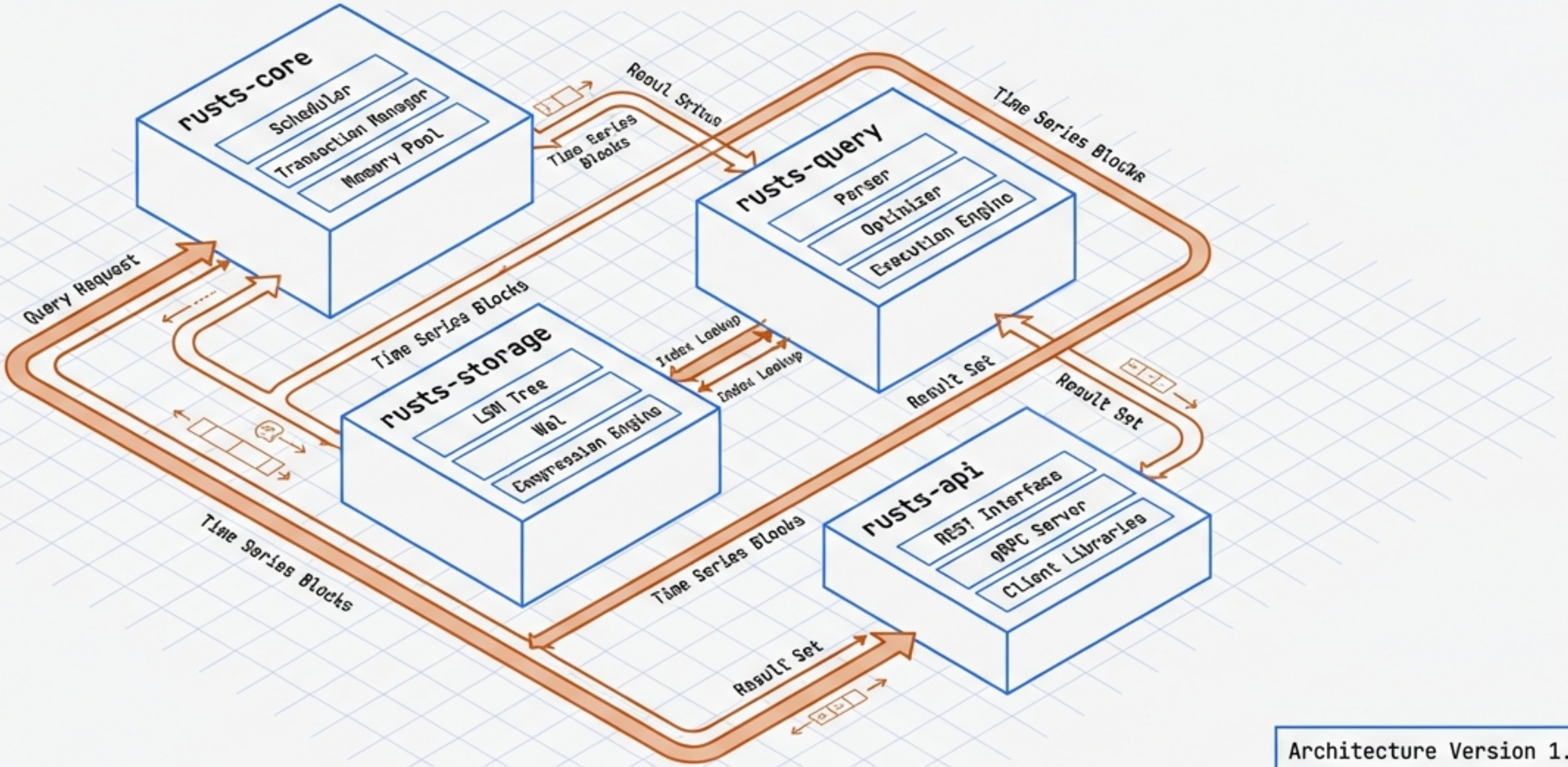
RusTs: Technical Deep Dive

Architecture, Internals, and Performance of a Rust-Based Time Series Database.



Architecture Version 1.0
Rust 1.75+

Performance Without the Garbage Collector

Performance First

Columnar storage with specialized compression (`Gorilla/DoD`) and zero-copy read paths.

