert only**
  
  – append, never delete, to keep the history complete
Differential Buffer: Motivation

• **Inserting** new tuples into a compressed structure can be expensive
  – New values may require a **reorganization** of the dictionary
  – Number of bits required to encode all dictionary values may change
  – Attribute vector has to be **reorganized**

• **Deletion** of tuples is expensive
  – All attribute vectors have to be **reorganized**
  – valuedIDs of subsequent tuples have to be adjusted

⇒ Differential Buffer
Differential Buffer

- **Concept**
  - New values are written to a dedicated differential buffer (delta)
  - Cache sensitive B+ trees (CSB+) user for faster search on delta

![Diagram of world_population with Differential Buffer and Main Store](Source: Plattner:2013)
Differential Buffer

- **Advantage**
  - Insert of new values are faster, because dictionary and attribute vector **does not need to be resorted**

- **Disadvantages**
  - Range select on differential Buffer is expensive, based on unsorted dictionary
  - Differential buffer requires more memory (no compression + CSB+ tree)
Differential Buffer: Tuple Lifetime

Michael Berg moves from Berlin to Potsdam

Table: world_population

<table>
<thead>
<tr>
<th>recli</th>
<th>fname</th>
<th>Iname</th>
<th>gender</th>
<th>country</th>
<th>city</th>
<th>birthday</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Martin</td>
<td>Albrecht</td>
<td>m</td>
<td>GER</td>
<td>Berlin</td>
<td>08-05-1955</td>
</tr>
<tr>
<td>1</td>
<td>Michael</td>
<td>Berg</td>
<td>m</td>
<td>GER</td>
<td>Berlin</td>
<td>03-05-1970</td>
</tr>
<tr>
<td>2</td>
<td>Hanna</td>
<td>Schulze</td>
<td>f</td>
<td>GER</td>
<td>Hamburg</td>
<td>04-04-1968</td>
</tr>
<tr>
<td>3</td>
<td>Anton</td>
<td>Meyer</td>
<td>m</td>
<td>AUT</td>
<td>Innsbruck</td>
<td>10-20-1992</td>
</tr>
<tr>
<td>4</td>
<td>Ulrike</td>
<td>Schulze</td>
<td>f</td>
<td>GER</td>
<td>Potsdam</td>
<td>09-03-1977</td>
</tr>
<tr>
<td>5</td>
<td>Sophie</td>
<td>Schulze</td>
<td>f</td>
<td>GER</td>
<td>Rostock</td>
<td>06-20-2012</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>8 × 10⁹</td>
<td>Zacharias</td>
<td>Perdopolus</td>
<td>m</td>
<td>GRE</td>
<td>Athen</td>
<td>03-12-1979</td>
</tr>
</tbody>
</table>

**Main Store**

**Differential Buffer**

```
UPDATE 'world_population'
SET city='Potsdam'
WHERE fname='Michael' AND Iname='Berg' AND city='Berlin'
LIMIT 1
```

Source: Plattner:2013
Michael Berg moves from Berlin to Potsdam
Table: world_population

<table>
<thead>
<tr>
<th>recid</th>
<th>fname</th>
<th>lname</th>
<th>gender</th>
<th>country</th>
<th>city</th>
<th>birthday</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Martin</td>
<td>Albrecht</td>
<td>m</td>
<td>GER</td>
<td>Berlin</td>
<td>08-05-1955</td>
</tr>
<tr>
<td>1</td>
<td>Michael</td>
<td>Berg</td>
<td>m</td>
<td>GER</td>
<td>Berlin</td>
<td>03-05-1970</td>
</tr>
<tr>
<td>2</td>
<td>Hanna</td>
<td>Schulze</td>
<td>f</td>
<td>GER</td>
<td>Hamburg</td>
<td>04-04-1968</td>
</tr>
<tr>
<td>3</td>
<td>Anton</td>
<td>Meyer</td>
<td>m</td>
<td>AUT</td>
<td>Innsbruck</td>
<td>10-20-1992</td>
</tr>
<tr>
<td>4</td>
<td>Ulrike</td>
<td>Schulze</td>
<td>f</td>
<td>GER</td>
<td>Potsdam</td>
<td>09-03-1977</td>
</tr>
<tr>
<td>5</td>
<td>Sophie</td>
<td>Schulze</td>
<td>f</td>
<td>GER</td>
<td>Rostock</td>
<td>06-20-2012</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>8 x 10^9</td>
<td>Zacharias</td>
<td>Perdopolus</td>
<td>m</td>
<td>GRE</td>
<td>Athen</td>
<td>03-12-1979</td>
</tr>
</tbody>
</table>

