Building Hybrid Transactional/Analytical Processing (HTAP) Systems

Gabriel Campero Durand
HTAP systems are about 4 fundamental things:

First, **freshness** of data, which requires fast connections.
HTAP systems are about 4 fundamental things:

First, freshness of data, which requires fast connections.
Second, HTAP systems are about keeping a stellar performance (efficiency) for both OLAP and OLTP
Third, smart **isolation** of both types of workloads, so one does not consume all the resources of the other.
Fourth, flexibility to tune between freshness, efficiency and isolation.
Today's HTAP systems offer us great insights about the state of the art on data management. And also, not only for a single data model.
Agenda

1. Goals
2. HTAP: Definition, Goals
3. Background:
   a. Storage models: PAX, logs.
4. Design choices for HTAP
   a. Architectural
   b. Storage models per operators
   c. Layout flexibility
   d. Execution model
5. Challenges ahead & Takeaways
Agenda: A tour of 11 HTAP systems and design alternatives

Note: We won’t consider loosely coupled HTAP systems in this talk.

HTAP design is heavily connected to both storage and query processing.
A tour of 11 (from 16) HTAP systems

3. OctopusDB (2011) - Univ. of Saarland
4. ES² (2011) - National Univ. of Singapore
5. SAP HANA (2012)
8. Oracle 12c Database In-Memory (2015)
9. Analytics in Motion (2015) - ETH Zurich
12. Caldera (2017) - EPFL
13. BatchDB (2017) - ETH Zurich
14. SnappyData (2017) - Univ. of Michigan
15. IBM Wildfire (2017)

* With CockroachDB being the open source version
1. Goals
Goals

1. To give you a clear understanding on:
   a. What can be used to design for HTAP workloads.
   b. Storage models available (Rows, columns, PAX, logs) & their effect on access patterns.
   c. Architectural choices for HTAP systems, with examples from contemporary products.

2. To give you a broad perspective about the design of data management systems, and some knobs in physical design for tuning them.
2. HTAP Systems
But first, what is \textit{really} an OLAP or an OLTP system?
OLAP vs OLTP

- The separation between OLAP and OLTP has its roots in the separation of **Decision Support Systems** and **Operational Systems**. Each was a system specialized for certain uses.
- This extends far back in time!
DSS workloads = OLAP workloads

- Primary task: **Decision Support** (e.g. Frequent Pattern Mining to decide on a marketing campaign), reports and analysis.
- Large amounts of **historical data**, usually a bit out of date.
- **Usually complex queries** that go through all the rows but a limited number of columns. <= Indexes might not be needed, but having a sorted copy of the column could be good!
- **Mostly read-only** workloads.
- **Long-running queries**, need to be optimized for individual response time.
- Star instead of snowflake schemas, Denormalization to reduce joins.
- **Massive parallelism**.
- Updates tend to be done in batch, entirely separate from the analysis tasks, with the system offline.
Operational workloads = OLTP workloads

- Operational systems’ Primary task: **Everyday transactional tasks**: Individual sales in retail or e-commerce (which have to keep an inventory updated), tracking product delivery in transport companies, etc.
- **Usually only recent data**, usually not long histories are kept for these systems.
- **Usually small and simple queries** that go through all the attributes of a few rows. Usually retrieving only one tuple! (Point-queries) $\leq$ Indexes are needed!
- A high amount of **inserts, updates and deletes**.
- **Many short-lived transactions**, need to be optimized for throughput.
- Normalization needed for fast writes and to keep the database from growing.
- **Parallelism is actually a challenge**. Concurrent writes can slow things down.
- Updates are frequent. The system cannot go offline for them.
Observation:

- These example of access patterns are just a rough approximation.
- In the wild, it’s not straightforward to identify queries as OLAP or OLTP
  - =>The real distinction happens at **workload level!**
Main-memory RDBMSs

- In former lectures we’ve heard about the generals evolution of RDBMSs:
  - Main-memory designs with a simpler architecture have emerged from disk-based ones. ✓
  - Cache-aware indexes and operations. ✓
  - Query processing: Vector-at-a-time. ✓
  - ...

