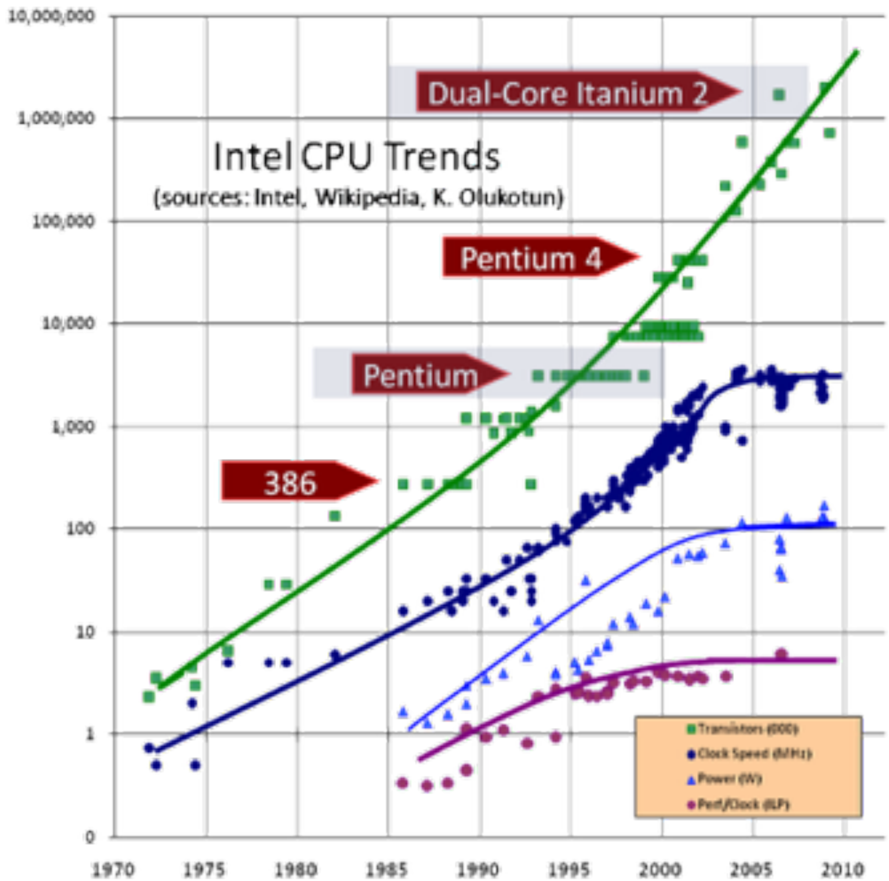
# Advanced Topics On In-Memory Database Servers

Martin Boissier, Carsten Meyer
Martin Faust, David Schwalb

October 2014

# “The Free Lunch Is Over”




– Number of transistors per CPU increases
– Clock frequency stalls



[Source: http://www.gotw.ca/publications/concurrency-ddj.htm]

# Capacity vs. Speed (latency)

- Memory hierarchy:
  – Capacity restricted by price/performance
  – SRAM vs. DRAM (refreshing needed every 64ms)
  – SRAM is very fast but very expensive

➡ **Memory is organized in hierarchies**
- Fast but small memory on the top
- Slow but lots of memory at the bottom

| | technology | latency | size |
|---|---|---|---|
| CPU | SRAM | < 1 ns | bytes |
| L1 Cache | SRAM | ~ 1 ns | KB |
| L2 Cache | SRAM | < 10 ns | MB |
| Main Memory | DRAM | 100 ns | GB |
| Magnetic Disk | | **~ 10 000 000 ns** (10 ms) | **TB** |

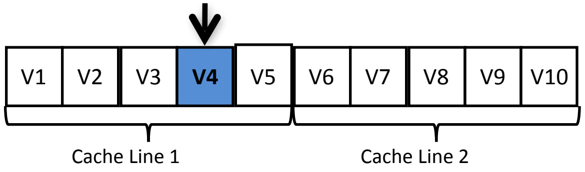
# Data Processing

- In DBMS, on disk as well as in memory, data processing is often:
  - Not CPU bound
  - But bandwidth bound
  - “I/O Bottleneck”

