

WHITEPAPER

A MULTI-DIMENSIONAL DATA MATRIX

A new data representation format for secure, consistent, and efficient software.

TETRA CA PBC, in collaboration with HorneSci LLC, introduces the data matrix: information represented as integer vectors to store, process, and retrieve data. This vector representation reduces the work that processors (CPU or GPU) perform to calculate a result.

SECTION 01

EXECUTIVE SUMMARY

TETRA CA PBC, in collaboration with HorneSci LLC, introduces the *data matrix*: information represented as integer vectors to store, process, and retrieve data. This vector representation reduces the work that processors (CPU or GPU) perform to calculate a result.

The impacts of this technology compound. The more this technology gets used in an ecosystem, the more resource usage (and costs) plummet. The “CA” in TETRA CA means “Collapsive Architecture,” because our systems operate with higher efficiency than you can build with open source tooling today.

The data matrix has endemic compression (~24% of original data size). When doing search and lookup operations, no additional indices (lookup pointers) get created or used. Instead, we use the integer vector representations to perform searches with math (linear algebra and geometry).

Modern systems must network and distribute data. Our data matrix costs less to transmit, read, and store, thereby using fewer resources for the sender, the networking components, and the receiver. As integrated dynamical systems, the gains from the data matrix over the wire can be challenging to quantify, but in rough terms we measure 6000% to 9000% more cost-effective end-to-end.

We have long known that computers perform linear algebraic operations more efficiently than “scalar math”, the type of logic encoded by every programming language. Because of our data matrix representation, we do work faster. This becomes more apparent as data systems scale, because the vector representation of data scales more cost-effectively than current systems. On most operations, we see anywhere from at 50% to a 10000% speedup, with more complex and nuanced operations seeing greater acceleration.

Our implementation of the data matrix gets encrypted at write time. We use cutting edge “ML-KEM” (quantum resistant) key encapsulation and AES encryption on the datastore. We also encrypt the data matrix in transit.

Because information has “spatial” or “geometric” representation, we do not require explicit indexing.

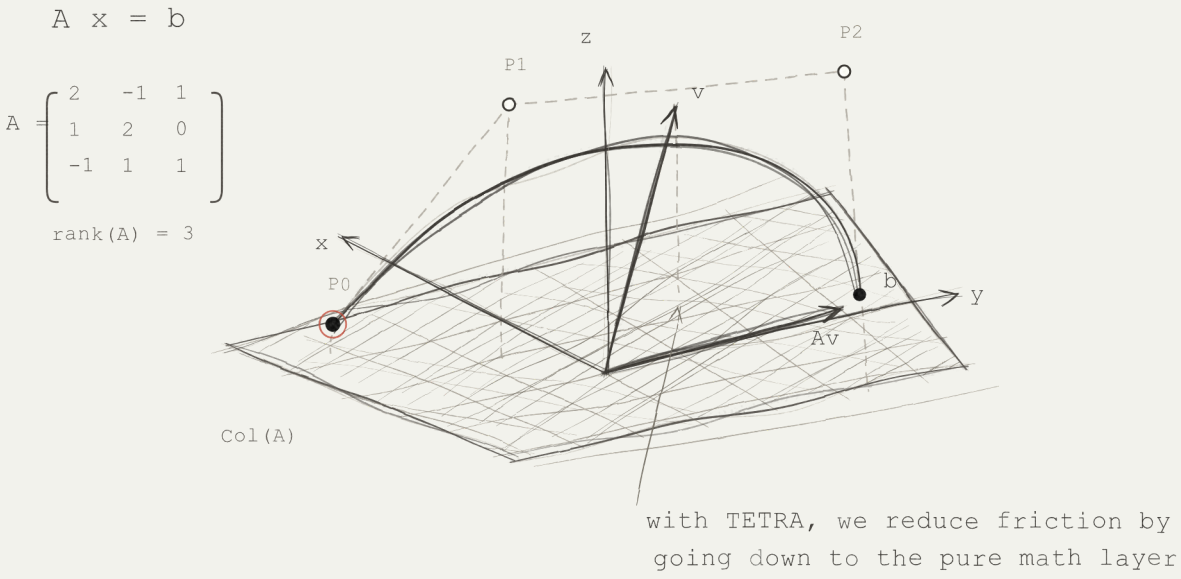


Figure 1. TETRA systems represent information using vector mathematics.

TETRA has no indices, no schema, no caches, and low operational overhead. The data representation can evolve over time, so systems require fewer if any rewrites over time.

> We collapse the cost, time, and complexity of scaling data systems.

SECTION 02

INTRODUCTION

“Information technology.” This blanket term encompasses everything related to how computers store and retrieve data. Applications then stitch together this “data” in a format that humans (or LLMs) can interpret and act upon. This is when the data becomes “information.” And how we represent the data to transmit information determines the efficiency of the technology.

IT fundamentally impacts the day-to-day lives of most people living today, whether they use computer systems or not. We use IT in our policy, governance, economic systems... everything. And this makes sense, because the technology enables people to create, transmit, and store information faster and more efficiently.

Consider the generally-known correlation between information representation (how information gets created, transmitted, and stored) and technological progress. The reuse of others’ learning enables us to build upon what we’ve already learned. Doing so faster means compounding technological progress. And our technology reflects this phenomenon.

But now we encounter the limits of our systems today.

As new technologies consume information at a faster rate than our current data systems support, scaling data becomes ever more expensive.

Our current methods of representing data for storing, transmitting, and reading data derive from mathematical data encodings that date back to Alan Turing (1940s). Turing sought to encapsulate how people think, the reasoning they use, in machines.

Building on the methods we use today, we represent data as algebraic/geometric vectors rather than scalar, calculated values.

Accelerating and complimenting our efforts, HorneSci optimizes tensor and vector calculations at the hardware and runtime level. This means that the business logic represented in TETRA gets optimized by HorneSci’s Hierarchical Sparse Tensor (HST) framework.

HST uses telemetry from the hardware to plan and execute the most efficient, least-change execution path for sparse and dense tensor operations. The framework builds on the founder’s background in High-Performance Compute (HPC) and consulting work with prestigious National Labs in the United State. By partnering with HorneSci, TETRA gains next-level delivery and optimization capacity; we translate your business requirements into a next-level data system that reliably and securely scales to your needs.

The Data Matrix takes a massive step forward for scaling data efficiently and securely. Let’s take a look at the representation and its benefits.

SECTION 03

DATA MATRIX

A matrix of integers

Put simply, the data matrix represents data using integer vectors.

`[1, 2, 3, 4]`.

If you open up a `.tetra` file, you find numbers. Nothing else. These numbers represent the information stored in TETRA as well as the relationships between the information (which is also information, of course).

This representation reflects, and drew inspiration from, machine learning or large language model "weights," the numbers that give models their "intelligence" or "predictive power." But weights use decimal numbers, and we used integers because computers process integers more easily than decimals.

Not only does processing integer vectors require less work on the processor (GPU or CPU), we also require less space to store the same amount of information. This representation applies to memory, disk storage, and transmission as well.

By sending and receiving the same format, the data matrix, we reduce system friction. Using the same data model across systems means low-impedance; nothing has to decode information. The data matrix never needs to be "unpacked" between systems, so it takes no RAM beyond its own compressed size. The data matrix gets read and processed and sent without translation or disruption; changes and additions become more matrix, so no new systems have to be adapted to extend the functionality.

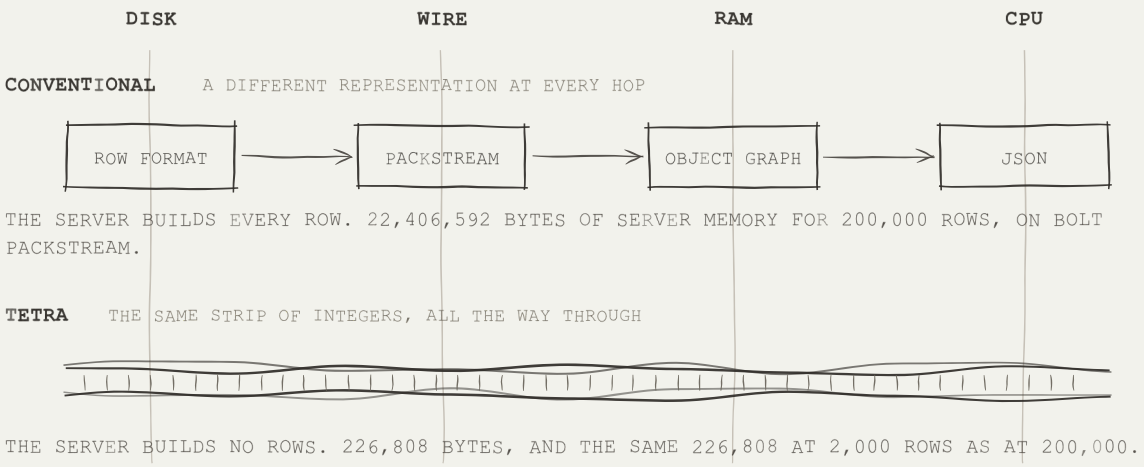


Figure 2. Disk, wire, RAM, CPU. Appendix 12.

The data matrix reflects TETRA's core idea: collapse all of the complexity, doing the minimal work to achieve maximal impact on data systems.

The format handles modifications (writes and deletes) with phenomenal, mathematical ease. We simply extend the matrix with overwrites, and we store these writes to a mutations file. This enables us to provide correct, current-matrix state reads.

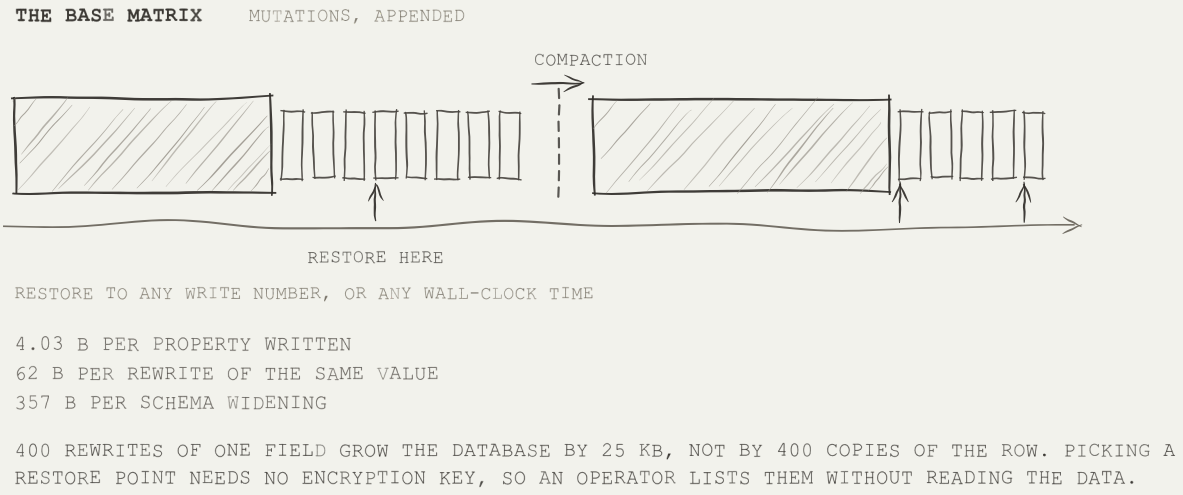


Figure 3. The mutations file and compaction. Appendix 7.