Safety & Durability

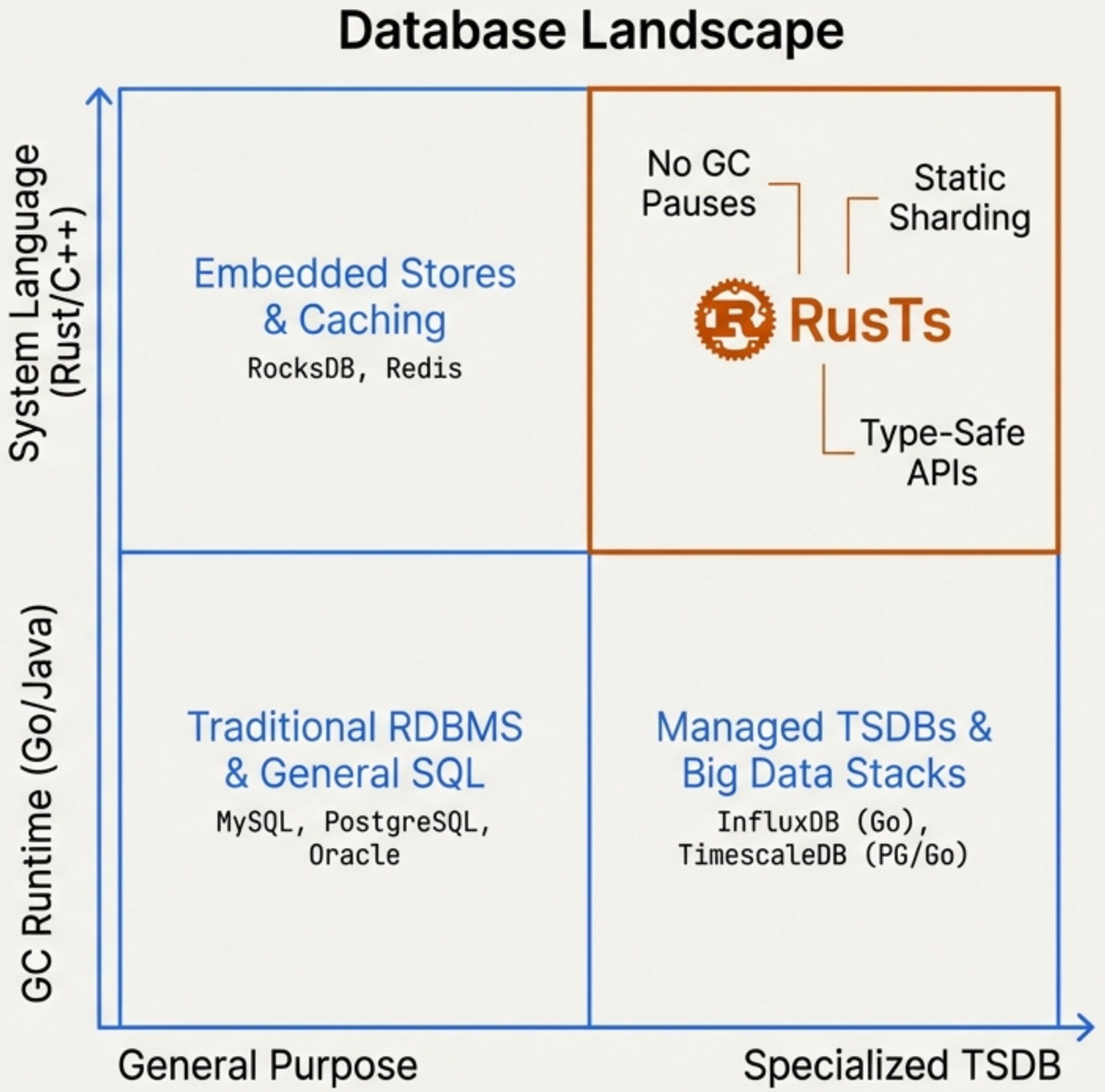
Memory-safe concurrency using `parking_lot` primitives and flexible WAL modes.

Modularity

Decoupled crate ecosystem allowing independent evolution.