➡ CPU could process data faster

- Memory Access:
  - **Not** truly random (in the sense of constant latency)
  - Data is read in blocks/cache lines
  - Even if only parts of a block are requested

➡ Potential waste of bandwidth



# Memory Hierarchy

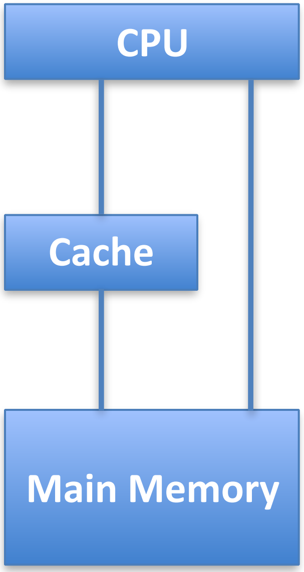
- Cache
  Small but fast memory, which keeps data from main memory for fast access.

➡ **Cache performance is crucial**

- Similar to disk cache (e.g. buffer pool)

**But:** Caches are controlled by hardware.

- Cache hit
  Data was found in the cache.
  Fastest data access since no lower level is involved.
- Cache miss
  Data was not found in the cache. CPU has to load data from main memory into cache (miss penalty).



# Locality is King!

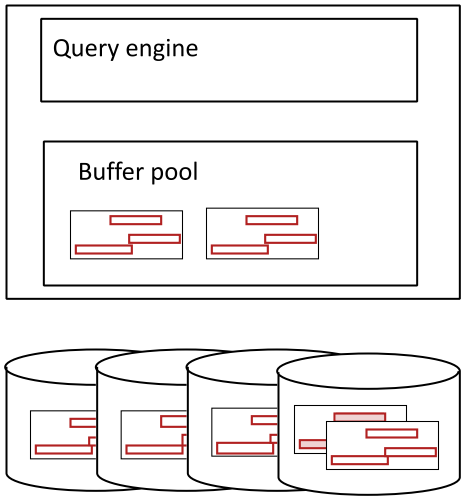
- To improve cache behavior
  - Increase cache capacity
  - Exploit locality
    - Spatial: related data is close (nearby references are likely)
    - Temporal: Re-use of data (repeat reference is likely)
- To improve locality
  - Non random access (e.g. scan, index traversal):
    - Leverage sequential access patterns
    - Clustering data to a cache lines
    - Partition to avoid cache line pollution
      (e.g. vertical decomposition)
    - Squeeze more operations/information into a cache line
  - Random access (hash join):
    - Partition to fit in cache (cache-sized hash tables)

# Motivation

- Hardware has changed
  - TB of main memory are available
  - Cache sizes increased
  - Multi-core CPU’s are present
  - Memory bottleneck increased
  - NUMA (and NUMA on a NUMA?)
- Data/Workload
  - Tables are wide and sparse
  - Lots of set processing
- Traditional databases
  - Optimized for write-intensive workloads
  - Show bad L2 cache behavior

# Problem Statement

- DBMS architecture has **not changed** over decades
- Redesign needed to handle the changes in:
  - Hardware trends (CPU/cache/memory)
  - Changed workload requirements
  - Data characteristics
  - Data amount



