Advanced Data Management (CSCI 680/490)

Graph Databases

Dr. David Koop
Recent History in Databases

- Early 2000s: Commercial DBs dominated, Open-source DBs missing features
- Mid 2000s: MySQL adopted by web companies
- Late 2000s: NoSQL does scale horizontally out of the box
- Early 2010s: New DBMSs that can scale across multiple machines natively and provide ACID guarantees
New SQL
NewSQL

- 451 Group’s Definition:
  - A DBMS that delivers the scalability and flexibility promised by NoSQL while retaining the support for SQL queries and/or ACID, or to improve performance for appropriate workloads.

- Stonebraker's Definition:
  - SQL as the primary interface
  - ACID support for transactions
  - Non-locking concurrency control
  - High per-node performance
  - Parallel, shared-nothing architecture
OLTP Workload

- **Operation Complexity**
  - Simple
  - Complex

- **Workload Focus**
  - OLTP
  - Social Networks
  - Data Warehouses

- **Axes**
  - Writes
  - Reads

References:
- A. Pavlo
Ideal OLTP System

- Main Memory Only
- No Multi-processor Overhead
- High Scalability
- High Availability
- Autonomic Configuration
Spanner Overview

• Focus on scaling databases focused on OLTP (not OLAP)
• Since OLTP, focus is on sharding **rows**
• Tries to satisfy CAP (which is impossible per CAP Theorem) by not worrying about 100% availability
• External consistency using multi-version concurrency control through timestamps
• ACID is important
• Structured: universe with zones with zone masters and then spans with span masters
• SQL-like (updates allow SQL to be used with Spanner)
Spanner and the CAP Theorem

- Which type of system is Spanner?
  - C: consistency, which implies a single value for shared data
  - A: 100% availability, for both reads and updates
  - P: tolerance to network partitions

- Which two?
  - CA: close, but not totally available
  - So actually CP
External Consistency

- Traditional DB solution: **two-phase locking**—no writes while client reads
- "The system behaves as if all transactions were executed sequentially, even though Spanner actually runs them across multiple servers (and possibly in multiple datacenters) for higher performance and availability" [Google]
- Semantically indistinguishable from a single-machine database
- Uses multi-version concurrency control (MVCC) using **timestamps**
- Spanner uses **TrueTime** to generate monotonically increasing timestamps across all nodes of the system
## Google Cloud Spanner: NewSQL

<table>
<thead>
<tr>
<th>CLOUD SPANNER</th>
<th>TRADITIONAL RELATIONAL</th>
<th>TRADITIONAL NON-RELATIONAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Consistency</td>
<td>Strong</td>
<td>Strong</td>
</tr>
<tr>
<td>Availability</td>
<td>High</td>
<td>Failover</td>
</tr>
<tr>
<td>Scalability</td>
<td>Horizontal</td>
<td>Vertical</td>
</tr>
<tr>
<td>Replication</td>
<td>Automatic</td>
<td>Configurable</td>
</tr>
</tbody>
</table>

[https://cloud.google.com/spanner/]
Spanner as "Effectively CA"

- Criteria for being "effectively CA"
  1. At a minimum it must have very high availability in practice (so that users can ignore exceptions), and
  2. As this is about partitions it should also have a low fraction of those outages due to partitions.
- Spanner meets both of these criteria
- Spanner relies on Google's network (private links between data centers)
- TrueTime helps create consistent snapshots, sometimes have a commit wait

[E. Brewer, 2017]
Throughput: Spanner vs. MySQL

Median Latency as Throughput Increases

- MySQL (median)
- spanner 9 nodes (median)
- spanner 15 nodes (median)
- spanner 30 nodes (median)

[P. Bakkum and D. Cepeda, 2017]
Assignment 4

• Work on Data Integration and Data Fusion
• Integrate artist datasets from different institutions (The Met, The Tate, Smithsonian, Carnegie Museum of Art)
  - Integrate information about names, places, nationality, etc.
• Record Matching:
  - Which artists are the same?
  - Which nationalities are the same? (British/English)
• Data Fusion:
  - Year of birth/death differences
  - Nationality differences
Test 2

- Wednesday, April 6
- Covers material from the beginning of course, emphasizing material since Test 1
- Similar Format to Test 1
- We have discussed more papers since Test 1
Specific Types of Data
Graphs: Social Networks

[Image of a world map representing social networks]

[P. Butler, 2010]
What is a Graph?

