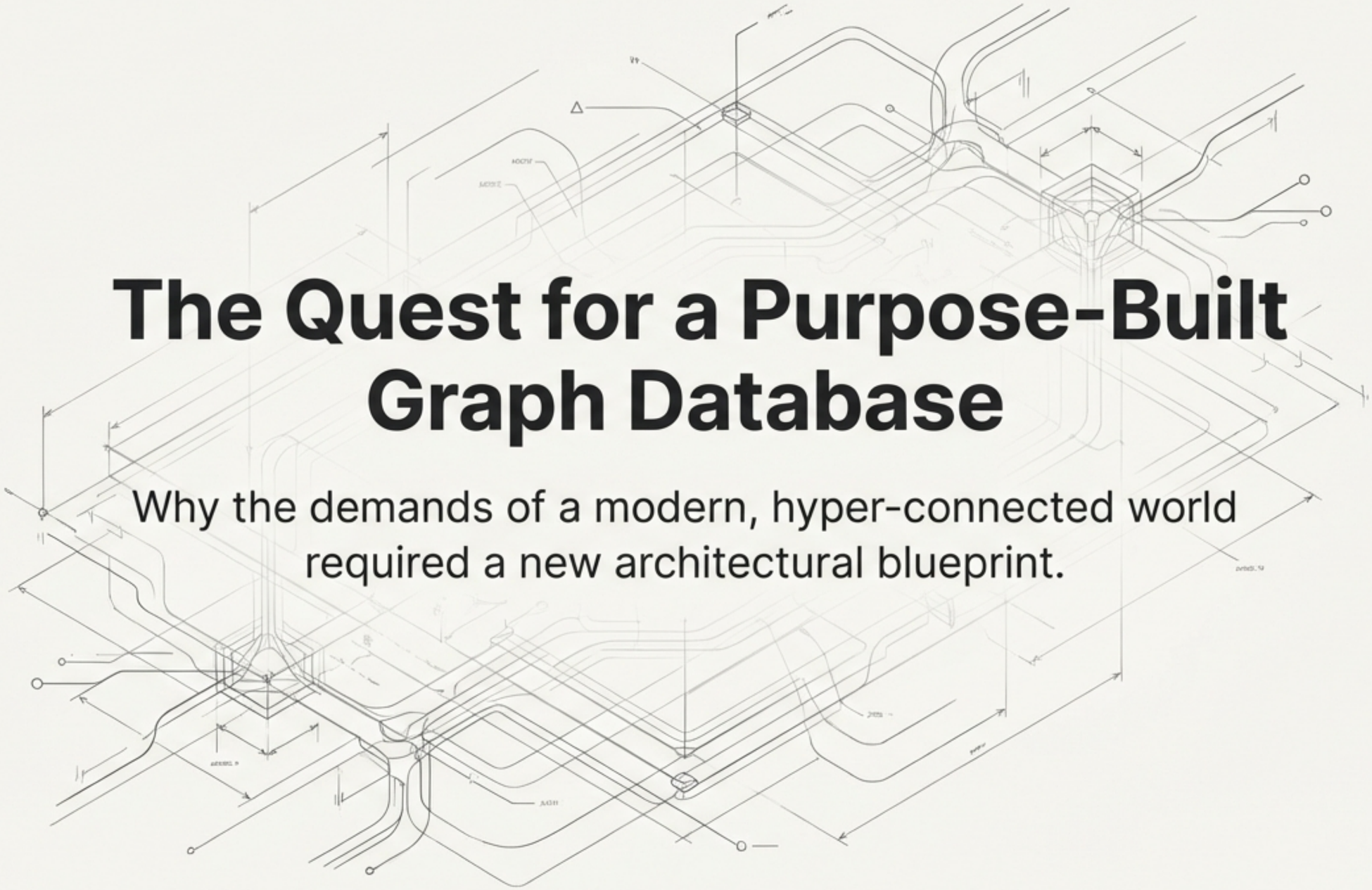


The Quest for a Purpose-Built Graph Database

Why the demands of a modern, hyper-connected world required a new architectural blueprint.

Existing Solutions Force Unacceptable Trade-offs

I set out to create a serverless graph database because no existing graph database met all of my requirements. Popular graph databases usually forced trade-offs or lacked certain features I considered essential.