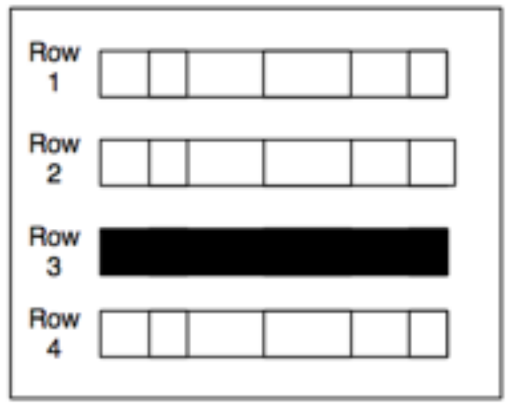

Traditional DBMS Architecture

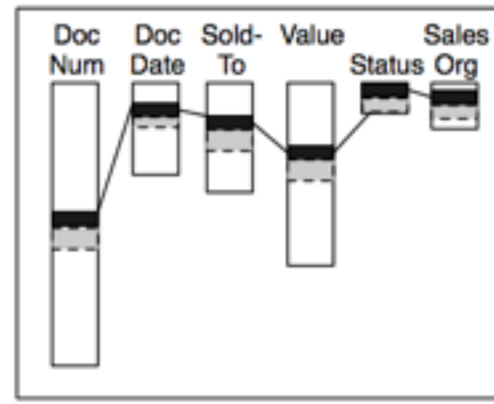
# Row- or Column-oriented Storage

Row Store

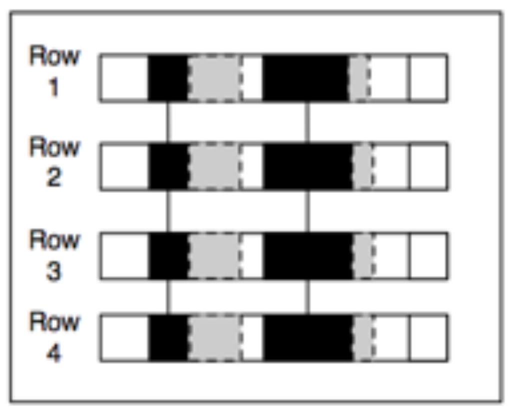
Column Store

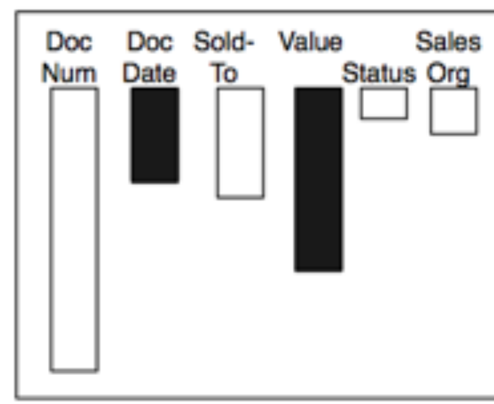
SELECT *
FROM Sales Orders
WHERE Document Number = ‘95779216’




SELECT SUM(Order Value)
FROM Sales Orders
WHERE Document Date > 2009-01-20



# Question & Answer

- How to optimize an IMDB?
  - Exploit sequential access, leverage locality

    -> Column store
  - Reduce I/O
    - Compression
  - Direct value access

    -> Fixed-length (compression schemes)
  - Late Materialization
  - Parallelize

# Seminar Organization

# Objective of the Seminar

- Work on advanced database topics in the context of in-memory databases (IMDB) with regards to enterprise data management
  - Get to know characteristics of IMDBs
  - Understand the value of IMDBs for enterprise computing
- Learn how to work scientifically
  - Fully understand your topic and define the objectives of your work
  - Propose a contribution in the area of your topic
  - Quantitatively demonstrate the superiority of your solution
  - Compare your work to existing related work
  - Write down your contribution so that others can understand and reproduce your results

# Seminar schedule

- Today (14.10.): Overview of topics, general introduction
- Thursday (16.10.): In-memory DB Basics and Topics Q&A (if you're interested)

- 21.10.: Send your priorities for topics to lecturers (martin.boissier@hpi.de)

- Planned Schedule
  - **09./11.12.2014:** Mid-term presentation
  - **10./12.02.2015:** Final presentation (tbc)
  - **20.02.2015:** Peer Reviewing (tbc)
  - **06.03.2015:** Paper hand-in (tbc)

- Throughout the seminar: individual coaching by teaching staff
- Meetings (Room V-2.16)

# Final Presentation

- Why a final presentation?
  - Show your ideas and their relevance to others
  - Explain your starting point and how you evolved your idea /implementation
  - Present your implementation, explain your implementations properties
  - Sell your contribution! Why does your work qualify as rocket science?