Data representation today

What does data representation look like without the data matrix?

Currently, we calculate values, and we store the results “in a box.” We can only work with certain block sizes, so we always have to decide how many boxes we’re going to need to use when we plan the work. If our information overflows the boxes we planned, this can mean trouble!

We have data, we put data into files, and we put files into storage.

For databases today, this means records in data tables. Then, on the back end, indices tell you which data points exist in each file. An index is basically a copy of some information pointing to a file. Software does this very efficiently. So efficiently, in fact, that we may have stopped looking for other ways of doing things!

You might call this approach “scalar” or “linear.” The more work you have to do, the longer doing the work takes. If you have to compare a large number of items to do the work, it’s going to take way more energy to accomplish the task.

Our data matrix scales “sub-linearly.” This means that 2x the data costs less than 2x the costs, while typical data systems cost more than twice as much to scale!

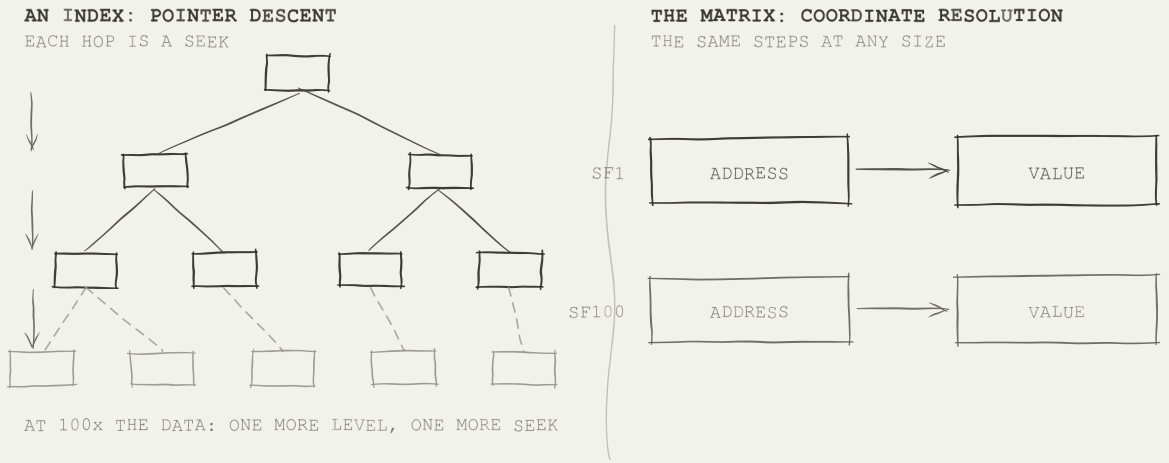


Figure 4. Pointer descent and coordinate resolution. Appendix 2.

These scalar systems that copy data to find it and chop data into boxes drive everything we do with compute today. We block data into boxes and we assemble and glue together boxes to represent and move information. Each instruction gets planned in computer code, then executed, step-by-step, between storage, memory, and processing. What a software engineer sees reflects the “tip of the iceberg” at the hardware and low-level instruction level.

The data you store into files must be “typed,” that is, there must exist instructions on how to represent the data as it got stored to the way it should be handled in the system.

We’ve been using the same type system since the 1980s because our programs build on top of it.

Current database technology must update indices and store logs of write operations (Write Ahead Logs, or WAL) to keep data in sync. This is because the relationships between information (topology) stay separate from the information itself (data).

Because databases have bloated, complex state management systems, we write APIs to handle all of the business logic that our databases cannot. These concerns get addressed throughout the paper, but in general we can think of them as “non functional requirements”, qualitative aspects about how a system operates. Securely, reliably, with good monitoring and operational insight.

You want software systems that provide this, so you must wrap your database with API calls. For processes requiring a lot of processing, data gets translated across formats between clients and servers and databases and file storage, all of these concerns eating memory and resource simply to transform the same data from one format to another.

The data matrix, in contrast, represents everything as an integer matrix. Security requirements. Telemetry signals. Queries and their results. The difference in representation means that the data matrix avoid many issues that make traditional data systems challenging, including:

- Storage size requirements
- Accuracy and precision
- Security & Identity
- Data representation across APIs and networks
- Data scalability / cost scaling factors
- Ecosystem integration

The following sections show how our data matrix revolutionizes data storage, retrieval, transmission, and processing.

Interlude: agentic systems and data

Agentic token costs add up, AND the infrastructure they use starts costing more. Companies already feel the crunch!

The problem: LLMs do not understand your human-inspired system-use expectations. They query the same API endpoint 12 times. They run queries against your database that you do not expect, and it adds a massive load to your system.

TETRA's data matrix needs no index, searches find coordinates in the matrix rather than values within data table files. This means agents can explore your data without racking up a devastating infrastructure bill.

TETRAbase, the database system, uniquely serves agentic processes as a "TETRAbrain," an agentic memory system that future agents can explore and write-to without friction.

Queryable compression

We know how to represent data to make it queryable. We know how to compress data to make it more economical to store and transmit. We do not know how to do both at the same time.

Admittedly, claiming queryable compression sounds bold. In general:

- A data matrix is just numbers
- Numbers have emergent patterns
- We create a representation graph of patterns to vectors.

This enables us to take 100 GB of uncompressed CSV data (the ldbcouncil.org social network benchmark dataset) and the .tetra file is ~23 GB. This is comparable to other compression techniques, but it's the only version that you can query like a database without blowing up memory usage.

We compress the data and represent it once, and this means we query against the compressed matrix. The same queryable format works across data storage, transmission, and processing.

Decompression / data reconstitution becomes a just-in-time read concern. We therefore amortize decompression at the query level, enabling more memory efficiency than other data storage and retrieval systems.

Because we can represent data in a compressed, but still accessible and readable format, we can host more data on the same servers, or co-locate the data with application servers.

The consistent representation of the data enables lower-impedance operations.

Additionally, because we compose vectors into matrices to represent the data, queries against the data (searches, graph-like traversals, and traditional SQL) get expressed as linear algebraic expressions; results get calculated rather than stored.

Of course, the efficiency of “queryable compression” means nothing if we cannot leverage it with confidence. For any data system, accuracy is paramount. We discuss how TETRA’s data matrix provides unparalleled data accuracy.

Uncompromising accuracy

Engineering around floating point loss presented a huge challenge to the pioneers of computer science.

Example: write 0.1 + 0.2 in almost any language and you get 0.30000000000000004.

Binary can only represent non-integer numbers (fractions/decimals) against powers of two. We cannot store 0.1 exactly, but we can store 1/8. The example above makes the issue clear. It’s a small error, but added up over millions to billions of calculations, it starts to matter.

To compensate for floating point loss, most systems just add more memory space to compensate and ignore the noise.

For our system, we only use integers. As such, we store values like 6.095 as the integer 6095, with a scale of 3. 50.00 is 5000, scale 2. A complete integer representation, no approximation.

COBOL stored money this way in the 1960s and SQL's DECIMAL leverages this too.

TETRA runs on integers, and integers mean (a + b) + c equals a + (b + c). This is the associative property. Sum a column in any order, across any number of threads, and get the same, exact, precisely correct answer every time.

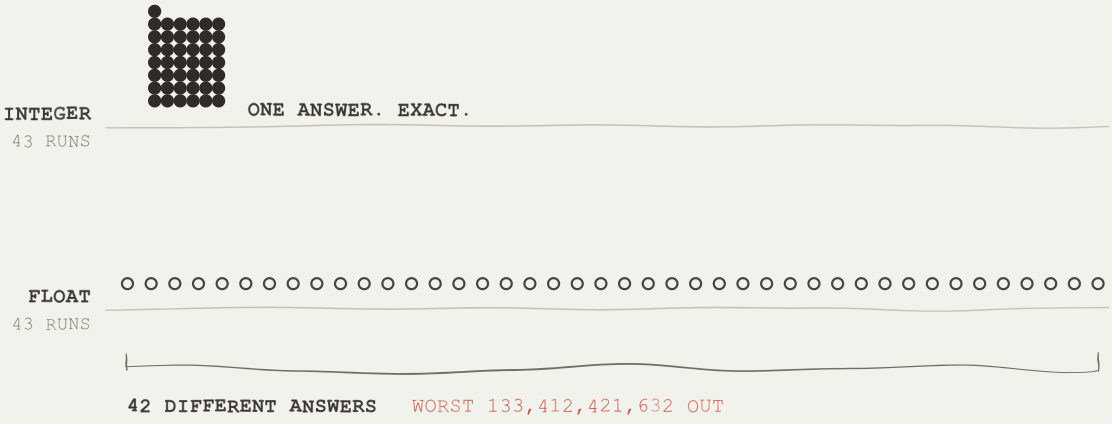


Figure 5. 43 summation orders with their answers. Appendix 3.

Because we forced everything into integer representation, we created a more precise way of representing information. When you use TETRA CA systems, you get accuracy. Full stop.

Security and identity

The data you save gets encrypted. Encryption happens before the matrix is stored, per file block. When a file block gets read during a query, it gets decrypted at read-time. Because the encrypted file is a block of encrypted integers, and the decrypted data is a block of decrypted integers, we incur minimal encryption overhead.

Identity and permissions concerns map naturally to “graph model.” What is this request trying to do? What resource does the request access? Should this person/process/agent have access to this data? These queries map to a node - relationship -> node abstraction, and our data matrix ensures these traversals run incredibly fast.

This means we can handle deep permission checks in real-time by giving you personalized and configurable access controls.

Our data matrix has a graph data API. Graph data, nodes and relationships, maps effectively to roles, permissions, attribute controls... All authorization models reflect graph-like structures, and we enable security permissions to be defined easily alongside your data model.

A secure and accurate system cannot meet your business needs unless it can reliably provide answers when you need it. Read on to learn about our scaling efficiency.

Scalability and cost efficiency

When talking about software data and architecture scalability, we describe two techniques:

- **Vertical scaling**: we make the machine that serves the data bigger; more RAM, more compute, more storage
- **Horizontal scaling**: We add more machines, such that we spread the load.

Vertical scaling may be considered “easier,” but it keeps a single-point-of-failure. Also, if your user experience gets impacted by data location, vertical scaling cannot help.

Horizontal scaling incurs an administration overhead, often adversely affecting overall latency/throughput, and has a massive negative impact on resource efficiency. Every replicated data system means replicated indexes and writes.

The larger a horizontally scaled system grows, the more cost-ineffective it becomes.

