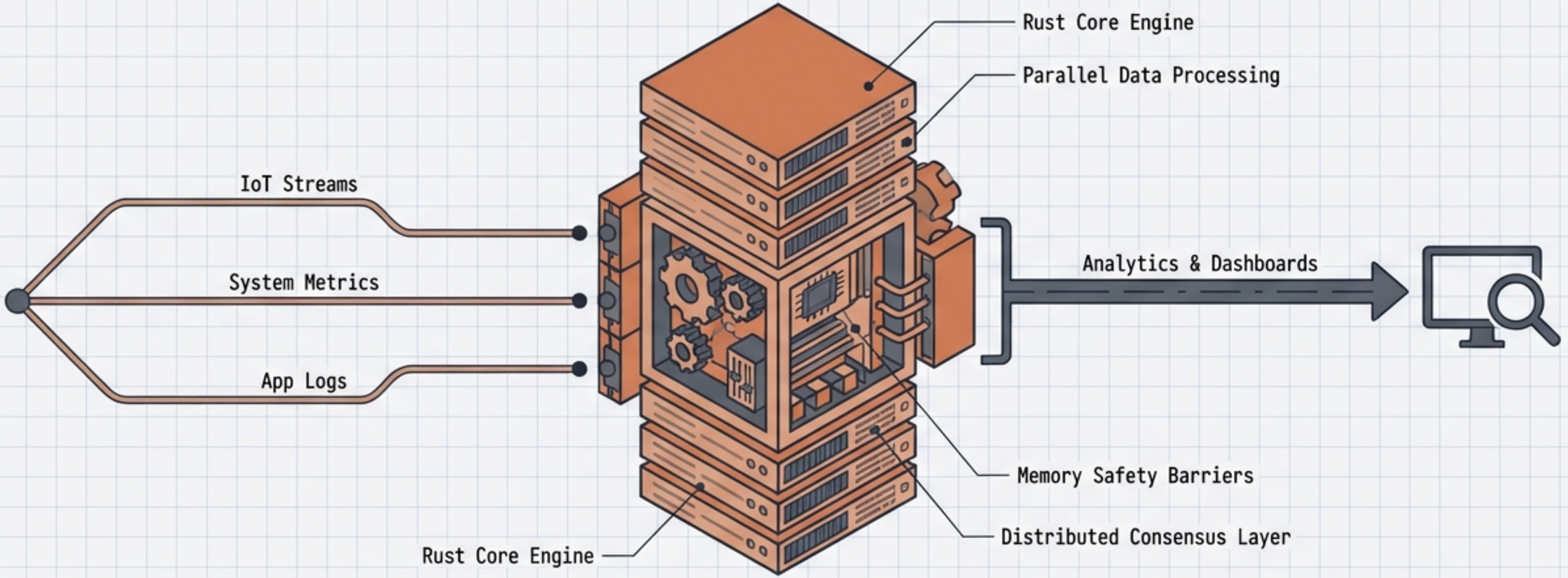
RusTs: Precision Engineering for Time Series Data

A high-performance, memory-safe, distributed database built for the velocity of IoT and real-time analytics.

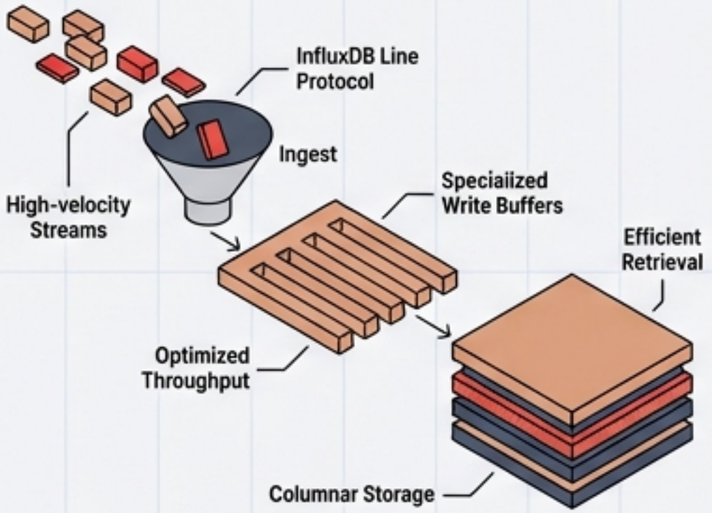


Implementation: Native Rust 1.75+ | **Protocol:** InfluxDB Line Compatible | **Storage:** Columnar + Gorilla/Delta Compression | **License:** Dual Apache 2.0 / MIT

Engineered for Speed, Efficiency, and Reliability

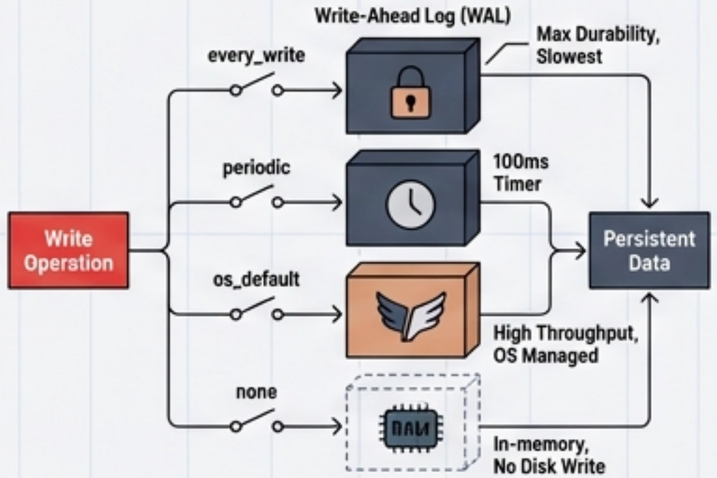
Performance & Compatibility

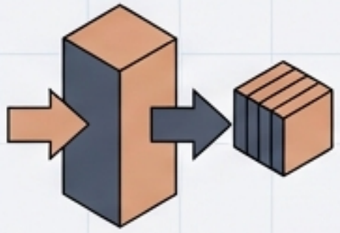
- Optimized for high-velocity time series workloads using columnar storage.
- Drop-in migration: Supports InfluxDB line protocol for immediate ecosystem integration.
- High-throughput ingestion via specialized write buffers.