- How did these changes take place in OLAP or OLTP-specific systems?
The evolution of OLTP systems (NewSQL)

- Given their needs, specialized OLTP systems have evolved since the 2010s:
  - Row-oriented
  - Distributed and scalable (competing with NoSQL)
  - ACID guarantees
  - Several specialized indexes for point-queries and frequent inserts.
  - Based on single-threaded partitions, with one transaction at a time→ Locking is avoided inside a partition.
    - Cross-partition transactions are usually handled with 2 phase locking.

![Image: Source [1]](image-url)
The evolution of OLTP systems (NewSQL)

- Given their needs, specialized OLTP systems have evolved since the 2010s:
  - Row-oriented
  - Distributed and scalable (competing with NoSQL)
  - ACID guarantees
  - Several **specialized indexes** for point-queries and frequent inserts.
  - Based on single-threaded partitions, with one transaction at a time=> Locking is avoided inside a partition.

- Cross-partition transactions are usually handled with 2 phase locking.

For a good definition of NewSQL:

Image: Source [1]
Specialized indexes for OLTP

**The goal:**
- To facilitate point-queries with $O(\log_n)$ access.

**The problem:**
- Balanced trees, the traditional structures, can require too much time for maintenance.
  - Not a good characteristic for systems that have continuous inserts!

**Some proposed solutions:**
- Skiplists, HyPer’s ART (prefix-oriented), Microsoft’s BWTree (using an idea like MVCC).

*Next: A quick test-drive of the ideas behind Skiplists.*
Skiplists: A specialized index for OLTP

- The core idea:
  - To rely on probabilities, instead than a balancing operation, to keep a tree-like structured balanced for fast point-queries.

---

Skip Lists: A Probabilistic Alternative to Balanced Trees

Skip lists are data structures that use probabilistic balancing rather than strictly enforced balancing. As a result, the algorithms for insertion and deletion in skip lists are much simpler and significantly faster than equivalent algorithms for balanced trees.

William Pugh
Skiplists: A specialized index for OLTP

Rotate, remove duplicate nodes, does it look familiar?
Skiplists: A specialized index for OLTP

It's somehow similar to a binary tree.

In fact, the expectation is that skiplists will probabilistically tend towards a structure with access times similar to those of a binary tree.
Skiplists: A specialized index for OLTP

Insertion of 19, with False, True:

- For insertion, we flip a coin several times (true or false) to decide, starting at the bottom, how high to place a key.
- When you insert a key, you also insert the complete vertical chain until the bottom row.
- The tree should be unlimited to grow in height, but on practical cases, limits can be given.
Skiplists: A specialized index for OLTP

Insertions in an AVL tree:

Skiplists: A specialized index for OLTP

- Let's find a couple of keys!

- For K5, comparisons: K4, K17, K5
- For K19, comparisons: K4, K17, K19
- For K1, comparisons: K4, K2, K1
- For K3, comparisons: K4, K2, K4, K3
Skiplists: Pros and Cons

Pros:
- Insertions and deletes do not require re-balancing.
- Lock-free implementation possible using Compare and Swap.
  - For this we need to add a delete attribute to the keys.

Cons:
- Random generator is costly.
- Skiplists need to be tuned for efficient cache usage (note: some improvements already exist in this direction, http://ticki.github.io/blog/skip-lists-done-right/)
OLAP systems

- Given their needs, specialized OLAP systems have been developed since the 2010s:
  - Column-oriented
  - They rely on complete column scans, accelerated with SIMD, encoding and compression. A sorted copy of the column could act as an index.
  - Multi-thread parallelism.
  - Avoidance of logging by using a standby server.
  - They could be distributed, but this is not the essential aspect of their design.
OLAP systems: Specialized indexes

The goal:
- Range queries, parallel access, serving multi-dimensional queries (not addressed here)

Some solutions:
- Encodings, Bitmap indexing (not efficient if we have lots of values in the attribute)
“Sorted column copy” as an index

The goal:
- Since having a copy of the column sorted can help in range queries, it would be a useful feature.

The problem:
- Sorting takes time on very large columns.
  - During the sorting, the column would have to be blocked from readers.