Common Limitations of Traditional Graph Databases

- Always-on clusters incurring constant cost.
- Monolithic architecture that slows down with scale.
- Rigid data models limited to uniform node types.
- Proprietary lock-in or feature paywalls.
- Reliance on external tooling for management and visualisation.

The Vision for a Modern Alternative

- A truly usage-based, serverless economic model.
- A distributed, network-of-graphs approach to scale.
- A flexible model where anything can link to anything.
- A foundation of open-source transparency.
- A self-contained, developer-centric ecosystem.

The Blueprint: A New Philosophy Built on Three Foundational Pillars

A New Economic & Operational Model

Architected for true serverless operation, horizontal scalability, and radical cost-efficiency.

Unprecedented Data Flexibility

Designed to connect heterogeneous data formats in a “graph of graphs” with a multi-layered, evolving schema.

A Developer-First Experience Built on Trust

Centred on open-source principles, robust security, fluent usability, and built-in intelligence.

Pillar I: A Radically New Economic & Operational Model

> “The system should be fundamentally ‘off’ when not in use, eliminating any always-on infrastructure and its associated costs.”

Key Components Covered in this Section:

- True Serverless & Stateless Operation
- Horizontal Scaling Without Slowdown
- Asymmetric Read/Write Performance
- Intelligent, Tiered Storage

Requirement: True Serverless, On-Demand Operation

No Idle Compute

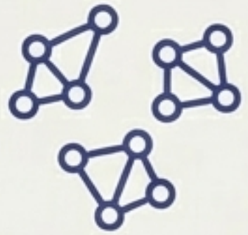
The database spins up only when in use and scales down to zero when idle. You pay only for actual usage, with zero cost for inactive periods. Compute instances are ephemeral.

No Persistent Servers

No long-lived server process or daemon to maintain, patch, or upgrade. The only state is the data storage itself; all query processing is transient, reducing administrative burden.

"Each request should stand up just enough compute to do the job and then terminate."

Requirement: Horizontal Scale and Asymmetric Performance



Scale Like a Network of Small Graphs.

Avoids the ‘monolithic slowdown’ where performance degrades as the dataset grows. The architecture encourages distributing data across multiple stores and linking them, so queries target smaller, relevant subsets.

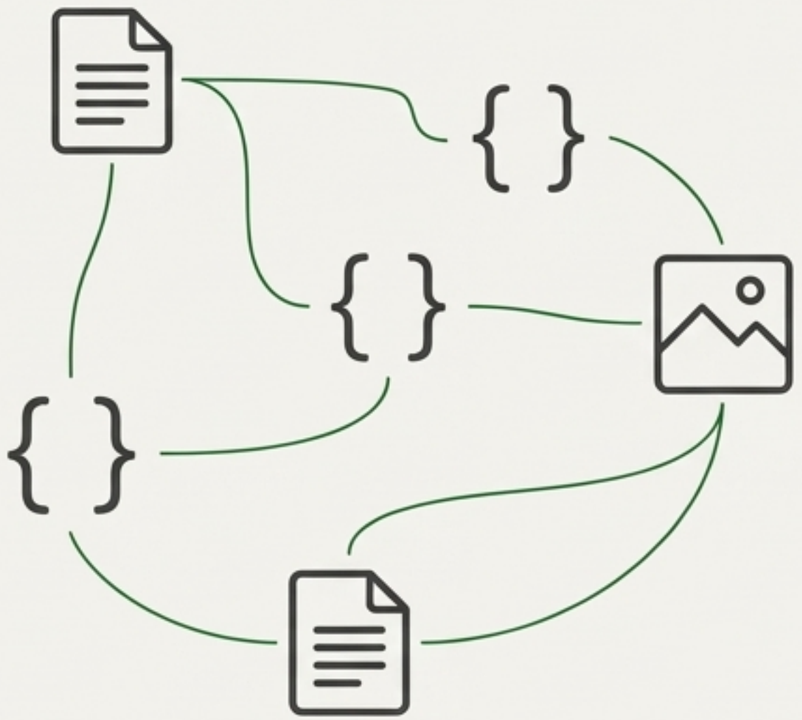
Optimised for Fast Reads, Tolerant of Slower Writes.