TETRA breaks this mold because our system uses the data matrix representation. The data matrix, being an array of numbers, transfers effortlessly between storage, processing, and transmission. We can scale vertically AND horizontally more efficiently than current data systems.

We have developed what we call the TETRAbeam, a data matrix optimized for sending across the wire. On our early benchmarks and tests, we see the TETRAbeam format being 60x to 90x more efficient (end to end) than other data format methods.

Recall, TETRA data systems scale sub-linearly. The efficiency we bring enables you to condense your cluster into a single machine. Or to co-locate your business logic and your data. But truly big systems require distribution, and we help with that.

At scale and under load, TETRA performs ~3x more memory-efficiently than comparable data systems. This means 3x the headroom before systems break.

As mentioned previously, TETRA takes 12.5% of the storage disk space when compared to traditional indices and storage (Postgres and Neo4j both measured similarly). This means every backup, every operation, every time you need to move data, the data matrix costs less. Way less.

But even if something costs less to store, transmit, and process, and even if we consider compounding savings on the infrastructure you *would* have required without TETRA, we must address one of the most important aspects of software: the ability for the software to cross-integrate with your existing data and tooling. And we talk about that next.

Ecosystem integration

Many times, software projects grow so problematic and unwieldy that the organization invests in a vNext; such as an identity system v4 at a Fortune 1000 company.

The company may have data from vendors, legacy systems that still serve business-critical requirements, or use third-party tooling for specific concerns (examples: observability, identity, data APIs). Any data system which cannot integrate with existing systems fails.

For this reason, we took an integration-first approach to building TETRA. We interoperate with existing systems. We emit signals in standard ways, and we use commodity hardware (CPUs, for example).

We started with a graph database model, supporting the open source OpenCypher query language and the standard connection protocol Bolt. This means we encode and decode our data matrix into and out of the formats that the connection requires.

Sometimes, external systems need to “phone in” to your systems. To accommodate this, we enable “webhooks” (listeners for events) as well as polling configuration. TETRA’s engineering efforts go towards ensuring that our systems integrate with your current tools to solve your current problems.

By integrating into existing ecosystems, we make it easy to adopt its benefits beyond cost saving; and there are many.

SECTION 04

BENEFITS OF THE DATA MATRIX

Compounding operational overhead reductions

While we broadcast 66% infrastructure cost reduction, the potential savings go much further. We applied over a decade of operations expertise using modern “development operations” (dev-ops) principles.

This means our data matrix datastore, TETRAbase, stores and retrieves information at parity or better with industry standard tooling while using a half to a third of the RAM. The data storage uses ~12% of the space, meaning costs to back up and restore stay low.

TETRAbase writes route to a mutations file (.mut) which serves as a write-ahead-log (WAL), we simply create a mutation and add it to a mutation file. To address consistency and point-in-time, (Recovery Time Objective (RTO) and Recovery Point Objective (RPO), mutations have a journal format for point in time recovery (PITR). We have a “compact” operation to merge your mutations files with your base file.

The impact of these design choices mean your operations team spends far less time fixing things. Human and agentic operators find the data matrix format easy-to-use.

This does not just reduce your cloud savings once and then leave you powerless; your data operations savings compound as the data you manage grows.

Developer friendly (agentic or human)

TETRAbase has no schema (however, you can enforce one with your business logic). This means we can represent legacy SQL data as easily as new graph-based data models. We work with the data you have, and as you add new functionality and relationships *you will have to restructure your data less*.

Most data requires no caching with TETRAbase, which means one layer less your system has to deal with, which means a little less support needed to keep your systems operational.

All of these little things add up to big savings for businesses that need to scale data products.

The TETRA ecosystem

But the benefits do not stop there. As we mentioned before, TETRAbase covers data storage and retrieval. Business systems today must consider:

- Networking
- Load balancing and distribution / workload delegation
- Security: identity and permissions, firewalls and network intrusions, bot detection and control, access auditing
- Observability (logging, metrics)
- Compliance: privacy laws, access controls and verifications
- Business logic: APIs, data pipelines, Extract Transform Load operations (ETL)

For every one of these, TETRA can help lower infrastructure costs and improve data efficiency.

We have developed 3 components, or “roles,” connected with the data matrix, that enable all of these workflows to operate orders of magnitude more efficiently. They communicate over existing protocols with a data matrix format called “TETRAbeam.”

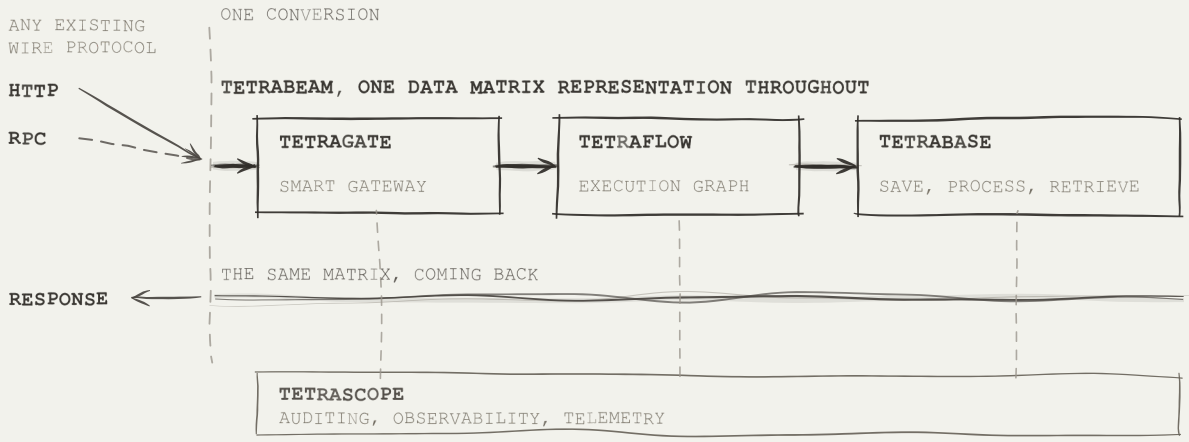


Figure 6. The three roles, and the data matrix between them. The conversion happens once, at the gateway.

> The result: no impedance, translations, conversions, or any complexity between your data at the edge, on the network, during processing, or in the database. One data matrix representation.

TETRAgate

TETRAgate acts as a smart gateway / delegator. It receives any existing wire protocol (RPCs over binary or http, JSON, or custom), converts the request to TETRAbeam (data matrix), then it sends the TETRAbeam to a TETRAflow.

TETRAflow

Replaces traditional APIs. These are composable execution graphs with success and error conditions baked in. No code, no code to break. Instead of code, we provide execution codes (operation codes) and outcome conditions. Same concerns, direct-to-matrix-graph.

APIs interface with third-party services and enforce security, provide observability, and execute your business logic. TETRAflows do exactly that, with lower overhead and more security.

TETRAbase

A massive upgrade to the database, TETRAbase saves, processes, and retrieves data from the matrix.

This means TETRAgate handles all of the traditional protocol resolution. If your identity and security data gets stored in a TETRAbase, you can use TETRAflows to query and resolve permissions in real-time. Then, TETRAflows compose to perform the data action that the system requested.

For interacting with outside systems, TETRAflows can go out through the TETRAgate, too. This is where we convert the data matrix to an external API call or query, and then we process the response (success or failure) into a data matrix again (TETRAbeam).

To provide observability, auditing, and operational insight, we developed TETRAscope. TETRAscope uses the data matrix to manage auditing, observability, and telemetry. This means that all of the systems above show you what’s happening, and you can show your auditors.

Use cases

TETRA CA PBC has had talks with mobile network operations teams, supply chain and retail, devops, enterprise ontology. The list is endless. TETRA partners with HorneSci, and they bring world-class expertise in high-performance-compute, linear algebraic mathematics, and chip-level interface optimization. Even on commodity hardware.

We thrive in edge and embedded use cases; our general purpose engine is ~50 MB, but we can create small for-purpose deployments of the same technology.

Using our TETRAbeam data matrix transport, we can also move “decoding” (RAM and CPU usage) client-side, enabling low-impedance end-to-end systems.

If you:

- Use data to drive your business
- Your data keeps getting bigger and expenses unmanageable
- Need to reduce your Cost of Goods & Services (COGS) over the long haul
- Have reliability and security requirements for your data
- Would like to access your data using LLM/”AI” agents

TETRA and HorneSci provide the data infrastructure to do the work without the headaches. We took everything hard about data management, we literally did the math, and then turned it into high-efficiency systems for your business systems.

> The sooner you collapse your complexity, the sooner your cost savings compound.

SECTION 05

CONCLUSION

TETRA and HorneSci present a compelling argument as to how the data matrix can benefit your organization. We remove all the boundaries and friction that hold organizations back from making the most of their data.

TETRA works better on your existing hardware resources, has less error propensity and higher reliability, security, and observability. We simplify every existing data workflow from API to ETL. Our roadmap (and next whitepaper) address blob storage and embeddings, too.

This representative technology means that scaling and operating data-intensive systems becomes easy and predictable.

> We did the math, and we conservatively estimate a two-thirds reduction on your current infrastructure costs.

To get started today, please email your cloud bill to inquiries@tetra-ca.com.

APPENDIX

Supporting benchmarks and test results

The data presented here reflects best-effort measurements from our engineering team; improvements continue to be made, so these figures are “worst of.” We err on the side of conservatism on the data we select and present.

Test environments

Two machines produced everything below.

	MACHINE	WHAT IT RAN
laptop	Apple M4, 10 cores, macOS	small benchmarks, local databases, engine size
server	AMD EPYC 7502P, 32 cores / 64 threads, 251 GB RAM, SSD array	the SF1 to SF100 scale ladder, the head-to-head runs

Other databases, a PostgreSQL and a Neo4j sat resident on the server throughout every run. That inflates our own numbers rather than flattering them.

SF 1 - 100 refers to the Graph Data Council (ldbcouncil.org) benchmark dataset. The 1 - 100 refer to gigabytes of uncompressed CSV data from data tables.

Every claim in the whitepaper derives from actual measurements of our system.

