

The future is big graphs: a community view on graph processing systems

Journal Article

Author(s):

Sakr, Sherif; Bonifati, Angela; Voigt, Hannes; Iosup, Alexandru; Ammar, Khaled; Angles, Renzo; Aref, Walid; Arenas, Marcelo; Besta, Maciej; Boncz, Peter A.; Daudjee, Khuzaima; Della Valle, Emanuele; Dumbrava, Stefania; Hartig, Olaf; Haslhofer, Bernhard; Hegeman, Tim; Hidders, Jan; Hose, Katja; Iamnitchi, Adriana; Kalavri, Vasiliki; Kapp, Hugo; Martens, Wim; Özsu, Tamer M.; Peukert, Eric; Plantikow, Stefan; Ragab, Mohamed; Ripeanu, Matei R.; Salihoglu, Semih; Schulz, Christian; Selmer, Petra; Sequeda, Juan F.; Shinavier, Joshua; Szárnyas, Gábor; Tommasini, Riccardo; Tumeo, Antonino; Uta, Alexandru; Varbanescu, Ana L.; Wu, Hsiang-Yun; Yakovets, Nikolay; Yan, Da; Yoneki, Eiko

Publication date:

2021-09

Permanent link:

<https://doi.org/10.3929/ethz-b-000504426>

Rights / license:

[Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International](#)

Originally published in:

Communications of the ACM 64(9), <https://doi.org/10.1145/3434642>

CONTRIBUTORS

KHALED AMMAR
Borialis AI

RENZO ANGLES
University of Talca

WALID AREF
Purdue University

MARCELO ARENAS
PUC & IMFD

MACIEJ BESTA
ETH Zürich

PETER A. BONCZ
CWI

KHUZAIMA DAUDJEE
University of Waterloo

EMANUELE DELLA VALLE
Polytechnic University of Milan

STEFANIA DUMBRAVA
ENSIE

OLAF HARTIG
Linköping University

BERNHARD HASLHOFER
Austrian Institute of Technology

TIM HEGEMAN
VU University Amsterdam

JAN HIDDERS
Birkbeck, University of London

KATJA HOSE
Aalborg University

ADRIANA IAMNITCHI
University of South Florida

VASILIKI KALAVRI
Boston University

HUGO KAPP
Oracle Labs Switzerland

WIM MARTENS
Universität Bayreuth

M. TAMER ÖZSU
University of Waterloo

ERIC PEUKERT
Universität Leipzig

STEFAN PLANTIKOW
Neo4j

MOHAMED RAGAB
University of Tartu

MATEI R. RIPEANU
University of British Columbia

SEMIH SALIHOGLU
University of Waterloo

CHRISTIAN SCHULZ
Heidelberg University and Universität Wien

PETRA SELMER
Neo4j

JUAN F. SEQUEDA
data.world

JOSHUA SHINAVIER
Uber Engineering

GÁBOR SZÁRNYAS
Budapest Univ. of Technology and Economics

RICCARDO TOMMASINI
University of Tartu

ANTONINO TUMEO
Pacific Northwest National Lab

ALEXANDRU UTA
VU University Amsterdam

ANA LUCIA VARBANESCU
University of Amsterdam

HSIANG-YUN WU
TU Wien

NIKOLAY YAKOVETS
TU Eindhoven

DA YAN
The University of Alabama

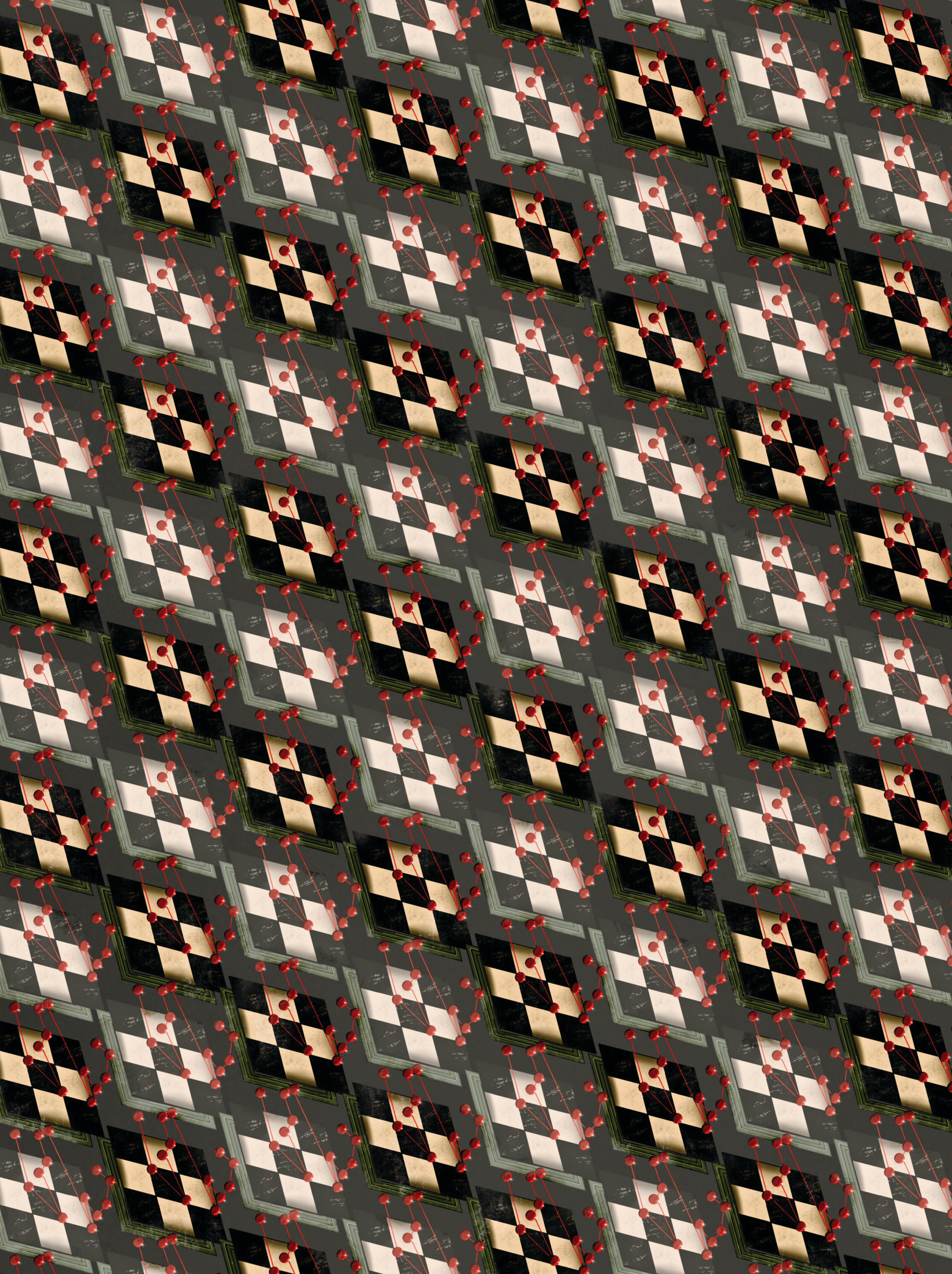
EIKO YONEKI
University of Cambridge

Ensuring the success of big graph processing for the next decade and beyond.

BY SHERIF SAKR, ANGELA BONIFATI,
HANNES VOIGT, AND ALEXANDRU IOSUP

The Future Is Big Graphs: A Community View on Graph Processing Systems

GRAPHS ARE, BY nature, ‘unifying abstractions’ that can leverage interconnectedness to represent, explore, predict, and explain real- and digital-world phenomena. Although real users and consumers of graph instances and graph workloads understand these abstractions, future problems will require new abstractions and systems. What needs to happen in the next decade for big graph processing to continue to succeed?



A Joint Effort by the Computer Systems and Data Management Communities

The authors of this article met in Dec. 2019 in Dagstuhl for Seminar 19491 on Big Graph Processing Systems.^a The seminar gathered a diverse group of 41 high-quality researchers from the data management and large-scale-systems communities. It was an excellent opportunity to start the discussion about next-decade opportunities and challenges for graph processing.

This is a community publication. The first four authors co-organized the community event leading to this article and coordinated the creation of this manuscript. All other authors contributed equally to this research. Unfortunately, Sherif Sakr passed away during the period following the event and the completion of this article. This article is published in memoriam.

a <https://www.dagstuhl.de/en/program/calendar/semhp/?semnr=19491>

We are witnessing an unprecedented growth of interconnected data, which underscores the vital role of graph processing in our society. Instead of a single, exemplary (“killer”) application, we see big graph processing systems underpinning many emerging but already complex and diverse data management ecosystems, in many areas of societal interest.^a

To name only a few recent, remarkable examples, the importance of this field for practitioners is evidenced by the large number (more than 60,000) of people registered^b to download the Neo4j book *Graph Algorithms*^c in just over one-and-a-half years, and by the enormous interest in the use of graph processing in the artificial intelligence (AI) and machine learning (ML) fields.^d

a As indicated by a user survey¹² and by a systematic literature survey of 18 application domains, including biology, security, logistics and planning, social sciences, chemistry, and finance. See <http://arxiv.org/abs/1807.00382>

b See <https://app.databox.com/datawall/551f309602080e2b2522f7446a20adb705cabbde8>

c See <https://www.oreilly.com/library/view/graph-algorithms/9781492047674/>

d Many highly cited articles support this statement, including “Inductive Representation Learning on Large Graphs” by W. Hamilton et al. (2017) and “DeepWalk: Online Learning of Social Representations” by B. Perozzi et al. (2014); <https://arxiv.org/pdf/1403.6652.pdf>

» key insights

- 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.

Furthermore, the timely Graphs 4 COVID-19 initiative^e is evidence of the importance of big graph analytics in alleviating the pandemic.

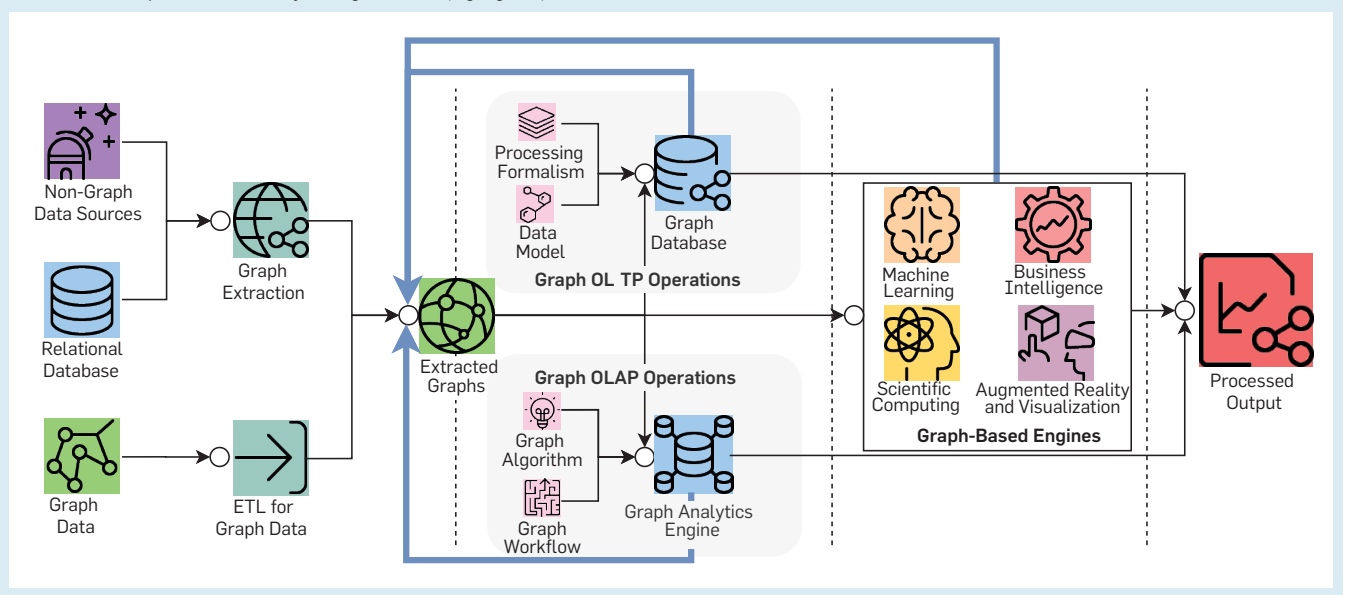
Academics, start-ups, and even big tech companies such as Google, Facebook, and Microsoft have introduced various systems for managing and processing the growing presence of big graphs. Google’s PageRank (late 1990s) showcased the power of Web-scale graph processing and motivated the development of the MapReduce programming model, which was originally used to simplify the construction of the data structures used to handle searches, but has since been used extensively outside of Google to implement algorithms for large-scale graph processing.