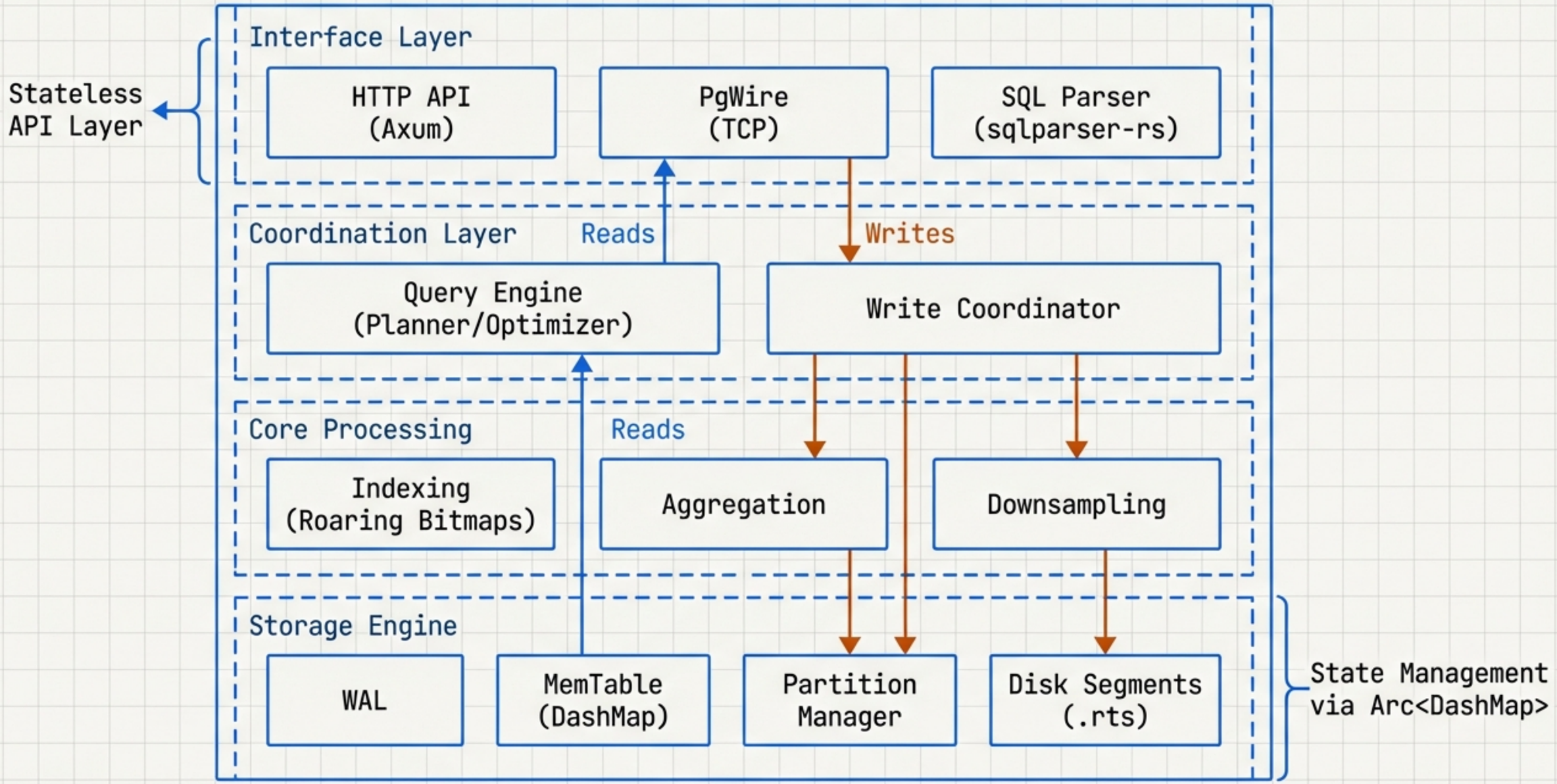
Interoperability

InfluxDB Line Protocol + PostgreSQL Wire Protocol.



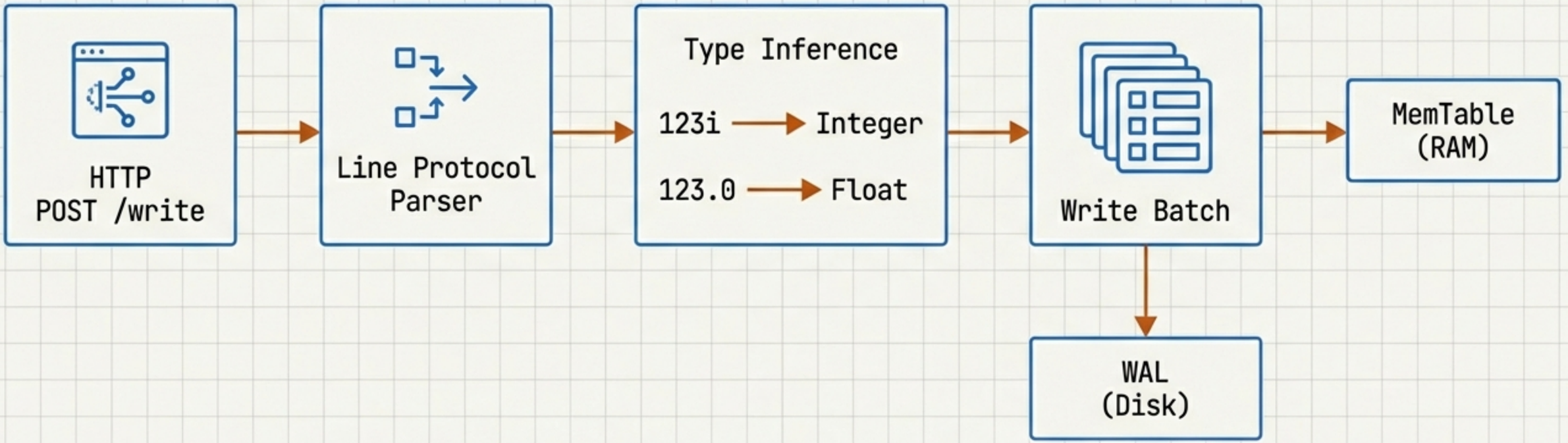
System Architecture & Module Map

RusTs



The Write Path: Ingest & Parsing

RusTs



Protocol

InfluxDB Line Protocol. Stateless parsing.

Error Handling

Function `parse_lines_ok` isolates invalid lines to prevent total batch failure.

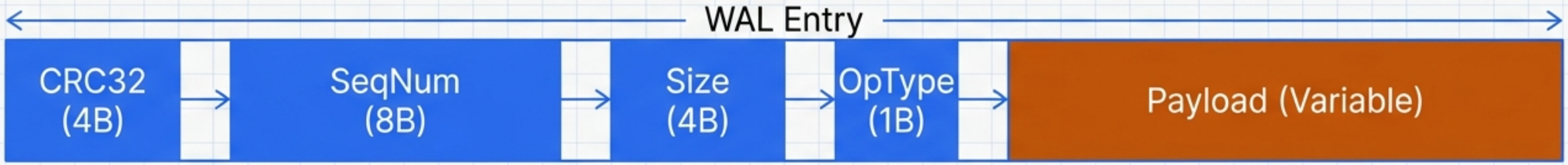
Concurrency

Shared state managed via `Arc<DashMap>`. No global lock.