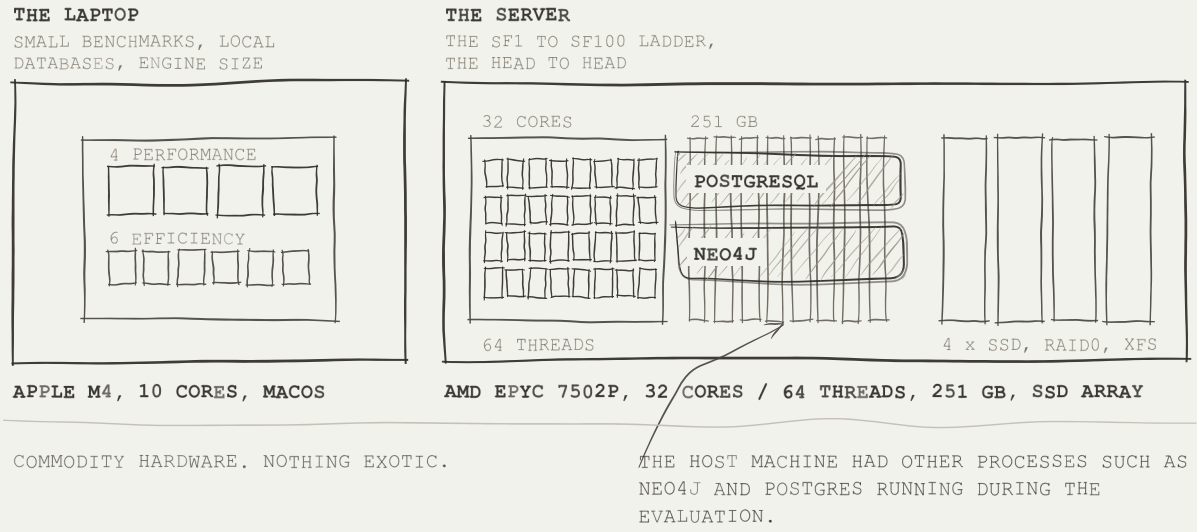


Figure 7. The two machines.

CLAIM	PAPER SECTION	APP.	MEASUREMENT
Endemic compression, ~24% of original data size	Executive Summary; Queryable compression	1	21.0% to 22.9%
100 GB of CSV becomes a ~23 GB queryable file	Queryable compression	1	101.0 GiB to 23.105 GiB
No additional indices get created or used	Executive Summary; INTERLUDE	2	Nothing on disk but the matrix; point lookups flat to SF100
A .tetra file is numbers and nothing else	A matrix of integers	2	Every database fully accounted for
Sum in any order, any thread count, same exact answer	Uncompromising accuracy	3	43 orderings, one answer, exact
6.095 stored as 6095, scale 3	Uncompromising accuracy	3	Confirmed
Scales sub-linearly	Data representation today; Scalability	4	102.89x the edges costs 16.7x the time
Never unpacked; no RAM beyond its compressed size	A matrix of integers; Queryable compression	5	23.105 GiB database open in 16.56 GiB
Decompression is a just-in-time read concern	Queryable compression	5	4,096 bytes read against 4,210,768
ML-KEM key encapsulation and AES on the datastore	Executive Summary; Security and identity	6	ML-KEM-768, AES-256-GCM
Minimal encryption overhead	Security and identity	6	0.392% on disk, 9.0 GB/s to read
Writes to a mutations file; PITR; compact	Compounding operational overhead reductions	7	8 invariants pass; 4.03 B per property

CLAIM	PAPER SECTION	APP.	MEASUREMENT
No schema	Executive Summary; Developer friendly	8	19 properties, 14 labels, 16 relationship types, all found in the data
Most data requires no caching	Developer friendly	8	40 distinct queries, flat latency
OpenCypher and Bolt	Ecosystem integration	9	3,897 of 3,897 conformance scenarios
General purpose engine ~50 MB	Use cases	10	49.8 MB
50% to 10000% speedup on most operations	Executive Summary	11	1.30x to 28.04x on kernels, to 1,854x on whole operations
TETRAbeam 60x to 90x more efficient end to end	Scalability and cost efficiency	12	64x on macOS, 90x on the server
12.5% of the storage of Postgres and Neo4j	Scalability; Benefits	13	211.5 MiB against 1.683 GiB
~3x more memory-efficient	Scalability and cost efficiency	13	1,290 MB against 3,590 MB
Parity or better on a half to a third of the RAM	Compounding operational overhead reductions	13	Faster at 8 of 9 concurrency levels
66% infrastructure cost reduction; two-thirds reduction	Benefits; Conclusion	13	12.3% of the disk, 36% of the RAM, more operations at 8 of 9 levels

APPENDIX 1

QUERYABLE COMPRESSION

Supports “endemic compression (~24% of original data size)” and “100 GB of uncompressed CSV … the .tetra file is ~23 GB.”

Source: generated CSV directory. The database is one file.

	SOURCE CSV	.TETRA FILE	RATIO
SF1	1,056,504,650 B	221,773,824 B	21.0%
SF3	3,206,291,176 B	691,486,720 B	21.6%
SF10	10,716,728,756 B	2,337,271,808 B	21.8%
SF30	33,234,100,003 B	7,349,903,360 B	22.1%
SF100	108,487,275,552 B	24,809,123,840 B	**22.9%**

SF100 holds 101.0 GiB of CSV in 23.105 GiB, and you query the 23.105 GiB.

Scale

	SF1	SF10	SF30	SF100
Nodes	3,181,724	29,987,835	88,789,833	282,637,871
Edges	17,256,038	176,623,433	540,915,383	1,775,513,741
Edges / SF1	1.00x	10.24x	31.35x	102.89x
On disk	211.5 MiB	2.177 GiB	6.845 GiB	23.105 GiB
Bytes per edge	12.9	13.2	13.6	14.0

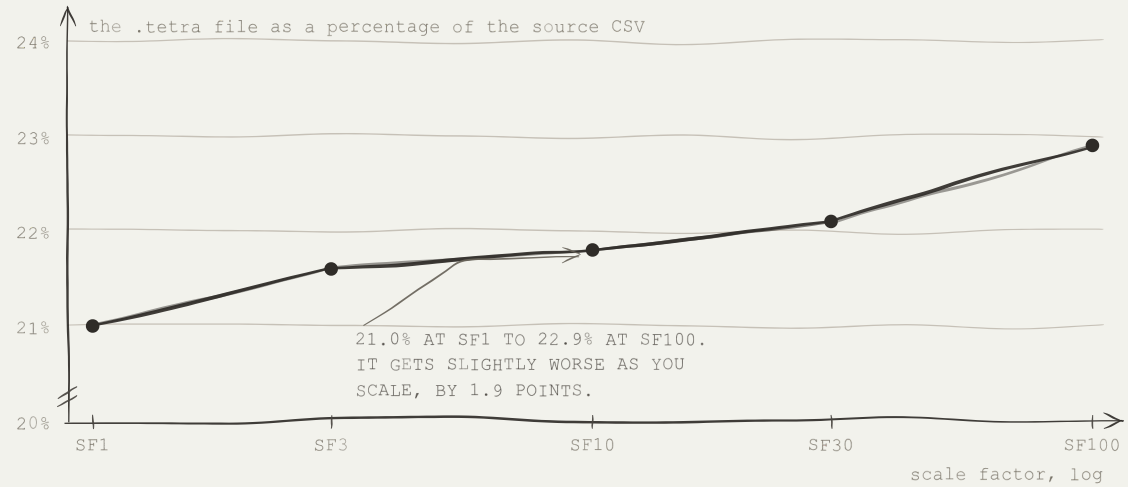


Figure 8. The .tetra file as a percentage of source CSV, five scale factors. Broken y axis, 20% to 24%.

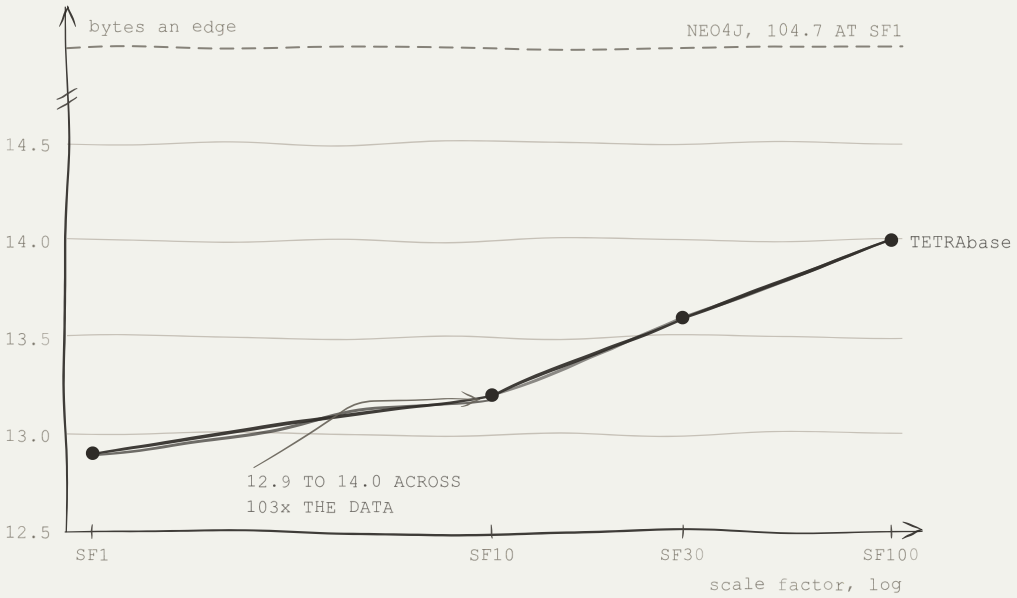


Figure 9. Bytes per edge, four scale factors, against Neo4j at SF1. Log axis.

APPENDIX 2

NO INDICES

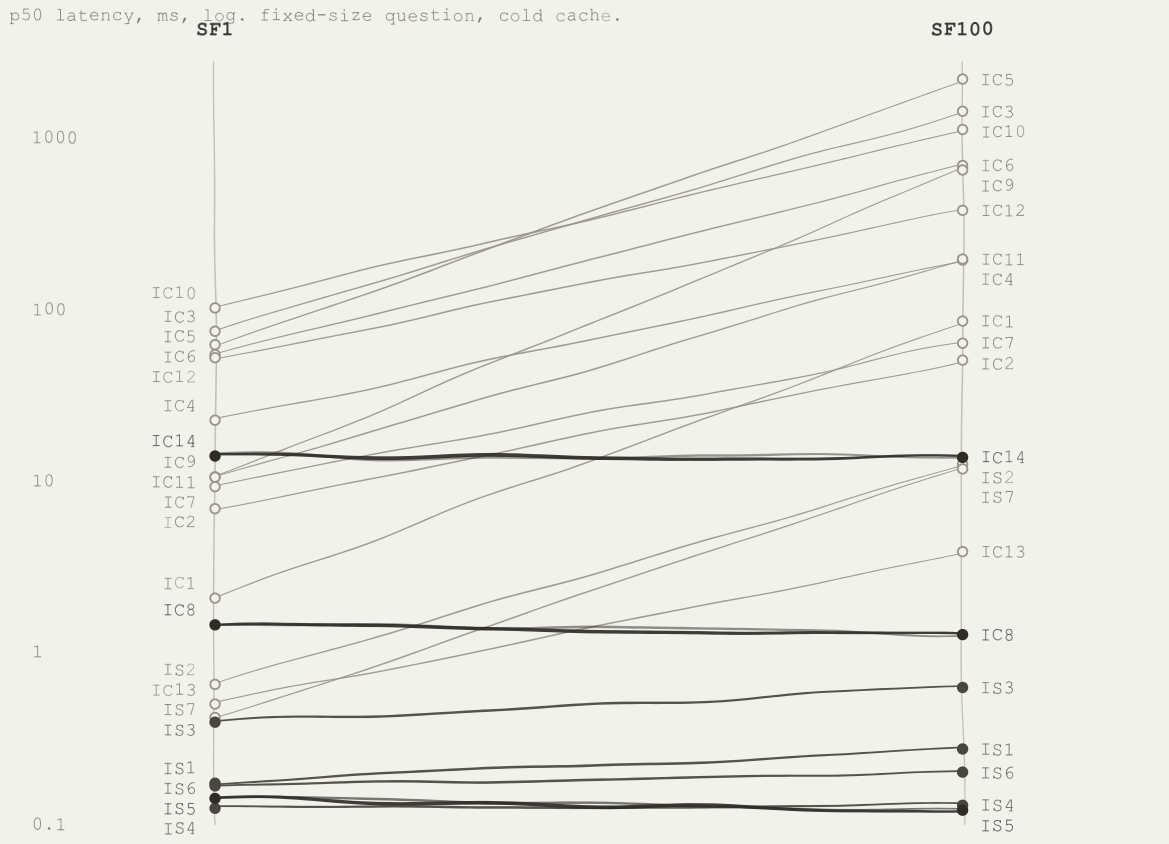
Supports “no additional indices (lookup pointers) get created or used,” “we do not require explicit indexing,” “all data is equally discoverable,” and “you find numbers. Nothing else.”