Motivated by scalability, the 2010 Google Pregel “think-like-a-vertex” model enabled distributed PageRank

e See <https://neo4j.com/graphs4good/covid-19/>

Figure 1. Illustration of a complex data 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.



computation, while Facebook, Apache Giraph, and ecosystem extensions support more elaborate computational models (such as task-based and not always distributed) and data models (such as diverse, possibly streamed, possibly wide-area data sources) useful for social network data. At the same time, an increasing number of use cases revealed RDBMS performance problems in managing highly connected data, motivating various startups and innovative products, such as Neo4j, Sparksee, and the current Amazon Neptune. Microsoft Trinity and later Azure SQL DB provided an early distributed database-oriented approach to big graph management.

The diversity of models and systems led initially to the fragmentation of the market and a lack of clear direction for the community. Opposing this trend, we see promising efforts to bring together the programming languages, ecosystem structure, and performance benchmarks. As we have argued, there is no killer application that can help to unify the community.

Co-authored by a representative sample of the community (see the sidebar, “A Joint Effort by the Computer Systems and Data Management Communities”), this article addresses the questions: What do the next-decade big-graph processing systems look like from the perspectives of the data management and the large-scale-systems communities?^f What can we say today about the guiding design principles of these systems in the next 10 years?

Figure 1 outlines the complex pipeline of future big graph processing systems. Data flows in from diverse sources (already graph-modeled as well as non-graph-modeled) and is persisted, managed, and manipulated with online transactional processing (OLTP) operations, such as insertion, deletion, updating, filtering, projection, joining, uniting, and intersecting. The data is then analyzed, enriched, and condensed with online analytical processing (OLAP) operations, such as grouping, aggregating, slicing, dicing, and rollup. Finally, it is disseminated and consumed by a variety of applications, including machine learning, such as



What needs to happen in the next decade for big graph processing to continue to succeed?



ML libraries and processing frameworks; business intelligence (BI), such as report generating and planning tools; scientific computing; visualization; and augmented reality (for inspection and interaction by the user). Note that this is not typically a purely linear process and hybrid OLTP/OLAP processes can emerge. Considerable complexity stems from (intermediate) results being fed back into early-process steps, as indicated by the blue arrows.

As an example, to study coronaviruses and their impact on human and animal populations (for example, the COVID-19 disease), the pipeline depicted in Figure 1 could be purposed for two major kinds of analysis: network-based ‘omics’ and drug-related search, and network-based epidemiology and spread-prevention. For the former, the pipeline could have the following steps:

1. Initial genome sequencing leads to identifying similar diseases.
2. Text (non-graph data) and structured (database) searches help identify genes related to the disease.
3. A network treatment coupled with various kinds of simulations could reveal various drug targets and valid inhibitors, and might lead to effective prioritization of usable drugs and treatments.

For the latter, social media and location data, and data from other privacy-sensitive sources, could be combined into social interaction graphs, which could be traversed to establish super-spreaders and super-spreading events related to them, which could result in the establishment of prevention policies and containment actions. However, the current generation of graph processing technology cannot support such a complex pipeline.

For instance, on the COVID-19 knowledge graph,^g useful queries can be posed against individual graphs^h inspecting the papers, patents, genes, and most influential COVID-19 authors. However, inspecting several data sources in a full-fledged graph processing pipeline across multiple graph datasets, as illustrated in Figure 1, raises many challenges for current graph da-

^g See <https://covidgraph.org/>

^h See <https://github.com/covidgraph/documentation/blob/master/helpful-queries.md>

^f The summary of the Dagstuhl seminar. See <https://www.dagstuhl.de/19491>

tabase technology. In this article, we formulate these challenges and build our vision for next-generation, big-graph processing systems by focusing on three major aspects: *abstractions*, *ecosystems*, and *performance*. We present expected data models and query languages, and inherent relationships among them in lattice of abstractions and discuss these abstractions and the flexibility of lattice structures to accommodate future graph data models and query languages. This will solidify the understanding of the fundamental