Durability: The Write-Ahead Log (WAL)

RustTs

Byte-Level Layout



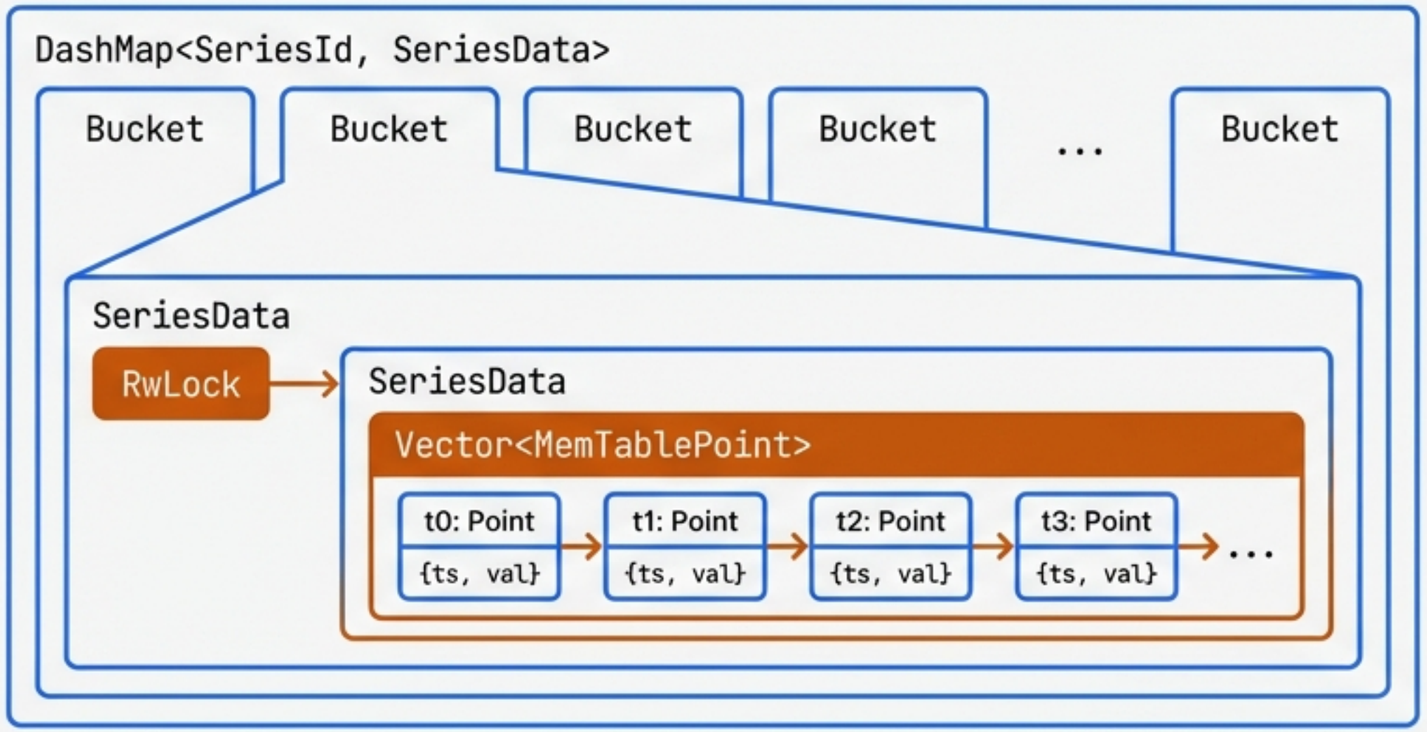
Configuration Modes

- `every_write`: Sync after each write (Max Safety).
- `periodic`: Background sync every 100ms (Default).
- `os_default`: Relies on OS page cache (Max Throughput).

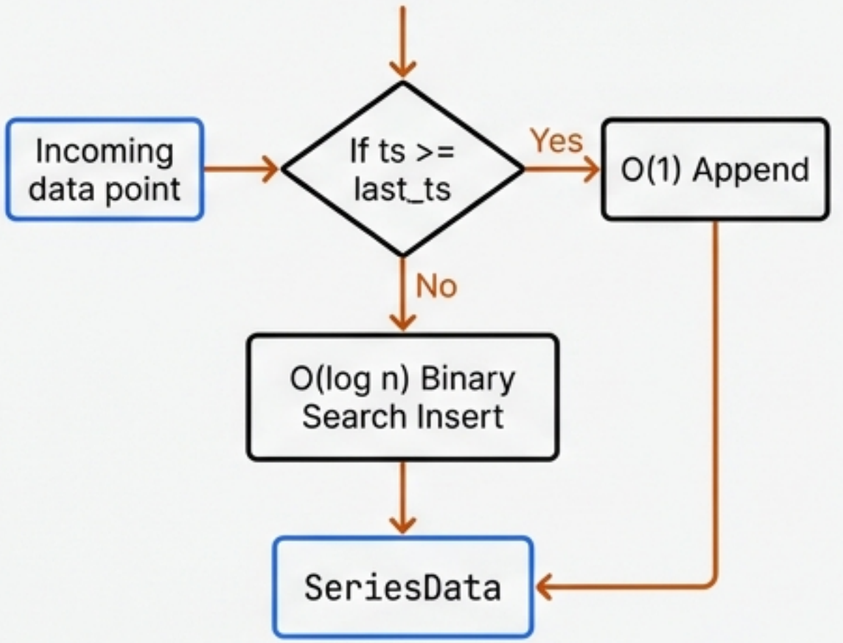
CDC Capability: Set `wal_retention_secs: null` to enable indefinite tailing for Change Data Capture.