DATABASE	FILE SIZE	NODES	RELATIONSHIPS	KINDS	UNACCOUNTED
product catalogue	6,549,504 B	28,863	195,124	5	none
LDBC SF1	216,219,648 B	2,997,352	20,194,128	16	none
LDBC SF3	654,176,256 B	8,513,157	59,548,384	16	none
LDBC SF10	2,209,210,368 B	27,231,349	197,575,294	16	none

An index deepens as the data grows, and a lookup slows down with it. Across SF1 to SF100, a hundredfold increase in the graph, six of the twenty-one standard queries do not slow down at all.

QUERY	SF1	SF100
IS1 profile of a person	0.168 ms	0.266 ms
IS3 friends of a person	0.381 ms	0.607 ms
IS4 content of a message	0.120 ms	0.125 ms
IS5 creator of a message	0.137 ms	**0.117 ms**
IS6 forum of a message	0.162 ms	0.195 ms
IC8 recent replies	1.407 ms	**1.234 ms**

Two of the six answer faster on the hundred-gigabyte database than on the one-gigabyte one.



ALL 21 INTERACTIVE QUERIES ACROSS 102.89x THE EDGES. 3 ANSWER FASTER, 4 STAY INSIDE A FACTOR OF TWO, 14 SLOW DOWN.
PRESSED = FASTER AT SF100. MID = WITHIN 2x. FAINT = SLOWED. NOTHING IS OMITTED AND NOTHING IS RANKED AWAY.

Figure 10. All 21 standard queries, SF1 and SF100, log axis.

APPENDIX 3

UNCOMPROMISING ACCURACY

Supports the “Uncompromising accuracy” section: the 0.1 + 0.2 example, storing 6.095 as 6095 with a scale of 3, and “sum a column in any order, across any number of threads, and get the same, exact, precisely correct answer every time.”

Adding 0.1 and 0.2 in binary floating point yields 0.30000000000000004. The true 0.3 rounds to 0.29999999999999999. That gap arrives before you have done anything with the number.

The same two values as integers with a scale, 1 and 2 at scale 1, add to 3, which is exactly 0.3. The paper’s own examples:

WRITTEN	STORED AS	SCALE
6.095	6095	3
50.00	5000	2
−0.01	−1	2

Summing a column. We took 1,048,576 values. Each one fits in a 64-bit integer; their total needs 82 bits. We added them forty-three ways and checked every answer against arbitrary-precision arithmetic.

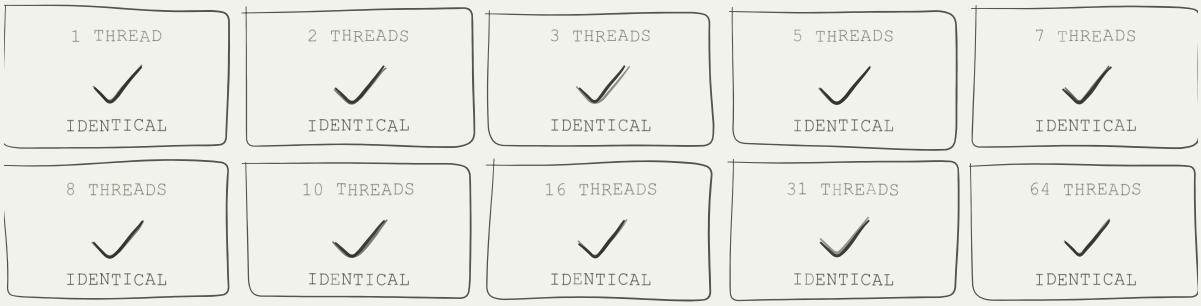
HOW WE ADDED THEM	ANSWER
straight through	exact
32 random orders	exact, all 32 identical
split across 1, 2, 3, 5, 7, 8, 10, 16, 31 and 64 threads	exact, all 10 identical
an ordinary 64-bit total	wrong; the total overflowed
floating point, the same 43 ways	**42 different answers**, worst off by 133,412,421,632

What accuracy costs, on 65,536 values

	NS/OP	THROUGHPUT
our exact total	19,163	27.4 GB/s
ordinary 64-bit total (wrong)	16,884	31.1 GB/s
floating point total (inexact)	37,543	14.0 GB/s

None of the three allocates memory.

THE RETURNED SUM, BY THREAD COUNT



THE SAME 1,048,576 VALUES, SPLIT ACROSS EACH THREAD COUNT. ALL TEN TOTALS IDENTICAL AND EXACT, CHECKED AGAINST ARBITRARY-PRECISION ARITHMETIC.

Figure 11. Ten thread counts, 1 to 64.

APPENDIX 4

SCALING SUB-LINEARLY

Supports “Our data matrix scales sub-linearly. This means that 2x the data costs less than 2x the costs.”

Twenty-one standard queries, four database sizes, ten timed runs each, page cache dropped before every size. The fixed anchor holds the question a constant size, so the curve isolates database size. The growing anchor asks about the most connected person in the graph, so the question itself grows with the data.

	SF1	SF10	SF30	SF100	SF1 TO SF100	SLOPE
Edges	17,256,038	176,623,433	540,915,383	1,775,513,741	102.89x	
All 21, fixed question	0.413 s	1.544 s	2.770 s	6.878 s	**16.7x**	0.60
All 21, growing question	0.834 s	4.826 s	11.199 s	22.989 s	**27.6x**	0.72

Slope is the regression of log time on log edge count. 1.00 tracks the data; below 1.00 is sub-linear.

Opening the database behaves the same way, cold:

	SF1	SF10	SF30	SF100
Open and ready	441 ms	1,051 ms	3,093 ms	9,818 ms
Per million edges	25.55 ms	5.95 ms	5.72 ms	**5.53 ms**

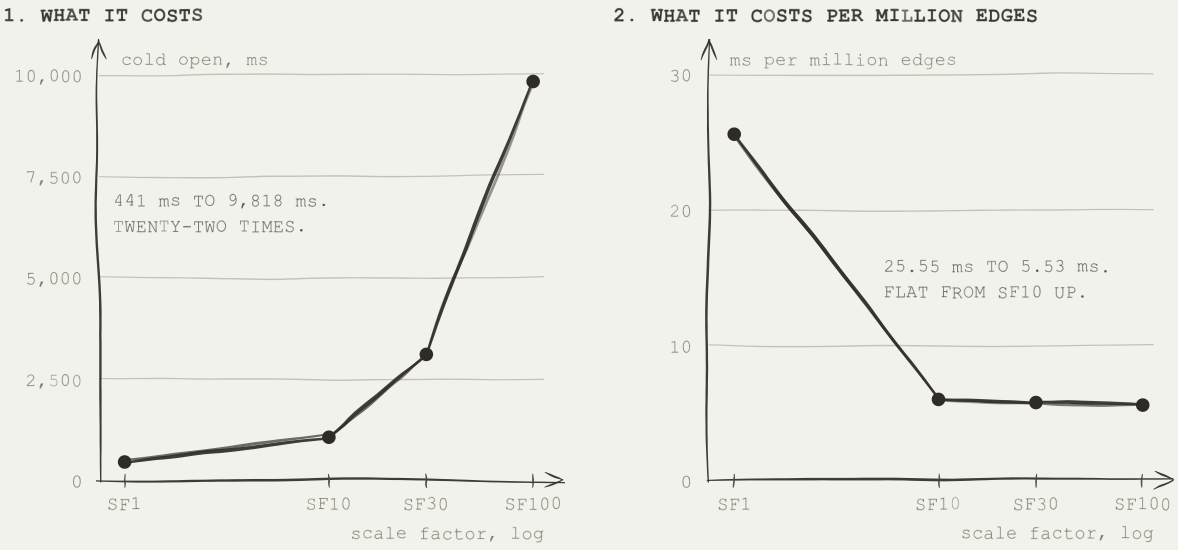


Figure 12. Cold open in milliseconds, then milliseconds per million edges.

APPENDIX 5

MEMORY

Supports “The data matrix never needs to be ‘unpacked’ between systems, so it takes no RAM beyond its own compressed size” and “Decompression / data reconstitution becomes a just-in-time read concern.”

Measured the moment the database answered its listener, before any query.

	SF1	SF10	SF30	SF100
File on disk	211.5 MiB	2.177 GiB	6.845 GiB	23.105 GiB
Resident, open and idle	289.6 MiB	1.70 GiB	4.89 GiB	**16.56 GiB**
Resident / file	1.37x	**0.78x**	**0.71x**	**0.72x**
Resident at peak in the run	1.63 GiB	9.42 GiB	28.04 GiB	88.16 GiB

A 23.105 GiB database answers in 16.56 GiB. The SF1 row exceeds 1.0 because the process floor takes a large share of a 211 MiB file.

