- What is the problem and why it is important?

  As we know data-intensive applications seek to obtain trill insights in real-time by analyzing a combination of historical data sets alongside recently collected data. The current trend, to handle fast ACID transactions and complex analytical queries on the same database, is to use specialized systems that are optimized for only one of these workloads, and thus require an organization to maintain separate copies of the database, which adds several types of unnecessary overheads. This paper is trying to solve this problem by presenting a hybrid DBMS architecture that efficiently supports varied workloads on the same database. They also present a technique to continuously evolve the database's physical storage layout by analyzing the queries' access patterns and choosing the optimal layout for different segments of data within the same table.

- Why is it hard and why older approaches are not enough?

  There are lot of past work to overcome the same problem by giving several storage models like for OLAP workloads, the DBMS I/O efficiency can be improved by adopting DSM, as it only fetches those columns that are required for query processing but incurs high tuple reconstruction costs for OLTP queries. The PAX model was introduced, where all the data for a single tuple are stored together in a disk page like NSM, but it clusters the values of a particular column together for multiple tuples within that page. It also determines the optimal partitioning for a given workload using a hill-climbing algorithm evaluated over query workload statistics. This paper claims that all these works examine the performance impact of storage models on OLTP and OLAP workloads. They assume, however, that the workload is known a priori, and thus are unable to adapt to dynamic HTAP workloads.

- What is key idea and why it works?

  The Key idea here is to store tables using hybrid layouts based on how the DBMS expects tuples will be accessed in the future. This means that a table's "hot" tuples are stored in a format that is optimized for OLTP operations, while the other "cold" tuples in that same table are stored in a format that is more amenable to OLAP queries. This paper proposed a logical abstraction over this data that allows the DBMS to execute query plans that span these different layouts without using separate engines and with minimal overhead. The other contribution is a novel on-line reorganization technique that continuously enhances each table's physical design in response to an evolving query workload. This enables the DBMS to migrate the database to the near-optimal storage layout for an arbitrary application without requiring a human administrator to configure the DBMS for the application and in a transactionally safe manner.

- What is missing and how can we improve this idea?

  I feel the idea is very good to perform different type of workload with the same database and make efficient use of FSM. The only thing I am thinking that can be improved is the efficient use of machine learning algorithms to predict the future queries and reconstruct the data layout. This will reduce the latency as the data layout will be organized beforehand. Another approach which we can experiment is the use of compression techniques with logical tiles to increase the CPU efficiency.

- Does the paper support its claims?

  I think the paper did a pretty good job of demonstrating the importance of the problem and their solution. They also went into the right number of details and talked about different ways to implement each step. They also performed lot of experiments to support their approach and benefits.

- Possible next steps of the work presented in the paper?

  The paper specified that Peloton contains several knobs for controlling the layout reorganization process. Although they attempted to provide good default settings for these knobs, they believe that the DBMS should automatically adjust these knobs based on the HTAP workload. They also planned to investigate the design of a self-driving module within the DBMS that dynamically adjusts these knobs to simplify the tuning process. They intend to explore code generation and data compression techniques for optimizing query execution as well as other facets of the DBMS's runtime operations.