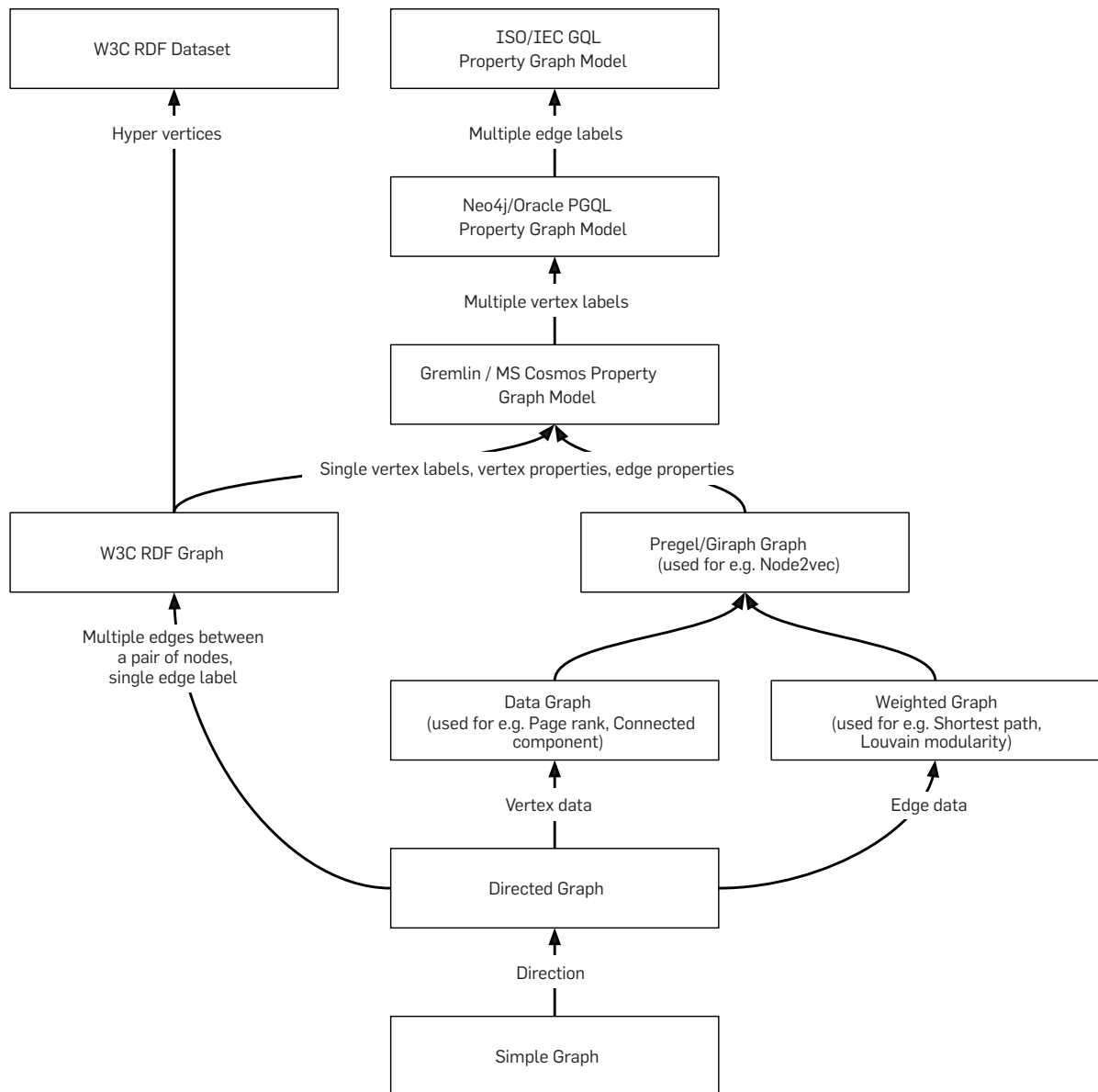
principles of graph data extraction, exchange, processing, and analysis, as illustrated in Figure 1.

A second important element, as we will discuss, is the vision of an ecosystem governing big graph processing systems and enabling the tuning of various components, such as OLAP/OLTP operations, workloads, standards, and performance needs. These aspects make the big processing systems more complicated than what was seen in the last decade. Figure 1 provides a high-level

perception of this complexity in terms of inputs, outputs, processing needs, and final consumption of graph data.

A third element is how to understand and control performance in these future ecosystems. We have important performance challenges to overcome, from methodological aspects about performing meaningful, tractable, and reproducible experiments to practical aspects regarding the trade-off of scalability with portability and interoperability.

Figure 2. Example lattice shows graph data model variants with their model characteristics.⁸



Abstractions

Abstractions are widely used in programming languages, computational systems, and database systems, among others, to conceal technical aspects in favor of more user-friendly, domain-oriented logical views. Currently, users have to choose from a large spectrum of graph data models that are similar, but differ in terms of expressiveness, cost, and intended use for querying and analytics. This ‘abstraction soup’ poses significant challenges to be solved in the future.


Understanding data models. Today, graph data management confronts many data models (directed graphs, RDF, variants of property graphs, and so on) with key challenges: deciding which data model to choose per use case and mastering interoperability of data models where data from different models is combined (as in the left-hand side of Figure 1).

Both challenges require a deeper understanding of data models regarding:


1. How do humans conceptualize data and data operations? How do data models and their respective operators support or hinder the human thought process? Can we measure how “natural” or “intuitive” data models and their operators are?

2. How can we quantify, compare, and (partially) order the (modeling and operational) *expressive power* of data models? Concretely, Figure 2 illustrates a lattice for a selection of graph data models. Read bottom-up, this lattice shows which characteristic has to be added to a graph data model to obtain a model of richer expressiveness. The figure also underlines the diversity of data models used in theory, algorithms, standards, and relevant industry systems. How do we extend this comparative understanding across multiple data model families, such as graph, relational, or document? What are the costs and benefits of choosing one model over another?

3. Interoperability between different data models can be achieved through mappings (semantic asser-



We are witnessing an unprecedented growth of interconnected data, which underscores the vital role of graph processing in our society.



tions across concepts in different data models) or with direct translations (for instance, W3C’s R2RML). Are there general ways or building blocks for expressing such mappings (category theory, for example)?

Studying (1) requires foremost investigators working with data and data models, which is uncommon in the data management field and should be conducted collaboratively with other fields, such as human-computer interaction (HCI). Work on HCI and graphs exists, for example, in HILDA workshops at Sigmod. However, these are not exploring the search space of graph data models.