Flexible Durability

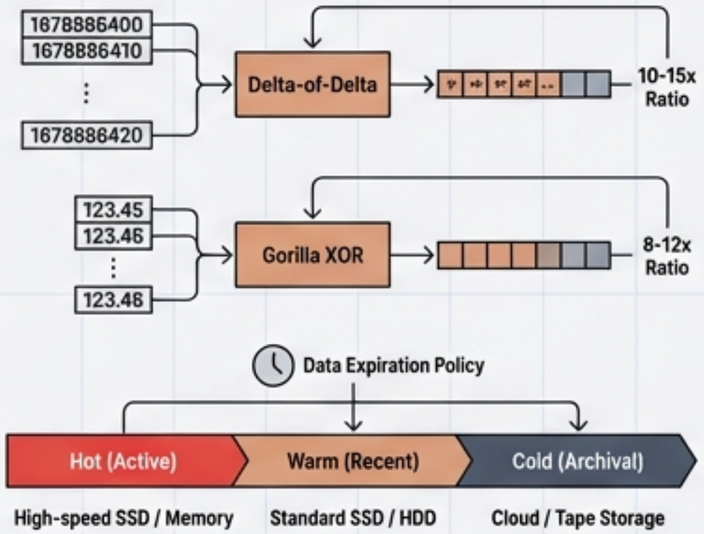
- Configurable Write-Ahead Log (WAL) modes allow tuning the consistency-performance trade-off.
- Modes:
  - every_write (Sync, max durability)
  - periodic (Default 100ms sync)
  - os_default (High throughput)
  - none (In-memory/Testing)



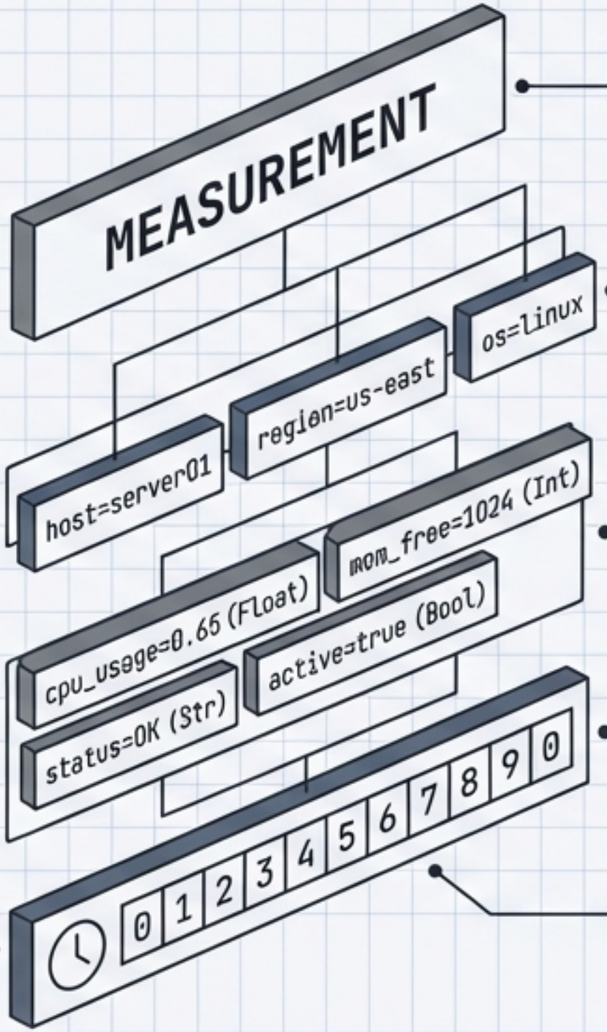


Advanced Optimization

- 10-15x compression ratios on timestamps using Delta-of-Delta.
- 8-12x compression on floats using Gorilla XOR.
- Automatic data expiration and storage tiering (Hot/Warm/Cold).



The Atomic Structure of Time Series Data



MEASUREMENT
Logical grouping / SQL Table

TAGS
Indexed Metadata
KEY=VALUE PAIRS

FIELDS
Actual Data Values
PAYLOAD BLOCKS / NOT INDEXED

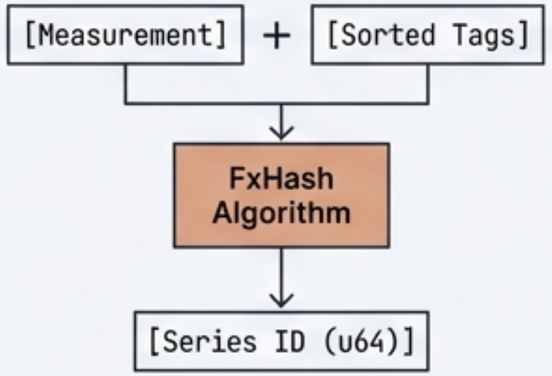
64-bit
Nanosecond Epoch

TIME BAR

TIMESTAMP
JetBrains Mono

The Series ID (u64)

A unique identifier representing a specific time series.



Constraint: Tags are always sorted alphabetically before hashing to ensure deterministic IDs.

