# Reminiscences on Influential Papers

This issue's contributors cover the impact of paying attention to the low-level implementation details, a paradigm shift in the way we approach stream processing, and the value of combining theoretical analysis with experimental evaluation. Furthermore, one of our contributors, rather than picking one paper, highlights the importance of putting the time to practice reading, reviewing, and learning from papers, not only from one's own field of interest but also from other fields. VLDB, similar to some systems conferences, launched a Shadow Program Committee for this purpose following the VLDB 2026 (Vol 19) submission cycles[1]. We wish to continue this effort in the future VLDB cycles. Enjoy reading!

While I will keep inviting members of the data management community, and neighboring communities, to contribute to this column, I also welcome unsolicited contributions. Please contact me if you are interested.

Pınar Tözün, *editor*
IT University of Copenhagen, Denmark
pito@itu.dk

---

**Viktor Leis**
Technical University of Munich
leis@in.tum.de

Stephen Tu, Wenting Zheng, Eddie Kohler, Barbara Liskov, and Samuel Madden.

***Speedy Transactions in Multicore In-Memory Databases.***

In Proceedings of the Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP), 2013.

In 2013, as I was starting my PhD, the database community was in the middle of an in-memory DBMS wave. Projects like H-Store [12], HANA [6], and HyPer [7] were rethinking architecture for abundant main memory and many-core processors. H-Store championed a shared-nothing, partitioned design with single-threaded execution per partition – a beautifully simple approach when workloads partition cleanly. Silo [13] took a different path: a shared-everything single-node system in which any thread can access any data, a choice many considered inefficient and unscalable at the time.

The paper presents an in-memory transactional database optimized for modern multi-core CPUs. Its organizing principle is to minimize coordination among threads. Silo's key contribution is an OCC-based commit protocol that lets read-only transactions avoid shared-memory writes, relying on version checks rather than read latches for safety. Transactions perform writes locally and synchronize only at commit. To avoid deadlocks, Silo sorts the write set and acquires locks in that order; conflicts simply trigger aborts.

What stood out to me was the care in low-level implementation. Each record carries a 64-bit TID word that encodes the commit timestamp plus lock / status bits, enabling a thread to lock and update a version in one atomic operation. Silo also addresses phantoms without heavy locking: leveraging Masstree's [9] node versioning, a transaction records the nodes it scanned and, at commit, aborts if any of those node versions changed – preventing range anomalies while still avoiding next-key locks.

Silo scaled extraordinarily well on a 32-core machine, reaching 700K TPC-C transactions per second – much higher throughput than prior reports at the time. The experiments were unusually thorough, including ablations that explain why it performs well, and the open-source code allowed others to learn from and build upon their implementation.

The paper strongly influenced my approach to

[1] https://vldb.org/2026/shadow-pc.html

systems research. It taught me that co-designing DBMS components often considered in isolation – such as concurrency control, latching, and indexing – can yield dramatic performance gains. It also showed me that on modern hardware, seemingly small low-level implementation choices can have outsized performance implications.

---

**Anja Gruenheid**
Microsoft Gray Systems Lab, Switzerland
anja.gruenheid@microsoft.com

In this column, I do not want to single out one paper but rather share a few experiences that helped me grow as a researcher by reading classic, influential works as well as learning to engage with and review ongoing research from others. During my Ph.D., I was fortunate to be part of activities that encouraged us to step outside our narrow focus and explore ideas from different corners of systems research. Looking back, those moments, reading papers from areas intersecting with databases, discussing them openly, and later practicing reviewing in a low-stakes setting, taught me lessons no formal course ever could. They showed me that many skills we often take for granted as researchers, like forming opinions about a paper and giving constructive feedback, do not simply emerge on their own. They need space, practice, and above all, a mindset that values understanding over judgment. As I started engaging more with research beyond my immediate area, I realized how much we can learn simply by looking closely at work that is different from our own. It is easy to assume that writing a paper equips us to judge another, but in reality, the real benefit of reading broadly is not about evaluation alone, it is about perspective. Systems research is wonderfully diverse, as databases intersect with distributed systems, networking, hardware, and now machine learning. Every paper reflects choices shaped by these contexts, and appreciating those choices requires us to step outside our familiar ground. This felt challenging early in my career as parts of a paper would sometimes seem only partly within reach. Over time, though, I came to see these moments not as obstacles but as opportunities to expand my understanding of the field. That broader view, in turn, makes both research and reviewing richer and more thoughtful. These are not skills that appear automatically, they develop through deliberate practice and above all, a willingness to approach unfamiliar ideas with curiosity.