The system is engineered for lightning-fast graph traversals and lookups. Writes can be slower and are handled with eventual consistency, a trade-off designed for read-heavy workloads.

“The system should scale like a network of small graphs rather than one giant graph.”

Pillar II: Unprecedented Flexibility for Hyper-Connected Data

“The database should not be limited to a single type of node or edge. It must follow a liberal, web-like approach where anything can link to anything.”



Key Components Covered in this Section:

Universal Hyperlinking Across Formats
A "Graph of Graphs" (Nested Databases)
Flexible, Multi-Layered Schemas
Rich, Strongly-Typed Data Fields

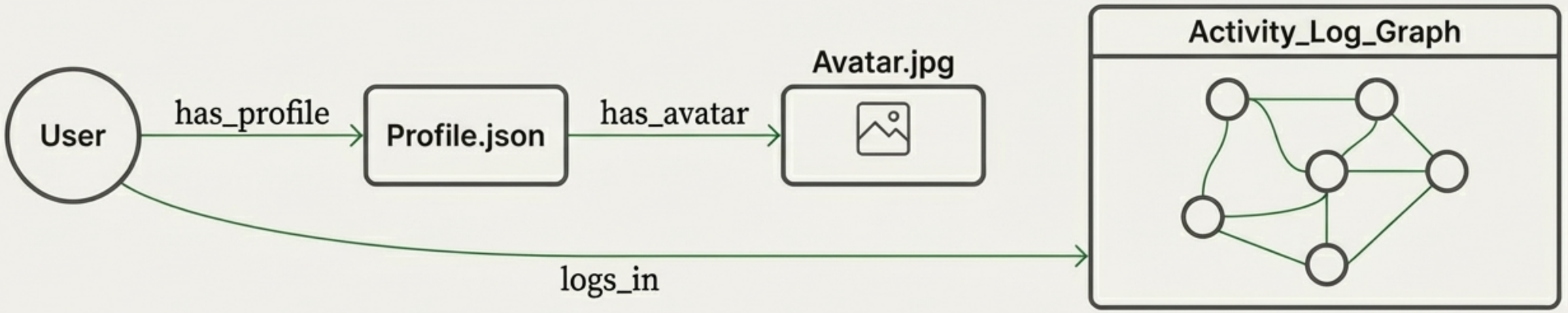
Requirement: A “Graph of Graphs” Where Where Anything Can Be Linked

Universal Linking Across Multiple Formats.

Every item—text records, JSON documents, binary blobs, CSVs—is a node with a unique ID that can link to any other item. This blurs the line between database records and external resources.

Databases Within Databases.

A complete graph can be treated as a node within another graph. This allows for modularity, isolation, and dynamic query scoping, enabling seamless traversal from one sub-graph to another.



Requirement: Multi-Layered Schemas and Rich Data Types

Flexible, Multi-Layered Schema Capability

Schema is optional. You can enforce a schema at the database level, while individual sub-graphs or files have their own schemas (or none at all). This ‘schemas of schemas’ approach supports ontologies and taxonomies without enforcing rigidity everywhere.

Rich and Strongly-Typed Data Fields

Go beyond primitives (string, integer). Define custom, domain-specific types like ‘EmailAddress’ or ‘GeoCoordinate’ with built-in validation rules. This ensures data quality and makes the data self-describing.

“It’s like having a schema with real data types rather than just saying ‘property X is a string’ – instead, ‘property X is a Password or an Email’.”

Pillar III:
A Developer-First Experience Built on Trust

> An open-source foundation, state-of-the-art security, and data transparency are not optional features; they are essential for building trust and enabling developers.

Key Components Covered in this Section:

- Open Source and No Lock-In
- Strong, Granular Security Controls
- Built-in Version Control for Data
- Fluent Querying & Integrated Visualisation

Requirement: Transparency, Security, and Verifiability

Open Source & No Lock-In

A transparent codebase with no feature paywalls. Core capabilities are available to everyone, fostering community collaboration and eliminating vendor dependency.

Strong Security & Access Control

Support for state-of-the-art practices, including Role-Based Access Control (RBAC), flexible security policies, and easy integration with encryption at rest and in transit.