# Peer Reviewing

- Each student will be assigned a colleague's paper version (~2 weeks before paper hand-in)
  - Review will be graded
  - Annotate PDF for easy fixes as typos
  - Short summary (2-3 pages in Word) about the paper's content and notes to the author how to further improve his paper
- Expected to be done in the week from Feb 16 to Feb 20

# Final Documentation

- 6-8 pages, IEEE format [1]
- Suggested Content: Abstract, Introduction into the topic, Related work, Implementation, Experiment/ Results, Interpretation, Future Work
- Important!
  - Related work needs to be cited
  - Quantify your ideas / solutions with measurements
  - All experiments need to be reproducible (code, input data) and the raw data to the experiment results must be provided



[1] http://www.ieee.org/conferences_events/conferences/publishing/templates.html

# Grading

- 6 ECTS
- Grading:
  - 30% Presentations (Mid-term 10% / Final 20%)
  - 30% Results
  - 30% Written documentation (Paper)
  - 10% Peer Review

# Topic Assignment

- Each participant sends list of top three topics in order of preference to lecturers by 21.10.
- Topics are assigned based on preferences and skills by 24.10.

# HYRISE

- Open source IMDB
- Hosted at https://github.com/hyrise
- C++ 11
- Query Interface: Query plan or stored procedures

# Recommended Papers for Intro

- Plattner, VLDB 2014: *The Impact of Columnar In-Memory Databases on Enterprise Systems*
- Grund et al. VLDB 2010: *HYRISE—A Main Memory Hybrid Storage Engine*
- Krueger et al. VLDB 2012: *Fast Updates on Read-Optimized Databases Using Multi-Core CPUs*

# Topics

# TPC-(E|C) Workload Analysis

- Project:
  - *Are synthetical standardized benchmarks really that far off the the truth?*
  - [2] examined an enterprise system and found that TPC-C does not reflect the properties very well
  - TPC-E is a successor of TPC-C and appears to be more realistic
  - In parallel, we have an SQL workload trace of a large productive enterprise system (7 TB of data, 50 Million queries)
  - *How do both TPC-* suites and their SQL workload traces compare to a real SQL workload trace of an enterprise system?*

[2] Plattner, The Impact of Columnar In-Memory Databases on Enterprise Systems, VLDB 2014

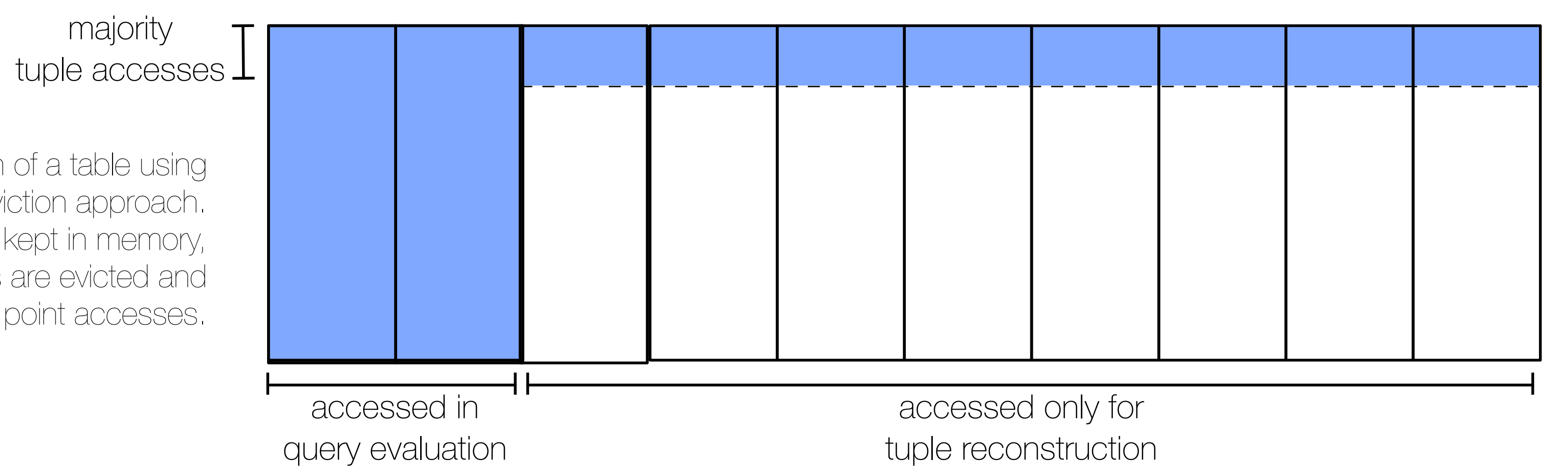
# TPC-(E|C) Workload Analysis

- This project is not a Hyrise project, it's more an analytical challenge to quantify the impacts of certain workloads on IMDBs
- Tasks:
  - Create *TPC-C* and *TPC-E* SQL workload traces
  - Find main characteristics (e.g., characteristics used related work)
  - Analyze each workload thoroughly and compare with the others
- Goal:
  - *Thorough comparison of a productive enterprise system with synthetical enterprise benchmarks*