Solutions:
- Database cracking
Database cracking

The idea:
- Perform the building of the sorted column, as part of normal query processing.
Quick recap

- OLAP vs. OLTP have their origins in the worlds of Decision Support Systems and Operational Support Systems.
- The difference between them is at workload level, not at the level of a specific query!
- New OLTP, OLAP systems: Characteristics and Indexes.
- Next: HTAP Systems
HTAP systems

Gartner defines an HTAP system as one that:

“can execute both I/U/D statements as well as analytical queries over all data, including the recently ingested in this transaction, within the same transaction”

This definition is about the freshness of data.
HTAP systems

T1:
1. Avg(R.salary)
2. Update(R.salary where id=30, increase by 10k)
3. Avg(R.salary)
4. Commit()

Gartner defines an HTAP system as one that:

“can execute both I/U/D statements as well as analytical queries over all data, including the recently ingested in this transaction, within the same transaction”

This definition is about the freshness of data.

Surprisingly (according to IBM researchers) some HTAP systems might not get to see the latest update at this point.

To compensate, some systems force single-statement transactions, so updates are propagated on each commit (e.g. ES²).
HTAP systems

By defining HTAP on this specific transactional aspect, this definition might be of little use.

We propose a more general definition
An HTAP system can be defined as: “A data management system that is as efficient as a standalone OLTP system or a standalone OLAP system, for those specific workloads. In addition it must be robust and efficient for handling these workloads at the same time, with the capability of performing analysis on the latest data.”
HTAP systems: The 3 goals and their challenges

From our straightforward definition we get the goals of HTAP:

- Recency of data/Freshness: Queries can be performed over historical and the latest data.
- Efficiency at OLAP, OLTP

A study by (Psaroudakis et al, 2014) unveiled other important design goal:
- Isolation in resource contention between OLAP and OLTP workloads/Robustness.

And finally, from the same study a characteristic of the system:
- Flexibility in the restrictions that a DBMS may impose for increasing optimization choices to enhance performance.

This set of possibilities is challenging because the goals require to consider tradeoffs:
- Strong isolation may give us less freshness.
- Freshness might reduce efficiency.

Scaling up Mixed Workloads: a Battle of Data
Freshness, Flexibility, and Scheduling

Iraklis Psaroudakis1,3, Florian Wolf1,4, Norman May1, Thomas Neumann2, Alexander Böhm1, Anastasia Ailamaki3, and Kai-Uwe Sattler4
HTAP systems: Expectations from goals and characteristics

Scaling up Mixed Workloads: a Battle of Data Freshness, Flexibility, and Scheduling

Iraakis Psaroudakis¹,², Florian Wolf¹,⁴, Norman Mayer¹, Thomas Neumann², Alexander Böhm¹, Anastasia Allamaki³, and Kai-Uwe Sattler¹
HTAP systems: The 3 goals and their challenges

Results comparing 2 state-of-the-art HTAP systems: HyPer and SAP HANA.

SAP HANA Deltastore

Image: Source [2]
HTAP systems: The 3 goals and their challenges

Results comparing 2 state-of-the-art HTAP systems: HyPer and SAP HANA.

Scaling up Mixed Workloads: a Battle of Data
Freshness, Flexibility, and Scheduling

Iraikis Paraskevakis¹,², Florian Wolf¹,³, Norman May¹, Thomas Neumann², Alexander Böhm¹, Anastasia Ailamaki³, and Kai-Uwe Satzler⁴
HTAP systems: The 3 goals and their challenges

Results comparing 2 state-of-the-art HTAP systems: HyPer and SAP HANA.

SAP HANA, default configuration
(High OLAP efficiency => Low isolation)

SAP HANA, no intra query parallelism
(Lower OLAP efficiency => Higher isolation, since OLTP throughput is less affected by more Analytical clients)
HTAP systems: The 3 goals and their challenges

Results comparing 2 state-of-the-art HTAP systems: HyPer and SAP HANA.

HyPer- Little freshness (High OLAP throughput)

HyPer- More freshness (Lower OLAP throughput)
HTAP systems: The ideal HTAP system

In the results we see a “house pattern” where increasing OLAP queries tend to cause resource starvation to OLTP queries. The more pronounced the impact, the worst the isolation.

We also see that:

● In SAP HANA, reducing intra-query parallelism (efficiency) leads to better isolation.
● In HyPer reducing freshness improves the OLAP throughput, but has little effect on the OLTP one (because the system is more isolated).

From these observations we can suggest the ideal HTAP system:

The ideal HTAP system offers Freshness, Efficiency and Isolation in such a Flexible way, in such a way that they could be adaptively tuned according to the workload.
After surveying the key ideas of HTAP, we will now present some background that we’ll use for our review of existing HTAP systems.
3. Background
Storage model

- From former lectures you must be familiar with storage models like N-ary storage model, NSM and the Decomposed storage model, DSM (a.k.a, row and columnar layouts).