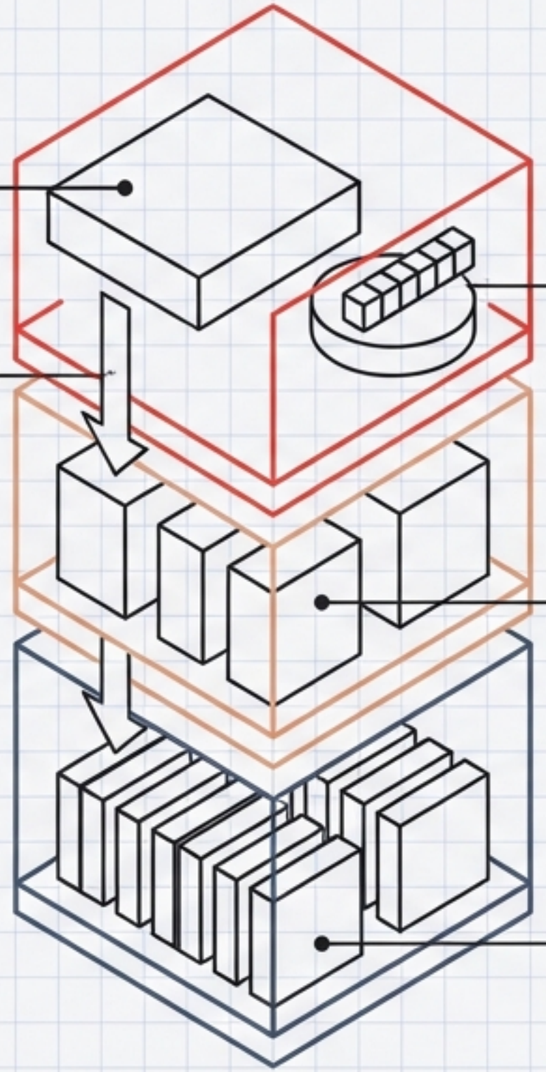
A Three-Tiered Storage Architecture

Tier 1: Active (Ingestion)
(In-Memory Buffer)
Max: 64MB or 1M points

Flush Trigger
(Size/Count/Age > 60s)

Tier 2: Hot (Recent Data)

Tier 3: Cold (Historical Data)



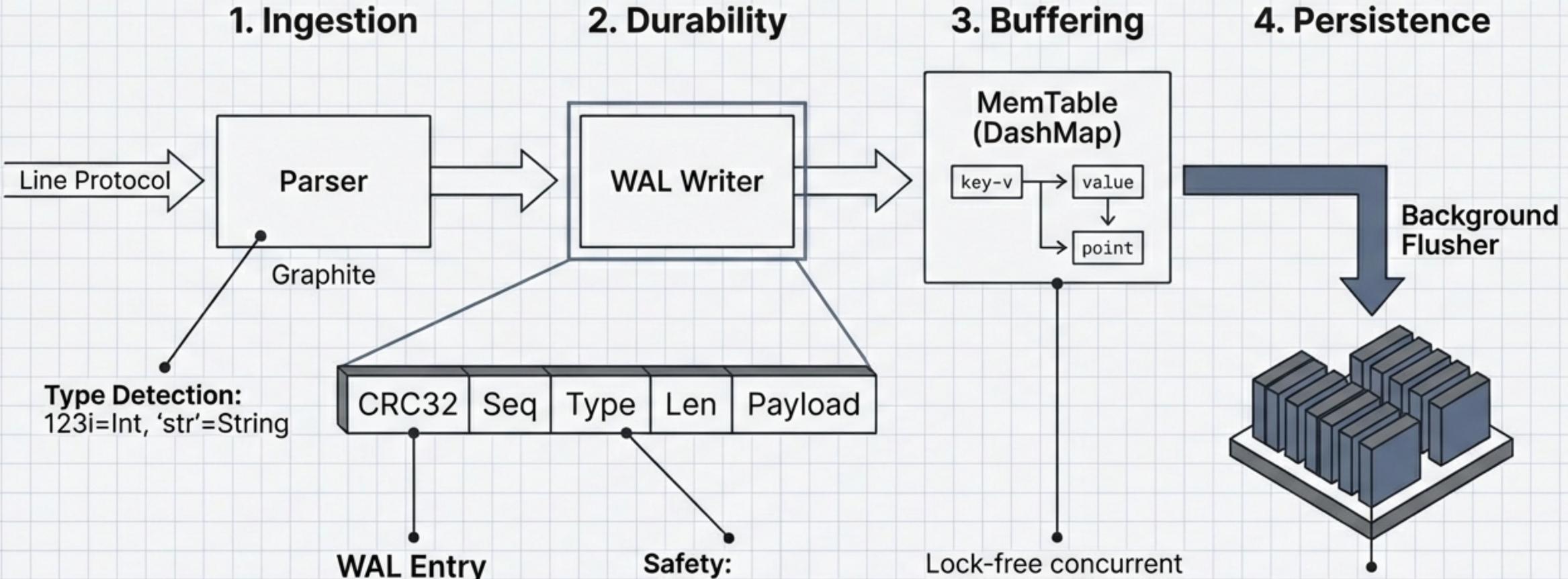
WAL
(Write-Ahead Log)
on disk

Native Columnar
Compression: LZ4
(Optimized for low latency)

High-Density
Archival Segments
Compression: Zstd
(Optimized for space efficiency)

LSM-Tree Design:
Data moves from mutable memory structures to immutable on-disk segments.

The Write Path and Durability Guarantees



Algorithmic Efficiency: The Compression Layer

Timestamps
Delta-of-Delta
10-15x Ratio

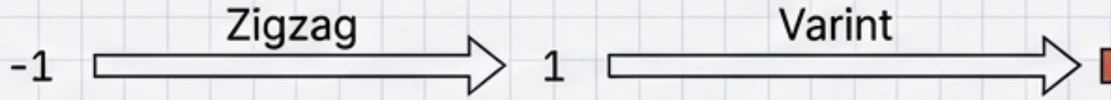
Stores difference between differences.
Regular sampling = 0 (1 bit).

Floats
Gorilla XOR
8-12x Ratio

`24.5 (01000001110001000...)` ⊗ `(01000001110001000...)` ⇒ `1100010...`

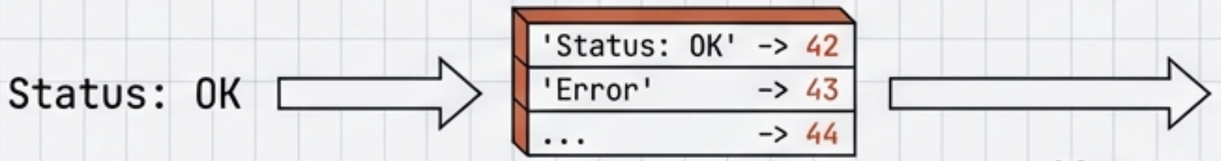
Facebook's Gorilla algorithm.
Strips leading/trailing zeros.