Studying (2) and (3) can build on existing work in database theory, but can also leverage findings from neighboring computer science communities on comparison, featurization, graph summarization, visualization, and model transformation. As an example, graph summarization²² has been widely exploited to provide succinct representations of graph properties in graph mining¹ but they have seldom been used by graph processing systems to make processing more efficient, more effective, and more user centered. For instance, approximate query processing for property graphs cannot rely on sampling as done by its relational counterpart and might need to use quotient summaries for query answering.

Logic-based and declarative formalisms. Logic provides a unifying formalism for expressing queries, optimizations, integrity constraints, and integration rules. Starting from Codd’s seminal insight relating logical formulae to relational queries,¹² many first order (FO) logic fragments have been used to formally define query languages with desirable properties such as decidable evaluation. Graph query languages are essentially a syntactic variant of FO augmented with *recursive* capabilities.

Logic provides a yardstick for *reasoning* about graph queries and graph constraints. Indeed, a promising line of research is the application of formal tools, such as model checking, theorem proving,¹⁵ and testing to establish the *functional correctness* of complex graph processing systems, in general, and of graph database systems, in particular.

The influence of logic is pivotal not

i The figure does not aim to provide a complete list of Graph DBMS products. Please consult, for example, <https://db-engines.com/en/ranking/graph+dbms> and other market surveys for comprehensive overviews.

Known Properties of Graph Processing Workloads

Graph workloads may exhibit several properties:

1. Graph workloads are useful for many, vastly diverse domains.^{24,25,26} Notable features include edge orientation, such as properties/timestamps for edges and nodes; graph methods (neighborhood statistics, pathfinding and traversal, and subgraph mining); programming models (think-like-a-vertex, think-like-an-edge, and think-like-a-subgraph); diverse graph sizes, including trillion-edge graphs;²⁶ and query and process selectivities.⁹
2. Graph workloads can be highly irregular, mixing (short-term) data-intensive and compute-intensive phases.²⁶ The source of irregularity, such as different datasets, algorithms, and computing platforms, greatly affects performance. Their interdependency forms the Hardware-Platform-Algorithm-Dataset (HPAD) Law.²⁹
3. Graph processing uses a complex pipeline, combining a variety of tasks other than querying and algorithms.^{1,24} From traditional data management, workloads include: transactional (OLTP) workloads in multi-user environments, with many short, discrete, likely atomic transactions; and analytical (OLAP) workloads with fewer users but complex and resource-intensive queries or processing jobs, with longer runtime (minutes). Popular tasks also include extract, transform, load (ETL); visualization; cleaning; mining; and debugging and testing, including synthetic graph generation.
4. Scalability, interactivity, and usability affect how graph users construct their workloads.²⁴

only to database languages, but also as a foundation for combining logical reasoning with statistical learning in AI. Logical reasoning derives categorical notions about a piece of data by logical deduction. Statistical learning derives categorical notions by learning statistical models on known data and applying it to new data. Both leverage the topological structure of graphs (ontologies and knowledge graphs^j or graph em-

beddings such as Node2vec^d to produce better insights than on non-connected data). However, both happen to be isolated. Combining both techniques can lead to crucial advancements.

As an example, deep learning (unsupervised feature learning) applied to graphs allows us to infer structural regularities and obtain meaningful representations for graphs that can be further leveraged by indexing and querying mechanisms in graph databases and exploited for logical reasoning. As another example, probabilistic models and causal relationships can be naturally encoded in property graphs and are the basis of advanced-graph neural networks.^k Property graphs allow us to synthesize more accurate models for ML pipelines, thanks to their inherent expressivity and embedded domain knowledge.

These considerations unveil important open questions as follows: How can statistical learning, graph processing, and reasoning be combined and integrated? Which underlying formalisms make this possible? How can we weigh between the two mechanisms?

Algebraic operators for graph processing. Currently, there is no standard graph algebra. The outcome of the Graph Query Language (GQL) Standardization Project could influence the design of a graph algebra alongside existing and emerging use cases.²⁵ However, next-generation graph processing systems should address questions about their algebraic components.

What are the fundamental operators of this algebra compared to other algebras (relation, group, quiver or path, incidence, or monadic algebra comprehensions)? What core graph algebra should graph processing systems support? Are there graph analytical operators to include in this algebra? Can this graph algebra be combined and integrated with an algebra of types to make type-systems more expressive and to facilitate type checking?

A “relational-like” graph algebra able to express all the first-order queries¹¹ and enhanced with a graph pat-

tern-matching operator¹⁶ seems like a good starting point. However, the most interesting graph-oriented queries are *navigational*, such as reachability queries, and cannot be expressed with limited recursion of relational algebra.^{3,8} Furthermore, relational algebra is a closed algebra; that is, input(s) and output of each operator is a relation, which makes relational algebra operators composable. Should we aim for a closed-graph algebra that encompasses both relations and graphs?

Current graph query engines combine algebra operators and ad hoc graph algorithms into complex workloads, which complicates implementation and affects performance. An implementation based on a single algebra also seems utopic. A query language with general Turing Machine capabilities (like a programming language), however, entails tractability and feasibility problems.² Algebraic operators that work in both centralized and distributed environments, and that can be exploited by both graph algorithms and ML models such as GNNs, graphlets, and graph embeddings, could be highly desirable for the future.

Ecosystems

Ecosystems behave differently from mere systems of systems; they couple many systems developed for different purposes and with different processes. Figure 1 exemplifies the complexity of a graph processing ecosystem through high-performance OLAP and OLTP pipelines working together. What are the ecosystem-related challenges?

Workloads in graph processing ecosystems. Workloads affect both the functional requirements (what a graph processing ecosystem will be able to do) and the non-functional (how well). Survey data²⁵ points to pipelines, as in Figure 1: complex workflows, combining heterogeneous queries and algorithms, managing and processing diverse datasets, with characteristics summarized in the sidebar “Known Properties of Graph Processing Workloads.”