• An abstract representation of a set of objects where some pairs are connected by links.

Object (Vertex, Node)

Link (Edge, Arc, Relationship)
What is a Graph?

- In computing, a graph is an abstract **data structure** that represents set objects and their relationships as **vertices** and **edges/links**, and supports a number of graph-related **operations**.
- Objects (nodes): \{A, B, C, D\}
- Relationships (edges):
  \{ (D, B), (D, A), (B, C), (B, A), (C, A) \}
- Operation: shortest path from D to A

[K. Salama, 2016]
Different Kinds of Graphs

- Undirected Graph
- Directed Graph
- Pseudo Graph
- Multi Graph
- Hyper Graph

[M. De Marzi, 2012]
Graphs with Properties

- Each vertex or edge may have properties associated with it
- May include identifiers or classes

```
Person
name = 'Tom Hanks'
born = 1956

Person
name = 'Robert Zemeckis'
born = 1951

Movie
title = 'Forrest Gump'
released = 1994

ACTED_IN
roles = ['Forrest']

DIRECTED
```
Types of Graph Operations

• Connectivity Operations:
  - number of vertices/edges, in- and out-degrees of vertices
  - histogram of degrees can be useful in comparing graphs
• Path Operations: cycles, reachability, shortest path, minimum spanning tree
• Community Operations: clusters (cohesion and separation)
• Centrality Operations: degree, vulnerability, PageRank
• Pattern Matching: subgraph isomorphism
  - can use properties
  - useful in fraud/threat detection, social network suggestions

[K. Salama, 2016]
What is a Graph Database?

- A database with an explicit graph structure
- Each node knows its adjacent nodes
- As the number of nodes increases, the cost of a local step (or hop) remains the same
- Plus an Index for lookups
How do Graph Databases Compare?

Graph Databases

Document Databases

BigTable Clones

Key-Value Store

Relational Databases

90% of Use Cases

Complexity

Size

[Source: M. De Marzi, 2012]
Graph Databases Compared to Relational Databases

- Optimized for aggregation
- Optimized for connections

[M. De Marzi, 2012]
Graph Databases Compared to Key-Value Stores

Optimized for simple look-ups

Optimized for traversing connected data

[M. De Marzi, 2012]
Graph Databases Compared to Document Stores

Optimized for “trees” of data

Optimized for seeing the forest and the trees, and the branches, and the trunks

[M. De Marzi, 2012]
The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing

S. Sahu, A. Mhedhbi, S. Salihoglu, J. Lin, and M. T. Özsu
The Future is Big Graphs

S. Sakr et al

CACM
Insights for the Future of Graph Processing

- Graphs are ubiquitous abstractions enabling reusable computing tools for graph processing with applications in every domain.
- Diverse workloads, standard models and languages, algebraic frameworks, and suitable and reproducible performance metrics will be at the core of graph processing ecosystems in the next decade.
Pipeline for Graph Processing

Data flows left to right, from data source to output, via a series of functionally different processing steps. Feedback and loopbacks flow mainly through the blue (highlighted) arrows.
Graph Databases

D. Lembo and R. Rosati
Why Graph Database Models?

- Graphs have been long ago recognized as one of the most simple, natural and intuitive knowledge representation systems.
- Graph data structures allow for a natural modeling when data has graph structure.
- Queries can address directly and explicitly this graph structure.
- Implementation-wise, graph databases may provide special graph storage structures, and take advantage of efficient graph algorithms available for implementing specific graph operations over the data.