# Simplified Data Eviction for Hyrise

- **Project:**
  - Workload analyses have revealed that large parts of a database are rarely or never accessed
  - The idea: while retaining the performance superiority of IMDBs, find columns that are never scanned and only accessed for point-accesses
  - These “cold” columns are swapped/evicted to disk
  - *Implement a data eviction strategy with minimal implementation effort and minimal performance impact (i.e., a “simplified data eviction”)*

Exemplary visualization of a table using an simplified eviction approach. Scanned columns are kept in memory, while other columns are evicted and only partially cached for point accesses.



# Simplified Data Eviction for Hyrise

- Tasks:
  - Adapt Hyrise in order to use EMT's `malloc()` to allocate data on disk
  - EMT provides an automated swapping and caching of files ("mmap done right") for their `malloc()` implementation
  - The disk will probably be a prototypical PCIe-connected Phase-Change Memory (PCM) device with 6 microseconds access times (16x faster than PCIe NAND SSD flash)
- Goal:
  - *Evaluate a simplified data eviction strategy and its applicability using a recent prototype of a PCM device*
  - *Further evaluate how much data can be evicted while e.g. retaining 90% of the original performance*

# Integrating Uncompressed Attributes in Hyrise

- Project:
  - Dictionary-Encoding has many advantages for scanning, range queries, and more
  - But one major shortcoming is the materialization of tuples
    - for each attribute to materialize two accesses have to be performed:
      - 1.) access to the attribute vector to get the valueID
      - 2.) lookup of the valueID in the dictionary
  - This overhead can be acceptable using main memory but is a major performance bottleneck for columns allocated on disk
  - *If we do not compress non-scanned columns, how much performance can we gain for materializations (and what is the compression loss)?*

# Integrating Uncompressed Attributes in Hyrise

- Tasks:
  - Build on the already existing (but unfinished) implementation of uncompressed columns in Hyrise
  - Implement missing interfaces & fix current issues when using uncompressed columns
  - Measure performance impact
- Goal:
  - *Evaluate performance of point-accesses (e.g., materialization) using dictionary-encoded data vs. uncompressed data for columns that are point-accessed*
  - *Quantification of losses in compression rate*

# Shared Domain Dictionary for Hyrise

- Order-preserving dictionaries (e.g. Hyrise, SAP HANA)
  - inefficient mapping structures for cross table operations (e.g. JOIN)
  - costly data re-encoding during merge
- Findings
  - Join operations always between columns of the same domain
  - Value-ranges of PK columns are typically incremental (but not FK)
- Idea
  - A shared dictionary (encoding) for PK and FK of the same domain
  - Direct join on (compressed) valueIDs
  - No re-encoding during merge for PK/ FK columns.
- Task
  - Implement a shared domain dictionary (SDD) as well as an adapted merge and join operation
  - Evaluate performance using HYRISE

# The Tiering Run in Hyrise

- Build on an implemented prototype in Hyrise:
  - Using given statistics about relevant data, tables are partitioned according to the data's relevance (i.e., the tiering run)
  - Relevant data is allocated with malloc()
  - Less relevant data is allocated on disk
  - (paper available with more details)
- Tiering Run:
  - Capture workload statistics and analyze
  - Create views that define relevant data
  - *Use views to partition data and re-allocate*
- Goal:
  - Implement or improve necessary steps for the tiering run, enable re-heating while providing existing database properties as transactionality et cetera

# Memory Mapped File Checkpointing

- A checkpoint is a consistent snapshot of the database to speed up recovery
- In-memory databases with main/delta concept need to write complete delta to storage for checkpoint
- Task: Implement checkpoint algorithm in Hyrise, by allocating delta data structures on Memory Mapped files on a Fusion ioDrive and perform a `msync()` of the file for the checkpoint
  - Working with newest FusionIO drive
  - Measure performance implications
  - Compare with ‘normal’ serialization of delta to storage