In Figure 1, graph processing links to general processing, including ML, as well as to domain-specific processing ecosystems, such as simulation and numerical methods in science

j A recent practical example is the COVID-19 Knowledge Graph: <https://covidgraph.org/>

k “A Comprehensive Survey on Graph Neural Networks” by Z. Wu et al, 2019; abs/1901.00596.

and engineering, aggregation and modeling in business analytics, and ranking and recommendation in social media.

Standards for data models and query languages. Graph processing ecosystem standards can provide a common technical foundation, thereby increasing the mobility of applications, tooling, developers, users, and stakeholders. Standards for both OLTP and OLAP workloads should standardize the data model, the data manipulation and data definition language, and the exchange formats. They should be easily adoptable by existing implementations and also enable new implementations in the SQL-based technological landscape.

It is important that standards reflect existing industry practices by following widely used graph query languages. To this end, ISO/IEC started the GQL Standardization Project in 2019 to define GQL as a new graph query language. GQL is backed by 10 national standards bodies with representatives from major industry vendors and support from the property graph community as

represented by the Linked Data Benchmarks Council (LDBC).¹

With an initial focus on transactional workloads, GQL will support composable graph querying over multiple, possibly overlapping, graphs using enhanced regular path queries (RPQs),³ graph transformation (views), and graph updating capabilities. GQL enhances RPQs with pattern quantification, ranking, and path-aggregation. Syntactically, GQL combines SQL style with visual graph patterns pioneered by Cypher.¹⁴

Long-term, it would also be worthwhile to standardize building blocks of graph algorithms, analytical APIs and workflow definitions, graph embedding techniques, and benchmarks.²⁸ However, broad adoption for these aspects requires maturation.

Reference architecture. We identify the challenge of defining a reference architecture for big graph processing. The early definition of a reference architecture has greatly benefited the discussion around the design, develop-

ment, and deployment of cloud and grid computing solutions.¹³

For big graph processing, our main insight is that many graph processing ecosystems match the common reference architecture of datacenters,¹⁸ from which Figure 3 derives. The Spark ecosystem depicted here is one among thousands of possible instantiations. The challenge is to capture the evolving graph processing field.

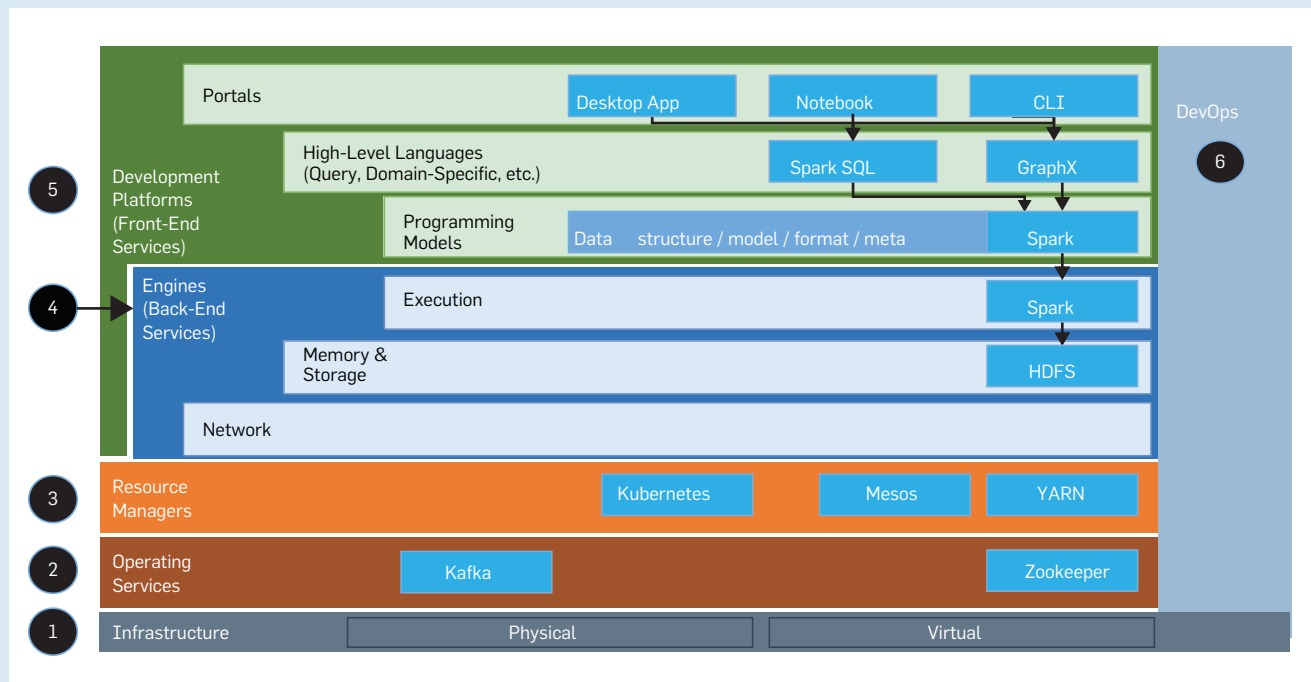
Beyond scale-up vs. scale-out. Many graph platforms focus either on scale-up or scale-out. Each has relative advantages.²⁷ Beyond merely reconciling scale-up and scale-out, we envision a *scalability continuum*: given a diverse workload, the ecosystem would automatically decide how to run it, and on what kind of heterogeneous infrastructure, meeting service-level agreements (SLAs).

Numerous mechanisms and techniques exist to enforce scale-up and scale-out decisions, such as data and work partitioning, migration, offloading, replication, and elastic scaling. All decisions can be taken statically or dynamically, using various optimization and learning techniques.

1 See <http://ldbncouncil.org/>

Figure 3. A reference architecture for graph processing ecosystems.