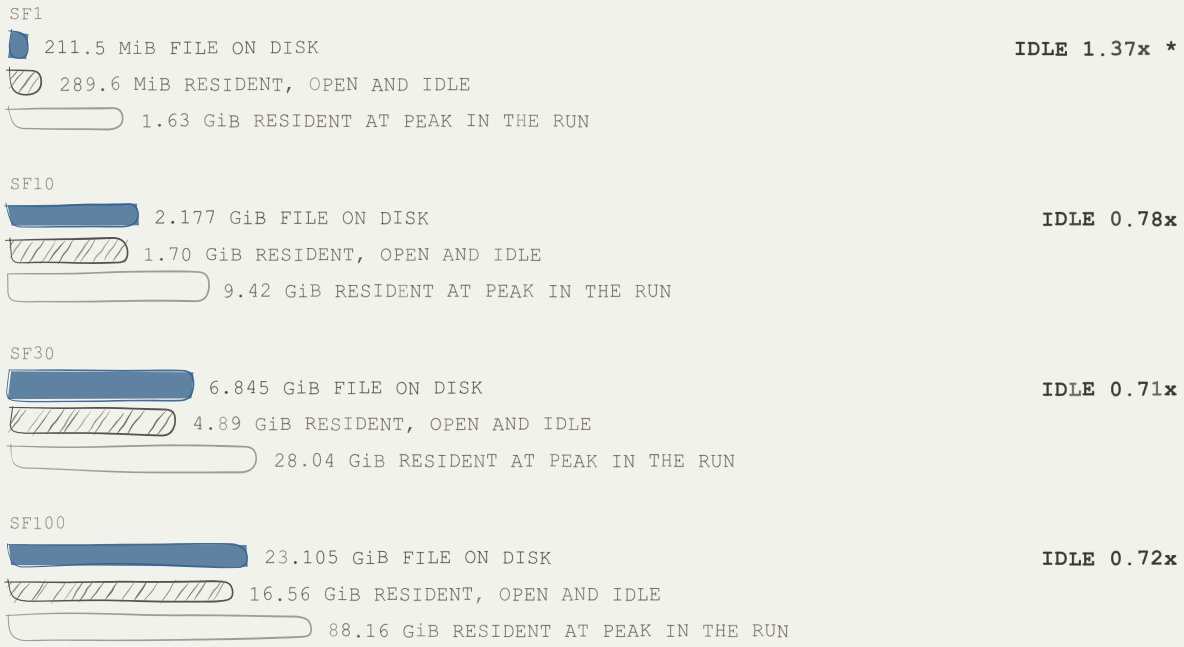


Figure 13. File, resident idle, and resident at peak, four scale factors. Log axis.

The laptop, watching a 206.2 MB database through its life:

	RESIDENT	/ FILE
just after opening	247.1 MB	1.20x
once the load settles	34.5 MB	0.17x
after the first traversal query	163.4 MB	0.79x
after 11 repeats of it	164.8 MB	0.80x
after 40 further, different queries	155.1 MB	0.75x

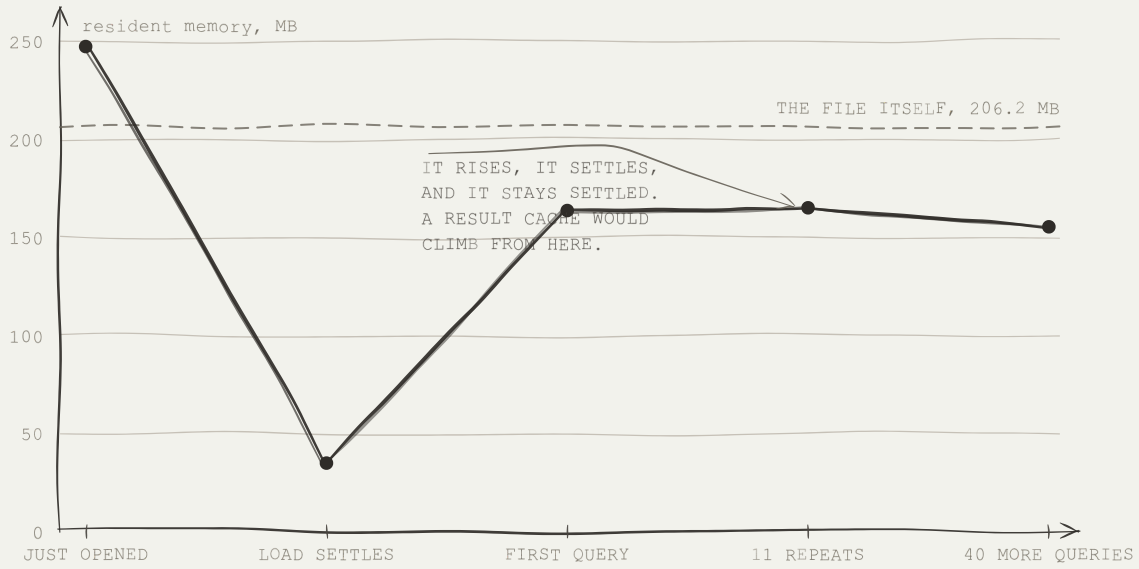


Figure 14. Resident memory of a 206.2 MB database through its life.

Reading just what a query needs, against reading the column it sits in

	NS/OP	BYTES OFF DISK
the whole 4.2 MB column	444,462	4,210,768
the part the query needs	**1,414**	**4,096**

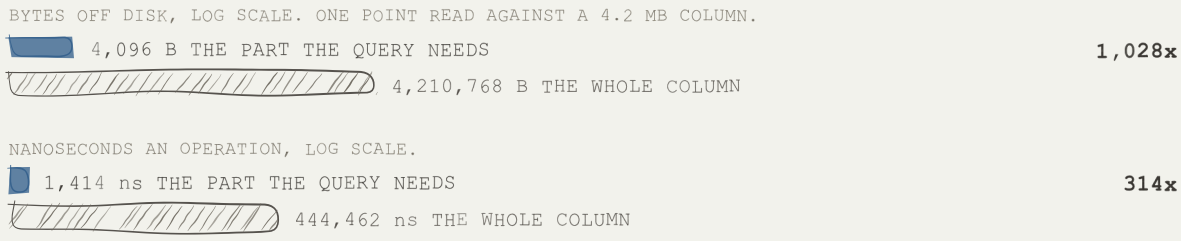


Figure 15. Bytes off disk and ns/op for one point read against the column it sits in. Log axes.

APPENDIX 6

SECURITY AND IDENTITY

Supports “cutting edge ‘ML-KEM’ (quantum resistant) key encapsulation and AES encryption on the datastore,” “Encryption happens before the matrix is stored, per file block … it gets decrypted at read-time,” and “we incur minimal encryption overhead.”

Every database in this appendix is encrypted. The compression figures in Appendix 1 compare encrypted files against plain CSV.

Key encapsulation	ML-KEM-768 (quantum resistant)
Encryption	AES-256-GCM
Granularity	per file block, each independently decryptable
Overhead on disk	**16 bytes per 4,080-byte block, 0.392%**

	COST	FREQUENCY
unlocking from a passphrase	43.2 ms	once per process
the quantum-resistant step	34.8 µs	**once per database opened**
preparing a column’s key	213 ns	once per column
decrypting one block	**840 ns**	only for blocks a query reads
reading a whole column	116,313 ns	**9.0 GB/s**
writing a whole column	188,712 ns	**5.6 GB/s**

The quantum-resistant step costs 34.8 microseconds once per database, however large it is. Every read after that runs on AES, which the processor implements in hardware.

APPENDIX 7

WRITES, RECOVERY AND COMPACTION

Supports “we simply extend the matrix with overwrites, and we store these writes to a mutations file” and the RTO / RPO / point-in-time-recovery discussion.

Merging a mutations file back into the database must answer every question identically. We test that, and seven other properties, on every build.

	PROPERTY	
1	An empty mutations file changes no answer	pass
2	Writes to different things are order-independent	pass
3	A deleted thing leaves every subsequent answer	pass
4	A newer value shadows an older one; latest write wins	pass
5	Labels added and removed resolve correctly in either order	pass
6	Identifiers stay stable and keep counting up	pass
7	Write ordering stays strictly monotonic across checkpoints	pass
8	**Compacting answers identically to not compacting, for every query**	pass

20,000 nodes with 8 properties each	645,089 B
Per node	32.3 B
Per property written	**4.03 B**
Rewriting one value 400 times	62 B each
Widening the schema, per change	357 B

Four hundred rewrites of one field grow the database by 25 KB, not by four hundred copies of the row. Widening the schema records the change beside the data instead of rewriting it.

You restore to any point by write number or by wall-clock time. Picking that point needs no encryption key, so an operator lists the available recovery points without reading the data.

APPENDIX 8

NO SCHEMA, AND NO CACHING

Supports “TETRAbase has no schema,” “you will have to restructure your data less,” and “Most data requires no caching with TETRAbase.”

Loading the LDBC dataset requires no schema definition. The engine finds the shape in the data.

	FOUND IN THE DATA
Properties	19
Labels	14
Kinds of relationship	16
Properties holding more than one type	**2**

Two of the nineteen properties hold values of more than one type. They load and query without complaint; a schema would have rejected the load or converted the column.

A result cache grows with what it keeps and leaves a permanent gap between a first query and a repeat. Neither happens.

	LATENCY	RESIDENT
first run of a traversal query	35.2 ms	42 to 167.4 MB
the same query, 11 more times	1.29 to 1.69 ms	167.4 to 164.8 MB
40 different queries	1.19 to 1.38 ms	157.7 to 158.8 MB

Forty different questions cost about 27 KB each and did not move the response time. Once a client holds a result, reading it a hundred times costs zero additional memory, at 20,000 rows and at 200,000 rows alike.

APPENDIX 9

ECOSYSTEM INTEGRATION

Supports “supporting the open source OpenCypher query language and the standard connection protocol Bolt.”

The openCypher Technology Compatibility Kit is the industry’s own conformance suite for the query language. It checks answers, not speed.

Scenarios	3,897
Passed	**3,897**
Failed	0
Skipped	0

MATCH, WHERE, RETURN, WITH, UNION, OPTIONAL MATCH, CREATE, SET, DELETE, MERGE, variable-length paths, pattern comprehensions, aggregations, CASE expressions, temporal types, list and map semantics, null handling.

PROTOCOL

Bolt, including TLS	Neo4j drivers and tools connect unchanged
PostgreSQL wire	takes a PostgreSQL credential table verbatim
HTTP, JSON and binary RPC	yes
TETRAbeam	yes
OpenTelemetry metrics and traces	exported

The scale ladder in Appendix 4 ran twenty-one unmodified reference Cypher queries over Bolt, checked against the reference implementation before each run.

APPENDIX 10

EDGE AND EMBEDDED

Supports “our general purpose engine is ~50 MB, but we can create small for-purpose deployments of the same technology.”

	SIZE
General purpose engine, macOS	**49.8 MB**
General purpose engine, Linux	51.8 MB
The same, symbols stripped	34.2 / 36.2 MB
A for-purpose build that only reads a database	**3.0 MB**
A for-purpose client	5.1 MB

Fifty megabytes covers server, loader, compactor, Bolt, PostgreSQL wire, TETRAbeam, encryption and the command line, in one file that installs nothing alongside it.

APPENDIX 11

ELEMENT-AT-A-TIME VS CONSTANT-STRIDE SWEEP

Supports “computers perform linear algebraic operations more efficiently than ‘scalar math’” and “we see anywhere from at 50% to a 10000% speedup.”

We keep both implementations of the same work and run them against each other. Both produce identical output.