Integers
Zigzag + Varint
4-8x Ratio



Maps signed to unsigned to utilize variable-length encoding.

Strings
Dictionary Encoding
3-10x Ratio

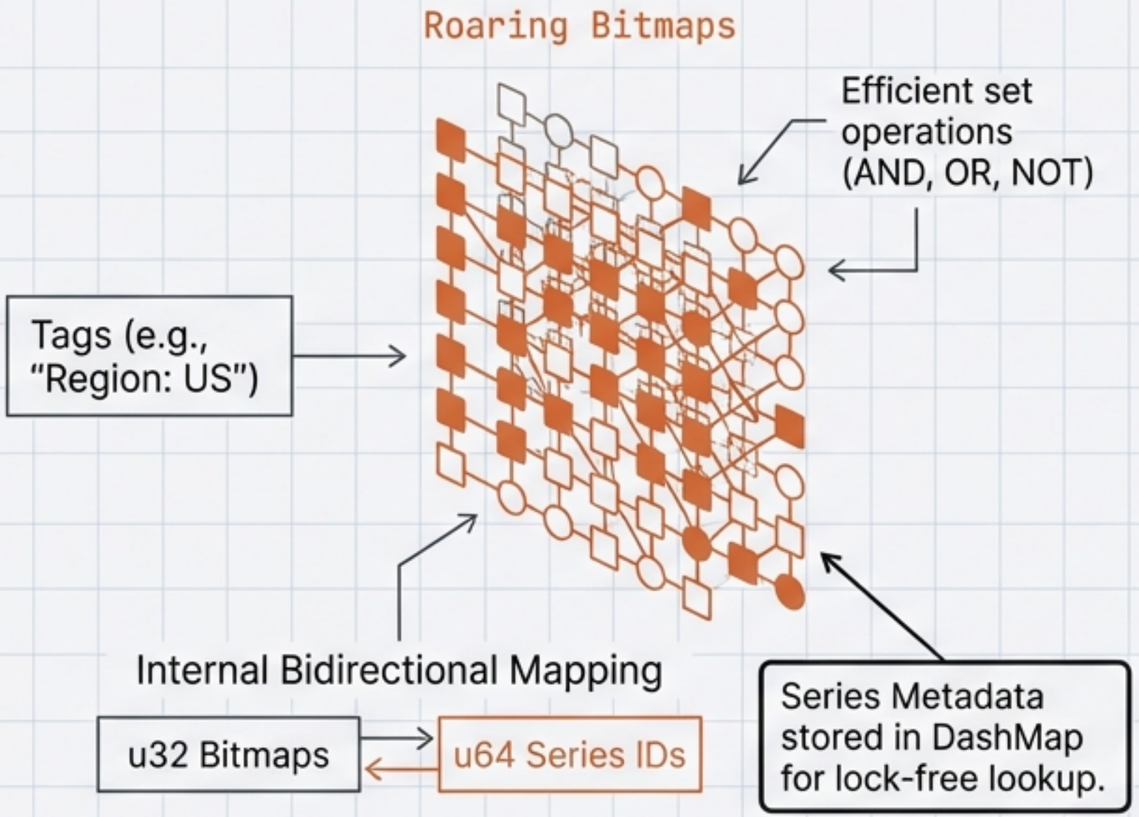


Maps repeated strings to integer IDs.

High-Performance Indexing and Pruning

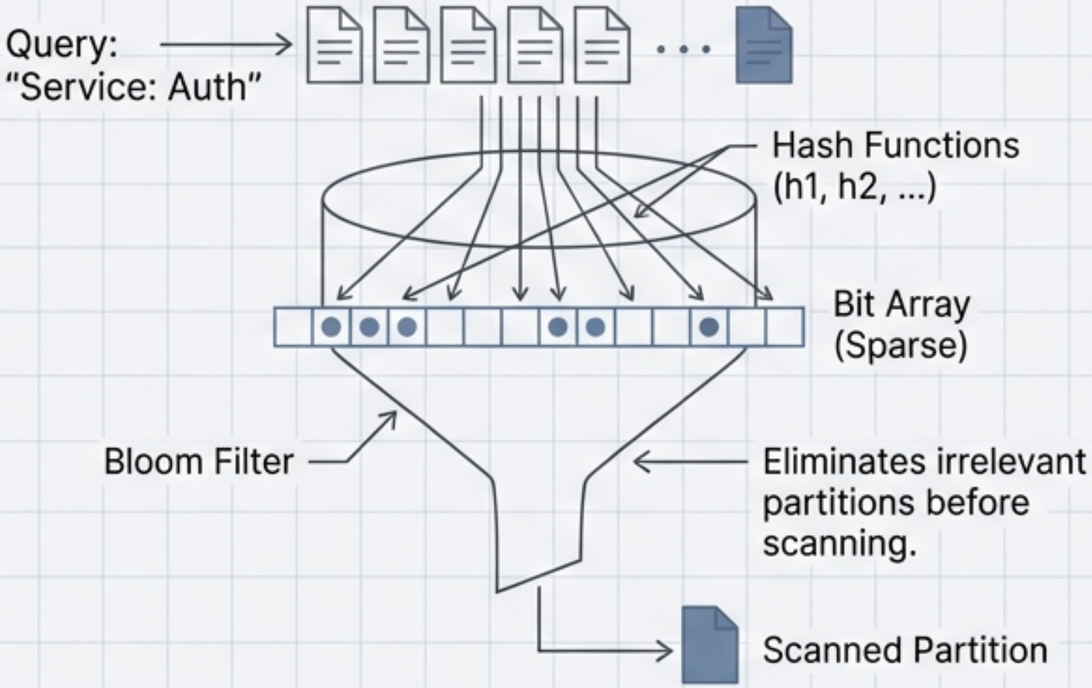
The Inverted Tag Index

Roaring Bitmaps



Partition Pruning

Bloom Filters



Specs

- **Hashing:** Double hashing h(i) = h1 + i * h2
- **Efficiency:** ~10 bits per item (1% False Positive Rate)

Query Planning and Execution Pipeline

The Planner

Partition Pruning (Time Range Overlap)

Cost Estimation Factors

Equals: 0.1x Exists: 0.7x
Regex: 0.3x NotEquals: 0.9x

Series Resolution

Resolve Tags to Series IDs via Inverted Index.