Differential Buffer

UPDATE `world_population`
SET city='Potsdam'
WHERE fname='Michael' AND lname='Berg' AND city='Berlin'
LIMIT 1

• Problem?
Differential Buffer: Tuple Lifetime (Cont.)

Michael Berg moves from Berlin to Potsdam
Table: world_population

<table>
<thead>
<tr>
<th>recld</th>
<th>fname</th>
<th>Iname</th>
<th>gender</th>
<th>country</th>
<th>city</th>
<th>birthday</th>
<th>valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Martin</td>
<td>Albrecht</td>
<td>m</td>
<td>GER</td>
<td>Berlin</td>
<td>08-05-1955</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Michael</td>
<td>Berg</td>
<td>m</td>
<td>GER</td>
<td>Berlin</td>
<td>03-05-1970</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>Hanna</td>
<td>Schulze</td>
<td>f</td>
<td>GER</td>
<td>Hamburg</td>
<td>04-04-1968</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Anton</td>
<td>Meyer</td>
<td>m</td>
<td>AUT</td>
<td>Innsbruck</td>
<td>10-20-1992</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Ulrike</td>
<td>Schulze</td>
<td>f</td>
<td>GER</td>
<td>Potsdam</td>
<td>09-03-1977</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Sophie</td>
<td>Schulze</td>
<td>f</td>
<td>GER</td>
<td>Rostock</td>
<td>06-20-2012</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>8 × 10⁹</td>
<td>Zacharias</td>
<td>Perdopolus</td>
<td>m</td>
<td>GRE</td>
<td>Athen</td>
<td>03-12-1979</td>
<td>1</td>
</tr>
</tbody>
</table>

Main Store

Differential Buffer

```
UPDATE 'world_population'
SET   city='Potsdam'
WHERE fname= "Michael" AND Iname= "Berg" AND city= "Berlin"
LIMIT 1
```
Merge: Motivation

- All write operations (insert, update, delete) are stored within a differential buffer first
- Read-operations on differential buffer are more expensive than on main store
- Outdated tuples values in main store are invalidated

⇒ To guarantee optimal performance the differential buffer **needs to be merged periodically into the main store**
## Differential Buffer: Example

### Main Store

<table>
<thead>
<tr>
<th>valueID</th>
<th>fname</th>
<th>city</th>
<th>reclID</th>
<th>fname</th>
<th>city</th>
<th>valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Albert</td>
<td>Berlin</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Michael</td>
<td>London</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Nadja</td>
<td></td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

### Differential Buffer

<table>
<thead>
<tr>
<th>valueID</th>
<th>fname</th>
<th>city</th>
<th>reclID</th>
<th>fname</th>
<th>city</th>
<th>valid</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source: Plattner:2013
Differential Buffer: Example

Michael moves from London to Berlin

### Main Store

<table>
<thead>
<tr>
<th>valueID</th>
<th>Dictionaries</th>
<th>Attribute Vectors</th>
<th>Validity Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fname</td>
<td>city</td>
<td>recID</td>
</tr>
<tr>
<td>0</td>
<td>Albert</td>
<td>Berlin</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>Michael</td>
<td>London</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Nadja</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

### Differential Buffer

<table>
<thead>
<tr>
<th>valueID</th>
<th>Dictionaries</th>
<th>Attribute Vectors</th>
<th>Validity Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fname</td>
<td>city</td>
<td>recID</td>
</tr>
<tr>
<td>0</td>
<td>Michael</td>
<td>Berlin</td>
<td>0</td>
</tr>
</tbody>
</table>

Source: Plattner:2013
Differential Buffer: Example (Cont.)