Block-FOR decode, same lane, same output values

VALUE WIDTH	SCALAR	VECTOR	SCALAR MB/S	VECTOR MB/S	RATIO
5-bit	645,609 ns	415,647 ns	6,497	10,091	1.55x
25-bit	652,525 ns	415,512 ns	6,428	10,094	1.57x
32-bit	641,053 ns	402,426 ns	6,543	10,423	1.59x
				median	**1.57x**

Row-keyed vs column-oriented reduce, a different structure, not just a different sweep

JOB	SCALAR (ROW-KEYED)	VECTOR (MEMBER AXIS)	RATIO
grouped sum, dense groups	118,126 ns	12,933 ns	**9.13x**
grouped sum, sparse groups	337,962 ns	12,055 ns	**28.04x**
allocations	18	14	

Sparse groups cost the row-keyed form nearly three times what dense groups cost it. They cost the column form nothing. The equality predicate sweep, nine column shapes, has a median of 1.62x.

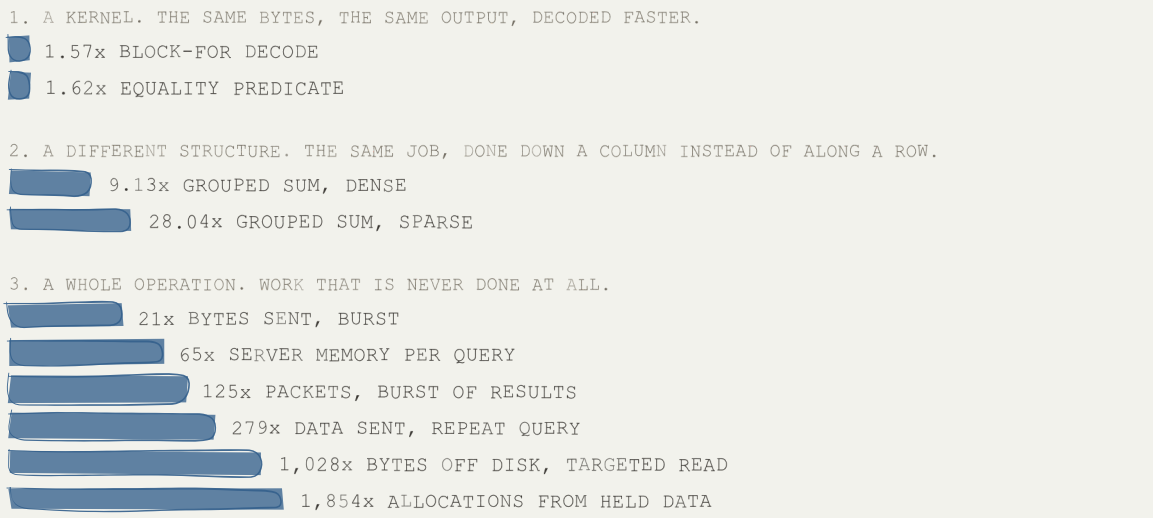


Figure 16. Kernel, structure, and whole-operation ratios. Log axis.

Whole operations, across a full request

	RATIO
bytes off disk for a targeted read	**1,028x**
memory allocations to answer from held data	**1,854x**
data sent for a repeat query	**279x**
network packets for a burst of results	**125x**
server memory to answer a query	**65x**
bytes sent for a burst of results	**21x**

APPENDIX 12

TETRABEAM OVER THE WIRE

Supports “the TETRAbeam format being 60x to 90x more efficient (end to end) than other data format methods.”

The same results, the same connection, the same encryption, framed two ways: one result per message, and packed. The workload is 5,000 distinct results from a 13,756-element fleet.

Framing (5,000 x 5), these byte figures reproduce identically on both machines

	FRAME-PER-RUN	PACKED
Frames	5,000	**40**
Segments	5,000	**40**
Sealed size	1,195,381 B	**56,989 B**
Bytes per result	239.08	**11.40**
Results per frame	1.0	**125.0**
Encode passes per frame	1	**1**

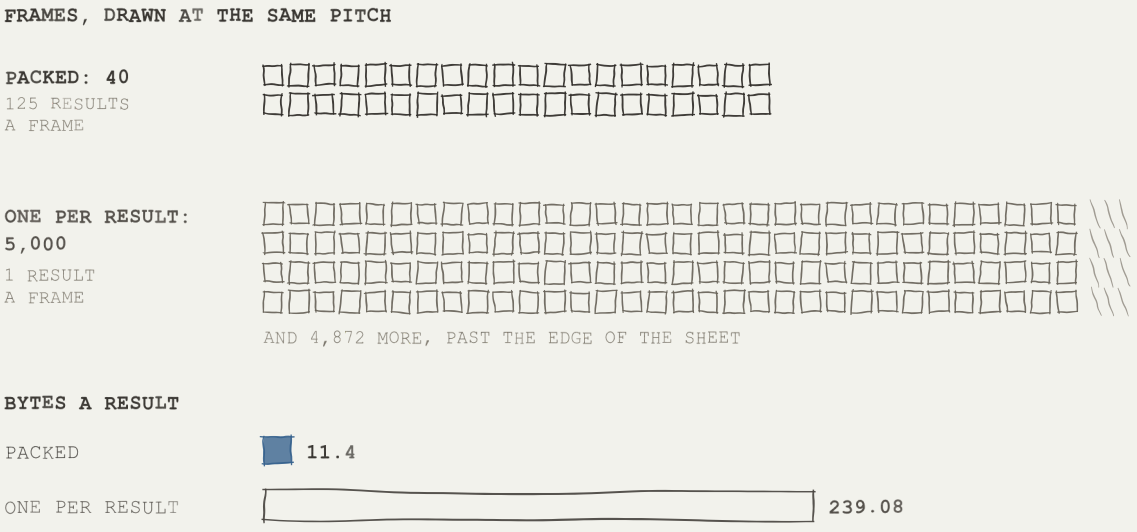
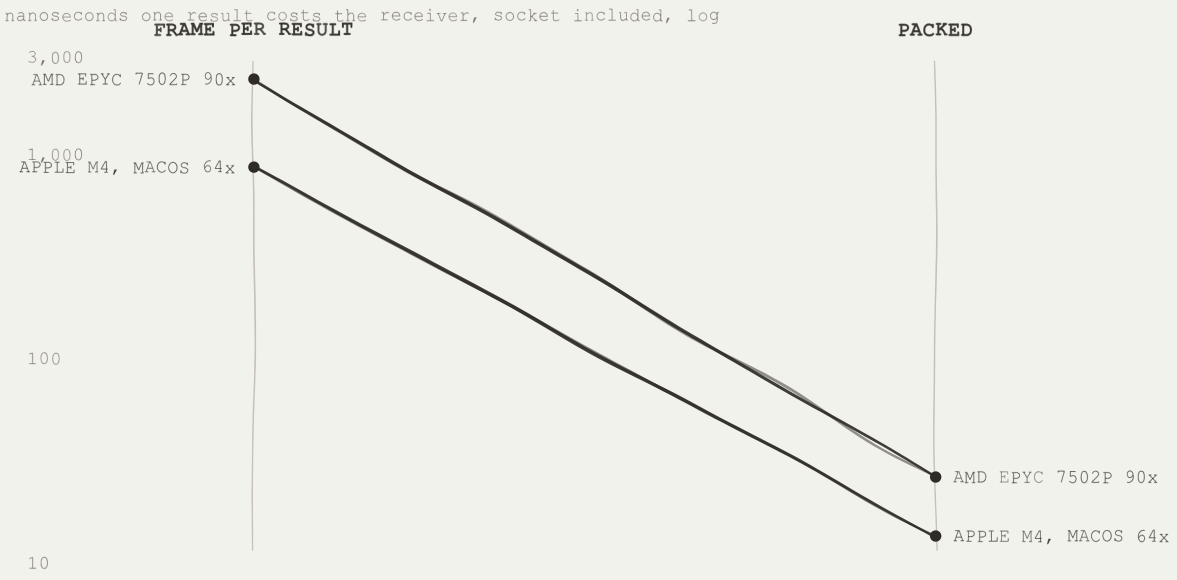


Figure 17. Frames and bytes per result.

End to end, socket included, what one result costs the receiver

	DARWIN/ARM64	EPYC 7502P
frame-per-run	851.0 ns	2,300.4 ns
packed	**13.2 ns**	**25.7 ns**
ratio	**64x**	**90x**

Under an MTU budget: 5,000 results, 39 to 40 messages, 39 to 40 encode passes, 55,386 to 56,989 B sealed, 11.1 to 11.4 B a result.



851.0 ns TO 13.2 ns ON THE LAPTOP, 64x. 2,300.4 ns TO 25.7 ns ON THE SERVER, 90x.
THE BYTE COUNTS BEHIND BOTH ARMS REPRODUCE IDENTICALLY ON THE TWO MACHINES. ONLY THE TIME DIFFERS.

Figure 18. One result, socket included, on two machines. Log axis.

A query result over the wire, 200,000 rows, three columns

FORMAT	WIRE BYTES	BYTES/ROW	SERVER MEMORY	BUILDS EVERY ROW?
TETRAbeam, cold session	1,320,763	6.60	10,772,480 B	no
TETRAbeam, warm session	**17,540**	**0.09**	**226,808 B**	no
Bolt packstream	5,345,370	26.73	22,406,592 B	yes, every row
JSON rows	10,888,403	54.44	190,907,696 B	yes, every row
JSON columnar	5,488,451	27.44	52,731,512 B	yes, every row
CSV	4,688,423	23.44	51,165,048 B	yes, every row

A server that builds every row spends memory in proportion to the result. Ours spends the same 226,808 bytes at 2,000 rows and at 200,000 rows.

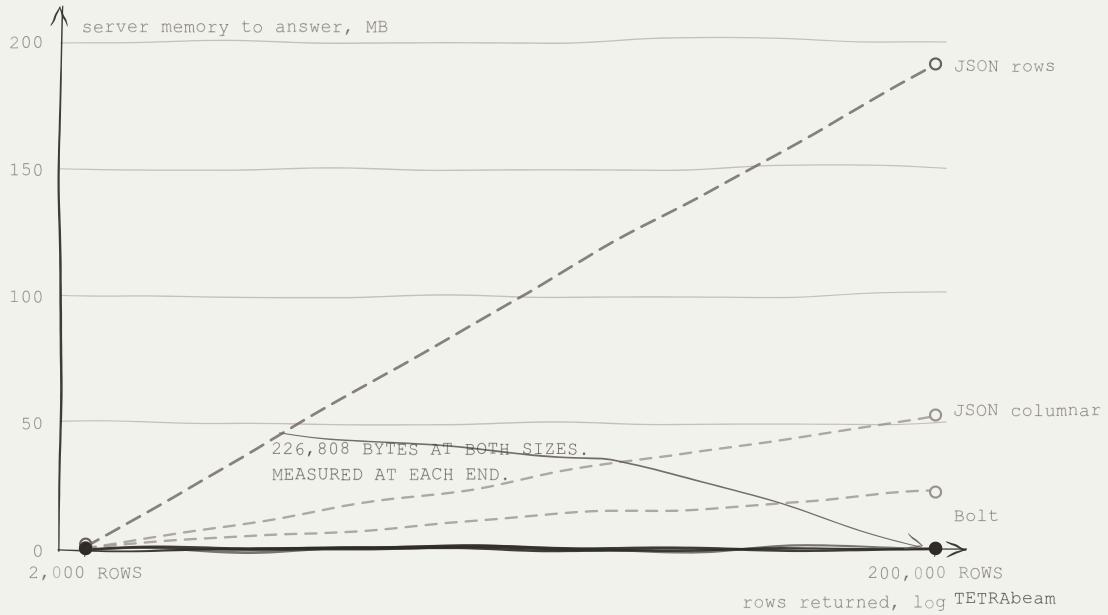


Figure 19. Server memory against rows returned. Log axis.



