



A Streaming Graph System For High-Volume Complex Event Processing

Designed to find complex patterns within high volume data streams composed of heterogeneous feeds without resort to time windows.

Origins and Challenges

Stream processing and event-driven microservices are complicated! They're complicated because they combine the hardest problems from the database domain with the hardest problems from the distributed systems domain.

At their core, both domains bring a long list of challenges that are due to the complex interplay of fundamental mathematical or physical constraints that will inevitably surface in any system that stores data and has more than one processor.

Researchers and developers spend entire careers developing novel ways to overcome the constraints and tradeoffs of a single aspect of these systems. Issues like consensus, concurrency, transactional logic, clustering behavior, fault tolerance, scalability, balancing read and write performance tradeoffs, and temporal ordering are all subjects of research and innovation.

However, when the system in question is primarily concerned with processing events as quickly as possible, one dimension in particular takes on outsized importance: time.

Quine is an open source streaming graph software project sponsored by thatDot. Visit quine.io to learn more.

Time and Distributed Systems

Processing vast amounts of event data, especially when the system is composed of many microservices that rely on a strict order of operations, presents particular challenges. How does such a system handle out-of-order data? What if data arrives hours, days, or months after it was generated? And how will such a system know when to execute a query? When will the system be guaranteed to have all the data and in proper order?

Consider for instance the previously mentioned problem of when to query data. Systems today must continuously query, or poll. Polling consumes system resources and increases query latency, both of which impact not just the process or service in operation at the moment, but have the knock on effect of delaying when the next service can guarantee that it has the necessary data.

A New Approach

What if we could eliminate many of the drawbacks of event processing and database systems and build a system from the ground up which confronts each of these fundamental challenges from a holistic perspective and with the goals of modern applications in mind? What if we could build a system to deliver high-throughput, low-latency reads and writes over unbounded data? One that is optimized to handle the constraints of distributed systems and databases under high concurrency workloads?



After seven years of research and development, and with major funding from DARPA, this is exactly what the team at thatDot has done. We call this system Quine.

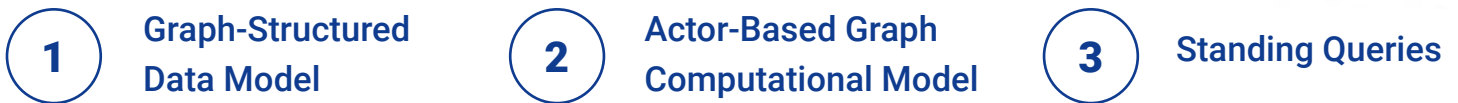
Quine: Streaming Graph for Real-time Applications

Quine is the open-source streaming graph and computation platform developed by the engineers at thatDot. It bridges the worlds of distributed event stream processing systems and databases.

Quine is designed to find complex patterns within high volume data streams composed of heterogeneous feeds without resort to time windows. It eliminates the headaches involved with building and operating event-driven microservices that enable real-time applications.

Core Design Choices and Their Implications

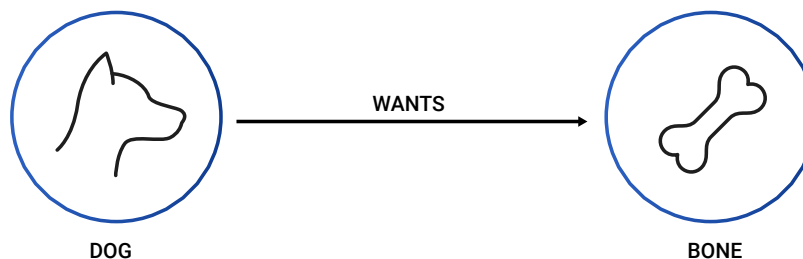
Three design choices define Quine, setting it apart from all event stream processing systems: a graph-structured data model, an asynchronous actor-based graph computational model, and standing queries, Quine's solution to the challenges time presents in distributed systems.