Nadja moves from Berlin to Potsdam

<table>
<thead>
<tr>
<th>Main Store</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dictionaries</strong></td>
</tr>
<tr>
<td>valueID</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Differential Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dictionaries</strong></td>
</tr>
<tr>
<td>valueID</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

Source: Plattner:2013
Michael moves from Berlin to Potsdam

### Main Store

<table>
<thead>
<tr>
<th>valueID</th>
<th>Dictionaries</th>
<th>Attribute Vectors</th>
<th>Validity Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><em>fname</em></td>
<td><em>city</em></td>
<td><em>recID</em></td>
</tr>
<tr>
<td>0</td>
<td>Albert</td>
<td>Berlin</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>Michael</td>
<td>London</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Nadja</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

### Differential Buffer

<table>
<thead>
<tr>
<th>valueID</th>
<th>Dictionaries</th>
<th>Attribute Vectors</th>
<th>Validity Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><em>fname</em></td>
<td><em>city</em></td>
<td><em>recID</em></td>
</tr>
<tr>
<td>0</td>
<td>Michael</td>
<td>Berlin</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>Nadja</td>
<td>Potsdam</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

*Source: Plattner:2013*
### Differential Buffer: Example (Cont.)

**Hanna is added, she lives in Dresden**

#### Main Store

<table>
<thead>
<tr>
<th>valueID</th>
<th>Dictionaries</th>
<th>Attribute Vectors</th>
<th>Validity Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fname</td>
<td>city</td>
<td>recID</td>
</tr>
<tr>
<td>0</td>
<td>Albert</td>
<td>Berlin</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>Michael</td>
<td>London</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Nadja</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

#### Differential Buffer

<table>
<thead>
<tr>
<th>valueID</th>
<th>Dictionaries</th>
<th>Attribute Vectors</th>
<th>Validity Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fname</td>
<td>city</td>
<td>recID</td>
</tr>
<tr>
<td>0</td>
<td>Michael</td>
<td>Berlin</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>Nadja</td>
<td>Potsdam</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Hanna</td>
<td>Dresden</td>
<td>2</td>
</tr>
</tbody>
</table>

**Source:** Plattner:2013
Merge Process Overview

• The merge process is working table wise

• Consist of 3 Phases
  1. Prepare merge
  2. Attribute merge
  3. Commit merge

• Is triggered by
  – Amount of tuples in buffer
  – Cost model, which e.g. takes query cost into account
Online Merge

- Working on data copies allows asynchronous merge
- Very limited interruption due to short lock
- But: At least twice the memory needed!

Source: Plattner:2013
Prepare Merge

Source: Plattner:2013
Attribut Merge

1. **Input**
   - Differential Buffer Attribute
     - Attribute Vector $A_{\text{Diff}}$
     - Dictionary $A_{\text{Diff}}$
   - Main Store Attribute
     - Attribute Vector $A_{\text{Main}}$
     - Dictionary $A_{\text{Main}}$

2. **Attribute Merge**
   - Dictionary $A_{\text{Diff}}$
   - Dictionary $A_{\text{Main}}$
   - Mapping Structure

3. **Output**
   - New Main Store Attribute
     - Attribute Vector $A_{\text{Diff}} + A_{\text{Main}}$
     - Dictionary $A_{\text{Diff}} + A_{\text{Main}}$

Source: Plattner:2013
Commit Merge Phase

1. Starts by acquiring a write lock of the table. This ensures that all running queries are finished before the switch to the new main store including the updated valueIDs takes place.

2. The valid tuple vector that was copied in the prepare phase is applied to the actual vector to mark invalidated tuples.