- As you might know, these models matter a lot because they determine the access pattern of operations and memory footprint.

- From our OLAP, OLTP discussion, you know that they also have an impact on the complete design of the DBMS.

- In this section we complement NSM and DSM with 2 models.
PAX- Partition Attributes Across

- PAX is a hybrid layout, between rows and columns.
- Accordingly it has a competitive performance with both layouts.

- It was proposed by Natassa Ailamaki, in 2001.

- PAX starts with the concept of a page, as a consecutive block of memory.
- Each PAX page is designed to hold a number of tuples. Cache lines might try to pack as many pages as possible.
- Inside it uses a columnar layout.
- The highest the number of tuples per PAX page, the closest it is to a column store, the lowest to a row store.

Some systems using PAX:

- OctopusDB
- IBM
- Parquet
“For most workloads, PAX is not optimal, but: it is relatively robust and performs well on many workloads”

Jens Dittrich.

Patterns in Data Management: A flipped textbook. 2016
Logs

- A log is perhaps the simplest storage abstraction.
  - Append-only, ordered sequence of records by time.

Historically, they have been confined to systems internals, but recently it’s being used as a mechanism for data subscription (e.g. Kafka) and also this idea has made its way into the design of some HTAP systems, like Octopus DB.

End of Part 1
Takeaways from Part 1

- OLAP, OLTP: Workloads and systems
- HTAP definition and challenges of managing: Freshness, Efficiency and Isolation, given a certain Flexibility.
  - HTAP - Efficiency at OLAP/OLTP, Freshness, Isolation + Flexibility allowing to use adaptive methods for tuning according to workloads.
- Necessary Background:
  - Storage models: PAX, Logs.
Note: We won't consider loosely coupled HTAP systems in this talk.
Offline Reorg =>
Layout might be good for yesterday's workload but not for today's.

Online Reorg => Can slow-down query processing.
References (apart from the material explicitly cited)


Parts of this lecture are based on material from: [1]
Interested in building data intensive systems? So are we!

Join us!

Collaboration opportunities:
- Thesis
- Scientific Team Project, Individual Projects
- http://www.dbse.ovgu.de/
- Contact me: campero@ovgu.de

Requirements:
- Motivation!
  - Willingness to learn and ask questions.
- Programming skills
Building HTAP Systems

Part 2: Design choices

Gabriel Campero Durand
Agenda for Part 2: A tour of 11 HTAP systems

1. HyPer (2010)- TU Munich
2. Hyrise (2010)- Hasso Plattner Institute
3. OctopusDB (2011)- Univ. of Saarland
4. ES² (2011)- National Univ. of Singapore
5. SAP HANA (2012)
8. Oracle 12c Database In-Memory (2015)
9. Analytics in Motion (2015)- ETH Zurich
12. Caldera (2017)- EPFL
13. BatchDB (2017)- ETH Zurich
14. SnappyData (2017)- Univ. of Michigan
15. IBM Wildfire (2017)

Systems in blue can be considered large-scale HTAP and will not be discussed
4. Design Choices for HTAP
4.a- Architectural choices

In our review, we looked only at single systems.
4.a- Architectural choices: Loosely-coupled systems

Why loosely-coupled systems?

For our discussion, let’s consider an HTAP system to be loosely-coupled if: “the query engine has limited information/actionable insights from the storage engine about the data.”

In this context, a tightly-coupled architecture could consist of an SQL engine running on top of Cassandra (without HDFS). This, however might not be an efficient example, because the SQL-to-NoSQL connectors might be slow.

Good examples of tight-couplings include:

- Splice Machine & Phoenix (custom SQL engines over HBase)
- SAP HANA Vora (SparkSQL, for scale-out OLAP, over HANA for OLTP)
- IBM Wildfire (Wildfire engine for OLTP over Spark for OLAP)
- SnappyData (Apache GemFire in-memory grid for OLTP over the Spark engine for OLAP)

Tight integration is also improved by adding specialized features to the query engine (e.g. new rules for pushing down the computation) or storage engine (e.g. co-location mechanisms).
4.a- Architectural choices: Loosely-coupled systems

- An example:

Lambda architecture.