RustTs

The Hot Layer: MemTable Internals



Insertion Logic



Flush Triggers (Mutable -> Immutable)

- Size > 64MB
- Count > 1,000,000 Points
- Age > 60 Seconds

The Flush: Segments & File Format

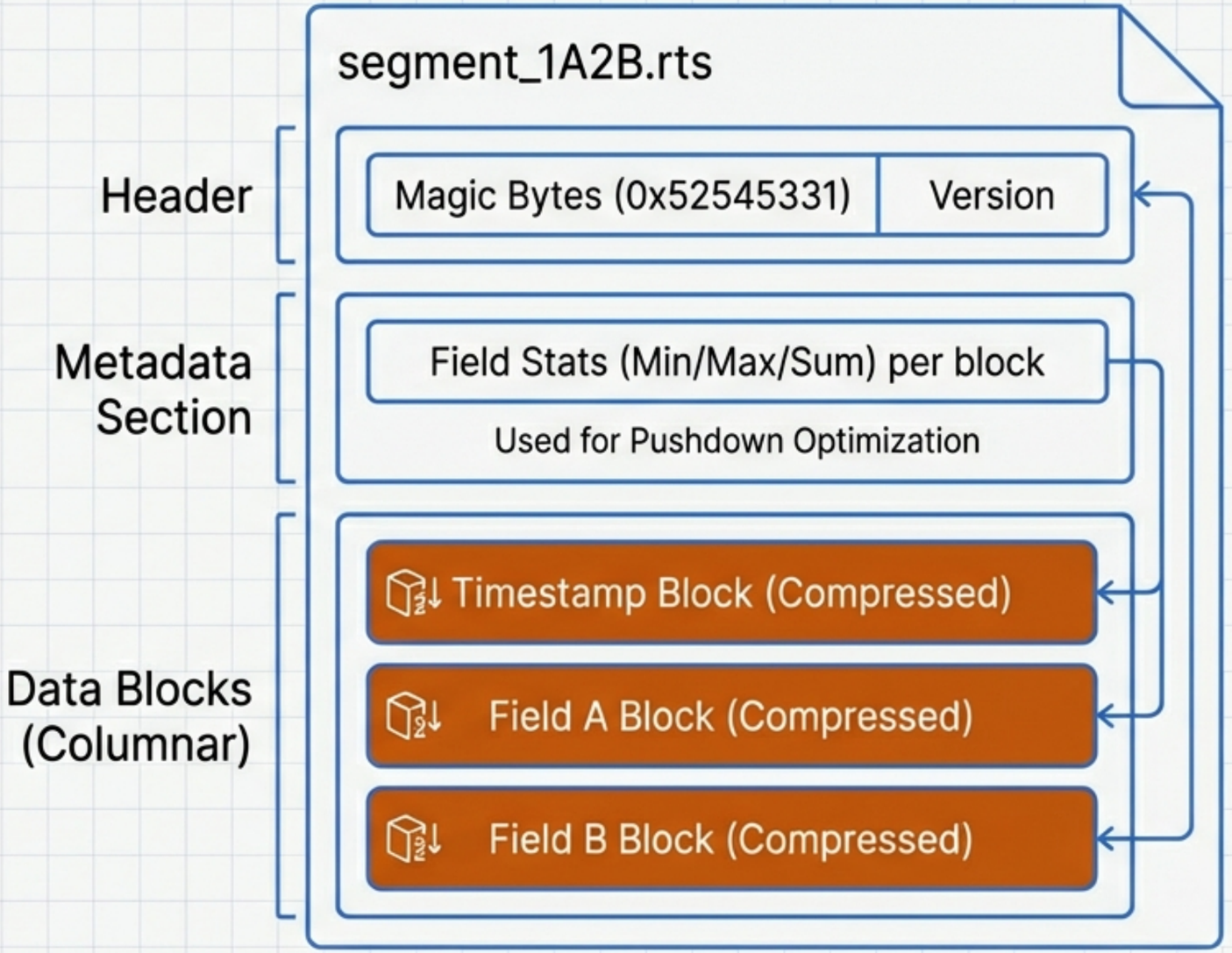
RustTs

LSM-Tree Organization

Immutable segments created from flushed MemTables.