That realization brings me to the experiences that shape such habits of mind. Early in my Ph.D., I had the good fortune to be part of an effort that explicitly allowed us students to learn from classic research papers and sharpen our reviewing skills at the same time. The professors in my group recognized that the ability to review well stems from perspective, an understanding not only of technical details but of the broader landscape and its history. To that end, they curated a list of about twenty papers spanning databases, operating systems, networking, and distributed systems, works that had left a lasting imprint on the field. I remember learning how statistics like histograms and sampling play a vital role in query plan generation and indexing strategies through the work of Chaudhuri and colleagues, which provided fascinating insights into how DBMS actually leverage these techniques. I also encountered Lamport's work on Paxos, a paper whose ideas, to my surprise, I would see surface in different guises again and again as a reviewer. And then there were papers on topics such as microkernels, which I have not really crossed paths with since, yet they opened my eyes to the rigor and elegance of fundamental research in adjacent communities. Despite several attempts, I have not been able to locate the original list. In hindsight, though, the specific papers mattered less than the way we engaged with them. For each paper, we organized a discussion session. A graduate student, chosen at random, would present the work, explain its contributions, and lead the conversation. We would ask what problem the paper was trying to solve, why that problem mattered in its original context, what conceptual leap the authors had made, and to what extent their ideas influenced the systems that came after. The element of randomness was important, as it meant that any of us could be the presenter for any given paper, which in turn meant that all of us needed to come well prepared. Skimming was not an option. At the time, this felt demanding. After all, each of us had our own work, deadlines to chase, and code to debug. Spending hours poring over a decades-old paper on distributed operating systems when your own work was on data integration might seem like an indulgence. But this was no indulgence, it was, in retrospect, a gift. Those sessions pushed us beyond the confines of our chosen topics. They taught us intellectual humility and curiosity, a desire to appreciate the reasoning behind design decisions, algorithmic choices, and experiments crafted under very different assumptions about hardware and software than those that dominate today.

Looking back, I see several reasons why this experience mattered so much. It instilled habits of critical thought, yes, but also modeled a tone for that critique, one that aimed to understand rather than dismiss. And perhaps without our noticing at the time, it established a standard for clarity, as we began to see that the most influential papers were often those that told their story with simplicity and precision even when the underlying idea was subtle or complex. Above all, it showed us a truth that extends beyond any one field, that breadth is not the enemy of depth but its complement, and that the discipline of engaging seriously with ideas outside one's immediate path strengthens one's ability to make meaningful contributions within it. Of course, not every group can replicate this exact format, and not every advisor has the time to lead such sessions regularly. But if we agree that good reviewing matters and by extension, that the quality of dialogue in our conferences and journals matters, then we as a community must think creatively about how to offer similar opportunities. The responsibility does not rest with advisors alone. We all have a shared stake in this process because the benefits ripple outward, strong reviewers make for stronger feedback, which makes for stronger papers, which makes for a stronger field. Finding practical ways to create such learning experiences is not always straightforward but it is possible.

One mechanism I have come to value deeply in recent years is the shadow program committee. For those unfamiliar, a shadow PC runs in parallel with the official review process of a conference. Its members read and review the same papers, often following the same guidelines, and later compare their assessments with the real decisions. When I was a student, I joined a EuroSys shadow PC and found the experience transformative. Until then, I had thought of reviewing as a mostly solitary act, you read, you form an opinion, you write it down. What I saw instead was the collective effort that underlies every acceptance and rejection. I saw reviewers with different backgrounds weigh novelty in different ways. I saw discussions wrestle with incomplete evaluations, ambiguous claims, or competing intuitions about practicality versus elegance. And I saw how much thought goes into offering feedback that is both candid and constructive. In addition to EuroSys, we also organized an internal shadow PC in our group for SoCC papers to practice good reviewing practices, and that too left an enduring impression. Together, these experiences made me not only a better reviewer but also a better author. Understanding the questions reviewers routinely ask, such as What gap does this work fill? What prior work does it build upon or overlook? How do the experiments support the claims? helped me anticipate and address them in my own submissions. Shadow PCs offer a rare opportunity, they involve students in real decisions without real stakes and demystify a process that can otherwise feel opaque. In doing so, they reinforce a message we should all embrace, reviewing is a craft, and like any craft, it can be taught, practiced, and improved.