Select Path

Fetch raw points -> Filter -> Sort -> Return.

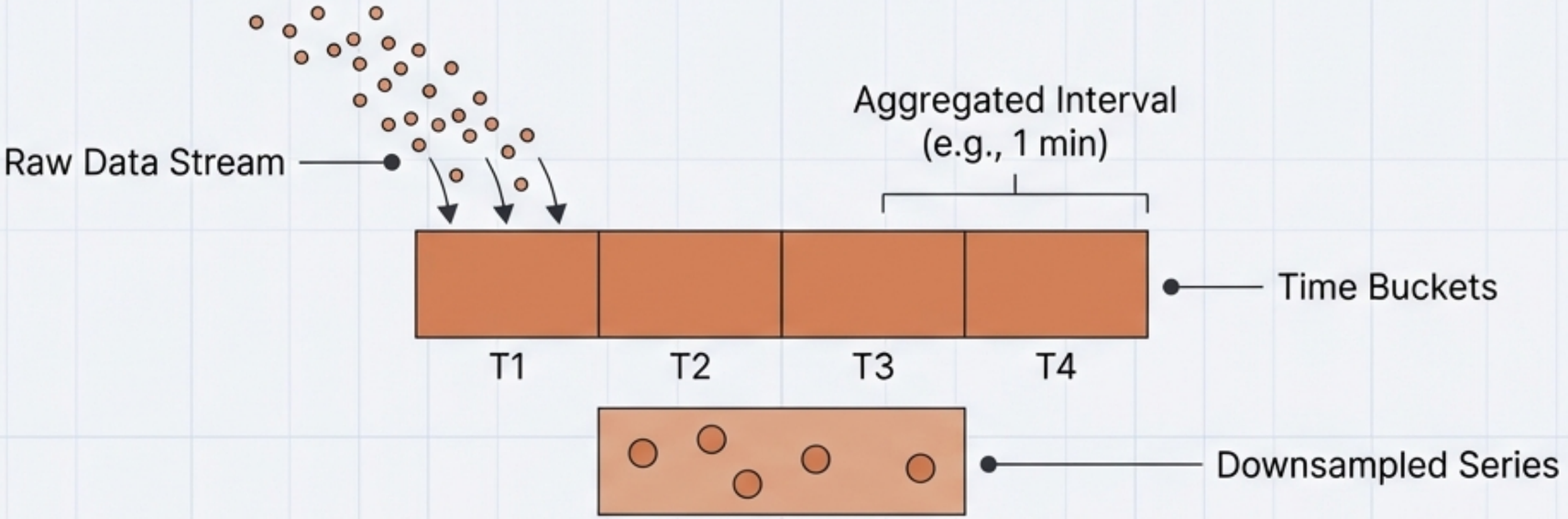
Simple Aggregate

Single-pass reduction (Sum, Mean, Count) via Aggregator.

Time-Bucketed

Map timestamps to bucket indices for downsampling.

Aggregation and Downsampling



Continuous Aggregates

Materialized view-like functionality. Decoupled from storage.
Logic:
Trigger: current_time - last_processed > interval

```
add_aggregation().group_by().retention()
```

Supported Functions

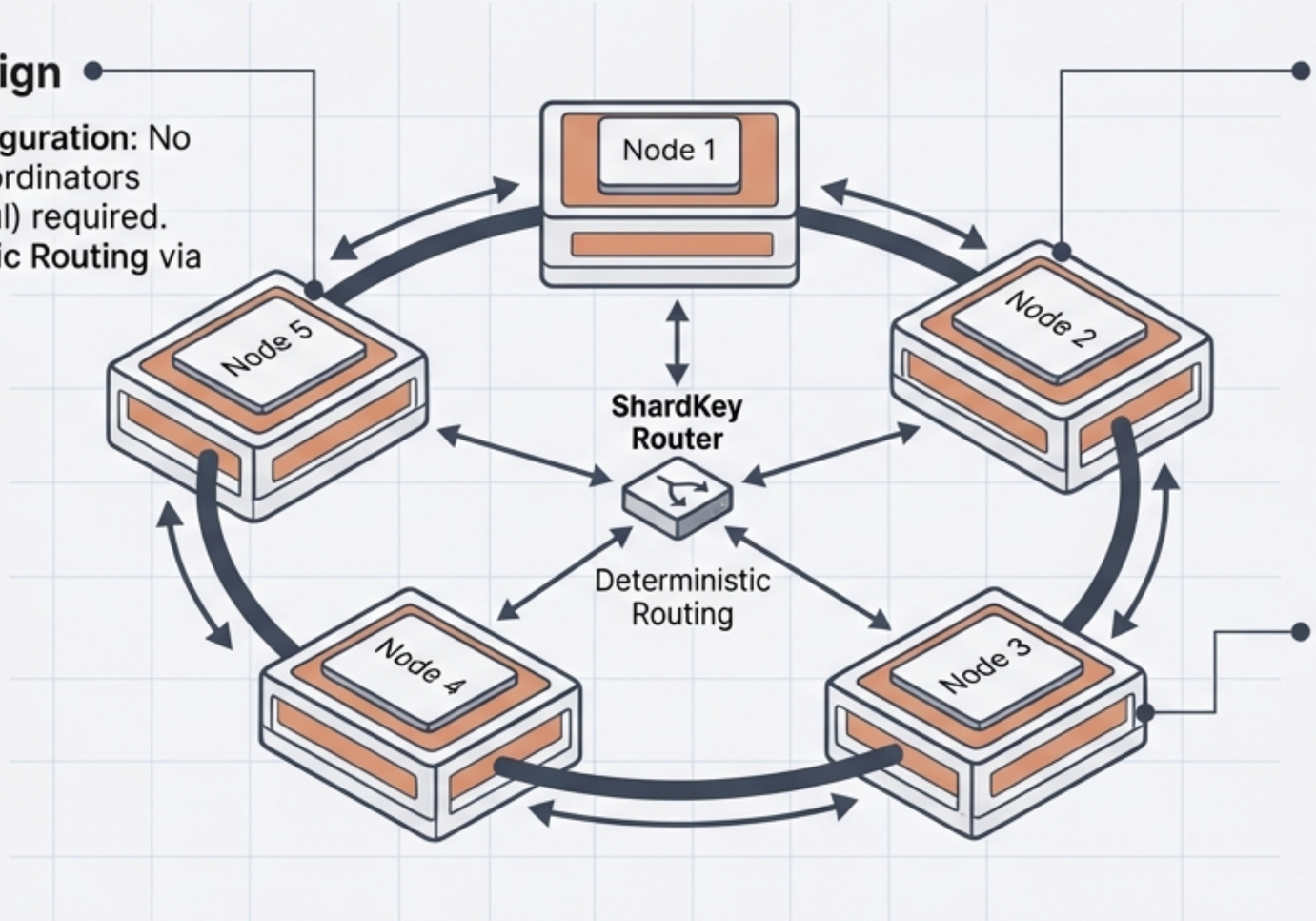
Standard: Count, Sum, Mean, Min, Max
Statistical: StdDev, Variance
Advanced: Percentile (Linear Interpolation), First/Last