ONE QUERY RESULT, 200,000 ROWS, THREE COLUMNS. FILLED POINTS DO NOT BUILD EVERY ROW ON THE SERVER; OPEN POINTS DO.
WARM MEANS THE SESSION ALREADY HOLDS THE COLUMN STATE, SO A REPEAT SENDS THE VALUES AND NOT THE DESCRIPTION OF THEM.
THAT IS WHY THE WARM POINT SITS IN A DIFFERENT QUADRANT: 17,540 BYTES AND 226,808 BYTES OF SERVER MEMORY.

Figure 20. Bytes per row against server memory, six formats. Log axes.

APPENDIX 13

HEAD TO HEAD

Supports “TETRA takes 12.5% of the storage disk space when compared to traditional indices and storage (Postgres and Neo4j both measured similarly),” “TETRA performs ~3x more memory-efficiently than comparable data systems,” “stores and retrieves information at parity or better with industry standard tooling while using a half to a third of the RAM,” and the 66% infrastructure cost reduction the paper broadcasts.

SF1, mixed read and write, eighteen query shapes, 30% writes, nine concurrency levels from 1 to 256 clients. One set of parameters sampled once and replayed against both systems, back to back on one machine.

Footprint

	TETRA	NEO4J 5.26
Nodes	3,181,724	3,181,724
Edges	17,256,038	17,256,038
On disk	**211.5 MiB**	1.683 GiB
Transaction logs	**none**	808 MiB
Bytes per edge	**12.9**	104.7
Indexes and constraints	**none**	9 constraints + 8 indexes
Encrypted at rest	**yes**	no

211.5 MiB against 1.683 GiB is 12.3%. Against PostgreSQL on a four-million-row corpus, with the four indexes the benchmark builds, 80.9 MiB against 653 MiB is 12.4%.

WHAT EACH SYSTEM HAD TURNED ON	TETRAbase	NEO4J 5.26	ADVANTAGE
ENCRYPTED AT REST	YES	NO	FAVOURS THEM
INDEXES AND CONSTRAINTS	NONE	9 + 8	FAVOURS THEM
TRANSACTION LOGS	NONE	808 MiB	FAVOURS THEM
VERSION	2026-09-03 BUILD	NEO4J 5.26	
ON DISK	211.5 MiB	1.683 GiB	FAVOURS US
BYTES PER EDGE	12.9	104.7	FAVOURS US

AGAINST POSTGRESQL	80.9 MiB	653 MiB	**12.4%**

A DIFFERENT MEASUREMENT ON A FOUR-MILLION-ROW CORPUS, NOT THE LDBC LADDER, WITH THE FOUR INDEXES THE BENCHMARK BUILDS. IT IS NOT PART OF THE HEAD TO HEAD ABOVE AND MUST NOT BE READ AS ONE OF ITS ROWS.

Figure 21. What each system had turned on.

Throughput and resident memory, every level

CLIENTS	TETRA OPS/S	NEO4J OPS/S	TETRA RSS	NEO4J RSS
1	**956**	442	**1,290 MB**	3,590 MB
2	**1,743**	860	**1,668 MB**	3,599 MB
4	**2,849**	1,534	**1,818 MB**	3,615 MB
8	**4,278**	2,791	**1,805 MB**	3,622 MB
16	**7,463**	4,708	**1,996 MB**	3,652 MB
32	**8,299**	7,056	**2,141 MB**	3,777 MB
64	8,111	**8,411**	**2,586 MB**	3,788 MB
128	**9,173**	7,512	**2,629 MB**	5,691 MB
256	**7,524**	6,692	**2,716 MB**	7,050 MB

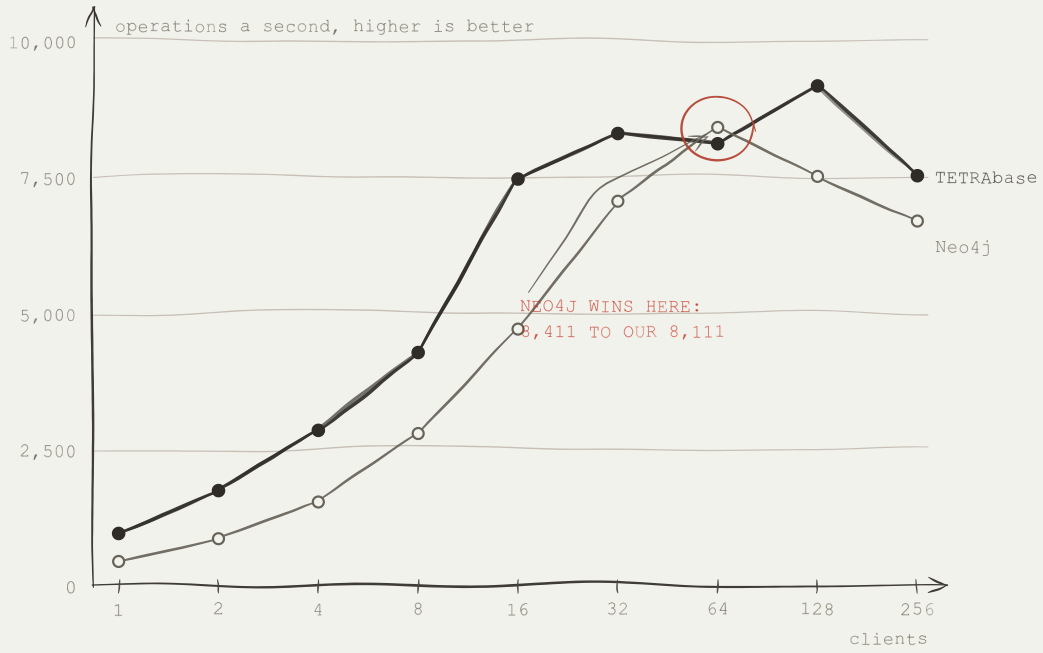


Figure 22. Operations a second against clients. Log axis.

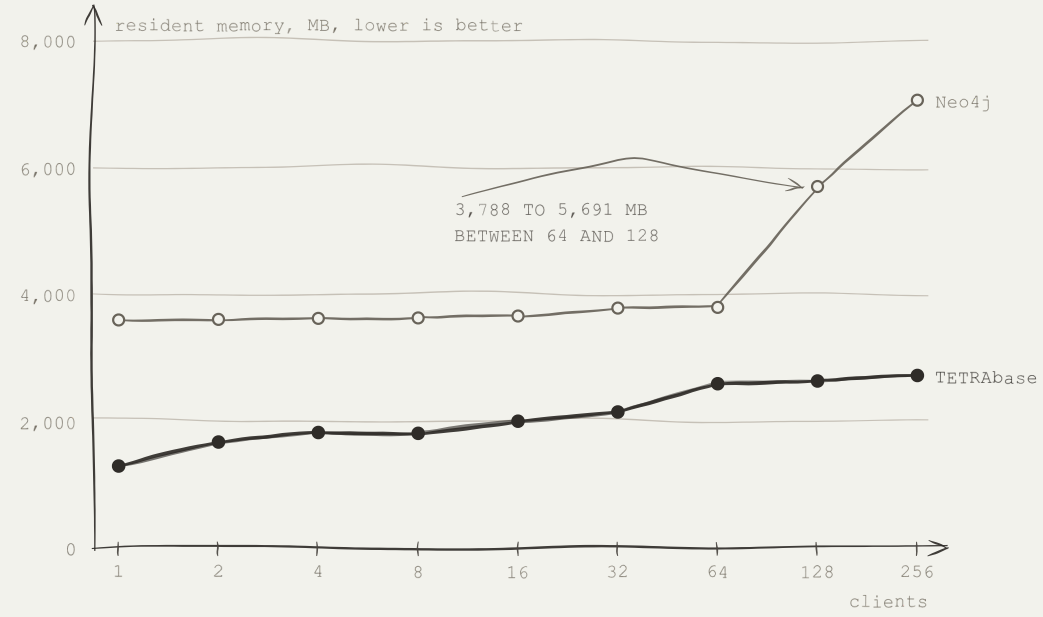


Figure 23. Resident memory against clients. Log axis.

Under load

	TETRA	NEO4J 5.26
Slowest 1% of responses, 256 clients	**102.40 ms**	344.06 ms
Worst single response across the sweep	**194.84 ms**	1,343.34 ms
Operations served across the sweep	**1,011,198**	803,014
Writes readable immediately after acknowledgement	2,223 of 2,223	1,651 of 1,651
Query errors	**0**	4
CPU per operation, 16 clients	**2.68 ms**	2.96 ms
CPU per operation, 128 clients	**4.58 ms**	6.74 ms

A further cross-platform run is under way and will extend these figures.

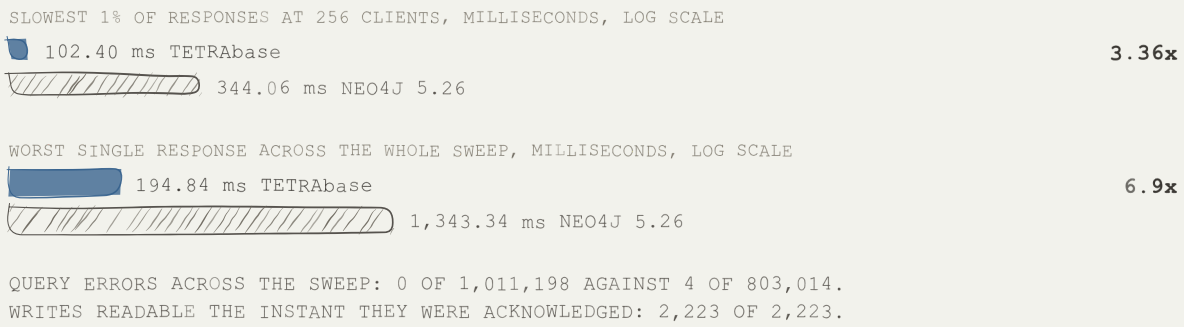


Figure 24. p99 at 256 clients and worst single response. Log axis.

To get started today

Please email your cloud bill.

We did the math, and we conservatively estimate a two-thirds reduction on your current infrastructure costs.

TETRA AND HORNESCI

inquiries@tetra-ca.com

tetra-ca.com

DRAFT · 2026-09-06
TETRA CA PBC · HorneSci LLC