[R. Angles and C. Gutierrez, 2017]
### Relational Model

<table>
<thead>
<tr>
<th>NAME</th>
<th>LASTNAME</th>
<th>PERSON</th>
<th>PARENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>George</td>
<td>Jones</td>
<td>Julia</td>
<td>George</td>
</tr>
<tr>
<td>Ana</td>
<td>Stone</td>
<td>Julia</td>
<td>Ana</td>
</tr>
<tr>
<td>Julia</td>
<td>Jones</td>
<td>David</td>
<td>James</td>
</tr>
<tr>
<td>James</td>
<td>Deville</td>
<td>David</td>
<td>Julia</td>
</tr>
<tr>
<td>David</td>
<td>Deville</td>
<td>Mary</td>
<td>James</td>
</tr>
<tr>
<td>Mary</td>
<td>Deville</td>
<td>Mary</td>
<td>Julia</td>
</tr>
</tbody>
</table>

![Genealogy Diagram](image)

Figure 1: Example of a genealogy expressed in the relational model (i.e. as tables on the left) and a diagram of its scheme on the right.

Regarding simplicity, one of the most popularized models is the semi-structured model, which uses the most simple version of a graph, namely a tree, the most common and intuitive way of organizing our data (e.g. directories).

Finally, the most common models are slightly enhanced versions of the plain graphs. One of them, the RDF model, gives a light typing to nodes, and considers edges as nodes, giving uniformity to the information objects in the model. The other, the property graph model, allows to add properties to edges and nodes.

Next, we will present these models and show a paradigmatic example of each. We will use the genealogy toy example modeled as tables and a simple schema in Figure 1.

#### 3.1 The basics: Labeled graphs

The most basic data structure for graph database models is a directed graph with nodes and edges labeled by some vocabulary. A good example is Gram \[37\], a graph data model motivated by hypertext querying.

A schema in Gram is a directed labeled multigraph, where each node is labeled with a symbol called a **type**, which has associated a domain of values. In the same way, each edge has assigned a label representing a relation between types (see example in Figure 2). A feature of Gram is the use of regular expressions for explicit definition of paths called **walks**. An alternating sequence of nodes and edges represents a walk, which combined with other walks conforms other special objects called **hyperwalks**.

For querying the model (particularly path-like queries), an algebraic language based on regular expressions is proposed. For this purpose a hyper-

---

[R. Angles and C. Gutierrez, 2017]
Basic Labeled Model (Gram)

- Directed graph with nodes and edges labeled by some vocabulary
- Gram is a directed labeled multigraph
  - Each node is labeled with a symbol called a **type**
  - Each edge has assigned a label representing a **relation** between types

---

[R. Angles and C. Gutierrez, 2017]
Hypergraph Model (Groovy)

- Notion of edge is extended to **hyperedge**, which relates an arbitrary set of nodes.
- Hypergraphs allow the definition of complex objects (undirected), functional dependencies (directed), object-ID and (multiple) structural inheritance.

![Hypergraph Diagram](image)

[R. Angles and C. Gutierrez, 2017]
Hypernode Model

- Hypernode is a directed graph whose nodes can themselves be graphs (or hypernodes), allowing **nesting** of graphs
- **Encapsulates** information
Semistructured (Tree) Model: (OEM Graph)

- "Self-describing" data like JSON and XML
- OEM uses pointers to data in the tree

![OEM Syntax](image)

```
{ person : &p1 { name : "George" ,
    lastname : "Jones" } 

person : &p2 { name : "Ana" ,
    lastname : "Stone" } 

person : &p3 { name : "Julia" ,
    lastname : "Jones" ,
    parent : &p1 ,
    parent : &p2 } 

person : &p4 { name : "James" ,
    lastname : "Deville" } 

person : &p5 { name = "David",
    lastname : "Deville" ,
    parent : &p3 ,
    parent : &p4 } 

person : &p6 { name = "Mary",
    lastname : "Deville" ,
    parent : &p4 ,
    parent : &p5 } } 
```