Built-in Version Control for Data

Any file or node in the graph can be versionable. Enables historical tracking of what changed and when, similar to Git but for database entries, increasing auditability and trust.

Requirement: Fluent Querying and Integrated Self-Management

Fluent and Expressive Querying.

A developer-friendly interface that feels like writing code that navigates objects. Promotes composable, high-level graph traversals without the steep learning curve of some query languages.

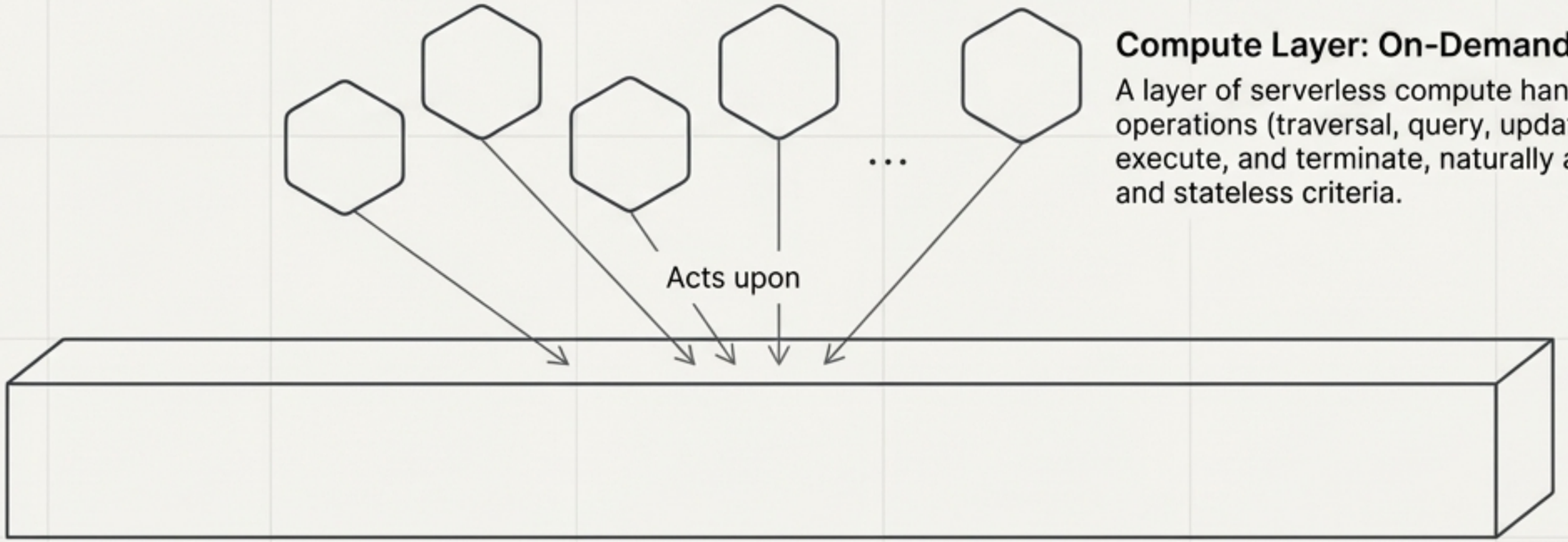
Integrated Visualisation & Self-Management.

Visualisation is a core feature, not an external tool. The graph can store its own visualisation metadata and even its own configuration (schemas, security rules), “eating its own dog food.”

The goal is an environment where you don’t leave the graph ecosystem to manage or understand the graph.

The Realisation: A Custom Solution on a File System and Serverless Functions

After listing these requirements, I realised no off-the-shelf database met every one. This gap is why I decided to build my own solution.



Compute Layer: On-Demand Serverless Functions

A layer of serverless compute handles all database operations (traversal, query, update). Functions spin up, execute, and terminate, naturally achieving the serverless and stateless criteria.

Data Store Layer: Distributed File System (e.g., Object Storage)

Inherently fiexible, hierarchical, and scalable. Files represent nodes, edges, and content, enabling tiered storage and massive capacity.