Nathan Marz: [http://lambda-architecture.net/](http://lambda-architecture.net/)

A possible implementation could include:

**Batch layer:** Parquet or ORC files on HDFS (OLAP-tuned, PAX formats, often called columnar).

**Serving layer:** SQL on Hadoop system (e.g. Hive, Drill, Impala)

**Speed Layer:** Key-column value store databases like Apache Cassandra/HBase (OLTP).
4.a- Architectural choices: Single system approach

In contrast to loosely coupled systems, a single system approach should be able to offer:

- Higher freshness
- Transparent integration
- Flexible tuning to the HTAP goals.

We'll look at some examples in the next sections.
### 4.b- Architectural choices: Storage Models Used

Let's look at the storage models used

<table>
<thead>
<tr>
<th>Storage model</th>
<th>Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSM (row), DSM (column), PAX</td>
<td>Caldera, OctopusDB, HyPer (optimized data block instead of PAX, NSM and DSM possible for uncompressed hot data)</td>
</tr>
<tr>
<td>NSM, DSM</td>
<td>ES², Peloton, SAP HANA, MemSQL, Oracle 12c DIM, Microsoft SQL Server, SnappyData, H₂O</td>
</tr>
<tr>
<td>PAX only</td>
<td>Google Cloud Spanner (Ressi), AIM (ColumnarMaps), IBM Wildfire</td>
</tr>
<tr>
<td>NSM only</td>
<td>BatchDB</td>
</tr>
<tr>
<td>DSM only</td>
<td>Hyrise (with different column groups)</td>
</tr>
</tbody>
</table>
4.b- Architectural choices: Types of operators in query plan

What do we mean by “types of operators in a query plan”? First, we’re talking about physical operators; and for types we mean operator for rows, operator for columns, or operator for PAX.

Second, the principle is like “counting heads instead of counting people” =>$\Rightarrow$

If we just look at the type of operators in any given query plan (i.e., are they row, column, log, FSM or PAX-oriented) we can infer some things about the underlying data organization.
4.b- Architectural choices: Types of operators in query plan

- **Single**: Layout for all tables
  - BatchDB, AIM, IBM Wildfire, Google Cloud Spanner, ES²
  - Peloton

- **Multiple**: Complete duplication
  - Oracle 12c DIM

  - Partial duplication
    - Microsoft SQL Server 2016, OctopusDB, H₂O

  - No duplication
    - Caldera, HyPer, MemSQL, Hyrise, SnappyData

- **Types of operators in query plans/engine**
  - e.g. query plans only have operators that work over rows, over columns or over PAX.
  - e.g. within a query plan some physical operators work over rows, others over columns or over PAX.
4.b- Architectural choices: Types of operators in query plan

**Single vs. Multiple types:**

- **Single type of:**
  - Leads to a simpler query engine
    - Simpler query optimization choices.
    - Simpler code maintenance for the database.
  - Might fail to exploit hardware sensitive tuning available to other storage models.

- **Multiple types:**
  - Add complexity
  - Contribute to efficiency
  - Perhaps more flexible to balance between the HTAP goals.
4.b- Architectural choices: Types of operators in query plan

The traditional approach. In our review, systems like BatchDB, AIM, Spanner, IBM Wildfire and ES² fall into this category. We will talk about them later.

Types of operators in query plans/engine

- **Single**: Single layout for all tables
- **Multiple**: Complete duplication, Partial duplication, No duplication
- **FSM-based architecture**: Fractured mirrors data distribution, Deltastore data distribution
4.b- Architectural choices: Types of operators in query plan

A flexible storage model, FSM (only available in Peloton DB):
- Introduces a level of indirection between operators and data layout.
- Operators work on tiles (thanks to a tile-based relational algebra), transparent to the actual layout (which remains the responsibility of the storage engine).

In our review, only Peloton DB falls into this category. We will also talk about this system at a later point.
4.b- Architectural choices: Types of operators in query plan

Types of operators in query plans/engine

Single
- Single layout for all tables
- FSM - Tile-based architecture
- Complete duplication

Multiple
- Fractured mirrors data distribution
- Partial duplication
- Deltastore data distribution
- No duplication
4.b- Architectural choices: Types of operators in query plan

Deltastore data distribution:

OLTP Updates → Delta Store → DSM Historical Data
4.b- Architectural choices: Types of operators in query plan

Deltastore data distribution:

This is an approach exemplified by SAP HANA (2012)

- Hot data (recently updated values) are stored as rows.
- Warm data is stored as columns.
- Finally, cold data is stored in a read-optimized column layout.