Thread-safe access via parking_lot::RwLock. Computation happens in rusts-query.

Clustering and Horizontal Scale

Cluster Design

- **Static Configuration**: No external coordinators (etcd/Consul) required.
- **Deterministic Routing** via ShardKey.



Sharding Strategies

BySeries	Hash(measurement + tags) % shards. (Even distribution)
ByMeasurement	Isolation by measurement.
ByTime	Optimized for range queries.
Composite	SeriesID + TimeBucket. (Balanced)

Replication Modes

- **Sync**: Strong consistency. N acks. (High Latency)
- **Quorum**: Eventual consistency. (N/2)+1 acks.
- **Async**: Fire-and-forget. 0 acks. (Max Throughput)

The API Layer: Surface and Security

Interaction Model

Write Endpoint:
POST /write
(InfluxDB Line Protocol)

Data Ingestion

Query Endpoint:
POST /query
(JSON)

Data Request

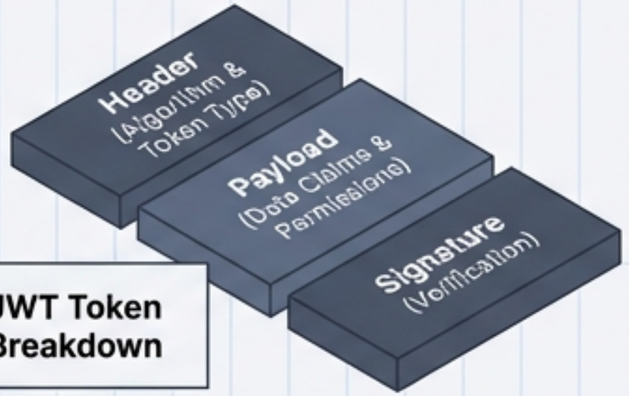
Read Operation

Health:
GET /health
(Load Balancer Ready)

Check Status

System Status Check

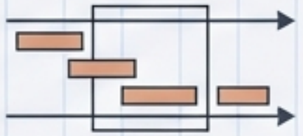
Security & Auth



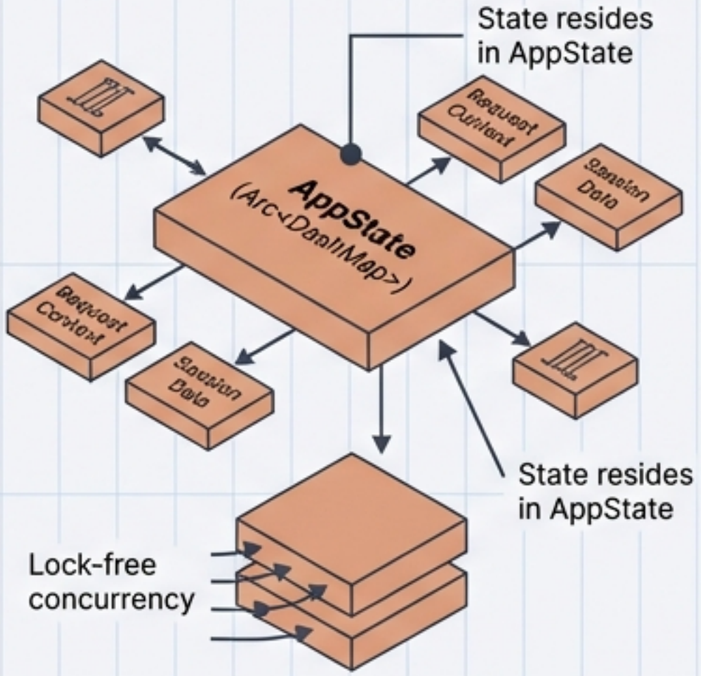
JWT Token Breakdown

Auth: JWT Tokens + API Keys
(Wildcard permissions)