The Result: A Database That Meets the Demands of a Modern World

Cost-Efficient and Scalable

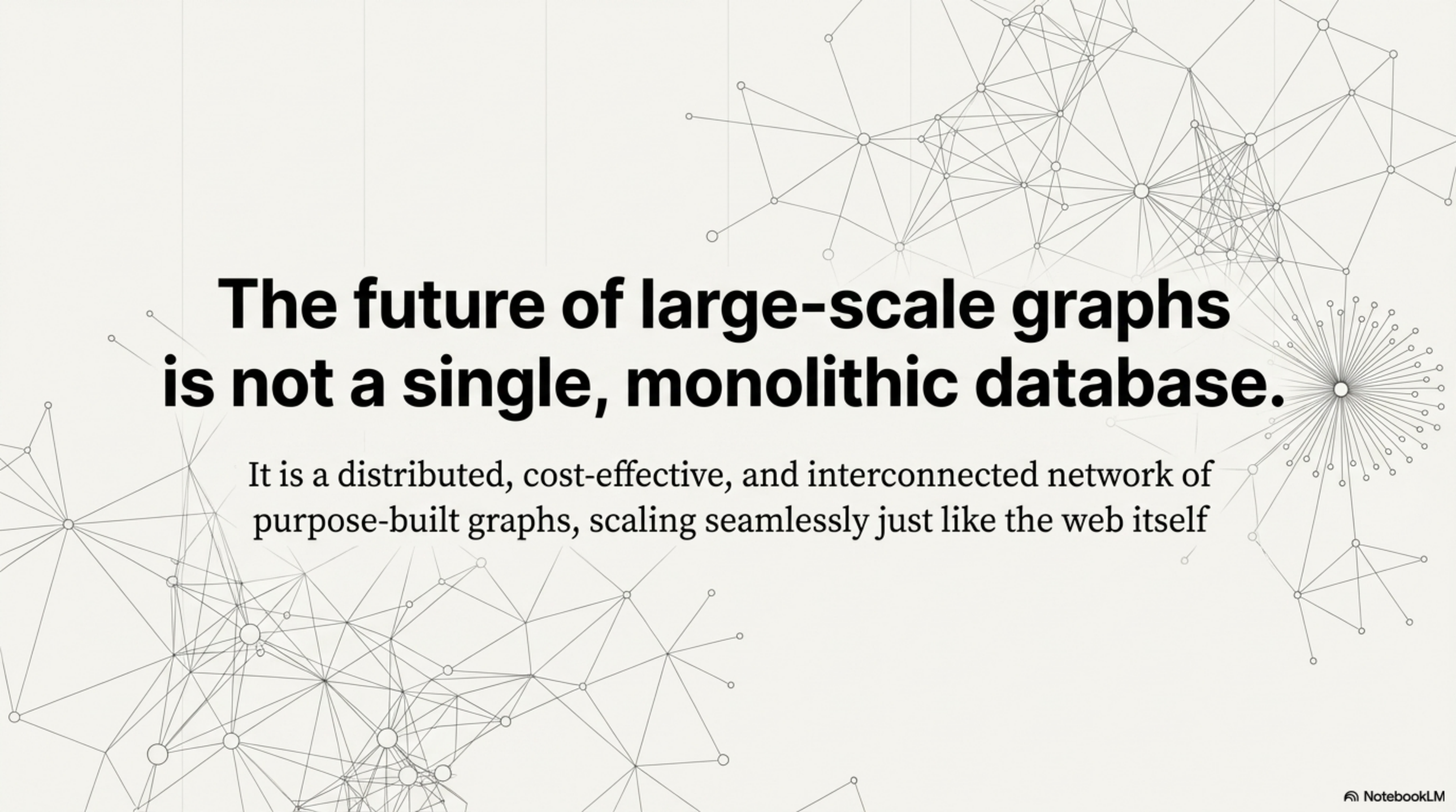
Only uses resources as needed and can store immense amounts of data in cheap, tiered storage. Grows with your data without linear performance degradation.

Highly Flexible and Interconnected

Capable of linking anything to anything, supporting evolving schemas and rich data types in a unique “graph-of-graphs” platform.

Evolving and Trustworthy

An open-source system that provides full control over its behaviour, security, and data history, with a developer-centric interface.



The future of large-scale graphs is not a single, monolithic database.

It is a distributed, cost-effective, and interconnected network of purpose-built graphs, scaling seamlessly just like the web itself