[R. Angles and C. Gutierrez, 2017]
RDF (Triple) Model

- Interconnect resources in an extensible way using graph-like structure for data
- Schema and instance are **mixed** together
- SPARQL to query
- Semantic web
Property Graph Model (Cypher in neo4j)

- Directed, labelled, attributed multigraph
- Properties are **key/value pairs** that represent metadata for nodes and edges

![Property Graph Model Diagram]
Types of Graph Queries

• Adjacency queries (neighbors or neighborhoods)
• Pattern matching queries (related to graph mining)
  - Graph patterns with structural extension or restrictions
  - Complex graph patterns
  - Semantic matching
  - Inexact matching
  - Approximate matching
• Reachability queries (connectivity)

[R. Angles and C. Gutierrez, 2017]
Types of Graph Queries (continued)

- Analytical queries
  - Summarization queries
  - Complex analytical queries (PageRank, characteristic path length, connected components, community detection, clustering coefficient)

[R. Angles and C. Gutierrez, 2017]
Graph Structures

- W3C RDF Dataset
  - Hyper vertices

- ISO/IEC JQL
  - Property Graph Model
  - Multiple edge labels

- Neo4j/Oracle POOL
  - Property Graph Model
  - Multiple vertex labels

- Gremlin / MS Cosmos Property Graph Model
  - Single vertex labels, vertex properties, edge properties

- W3C RDF Graph
  - Multiple edges between a pair of nodes, single edge label

- Pregel/Giraph Graph
  - (used for e.g. Node2vec)

- Data Graph
  - (used for e.g. Page rank, Connected component)

- Weighted Graph
  - (used for e.g. Shortest path, Louvain modularity)

- Vertex data

- Directed Graph
  - Direction

- Simple Graph

[S. Sakr et al.]
Figure 8: Evolution of graph query languages: G [63], G+ [64], Graphlog, HPQL [104], THQL [141], GRE, HNQL, PORL [72], SLQL, HQL [137], PRPQ [107], GraphQL, SPARQL, RLV [132], Cypher [14], ECRPQ, PDQL, GX-Path, SPARQL 1.1 and RQ.

For the sake of space we will not present a complete review of graph query languages. Instead we describe some of the languages we consider relevant and useful to show the developments in the area. Moreover, we restrict our review to “pure” GQLs, that is those languages specifically designed to work with graph data models. Figure 8 presents this subset of languages in chronological order.

As we mentioned before, Cruz et al. [63] proposed the query language G. This language introduced the notion of graphical query as a set of query graphs. A query graph (pattern) is a labeled directed multigraph in which the node labels may be either variables or constants, and the edge labels can be regular expressions combining variables and constants. The result of a graphical query $Q$ with respect to a graph database $G$ is the union of all query graphs of $Q$ which match subgraphs of $G$. For instance, Figure 9 presents an example of graphical query containing two query graphs, $Q_1$ and $Q_2$. This query finds the first and last cities visited in all round trips from Toronto (“Tor”), in which the first and last flights are with Air Canada (“AC”) and all other flights (if any) are with the same airline. Note that the last condition is expressed by the edge labeled with regular expression $w^+$. Thanks to the inclusion of regular expressions, G is able to express recursive queries more general than transitive closure. However, the evaluation of queries in G is of high computational complexity due to its semantics based on simple paths.