Rate Limiting:
Per-client sliding window algorithm



Stateless Design

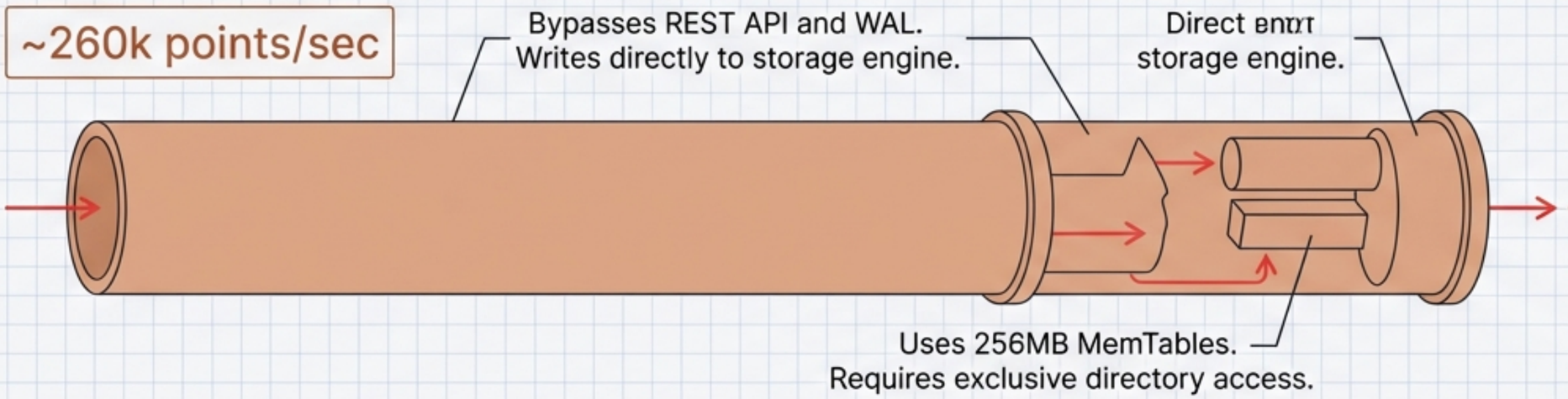


Middleware: tower-http for tracing/CORS

High-Throughput Data Import

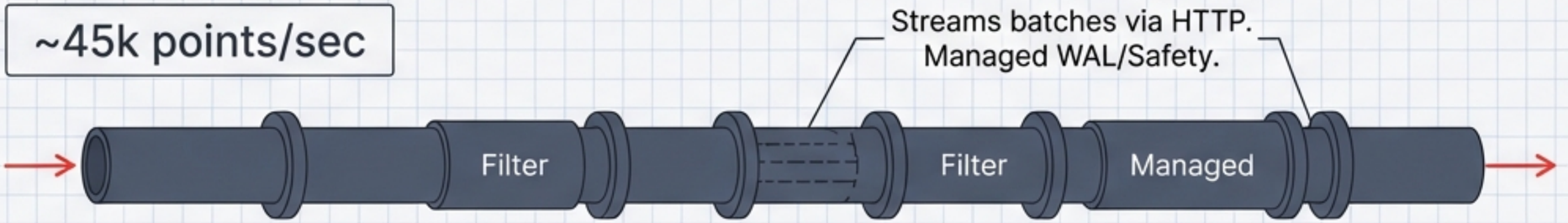
Direct Mode (The Performance King)

~260k points/sec



REST Mode (The Safe Option)

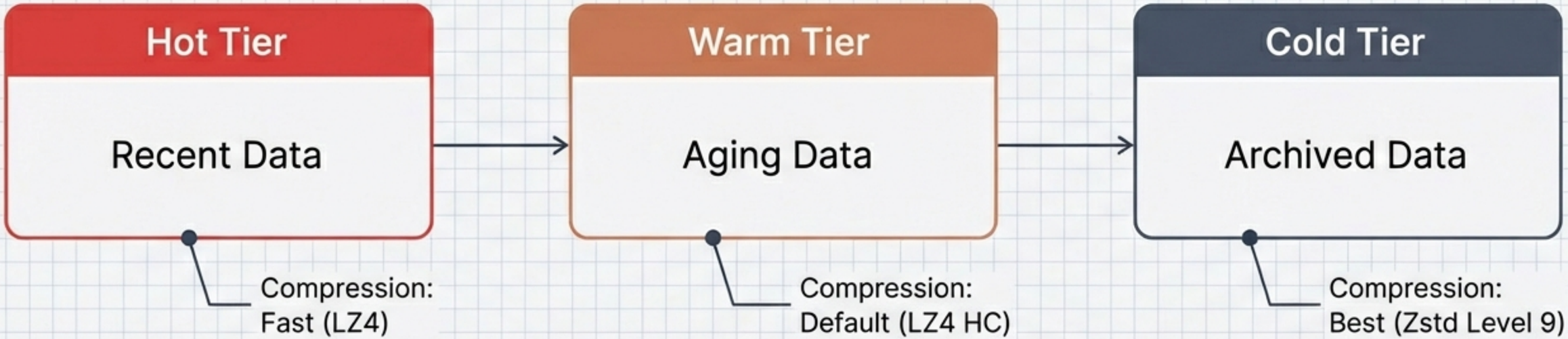
~45k points/sec



Deduplication

- Direct Mode: Dedups by Field only.
- REST Mode: Dedups by Field + Tag.