Partitions: Data physically grouped by time windows (default 24h).

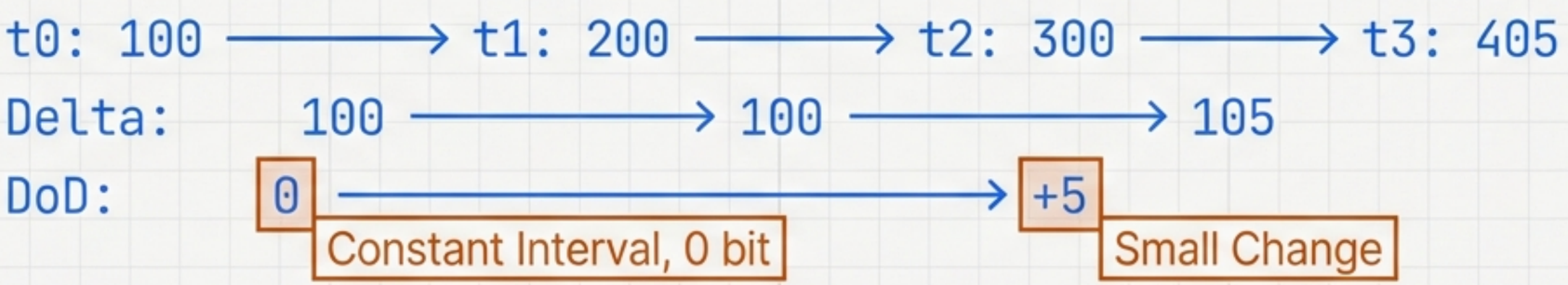
Columnar Layout: Enables reading specific fields without scanning full rows.



RustTs

Compression: The 10-15x Advantage

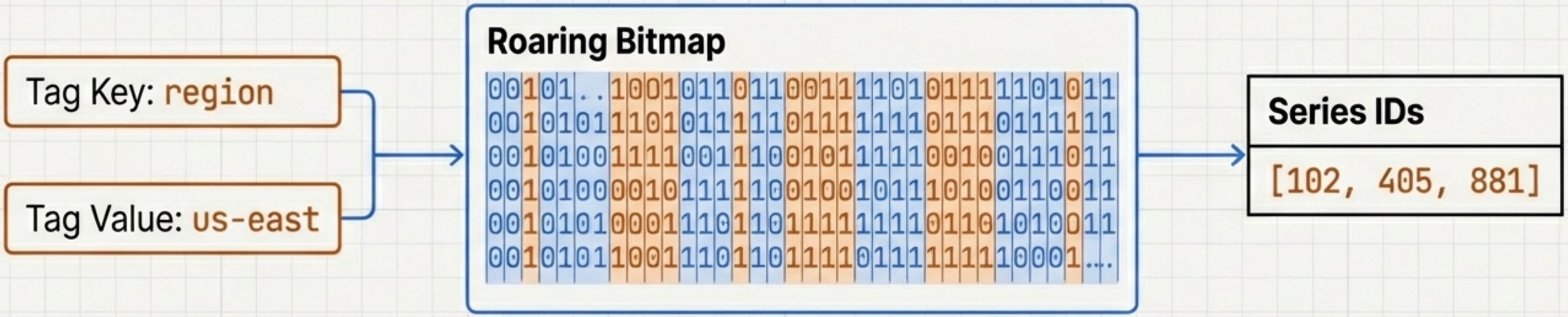
Data Type	Algorithm	Mechanism	Ratio
Timestamps	**Delta-of-Delta**	Stores difference of differences. `0` bit for constant intervals.	**10-15x**
Floats	**Gorilla XOR**	XORs with previous value; stores non-zero bits.	**8-12x**
Integers	**Zigzag + Varint**	Maps signed to unsigned; variable byte encoding.	**4-8x**
Strings	**Dictionary**	Maps low-cardinality strings to u32 IDs.	**3-10x**



Tiering: Hot segments use LZ4 (Speed), Cold use Zstd (Size).

RustTs

Indexing: Fiding the Needle

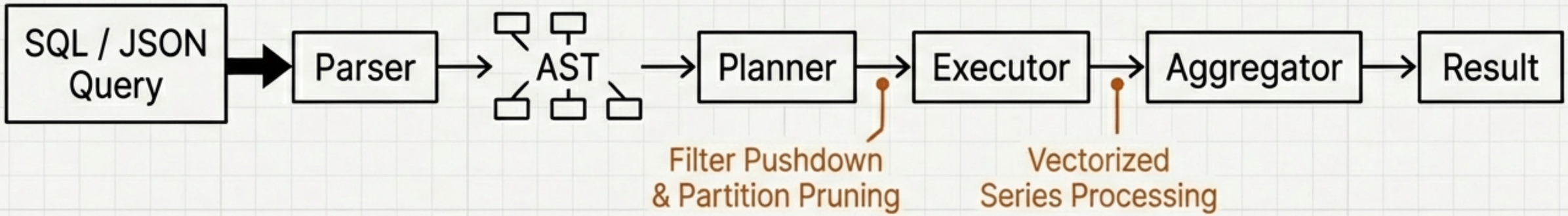


Query Optimization

- **Set Operations:** Bitwise AND/OR/NOT for queries like `region='us-east' AND host!='server01''`.
- **Cardinality Sorting:** Planner orders filters by lowest cardinality first.
- **Bloom Filters:** Used for Partition Pruning to skip files.