3. The new main store replaces the original differential buffer as well as the old main store and the memory allocations of both are freed.
• **Principles**
  – Never delete any data
  – Invalidate outdated tuples instead
  – Implemented with valid attributes

• **Advantages**
  – Time travel queries possible
  – Suitable for legal requirements (audit ability)
  – Implicit logging
  – Snapshot isolation and locking is simplified
  – Dictionary cleaning becomes obsolete

• **Disadvantage**
  – Increased memory consumption
In-Memory Database Systems

• Preface

• Impact of Recent Hardware Trends

• Architecture and Implementation
  – In-Memory DBMS Architecture
  – Implementation Techniques

• Application Architecture

• Trends
SAP HANA in a side-car approach

Source: Vey/Krutow:2012
SAP HANA in a side-car approach

Source: Vey/Krutow:2012
SAP HANA as a database for SAP NetWeaver BW

Source: Vey/Krutow:2012
Application Development

- Aggregation on the fly
- Extending data layout without downtime
- Moving business logic into the database?!
Application Development

- Three-tier architecture

Source: Plattner/Zeiger:2012
Application Development

- **Stored Procedures**
  - Well-known concept
  - „newly discovered“
  - Benefits
    - Reduce data transfer between the application and the database server
    - Pre-compilation of queries increases the performance for repeated execution
    - Reuse of business logic

Source: Plattner/Zeiger:2012
Performance and Sizing

- No established Benchmark with mixed OLTP and OLAP workload yet
- Performance example (be careful: provided by database vendor!)

SAP HANA server announcements, June 2014
- SGI: 8 to 32 processors (XEON-E7), 6 to 24 TB memory
- HP: XEON-E7, up to 32 TB memory (later 80+)

Source: SAP:2011
• **SAP HANA** (Appliance)

• **EXASolution** (by EXASOL) – very good performance!

• **VoltDB** (by M. Stonebraker et al.)
  – Targeted for OLTP workload

• …
Integration of In-Memory Technology in Relational Database Systems

- **DB2 10.5 (BLU, Blink Ultimate), April 2013**
  - Processor- and main-memory-optimized
  - Hybrid data layout

*Source: Stolze et al: 2013*
Integration of In-Memory Technology in Relational Database Systems (Cont.)

• Microsoft SQL Server 2014 (Project Hekaton), April 2014
  – Main-memory database engine integrated into SQL Server
  – Targeted for OLTP workload
  – Functionality
    • Convert performance critical tables to Hekaton tables
    • Convert performance critical stored procedures to Hekaton procedure \(\Rightarrow\) compiled to native code (machine code)

Integration of In-Memory Technology in Relational Database Systems (Cont.)

**Microsoft SQL Server 2014 (Project Hekaton) (Cont.)**

- Hash and range indexes exist only in memory
- Avoid global shared data structures
- Latch free
- Lock free
- MVCC
- Optimistic CC
- Logging at end of transaction only

• **Oracle Exalytics In-Memory Machine** (Appliance), October 2012
  – Targeted for BI (OLAP)
  – Flash, RAM, and optimized algorithms

• **Oracle 12c Enterprise Edition**, July 2014
  – Duplicate data column-oriented in main memory

• **PostgreSQL**
  – Third party extensions
    - IMCS (In-Memory Columnar Store)
    - Citus DB
In-Memory Analytics

• Integration of In-Memory approach in data analytic software
  – Teradata Intelligent Memory (2013)
  – SAS High Performance Analytics (2014)
  – ...

• Radical change of traditional data warehouse architecture
Trend: Combination of Scale Out and In-Memory

- Large scale, cluster-based in-memory data analysis
  - Apache Spark
    - https://spark.apache.org/
    - “In-Memory” Hadoop
  - ScaleOut Software
    - http://www.scaleoutsoftware.com/
    - In-Memory Grid Computing
    - ...
In-Memory Database Systems: Conclusion

- Vision: OLTP and OLAP running on same server
  - Information in Real Time: Anything, Anytime, Anywhere

- Benefit from recent hardware trends
  - Steadily declining memory prices
  - Many-core processor

- Changes in architecture and implementation necessary
  - In-Memory DBMS ≠ Disk-Based DBMS + Tons of Memory!

- „Hot“ Topic in research and industry