What stayed with me most from those early exercises was not just the specific ideas in any single paper but the habit of grappling with work outside my immediate comfort zone. Reading and truly trying to understand papers that fell well beyond the oftentimes narrow boundaries of my research opened my eyes to the sheer diversity of questions and approaches that systems research embraces. It taught me that there is no single mold for what constitutes important or elegant work, different subfields prize different virtues, and appreciating those differences deepens both our perspective as researchers and our fairness as reviewers.

---

**Paris Carbone**

KTH Royal Institute of Technology & RISE Research Institutes of Sweden, Sweden

parisc@kth.se

Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, and Sam Whittle.

***The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing.***

In Proceedings of the VLDB Endowment, 2015.

In the summer of 2015, beneath the tropical skies of the Kohala Coast, the conference room at Hilton Waikoloa Village was packed with anticipation. There, Tyler Akidau and fellow engineers took the stage, unveiling Google's ambitious quest to harmonize the realms of batch and streaming. VLDB 2015 was my very first exposure to the data management community as a fresh graduate student. Among a packed schedule that summer, three sessions stood out: Michael Stonebraker's journey at his turing award seminar, Peter Bailis' inspiring talk of coordination avoidance, and most captivating of all,

Google's kaleidoscopic presentation [2] of "The Dataflow Model" [2]. This column reflects on why the latter mattered so much then and why, even after ten years, it continues to shape our thinking.

**What** Tyler and colleagues called for, was a much-needed paradigm shift: *streaming should subsume batch*. Their model reads more like a manifesto for out-of-order processing; every element is stamped with its event-time, and windowing can be applied uniformly across both bounded and unbounded inputs. A bounded dataset is thus treated as a special case of an unbounded one. Dataflow discourages the runtime-specific terms "streaming" and "batch" in favor of the more precise "unbounded" and "bounded" datasets. The model evolved from two existing Google systems: FlumeJava [4] and MillWheel [1]: their lower-abstraction level dataflow runtime. Dataflow featured a somewhat overly lean unified programming abstraction based on two primitives: ParDo for parallel processing functions and GroupByKey for keyed grouping. These behave identically in both batch and streaming settings. The subsequent release of Apache Beam as the open source incarnation of Dataflow reinforced the sense that this was the product of a determined engineering team rather than a traditional industrial research group.

**When** the first bold statements by Tyler hit the stage the reactions were polarized. One could tell that from the diverse facial expressions of the audience. The Q&A session confirmed some of my own initial concerns. To some, naming the model "Dataflow" felt like appropriation. Dataflows indeed have a long history. Others highlighted that the model was restrictive and offered nothing fundamentally new. From a purist point of view the dataflow model deliberately ignored a wide set of complex data stream window types researchers have been building towards for decades. Yet, to younger me the simplicity was a revelation. I had just spent my first PhD year wrestling with ad-hoc window semantics, experimenting with every imaginable combination of complex windows. Google's model explicitly limited this proliferation: fixed, sliding and session windows were the canonical choices. Besides, "time" was the only dimension that mattered for correctness, beautifully captured using watermarks, triggers and accumulation modes. Personally, I remember feeling equal parts relieved and irritated: relieved to see a coherent framework for reasoning about time and correctness, and mildly irritated that much of the prior work on richer window types seemed destined to take a back seat.

**Where** the Dataflow model positioned itself was at the intersection of long-standing database theories and the pragmatic demands of cloud-scale applications. It was not a matter of industry versus research, but of industry distilling a favored set of research ideas into products that would endure. Much of the "Dataflow Model" drew on the influential contributions of David Mayer et al. [8] in stream processing. The model also built on the sophisticated MillWheel/Dataflow runtime [1], which delivered unprecedented performance for transactional, stateful streaming workloads. This combination left little room for competition; convergence, it seemed, was inevitable and just over the horizon.