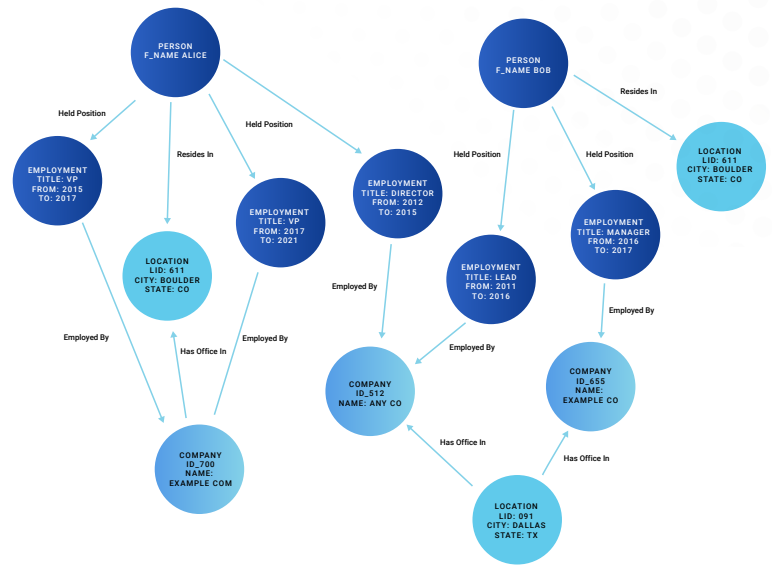
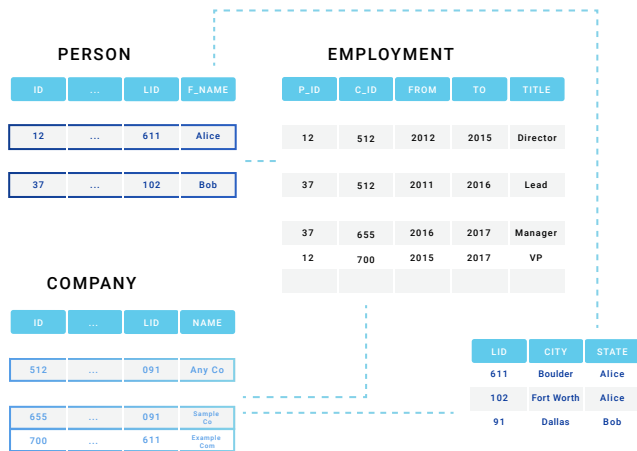
Graph is the Universal Data Structure

Quine uses a property graph model to store and query real-time data. In this regard, it functions similarly to graph databases; it even uses the most common graph query language, Cypher, to work with the data. In a graph model, data is represented as nodes connected to each other by edges. Nodes hold key-value pairs of "properties." The edges have a direction and a label, and connect exactly two nodes.

Graph data structures represent data and its relationships in a way strikingly similar to how humans often think and talk about data. The node-edge-node pattern in a graph corresponds directly to the subject-predicate-object pattern common to languages like English. This makes graphs both powerful and easy to understand. When visualized, a graph becomes endlessly intriguing.



Quine's property graph structure puts relationships at the same level as the data values themselves by encoding these relationships as edges. To discover relationships, one need only traverse a node's edges. Traversing the edge of a node to its neighbor is analogous to computing a join across tables in a relational data model.