Layer 1, the infrastructure layer, provides physical and virtual resources. Layer 2, the operating services layer, provides services across resources, including data streaming and synchronization. Resource managers, in layer 3, provide static and dynamic resource management and scheduling across resources. Back-end and front-end layers (layers 4 and 5, respectively) represent specialization efforts. Conversely, layers 2 and 3 may generalize techniques initially developed in layers 4 and 5.



Dynamic and streaming aspects.


Future graph processing ecosystems should cope with dynamic and streaming graph data. A *dynamic graph* extends the standard notion of a graph to account for updates (insertions, changes, deletions) such that the current and previous states can be seamlessly queried. Streaming graphs can grow indefinitely as new data arrives. They are typically unbounded, thus the underlying systems are unable to keep the entire graph state. The sliding window semantics⁶ allow the two notions to be unified, with insertions and deletions being considered as arrivals and removals from the window.

Since current streaming processing technologies are fairly simple, for instance aggregations and projections as in industrial graph processing libraries (such as Gelly on Apache Flink), the need for “complex graph data streams” is evident, along with more advanced graph analytics and ML ad hoc operators. Another research challenge is to identify the graph-query processing operators that can be evaluated on dynamic and streaming graphs while taking into account recursive operators^{7,23} and path-oriented semantics, as needed for standard query languages such as GQL and G-Core.⁴


Graph processing platforms are also dynamic; discovering, understanding, and controlling the *dynamic phenomena* that occur in complex graph processing ecosystems is an open challenge. As graph processing ecosystems become more mainstream and are embedded in larger data-processing pipelines, we expect to increasingly observe known systems phenomena, such as performance variability, the presence of cascading failures, and autoscaling resources. What new phenomena will emerge? What programming abstractions²⁰ and systems techniques can respond to them?

Performance

Graph processing raises unique performance challenges, from the lack of a widely used performance metric other than response time to the methodological problem of comparing graph processing systems across architectures and tuning processes to performance portability and reproducibility. Such



Instead of a single, exemplary (“killer”) application, we see big graph processing systems underpinning many emerging but already complex and diverse data management ecosystems.



challenges become even more daunting for graph processing ecosystems.

Benchmarks, performance measurement, and methodological aspects. Graph processing suffers from methodological issues similar to other computing disciplines.^{5,24} Running comprehensive graph processing experiments, especially at scale, lacks tractability⁹—that is, the ability to implement, deploy, and experiment within a reasonable amount of time and cost. As in other computing disciplines,^{5,24} we need new, reproducible, experimental methodologies.

Graph processing also raises unique challenges in performance measurement and benchmarking related to complex workloads and data pipelines (Figure 1). Even seemingly minute HPAD variations, for example the graph’s degree distribution, can have significant performance implications.^{17,26} The lack of interoperability hinders fair comparisons and benchmarking. Indexing and sampling techniques might prove useful to improve and predict the runtime and performance of graph queries,^{8,21,30} challenging the communities of large-scale systems, data management, data mining, and ML.

Graph processing systems rely on complex runtimes that combine software and hardware platforms. It can be a daunting task to capture system-under-test performance—including parallelism, distribution, streaming vs. batch operation—and test the operation of possibly hundreds of libraries, services, and runtime systems present in real-world deployments.

We envision a combination of approaches. As in other computing disciplines,^{5,24} we need new, reproducible experimental methodologies. Concrete questions arise: How do we facilitate quick yet meaningful performance testing? How do we define more faithful metrics for executing a graph algorithm, query, program, or workflow? How can we generate workloads with combined operations, covering temporal, spatial, and streaming aspects? How do we benchmark pipelines, including ML and simulation? We also need organizations such as the LDBC to curate benchmark sharing and to audit benchmark usage in practice.

Specialization vs. portability and

interoperability. There is considerable tension between specializing graph processing stacks for performance reasons and enabling productivity for the domain scientist, through portability and interoperability.

Specialization, through custom software and especially hardware acceleration, leads to significant performance improvements. Specialization to graph workloads, as noted in the sidebar, focuses on diversity and irregularity^m in graph processing: sheer dataset-scale (addressed by Pregel and later by the open source project, Giraph), the (truncated) power-law-like distributions for vertex degrees (PowerGraph), localized and community-oriented updates (GraphChi), diverse vertex-degree distributions across datasets (PGX.D, PowerLya), irregular or non-local vertex access (Mosaic), affinity to specialized hardware (the BGL family, HAGGLE, rapids.ai), and more.

The high-performance computing domain proposed specialized abstractions and C++ libraries for them, and high-performance and efficient runtimes across heterogeneous hardware. Examples include BGL,²⁸ CombBLAS, and GraphBLAS. Data management approaches, including Neo4j, GEMS,¹⁰ and Cray’s Urika, focus on convenient query languages such as SPARQL and Cypher to ensure portability. Ongoing work also focuses on (custom) accelerators.

Portability through reusable components seems promising, but no standard graph library or query language currently exists. More than 100 big graph processing systems exist, but they do not support portability: graph systems will soon need to support constantly evolving processes.

Lastly, interoperability means integrating graph processing into broader workflows with multi-domain tools. Integration with ML and data mining processes, and with simulation and decision-making instruments, seems vital but is not supported by existing frameworks.

A memex for big graph processing systems. Inspired by Vannevar Bush’s

^m Irregularity could be seen as the opposite of the locality principle commonly leveraged in computing.

1940s concept of personal memex, and by a 2010s specialization into a Distributed Systems Memex,¹⁹ we posit that it would be both interesting and useful to create a Big Graph Memex for collecting, archiving, and retrieving meaningful operational information about such systems. This could be beneficial for learning about and eradicating performance and related issues, to enable more creative designs and extend automation, and for meaningful and reproducible testing, such as feedback building-block in smart graph processing.

Conclusion

Graphs are a mainstay abstraction in today’s data-processing pipelines. How can future big graph processing and database systems provide highly scalable, efficient, and diversified querying and analytical capabilities, as demanded by real-world requirements?