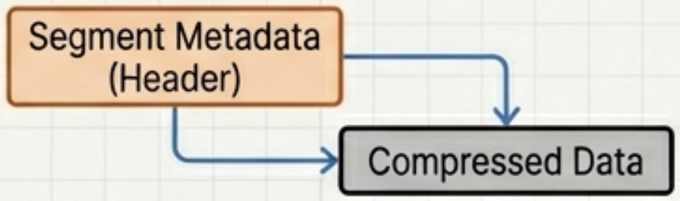
RustTs

The Query Engine & Optimization



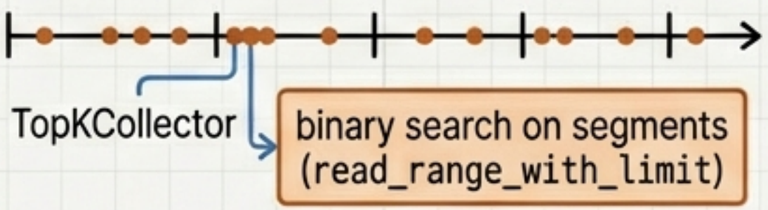
Pushdown Strategies

Metadata Pushdown



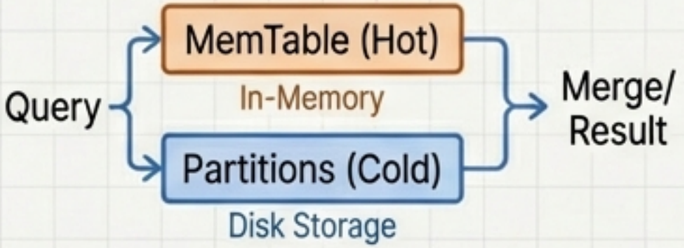
Executes `COUNT`, `MIN`, `MAX` on Segment Metadata (Header) without decompression.

LIMIT Optimization



Uses `TopKCollector` and binary search on segments (`read_range_with_limit`) to avoid full scans.

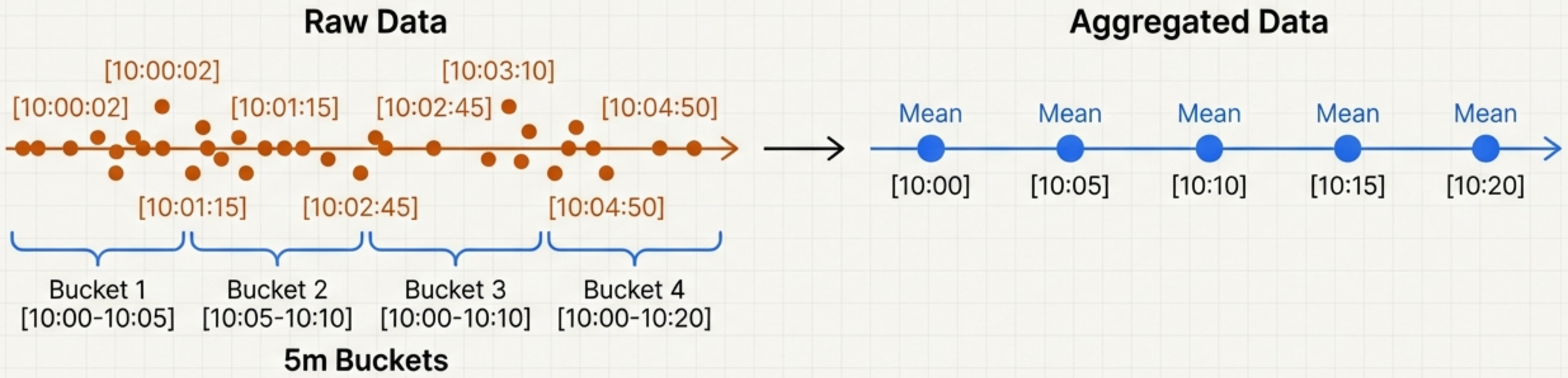
Routing



Queries MemTable (Hot) and Partitions (Cold) in parallel.

Engineering Blueprint / Technical Whitepaper

Aggregation & Downsampling



Aggregation Types

- **On-the-fly:** `TimeBucketAggregator` handles `GROUP BY time(...)` queries.
- **Continuous Aggregates:** Materialized views defined in `rusts-aggregation` crate.
- **Functions:** Sum, Mean, Last, StdDev, Percentile (linear interpolation).
- **Downsampling:** Configurable retention (e.g., Raw: 24h, 1m Avg: 7d).