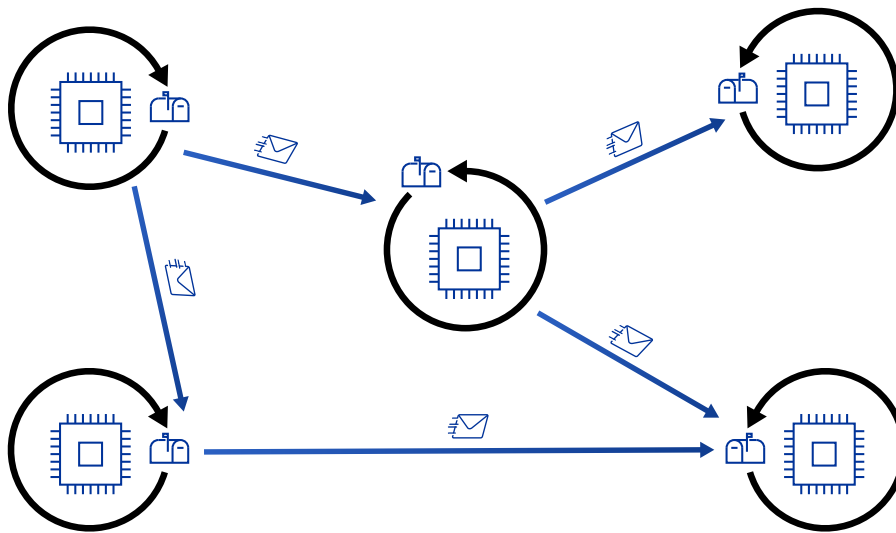


However, unlike relational databases, these relationships are essentially pre-computed joins. They make it easy and extremely efficient to use the relationships in data—which is how Quine is able to find complex patterns across both high-volume heterogeneous data streams and large historical data sets in real time.

A Graph Model for Asynchronous Computation: the Actor Model

In Quine, the graph data model is paired with a graph computational model. Computation is implemented as a native graph interpreter which occurs directly on the data. Ingested data, queries, or other instructions are inserted into the graph and propagate through the network of nodes and edges to compute the appropriate answer or trigger an action. The result is a fast and efficient process for highly parallel and fully asynchronous computation that executes inside the graph.

Computation in Quine is built on the Actor Model using [Akka](#). First described by [Carl Hewitt](#) in 1973, an actor is a lightweight, single-threaded process that encapsulates state and communicates with the outside world only through message passing. An actor receives messages in its mailbox and performs the corresponding small-scale computation.



MESSAGE HANDLING

Provides universal, event-driven, incremental computation

Implements complex protocols

Mixed in to nodes

Easily extended

Built on the battle-tested Akka OSS library

Actors are scheduled for computation concurrently with other actors. This makes the overall system computation highly parallel. Under heavy load, the scheduler utilizes all available CPU cores automatically, scheduling multiple separate actors to process their messages concurrently. Actor scheduling is done on a highly-efficient “work-stealing” fork-join thread pool, running efficiently on a small machine, or taking advantage of massive compute resources available on large machines. The net result is a fast, efficient, reactive system that provides very high throughput for complex event processing.

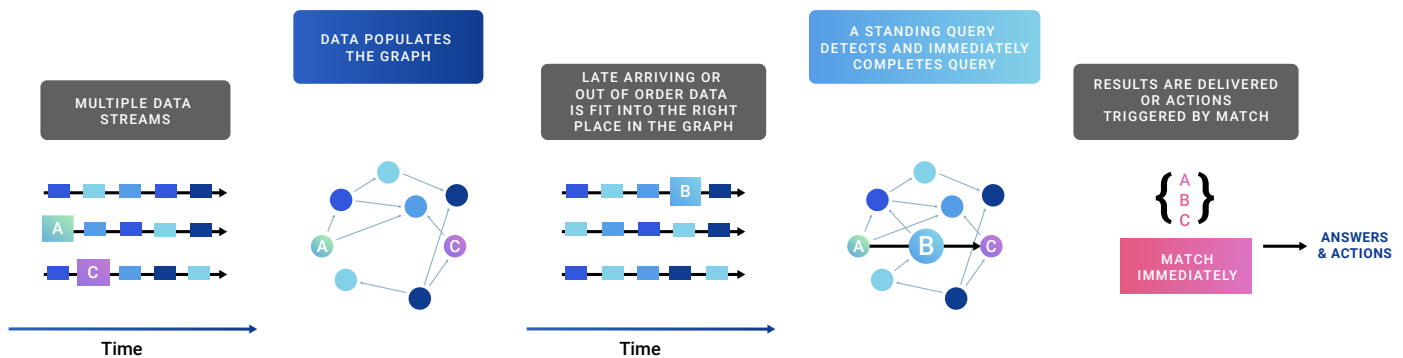
Standing Queries Change Everything!

Standing queries are the central innovation at the heart of Quine. Built on the pillars of the graph data and computational models, standing queries in Quine eliminate the time-based challenges otherwise inherent to distributed systems. But the implications of what they mean for building complex systems with Quine reach far beyond that.