To tackle this question, we have undertaken a community approach. We started through a Dagstuhl Seminar and, shortly after, shaped the structured connections presented here. We have focused in this article on three interrelated elements: abstractions, ecosystems, and performance. For each of these elements, and across them, we have provided a view into what’s next.

Only time can tell if our predictions provide worthwhile directions to the community. In the meantime, join us in solving the problems of big graph processing. The future is big graphs. **□**

References

1. Aggarwal, C.C. and Wang, H. Managing and mining graph data. *Advances in Database Systems 40*. Springer, (2010).
2. Aho, A.V. and Ullman, J.D. Universality of data retrieval languages. In *Proceedings of the 6th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages* (1979) 110–119.
3. Angles, R. et al. Foundations of modern query languages for graph databases. *ACM Computing Surveys 50*, 5 (2017), 68:1–68:40.
4. Angles, R. et al. G-CORE: A core for future graph query languages. *SIGMOD Conf.* (2018), 1421–1432.
5. Angriman, E. et al. Guidelines for experimental algorithmics: A case study in network analysis. *Algorithms 12*, 7 (2019), 127.
6. Babcock, B., Babu S., Datar, M., Motwani, R., and Widom, J. Models and issues in data stream systems. *PODS* (2002), 1–16.
7. Bonifati, A., Dumbrava, S., and Gallego Arias, E.J. Certified graph view maintenance with regular datalog. *Theory Pract. Log. Program.* 18, 3–4 (2018), 372–389.
8. Bonifati, A., Fletcher, G.H.L., Voigt, H., and Yakovets, N. Querying graphs. *Synthesis Lectures on Data Management*. Morgan & Claypool Publishers (2018).
9. Bonifati, A., Holubová, I., Prat-Pérez, A., and Sakr, S. Graph generators: State of the art and open


- challenges. *ACM Comput. Surv.* 53, 2 (2020), 36:1–36:30.
10. Castellana, V.G. et al. In-memory graph databases for web-scale data. *IEEE Computer* 48, 3 (2015), 24–35.
11. Chandra, A.K. Theory of database queries. *PODS* (1988), 1–9.
12. Codd, E.F. A relational model of data for large shared data banks. *Commun. ACM* 13, 6 (June 1970), 377–387.
13. Foster, I. and Kesselman, C. *The Grid 2: Blueprint for a New Computing Infrastructure*. Elsevier (2003).
14. Francis, N. et al. Cypher: An evolving query language for property graphs. *SIGMOD Conference* (2018), 1433–1445.
15. Gonthier, G. et al. A machine-checked proof of the odd order theorem. *Intern. Conf. Interactive Theorem Proving* (2013), 163–179.
16. He, H. and Singh, A.K. Graphs-at-a-time: Query language and access methods for graph databases. *SIGMOD Conference* (2008), 405–418.
17. Iosup, A. et al. LDBC Graphalytics: A benchmark for large-scale graph analysis on parallel and distributed platforms. In *Proc. VLDB Endow.* 9, 13 (2016), 1317–1328.
18. Iosup, A. et al. Massivizing computer systems: A vision to understand, design, and engineer computer ecosystems through and beyond modern distributed systems. *ICDCS* (2018), 1224–1237.
19. Iosup, A. et al. The AtLarge vision on the design of distributed systems and ecosystems. *ICDCS* (2019), 1765–1776.
20. Kalavri, V., Vlassov, V., and Haridi, S. High-level programming abstractions for distributed graph processing. *IEEE Trans. Knowl. Data Eng.* 30, 2 (2018), 305–324.
21. Leskovec, J. and Faloutsos, C. Sampling from large graphs. *KDD* (2006), 631–636.
22. Liu, Y., Safavi, T., Dighe, A., and Koutra, D. Graph summarization methods and applications: A survey. *ACM Comput. Surv.* 51, 3 (2018) 62:1–62:34.
23. Pacaci, A., Bonifati, A., and Özsu, M.T. Regular path query evaluation on streaming graphs. *SIGMOD Conf.* (2020), 1415–1430.
24. Papadopoulos, A.V. et al. Methodological principles for reproducible performance evaluation in cloud computing. *IEEE Trans. Software Engineering* (2020), 93–94.
25. Sahu, S. et al. The ubiquity of large graphs and surprising challenges of graph processing: Extended survey. *Proc. VLDB Endow. J.* 29, 2 (2020), 595–618.
26. Saleem, M. et al. How representative is a SPARQL benchmark? An analysis of RDF triplestore benchmarks. *WWW Conf.* (2019), 1623–1633.
27. Salihoğlu, S. and Özsu, M.T. Response to “Scale up or scale out for graph processing.” *IEEE Internet Computing* 22, 5 (2018), 18–24.
28. Siek, J.G., Lee, L.Q., and Lumsdaine, A. The boost graph library: User guide and reference manual. Addison-Wesley (2002).
29. Uta, A., Varbanescu, A.L., Musaafir, A., Lemaire, C., and Iosup, A. Exploring HPC and big data convergence: A graph processing study on Intel Knights Landing. *CLUSTER* (2018), 66–77.
30. Zhao, P. and Han, J. On graph query optimization in large networks. In *Proc. VLDB Endow.* 3, 1 (2010), 340–351.

Sherif Sakr was a professor at the Institute of Computer Science at University of Tartu, Estonia. He passed away on March 25, 2020 at the age of 40.

Angela Bonifati (angela.bonifati@univ-lyon1.fr) is a professor at Lyon 1 University and Liris CNRS in Villeurbanne, France.

Hannes Voigt is a software engineer at Neo4j, Germany.

Alexandru Iosup is a professor at Vrije Universiteit Amsterdam and a visiting professor at Delft University of Technology, The Netherlands.

 This work is licensed under a <https://creativecommons.org/licenses/by-nc-sa/4.0/>



Watch the authors discuss this work in the exclusive *Communications* video. <https://caacm.acm.org/videos/the-future-is-big-graphs>