G evolved into a more powerful language called G+ [64]. The notion of graphical query proposed by G is extended in G+ to define a summary

Graph Query Languages
Cypher

• Implemented by neo4j system
• Expresses reachability queries via path expressions
  \[-p = (a)-[:knows*]->(b): \text{nodes from } a \text{ to } b \text{ following } knows \text{ edges}\]
• START x=node:person(name="John")
  MATCH (x)-[:friend]->(y)
  RETURN y.name
SPARQL (RDF)

- Uses SELECT-FROM-WHERE pattern like SQL
- SELECT ?N
  FROM <http://example.org/data.rdf>
  WHERE { ?X rdf:type voc:Person . ?X voc:name ?N }

[R. Angles and C. Gutierrez, 2017]
Comparing Graph Database Systems: Features

Data Storage

<table>
<thead>
<tr>
<th>Graph Database</th>
<th>Main memory</th>
<th>External memory</th>
<th>Backend Storage</th>
<th>Indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllegroGraph</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>DEX</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Filament</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>G-Store</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>HyperGraphDB</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>InfiniteGraph</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Neo4j</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Sones</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>vertexDB</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
</tbody>
</table>

Operations/Manipulation

<table>
<thead>
<tr>
<th>Graph Database</th>
<th>Data Definition Language</th>
<th>Data Manipulat. Language</th>
<th>Query Language</th>
<th>API</th>
<th>GUI</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllegroGraph</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>DEX</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Filament</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>G-Store</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>HyperGraphDB</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>InfiniteGraph</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Neo4j</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Sones</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>vertexDB</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
</tbody>
</table>

[R. Angles, 2012]
Comparing Graph Database Systems: Representation

**Graph Data Structures**

<table>
<thead>
<tr>
<th>Graph Database</th>
<th>Graphs</th>
<th>Nodes</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple graphs</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Hypergraphs</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Nested graphs</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Attributed graphs</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Node labeled</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Node labeled</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Directed</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Edge labeled</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Edge labeled</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

**Entites & Relations**

<table>
<thead>
<tr>
<th>Graph Database</th>
<th>Node types</th>
<th>Property types</th>
<th>Relation types</th>
<th>Object nodes</th>
<th>Value nodes</th>
<th>Complex nodes</th>
<th>Object relations</th>
<th>Simple relations</th>
<th>Complex relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllegroGraph</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>DEX</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Filament</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>G-Store</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>HyperGraphDB</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>InfiniteGraph</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Neo4j</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Sones</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>vertexDB</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

[R. Angles, 2012]
study of their complexity and expressive power difficult.

data. Neo4j is developing Cypher, a query language for RDF. SPARQL is based on graph pattern matching languages are not frequent in current graph databases. In fact, in order to specify the name of the entity. The same applies for object-oriented concepts (e.g., IDs for objects) for representing simple model for representing graph data. The inclusion of semi-structure data models). For example, the definition of a than a flexible schema. Moreover, an evolving schema can be data consistency in a database is equal or even more important constraints (for preserving the consistency of the database).

AllegroGraph supports SPARQL, the standard query lan-

HyperGraphDB

G-Store

InfiniteGraph

Neo4j

Sones

vertexDB


Comparing Graph Database Systems: Queries

<table>
<thead>
<tr>
<th>Graph Database</th>
<th>Query Lang.</th>
<th>API</th>
<th>Graphical Q. L.</th>
<th>Retrieval</th>
<th>Reasoning</th>
<th>Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllegroGraph</td>
<td>o</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>DEX</td>
<td>●</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Filament</td>
<td>●</td>
<td></td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>G-Store</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HyperGraphDB</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InfiniteGraph</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neo4j</td>
<td>o</td>
<td>●</td>
<td></td>
<td>●</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>Sones</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>vertexDB</td>
<td>●</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Graph Database</th>
<th>Adjacency</th>
<th>Reachability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Node/edge adjacency</td>
<td>k-neighborhood</td>
</tr>
<tr>
<td>Allegro</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>DEX</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Filament</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>G-Store</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>HyperGraph</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Infinite</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Neo4j</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Sones</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>vertexDB</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

[R. Angles, 2012]
The (sorry) State of Graph Database Systems

Peter Boncz

Keynote, EDBT-ICDT 2022