To leverage this approach, physical operators need to be able to combine information from the 3 stores \(\Rightarrow\) This guarantees freshness.

Deltastores do not provide, by default, isolation nor flexibility.

In SAP Hana, SIMD processing boosts efficiency.
4.b- Architectural choices: Types of operators in query plan

Types of operators in query plans/engine:

- Single
  - Single layout for all tables
  - FSM - Tile-based architecture

- Multiple
  - Fractured mirrors data distribution
  - Deltastore data distribution
  - Partial duplication
  - Complete duplication
  - No duplication
4.b- Architectural choices: Types of operators in query plan

**Fractured mirrors:**

The original proposal for this approach was disk based and it included redundant DSM data in a B-tree, where the leaves contained values and the IDs were implicit.

The proposal was about having data duplicated into 2 storage layouts (DSM and NSM).

Expanding from the original proposal, the fractured mirrors consists in allowing (not enforcing) flexible data duplication into different layouts (*data can be stored simultaneously in rows and columns*), with the query engine being aware of the different layouts.

FMs are the more flexible counterparts of the Deltastores, and also they facilitate isolation.

FMs do not guarantee freshness, but can offer freshness like a tunable factor.
4.b- Architectural choices: Types of operators in query plan

Complete, Partial and No Duplication:

- Fractured mirrors data distribution
- Complete duplication
- Partial duplication
- No duplication
Fractured Mirrors: Complete Duplication

Oracle In Memory can serve as an example of this approach.

Just like in a Deltastore, data flows from NSM to DSM format. However the row format is persisted. It is further indexed for point-queries, treated like in a full-blown OLTP system.

Contents of the column store can be selectively populated from the row store (but complete duplication is a default option).

Updates are propagated in a trickle, or threshold based approach (this means, that there is flexibility to tune the freshness).

Rows can be horizontally partitioned, with different compression schemes applied to the corresponding columns.
Fractured Mirrors: Partial Duplication

- Microsoft's SQL Server is an example of this approach, it integrates the Apollo OLAP engine with the Hekaton in-memory OLTP engine.

- In Apollo (DSM), rows are grouped and each group is encoded into different column segments. These are paired with a row-wise deltastore (just like SAP HANA). Bulk updates can skip these deltastores and write directly into the column segments.

- Hekaton (NSM) tables can be indexed with “Column-wise indexes” (CSI).

- Users decide whether to have a table in Hekaton or in Apollo.

- In both cases, the writes proceed generally from rows to columns (like in Deltastores).

In our review, Octopus DB and H₂O also fall into this category.

Is the distinction between example and category clear?
4.b- Architectural choices: Types of operators in query plan

**Fractured Mirrors: No Duplication**

- MemSQL is an example of this approach.
- In MemSQL, tables can be stored either as rows or columns.
- This decision happens at schema-definition time, so users or DBAs decide the format of the table.
- ColumnStores are also updated with small row stores.

In our review, Hyrise, Caldera, SnappyData and HyPer fall into this category.

Is the distinction between example and category clear?
4.b- Architectural choices: Types of operators in query plan

Fractured Mirrors: No Duplication

- HyPer, the backend of Tableau is an example of this approach.
- In HyPer, data is divided into Hot (uncompressed) and Cold data (compressed data blocks).
- Uncompressed data could be in any format. The compressed data is in a PAX-like layout, with special optimizations for compression. Within each block, the layout and compression is unique.
- Data is aged towards a cold format.
Fractured Mirrors: Complete vs. Partial vs. No Duplication

- All FM systems are similar to Deltastores, in the fact that writes to columns tend to start from row representations.
- In a system with complete duplication
  - freshness might be harder to tune.
- In a system with no duplication
  - isolation might be harder to achieve (except for HyPer), but individual latency should be better.
- Systems with partial duplication seem like a balanced solution.
- Evaluations are needed on these design aspects.
4.b- Architectural choices: Types of operators in query plan

Fractured Mirrors: Complete vs. Partial vs. No Duplication

- All FM systems are similar to Deltastores, in the fact that writes to columns tend to start from row representations.
- In a system with complete duplication
  - freshness might be harder to tune.
- In a system with no duplication
  - isolation might be harder to achieve (except for HyPer), but individual latency should be better.
- Systems with partial duplication seem like a balanced solution.
- Evaluations are needed on these design aspects.
4.c- Architectural choices: Layout flexibility
4.c- Architectural choices: Layout flexibility

A system with an adaptive layout is a system that allows optimizations like this.