# Transparent Allocator Mechanism for Hyrise

- Currently, custom allocator for non-volatile memory exists, but needs integration
- Implement transparent allocator principle for Hyrise, allowing to switch allocation strategies transparently

# Read-Only Replication

- Basic read-only replication functionality exists in Hyrise, allowing for single master multiple slave replication
- Implement k-safety replication mechanisms and correct failover handling mechanisms
- Measure replication performance and replication delay and quantify robustness of replication mechanism

**Video:** https://dl.dropboxusercontent.com/u/2529895/hyrise_hotstandby.mp4

# Hyrise Frontend / Cluster Manager

- Browser based JavaScript frontend
- Managing database settings and cluster
- Displaying live performance data
  - Develop an HTML5 application that visualizes heartbeats from Hyrise during the execution of a TPC-C workload
  - Showing multiple live charts visualizing query performance and database statistics
  - Take inspiration from: memSQL

**Video:** https://dl.dropboxusercontent.com/u/2529895/mp-screencast%201080p.mov

# Hyrise SQL

- Implement basic SQL functionality for Hyrise, including:
    - Parsing end execution of (simple) SQL queries
    - Database connection handling to alleviate current usage of *evLoop* and *ODBC* interface building on top of open source *unixOBDC*
    - Frontend integration allowing to formulate SQL queries in browser based frontend

# Hyrise Clustered Index

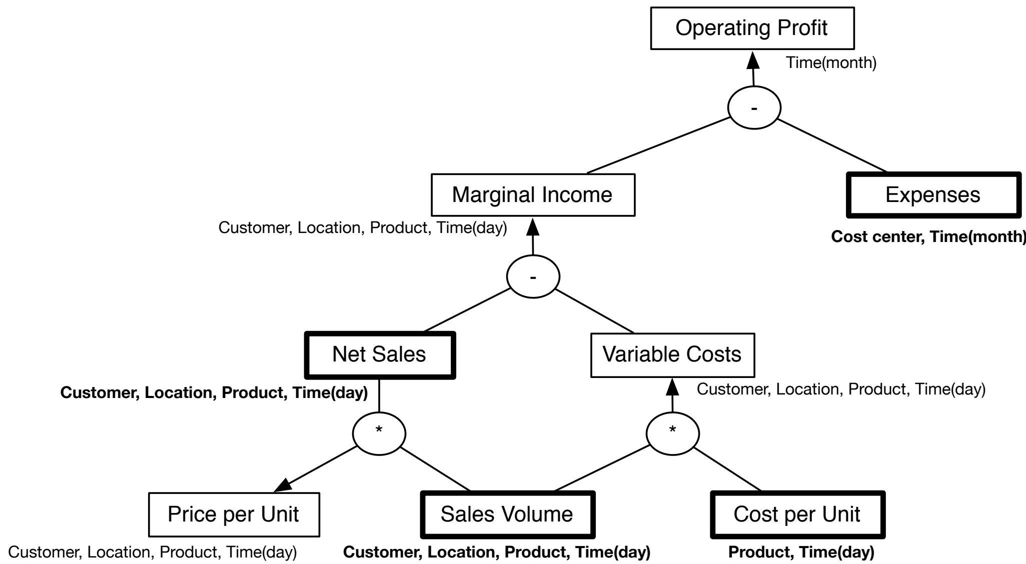
- Improve an existing implementation of a "sorting merge" algorithm and integrate into Hyrise Master Branch
- Performance Evaluation with real-world table data
- Implications on select performance, secondary index vs. clustered index, aggregations by cluster attribute

# HANA Primary Key Index

- Measure Insert & Select performance of different primary-key implementations in SAP HANA
- Evaluate memory footprint based on real-world key columns
- Evaluate multiple datatypes

# Support Enterprise Simulations with IMDBs

- Enterprise simulations define changes on multidimensional hierarchical data.




- How to support hypothetical queries?
- How to optimize hypothetical queries with groupings on various granularity level?

Thank you.