RustTs

Interface: SQL & PostgreSQL Wire Protocol

PostgreSQL Wire Protocol (TCP)

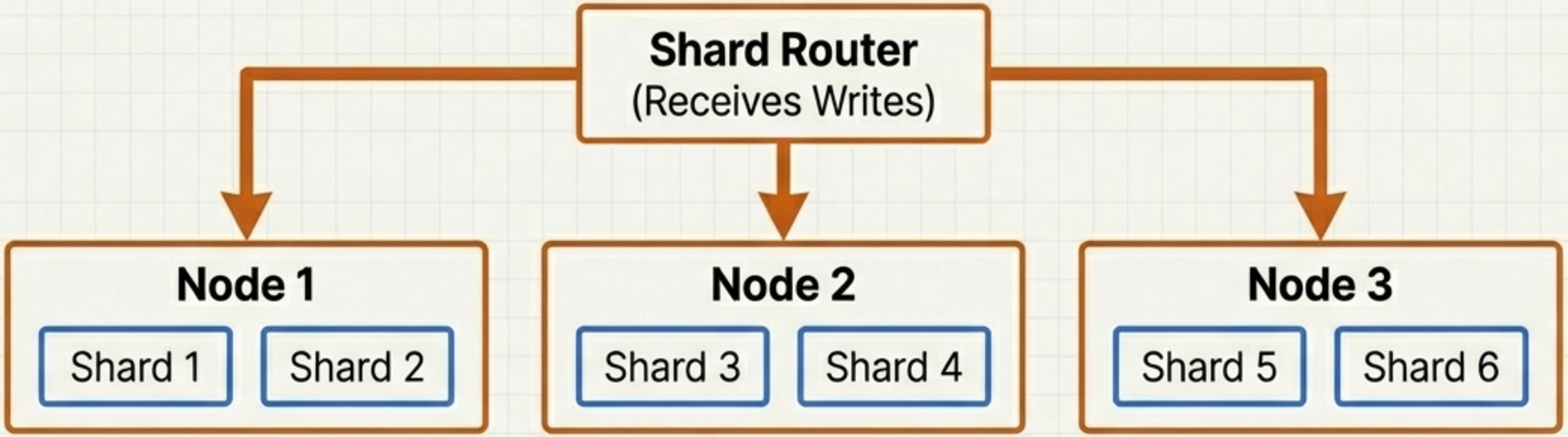
```
$ psql -h localhost -p 5432 -d rusts

rusts=> SELECT region, AVG(cpu) FROM metrics
        WHERE time > now() - interval '1h'
        GROUP BY region;
 region  | avg
---------+------
 us-east | 45.2
 eu-west | 32.8
(2 rows)
```

HTTP API

```
POST /query
{
  "query": "SELECT MEAN(cpu) FROM metrics..."
}
```

SQL Engine: `sqlparser-rs` → AST → RusTs Query Model.
Performance: PgWire is 5-15x faster than REST (Binary format, persistent connections).

Clustering & Distributed Architecture



Sharding Strategies

- **BySeries**: Hash of (Measurement + Tags). Even distribution.
- **ByMeasurement**: Isolates measurements to specific nodes.
- **ByTime**: Optimizes for time-range queries.

Replication Modes

- **Sync** (Strong Consistency)
- **Quorum** (Eventual Consistency)
- **Async** (High Throughput)

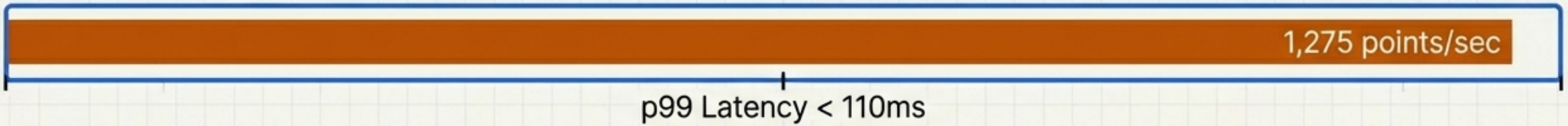
⚙ **Coordination** via static `cluster.toml` (No ZooKeeper required).

RustTs

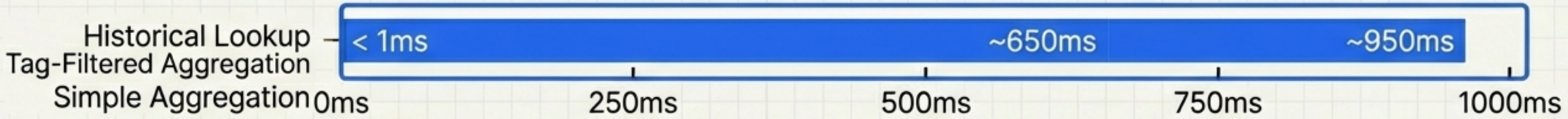
Performance Benchmarks

Dataset: NYC Taxi Trip Records (1.25M Points)

Write Throughput



Query Latency



Import Speed

RustTs

Roadmap & Getting Started

Current Status vs Future

Current

- SQL & PgWire
- Tiered Storage (Hot/Warm/Cold)
- Flexible Sharding

Future

- Kubernetes Operator
- DataFusion Integration
- Parameterized SQL

Quick Start

terminal

```
$ git clone https://github.com/rusts/rusts.git
$ cd rusts
$ cargo build --release
$ ./target/release/rusts-server --config rusts.yml
```

Docker Ready: `docker pull rusts:latest`