**How** these ideas reshaped systems and research became evident in the evolution of Apache Flink, Kafka Streams, Spark Streaming, and their peers. Several experimental Flink window types that fellow committers and I had added only months earlier had to be rewritten or removed to conform to the deterministic semantics championed by Dataflow. In retrospect, this process resembled what Schumpeter described as *creative destruction* in "Capitalism, Socialism and Democracy" [11]; tearing down existing designs was painful at the time, yet it cleared the way for a stronger and more coherent foundation for stream processing. The shift spread through research and industry like a major wave of innovation: Ververica (then Data Artisans) launched its "out-of-order" alignment mission, and shortly after Databricks and Confluent adopted similar principles in Structured Streaming and Kafka Streams respectively. Within just a year, the community's vocabulary and expectations had converged on event time, watermarks, and bounded versus unbounded data as the universal frame of reference. This was a truly impressive impact feat on its own. Apache Flink emerged as a popular runner of Apache Beam, Google's open incarnation of the Dataflow model, and this alignment greatly accelerated its industry adoption. At the same time, because the Dataflow model did not prescribe how a runtime should operate internally, much of the foundational work we had done on Flink, such as state checkpointing, remained not only relevant but essential, and continues to be so today [3, 10].

A decade later, the waters remain calm, perhaps too calm. No shift since has reshaped cloud data processing as profoundly as the Dataflow model. As for the runtimes, disaggregated state is now making a comeback with Flink 2.0 [10], an architecture el-

[2]The introductory slide deck for Google Dataflow is often remembered as a communication marvel in its own right: dark-mode, flashy, with precise animations that built up complex data processing ideas in a simple, understandable, and slightly "trippy" way.

ement already present in the very first version of Millwheel.

---

**Eleni Tzirita Zacharatou**
Hasso Plattner Institute & University of Potsdam, Germany
eleni.tziritazacharatou@hpi.de

Chee-Yong Chan and Yannis E. Ioannidis.
***Bitmap Index Design and Evaluation.***
In Proceedings of the International Conference on Management of Data (SIGMOD), 1998.

I first encountered this paper [5] in 2013 during the exploratory phase of my PhD, when I was searching for a research direction as a newcomer to the database field. Although I ultimately did not work much in the area of bitmap indexing, this study of bitmap indexes left a lasting impression during those formative first months of my PhD journey as I began to understand the landscape of database research. Beyond its significant technical contributions, Chan and Ioannidis's paper served as a model for how I approach problems, structure my research methodology, and communicate my findings. Essentially, their work was instrumental in shaping my understanding of what constitutes high-quality database research.

In today's research taxonomy, Chan and Ioannidis's paper would be classified as an Experiments and Analysis (E&A) study, but one that goes well beyond conventional experimental evaluation by incorporating both analytical modeling and novel algorithmic contributions. The work presents a comprehensive framework for understanding the design space of bitmap indexes, systematically investigating key design dimensions including attribute value decomposition and encoding approaches, selection query algorithms, and compression and caching techniques. Their analysis identified four critical points in the space-time tradeoff curve – ranging from space-optimal to time-optimal configurations – and provided what they described as "a first set of guidelines for physical database design using bitmap indexes." In subsequent years, bitmap indexes became widely adopted in commercial systems like Oracle for data warehousing, were a core component in early column stores, and powered specialized libraries such as FastBit.

What stood out to me was the paper's effective combination of theoretical analysis and practical evaluation. This inspired me early in my research career to always strive for principled analysis alongside thorough experimental validation. But more fundamentally, this paper taught me an important research philosophy: the value of taking a step back before moving forward. Chan and Ioannidis demonstrated that truly understanding the current landscape, identifying existing trade-offs, and systematically mapping the design space are essential tools for guiding innovation. The elegance of their framework lies not just in organizing existing knowledge, but in articulating the underlying design principles in a way that reveals previously unconsidered alternatives.