Standing queries live inside the graph and automatically propagate their incremental results computed from both historical data and incoming streaming data. Once matches are found, standing queries trigger actions using those results (e.g. report results, execute code, transform other data in the graph, publish data to another source). Because standing queries persist in the graph, incrementally updating partial results as new data arrives, you are not just querying the past and present state, you are querying the future for any matches from data yet to arrive.

Out-of-order and Late-Arriving Data With No Time Windows

Quine’s unique standing query capability allows it to easily handle out-of-order data and late-arriving data. A standing query is issued, ready to complete the sought-after pattern regardless of what order the data arrives, even if delayed by hours or months. Because Quine lets the user focus on the structure of the data instead of the order of events—even if data arrives entirely backwards—Quine will provide the correct answers immediately when all the relevant data has arrived.



Instead of needing to continuously poll to determine if data has arrived, as you must with other event processing systems, Quine’s state is continuously updated and each match triggers the appropriate action automatically. Because data is stored on disk, retrieved and managed automatically, new data can easily be combined transparently with very old data. Because results stream out immediately when a match is found, standing queries eliminate the need to keep checking whether the data is complete—eliminating all the operational complexity and wasted resource overhead that entails. Instead you can focus attention on the business problems that led you to implement an event-driven architecture in the first place.

Performance and Flexibility

Quine’s architecture gives rise to some novel and quite powerful capabilities that directly address the tradeoffs and limitations of existing distributed stream processing systems.

High Read and Write Performance

Quine's graph-based data and computational model also make it possible to achieve both high read and high write performance.

High read performance is achieved through an approach we call "semantic caching." The graph data structure, and specifically the edges between nodes, provides what are effectively pre-computed joins for the entire data set, but they are computed incrementally and the moment when it is most efficient. With the connections in the data easily accessible, they become the ideal clue to what other data is worth keeping warm in the cache. Semantic caching is another key to Quine's high-throughput and low-latency computation made possible by the unified graph representation of both data and computation. With related data kept in memory, the result is a remarkably high cache hit rate and great performance.

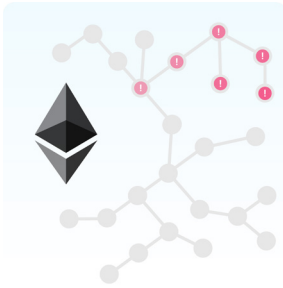
Quine's actor-based compute model is also key to achieving high write throughput. Since each node is an actor and has the ability to do arbitrary computation, Quine can efficiently test whether updates or writes actually need to be stored on disk. If they do, Quine writes only the minimum possible delta. This operation is then combined with a write-optimized data store like RocksDB or Cassandra to deliver fast and efficient write operations.

Balance: schema-less flexibility with schema-full data structure

Quine's unified graph provides the ideal sweet spot between flexibility and structure in the schema. You do not always have to know the shape of all data before the first write, and yet can still query and compute on the data efficiently and reliably.

Quine's high-throughput and low-latency computation made possible by the unified graph representation of both data and computation

Recipes Make it Easy to Try Quine Yourself



Real-Time Tag Propagation Across a Blockchain

The Ethereum blockchain is ingested live from the Web, transactions are modeled in the Quine streaming graph and a Quine Standing Query propagates a “dirty money” tag in real-time across the graph to trace money laundering.



CDN Cache Efficiency By Segment

Ingest CDN Logs and calculate cache hit rate in real-time by segments: country, state, PoP, ASN to generate alerts or dashboards.



Kubernetes Event Observability

Ingest Kube events and calculate state by component, pod & service to generate alerts & trace root causes.

<https://quine.io/recipes.html>

About Quine

Quine is the open source streaming graph project created by the team at thatDot, a Portland, Oregon-based streaming event processing company. Released under the MIT with Commons Clause license in February 2022, Quine already boasts almost a dozen prominent users in the systems management, security, and CDN/networking industries.

The decision to open source the Quine graph streaming engine underscores the thatDot team’s conviction that the best infrastructure software thrives within an open, diverse community of contributors and that well-made software freely available benefits everyone.

Interested in trying Quine out for yourself?

Visit quine.io for downloads, recipes, docs, and blogs.

 **Quine Slack** <https://quine-io.slack.com/>

 **Github** <https://github.com/thatdot/quine>



<https://quine.io>