The key feature for adaptive layouts.

Accordingly, systems with this strive to develop self-managing features.

The opposite of adaptive layouts are either static layouts or manually decided layouts.
4.c- Architectural choices: Layout adaptivity

Adaptive Layouts: Offline reorganization

- **Hyrise** is a framework for storing different vertical partitions (in DSM form)
- Compression is used.
- Columns are divided into a read-only area and one for updates. These areas are merged in batches.
- The capacity for working efficiently with ad-hoc column-groups is partly achieved by generating operators on the fly. => Not a single type of operator per query plan.

Layout might be good for yesterday’s workload but not for today’s.

Is the distinction between example and category clear?
4.c- Architectural choices: Layout adaptivity

Adaptive Layouts: Online reorganization, as part of query execution

- \( H_2O \) proposes a framework for continuously adapting the layout to the workload (through vertical partitions and a PDSM inside each group).
- Changes are done to a copy.
  - Think of it like a mini fractured mirror.
- The capacity for working efficiently with ad-hoc column-groups is partly achieved by generating operators on the fly. => Not a single type of operator per query plan.
- At the moment it does not support transactions, not updates.

Organization might slow-down query processing.

Is the distinction between example and category clear?
4.c- Architectural choices: Layout adaptivity

A copy that is swapped during query execution with the original
Adaptive Layouts: Online reorganization, as a background process

- **Peloton** is an innovative DBMS design built around the concept of a *flexible schema model* (FSM) based on tiles.
- To support this concept, Peloton offers a design of tile-based algebra—Single type of operator per query plan.
- Given that physical layout is abstracted from operators, the reorganization can happen as a background process.
- Another part of Peloton is the proposal of tunable knobs (e.g. view maintenance) exposed to a learning process=> Self-driving features.
4.c- Architectural choices: Layout adaptivity

Adaptive Layouts: Online reorganization, as a background process

Figure 4: Logical Tile – An example of a logical tile representing data spread across a couple of physical tiles (A-1, A-2).
4.c- Architectural choices: Layout adaptivity

Adaptive Layouts: Online reorganization, as a background process

- Operators in Peloton:
  - Bridge operators
    - The only operators that interact with the PT are either at the top (materialize operator) or at the bottom (sequential and index scans) of the tree. They create LT from PT or Tables. There is a concept of a passthrough logical tile, which allows to express partial materialization of logical tiles.
  - Metadata operators
    - They operate only on LT. Project and Select operators need only modify the metadata to mark columns or rows that should be left out of the tile. Bitmaps are used for row access.
  - Pipeline breakers
    - They operate only on LT. Project and Select operators need only modify the metadata to mark columns or rows that should be left out of the tile. Bitmaps are used for row access.
  - Mutators
    - MVCC is used. Insertions happen in a default NSM tile, with a latch on the tuple. Deletions also happen with a latch, but are like metadata operators. Updates are delete and inserts.
4.c- Architectural choices: Layout adaptivity
4.c- Architectural choices: Layout adaptivity

Adaptive Layouts: Online reorganization alternatives

- While authors argue that background reorganization might be less demanding than optimization as part of query execution in reality the second method needs further development (i.e., support for transactions), so the design tradeoff can be properly evaluated.
4.c- Methods for vertical partitioning

5 columns can be (vertically) partitioned in 52 ways.

\[
\begin{array}{cccccc}
1 & 1 & 2 & 3 & 5 \\
2 & 3 & 5 & 7 & 10 & 15 \\
5 & 7 & 10 & 15 & 20 & 27 & 37 & 52 \\
15 & 20 & 27 & 37 & 52 & 67 & 87 & 114 & 151 & 203 \\
52 & 67 & 87 & 114 & 151 & 203 & 255 & 322 & 409 & 523 & 674 & 877 \\
877 & ... \\
\end{array}
\]

Bell numbers

It is a large search space, and is exponentially so as we increase the number of columns!
Note: These methods are really all variations on common clustering techniques.
4.c- Methods for vertical partitioning