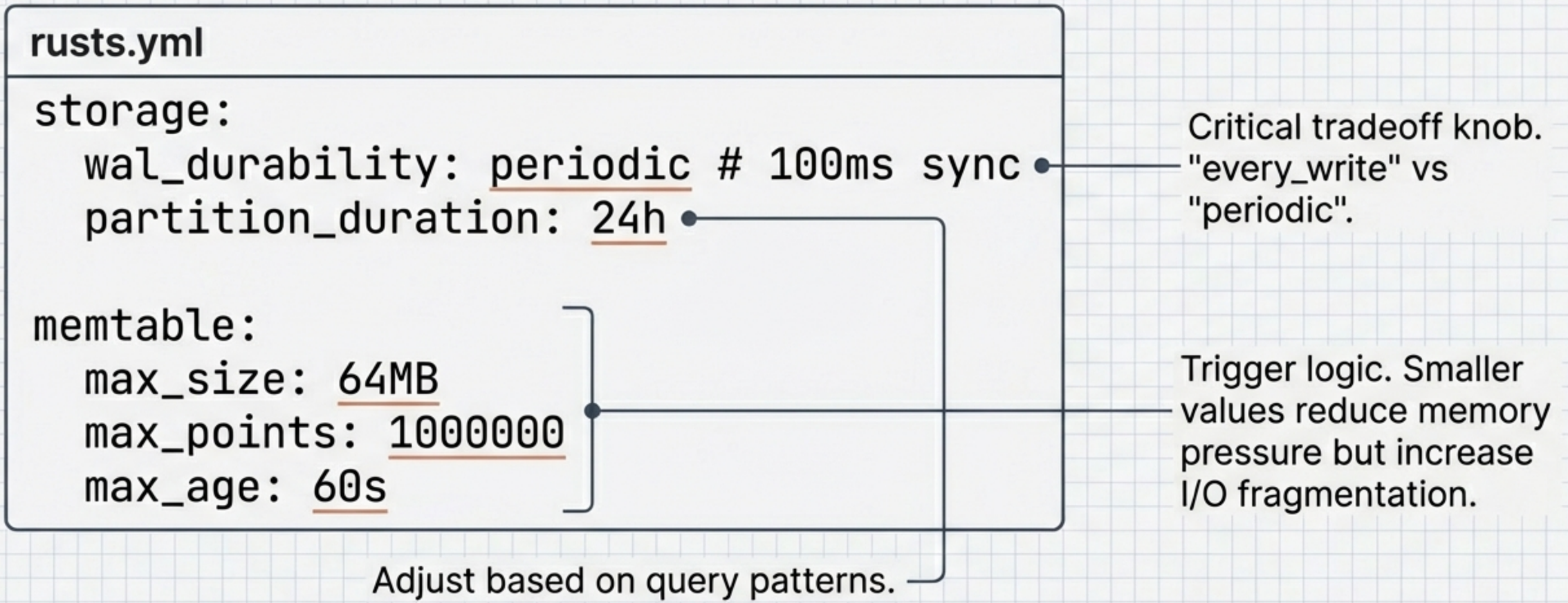
Data Lifecycle Management



Retention Policies

- Resolution Priority: Exact Match > Specific Pattern > Default.
- Pattern Matching: “cpu” (Exact), “cpu*” (Prefix), “*” (Wildcard).

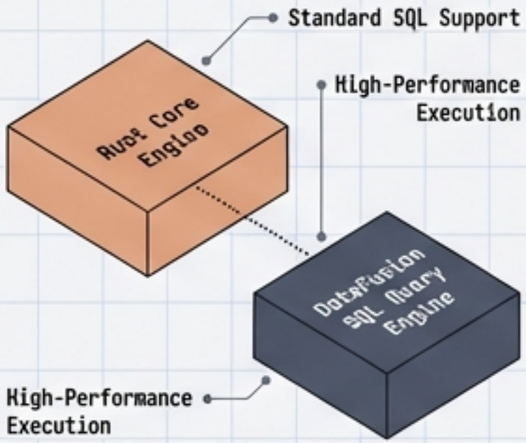
Configuration and Tuning



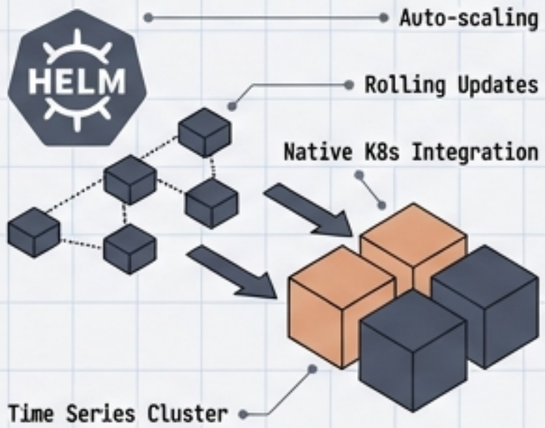
Default Network: Binds to 0.0.0.0:8086. Max body 10MB.

Roadmap and Future Horizons

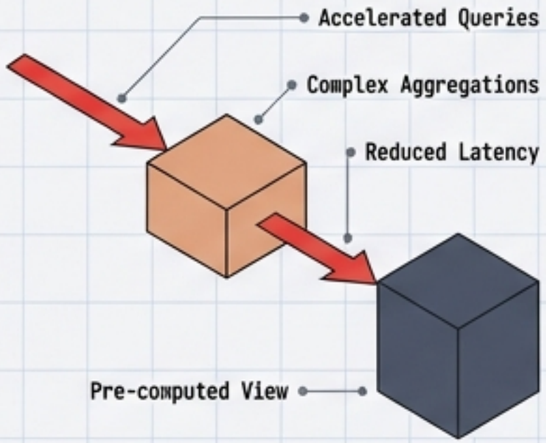
Upcoming Features (Wireframe Visuals)



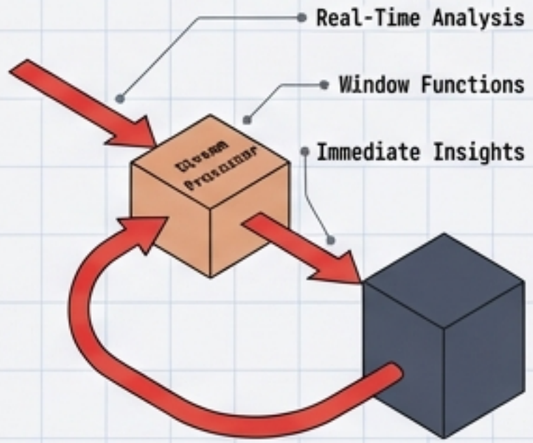
SQL Interface:
DataFusion Integration



Kubernetes Operator:
Native Lifecycle Management



Materialized Views:
Pre-computed complex queries



Continuous Queries:
Real-time stream processing

Getting Started

Prerequisites:
Rust 1.75+ and Cargo

```
> cargo run --release
```

Precision engineered for the future of time series.