Perhaps most profoundly, the paper illustrated the power of abstraction and decomposition in research. By breaking bitmap indexing into its fundamental elements, the authors enabled new compositions and revealed hidden trade-offs. This taught me that understanding complex systems requires first decoupling their components, and that this decoupling process itself often illuminates the path forward.

The combination of theoretical analysis and practical evaluation in the paper is evident not only in its content but also in its structure. Rather than grouping all experiments in a separate section, as is standard practice today, Chan and Ioannidis interleave experimental validation with analytical insights throughout the paper. This structure creates a more coherent reading experience, as each theoretical result is immediately supported by relevant experimental evidence, allowing readers to follow the argument without having to switch between different sections.

In conclusion, I believe this paper demonstrates that a careful analysis of trade-offs in physical database design is essential to database research. For anyone looking to understand bitmap indexing, or more importantly, to learn how to conduct a systematic design space exploration, this paper is an excellent guide.

## REFERENCES

[1] Tyler Akidau, Alex Balikov, Kaya Bekiroğlu, Slava Chernyak, Josh Haberman, Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, and Sam Whittle. MillWheel: Fault-Tolerant Stream Processing at Internet Scale. *Proc. VLDB Endow.*, 6(11):1033–1044, August 2013.

[2] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric

Schmidt, and Sam Whittle. The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing. *Proc. VLDB Endow.*, 8(12):1792–1803, August 2015.

[3] Paris Carbone, Stephan Ewen, Gyula Fóra, Seif Haridi, Stefan Richter, and Kostas Tzoumas. State Management in Apache Flink®: Consistent Stateful Distributed Stream Processing. *Proc. VLDB Endow.*, 10(12):1718–1729, August 2017.

[4] Craig Chambers, Ashish Raniwala, Frances Perry, Stephen Adams, Robert R. Henry, Robert Bradshaw, and Nathan Weizenbaum. FlumeJava: Easy, Efficient Data-Parallel Pipelines. In *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '10, page 363–375, New York, NY, USA, 2010. Association for Computing Machinery.

[5] Chee-Yong Chan and Yannis E. Ioannidis. Bitmap Index Design and Evaluation. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, SIGMOD '98, page 355–366, New York, NY, USA, 1998. Association for Computing Machinery.

[6] Franz Färber, Sang Kyun Cha, Jürgen Primsch, Christof Bornhövd, Stefan Sigg, and Wolfgang Lehner. SAP HANA Database: Data Management for Modern Business Applications. *SIGMOD Rec.*, 40(4):45–51, January 2012.

[7] Alfons Kemper and Thomas Neumann. HyPer: A Hybrid OLTP&OLAP Main Memory Database System Based on Virtual Memory Snapshots. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, ICDE '11, page 195–206, USA, 2011. IEEE Computer Society.

[8] Jin Li, Kristin Tufte, Vladislav Shkapenyuk, Vassilis Papadimos, Theodore Johnson, and David Maier. Out-of-Order Processing: A New Architecture for High-Performance Stream Systems. *Proc. VLDB Endow.*, 1(1):274–288, August 2008.

[9] Yandong Mao, Eddie Kohler, and Robert Tappan Morris. Cache Craftiness for Fast Multicore Key-Value Storage. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, page 183–196, New York, NY, USA, 2012. Association for Computing Machinery.

[10] Yuan Mei, Rui Xia, Zhaoqian Lan, Kaitian Hu, Lei Huang, Paris Carbone, Yanfei Lei, Vasiliki Kalavri, Han Yin, and Feng Wang. Disaggregated State Management in Apache Flink® 2.0. *Proc. VLDB Endow.*, 18(12):4846–4859, September 2025.

[11] Joseph A. Schumpeter. *Capitalism, Socialism and Democracy*. Routledge: London, UK, 1976.

[12] Michael Stonebraker, Samuel Madden, Daniel J. Abadi, Stavros Harizopoulos, Nabil Hachem, and Pat Helland. The End of an Architectural Era: (It's Time for a Complete Rewrite). In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, page 1150–1160. VLDB Endowment, 2007.

[13] Stephen Tu, Wenting Zheng, Eddie Kohler, Barbara Liskov, and Samuel Madden. Speedy Transactions in Multicore In-Memory Databases. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, page 18–32, New York, NY, USA, 2013. Association for Computing Machinery.