- **Navathe**: Based on an affinity matrix and matrix clustering with a bond energy algorithm.
- **HillClimb**: From Column layout, increasing 1 by 1 the partition sizes, until no improvement is possible.
- **AutoPart**: First horizontal partitions, based on selection predicates. Second, a set of *atomic partitions* is generated, such that all queries accessing it reference the partition. Fragments are extended in each iteration. Repetition of attributes is possible.
- **Hyrise**: Starts with atomic partitions, like AutoPart but uses affinity graphs.
- **O2P**: Dynamic programming approach based on Navathe. Works online.
4.c- Methods for vertical partitioning

Figure 14: The computed partitions for the TPC-H workload.

Figure 3: Estimated workload runtime for different algorithms.
4.c- Methods for vertical partitioning

![Figure 1: Optimization time for different algorithms](image)

![Figure 2: Optimization time over varying workload size](image)
To wrap-up our discussion about layouting.

One essential observation:

**It's not only about the layouting, but about the further optimizations that it enables.**
Where does the efficiency of columns come from?

3x late materialization
2x-10x compression
2x invisible join

=> It’s not only the layout, but using it smartly.

Figure 7: (a) Performance numbers for C-Store by query flight with various optimizations removed. The four letter code indicates the C-Store configuration: T=tuple-at-a-time processing, t=block processing; I=invisible join enabled, i=disabled; C=compression enabled, c=disabled; L=late materialization enabled, l=disabled. (b) Average performance numbers for C-Store across all queries.
4.d: Execution Model
4.d- Execution model

While traditionally the query engines of DBMSs work with a query-at-a-time model (i.e., each operator is answering to a single query). It is possible to implement a multi-query approach.

Some HTAP systems exploited this potential, to enable fast OLAP queries over non-indexed rows.
4.d- Execution model: Multi-query

- **BatchDB is an example of this approach**
  - *Isolation* as a key design component: OLAP and OLTP archipelagos with multiple replicas.
  - Partitioning for OLTP is similar to H-store (one thread per OLTP partition)
  - Writes and queries are interleaved, queued and propagated in batches.
    - First writes, then queries (*Freshness can be tuned flexibly*)
  - Cons: Efficiency of static layout depends highly on the workload.

Operators process queries in a batch.
Design space for HTAP systems.

Architecture
- Single System
  - Loosely-coupled systems
- Storage models used
  - NSM
  - PAX
  - DSM
  - Log

Types of operators in query plans/engine
- Single
  - Single layout for all tables
- Multiple
  - FSM-Tile-based architecture
  - Complete duplication
  - Fractured mirrors data distribution
  - Partial duplication
  - Deltastore data distribution
  - No duplication

Layout adaptivity
- Static layout per table
- Adaptive layout
  - In-place reorganization
  - Copy-based reorganization
- Online reorganization
  - As batch process
  - As part of query execution
- Offline reorganization

Execution model
- Query-at-a-time
- Multi-query
Ideal characteristics for an HTAP system

In yellow: some decisions that need evaluation
In red: the ideal choices for HTAP designs
End of Part 2
Takeaways from Part 2

- Different solutions for HTAP
  - Architectural alternatives for HTAP: Single system vs. separate ones.
  - Types of operators per query plan for HTAP
    - Single
    - Multiple
  - Fractured mirrors, Deltastores.
    - Some counter-intuitive findings: Fractured mirrors, while criticizable for not being fresh, might actually have benefits because it allows freshness to be tuned. Furthermore, it is not so different to deltastores w.r.t the write path (usually from rows to columns).
  - Layout flexibility: Reorganization process.
    - Methods for vertical partitioning are key.
  - Execution model:
    - Opportunities from cooperative-execution.
Thanks for the attention!

See you soon for Second Part on AI Techniques for Data Management (optional) :)

campero@ovgu.de
References (apart from the material explicitly cited)


Parts of this lecture are based on material from: [1]
Interested in building data intensive systems? So are we!

Join us!

Collaboration opportunities:
- Thesis
- Scientific Team Project, Individual Projects
- http://www.dbse.ovgu.de/
- Contact me: campero@ovgu.de

Requirements:
- Motivation!
  - Willingness to learn and ask questions.
- Programming skills
Thanks for the attention
&
All the best for your exam!

campero@ovgu.de