Abstract—Modern enterprise applications are a challenge for traditional database systems. In the recent years, the workload evolved from simple record-lookups and modifications to integrating analytical and transactional in a single database engine. Today, specialized graph structures and queries need to be integrated for applications like materials requirements planning, that further change the workload. Now, we can observe a new challenge for these application specific database systems: On the one hand, to achieve the best performance, we cannot store this graph data in a traditional relational database management system. On the other hand, to improve the business performance it becomes necessary to semantically enrich business processes with additional data.

In this vision paper we describe a future database storage architecture that is able to work with multiple different storage types like traditional relation tables stored in rows, columns and hybrid partitions, as being able to store graphs natively. In-Memory technology is the technological foundation, since it provides the required performance and flexibility to combine these different storage types in a single storage engine without loosing performance.

I. INTRODUCTION

The recent changes in computer architecture lead to a radical change of possibilities for enterprise software. The increase of parallelism and the availability of main memory allow rethinking of the storage layer of enterprise applications. This change allows overcoming the separation of analytical and transactional database system and to establish a single storage layer for enterprise applications[1].

However, there is a second side to enterprise application software that is often not taken into account: semantic and graph data processing. While more and more structured relational data is integrated into transactional processing, typically semantic and graph-structured data is left out. The reasons are manifold, but are mostly centered around data integration. Storing this graph data, like employee hierarchies and connected bill of materials, in a secondary system introduces additional latency and maintenance overhead. Tight integration of graph and relational data was ignored because both data types store data differently yielding different access patterns to disk which were not compatible. With in-memory technology we can overcome this separation. In this vision paper, we propose a new architecture of enterprise application specific database system that allows including semantic and graph data directly in the same storage engine combining the advantages of relational and graph data processing in a single in-memory database engine.

II. SCENARIOS

Graph processing for enterprise applications comes in two distinct facets: First, transactional data that favors graph representation and, second, additional semantic graph data that can be directly leveraged during transactional processing. Due to space constraints we focus on transactional graph processing.

A. Transactional Graph Processing

A typical example for the first facet is the bill-of-materials (BOM) that is maintained for all products. The BOM identifies which materials are used using a specified quantity during production. For highly integrated industries like the automotive industry, the BOM is very complex. In addition, products like cars come in many variants requiring to keep track of thousands of different possible BOMs that are than used in the production process. In traditional relational database system there exists different encoding schemes to store this information in a relational database and query it. First, there is the adjacency list pattern that stores each edge as a connection of two vertexes and each vertex is identified by an unique ID. Second, the overall material hierarchy is materialized in the relational table and thus allowing to quickly identify the level of hierarchy per part. However, for both storage patterns, querying is unintuitive and requires e.g. writing linear recursive SQL statements.

The applications that use this data have multiple goals. One example is materials requirements planning where based on available stock the optimal production cycle is planned. While the bill of material for a single product can be seen as a tree structure, the connection of multiple instances and planning over time to build the best schedule can generate a complex graph structure. The queries on this graph become more complex, when single components need to be traced through the complete supply chain.

It is possible to use specialized database architectures to store these kind of graph data elements, however it becomes of great importance to directly integrate query processing of the graph data with operational transactional data that is stored in the enterprise system. This shortcoming would require loading the data into a secondary system creating a lag that may
have a severe impact. Our initial experiments show that a custom in-memory based graph storage layer can significantly outperform an in-memory database system. However, this performance comes at the price of higher memory consumption. For our system the goal must be to achieve good compression ratio and still provide the necessary performance. Therefore, we devise the following research directions.

III. RESEARCH DIRECTIONS

Based on the motivational example, we will now outline the research directions for a hybrid relational-graph data storage engine. In [2], [3], we presented the architecture for a hybrid in-memory storage engine that we will use as foundation for our research. This architecture provides two fundamental properties that we can leverage in our envisioned work: First a customizable, compressed in-memory storage engine and a layout algorithm for optimal online vertical partitioning of relational tables. Therefore, we divide our work into two major aspects

A. Compression & Partitioning

While main memory of modern computer systems is large enough to store 2 terabytes of data this might not be enough to store all data uncompressed. Thus, we will investigate how to apply compression algorithms from compressed read-optimized database engines to storing graph data. Using compression we could further increase the locality that is crucial to query processing in in-memory database engines. The second part of this research direction directly appends to the previous topic: Graph partitioning for increasing locality for query execution. Related work, for example in [4], shows that by colocating important aspects of graph elements, we can achieve dramatically better performance. To summarize, research must address the following questions:

1) How can a graph container leverage compression to achieve better memory locality during query execution?
2) How can we define an optimal graph partitioning (e.g. with replication) to achieve the best performance?
3) How can we achieve the insert ratio required by enterprise applications?

B. Query Execution

In this second aspect, we cover the integration of two different kinds of storage engines, both requiring different algorithms to answer queries by the user. However, the question is not which language might be most suitable, but rather how can we extend an operator-based query execution flow so that graph and relational queries are not executed in separated phases, but can be freely mixed. An example would be the join of constructing the transitive closure for checking if materials can be freely mixed. An example would be the join.

1) Can traditional materialization strategies of relational databases be applied to graph databases to achieve the best performance?
2) What is the intersection of operators that are required for graph and relational data processing?

IV. RELATED WORK

Even though the size of available main memory drastically increased during the recent years, related work that specifically address issues in main memory is limited. The approach closest to ours is probably the idea of Storage Views in OctopusDB[5]. However, we focus on designing explicit storage containers for different data types rather than providing arbitrary transformations on relational database logs. In our approach we will try to embrace ideas from [6] and [7] for effectively storing data in main memory.

Unlike RDF3x[8], [9], we will focus on partitioning graphs instead of exhaustive indexing as it is done in DiplodocusRDF[4]. Large scale distributed graph processing is very important as shown in[10], but in a first version we will focus on improving the performance on a single node to assess the initial capacity.

V. SUMMARY

In this vision paper, we propose a new way of combining the execution of relational and graph data processing inside a single storage engine. In contrast to existing work, the goal is to provide specialized containers inside our prototype database system that allow storing the data in different representations. The advantage of this approach is many-fold: First, for each data item it is possible to chose what storage format fits best the requirements of the applications and thus our system can then avoid the overhead of maintaining data in an inappropriate format. Having the information distributed over different data stores inside a single main-memory database engine allows combining the advantages of the different algorithms to best process their data with the speed of main memory database.

REFERENCES