Composing Data Systems At Datadog

Alex Bianchi
Software Engineer, Cross-Product Queries, Datadog

Wendell Smith
Staff Engineer, Cross-Product Queries, Datadog

DATADOG

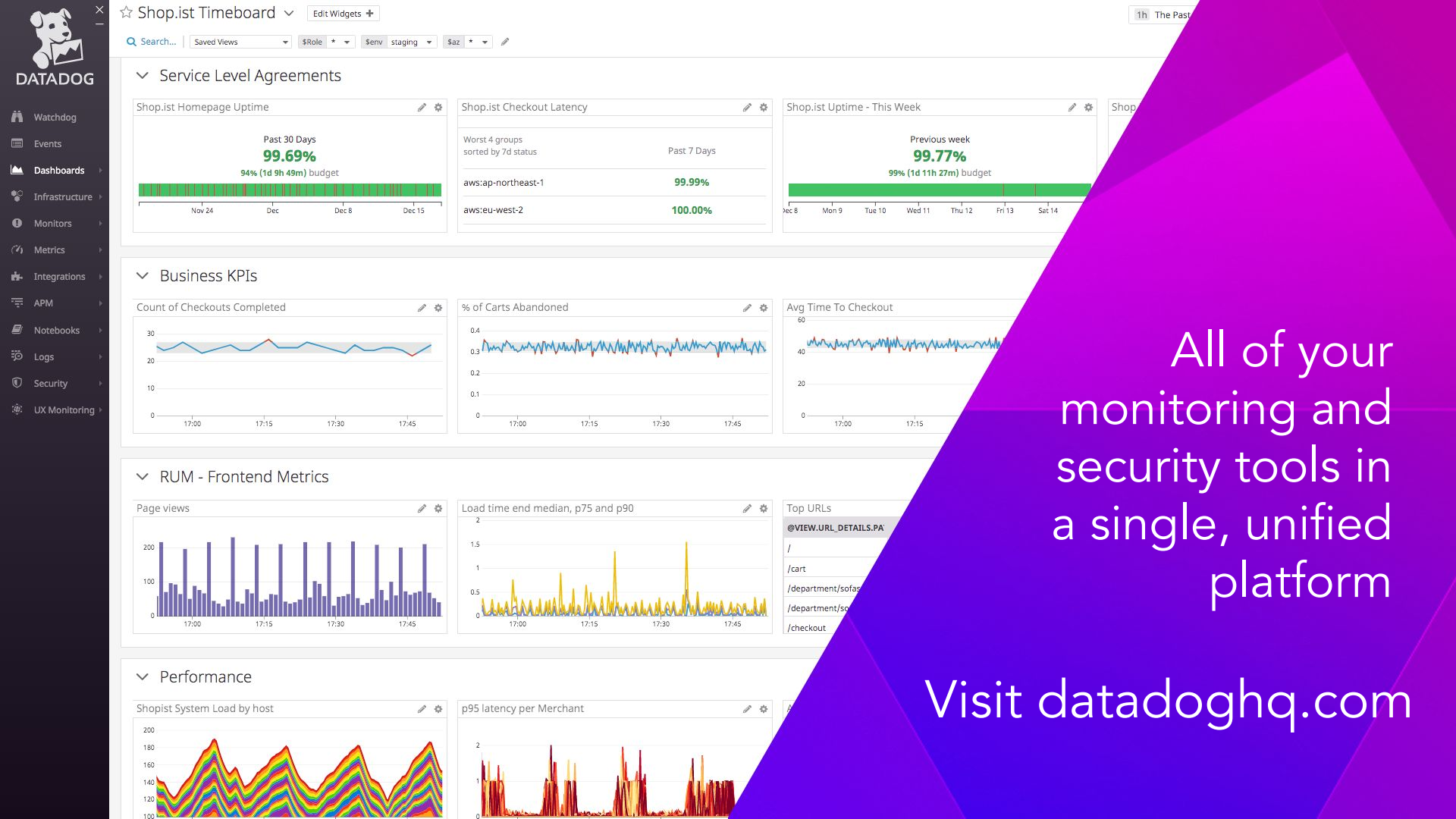
Agenda

01 What is Datadog?

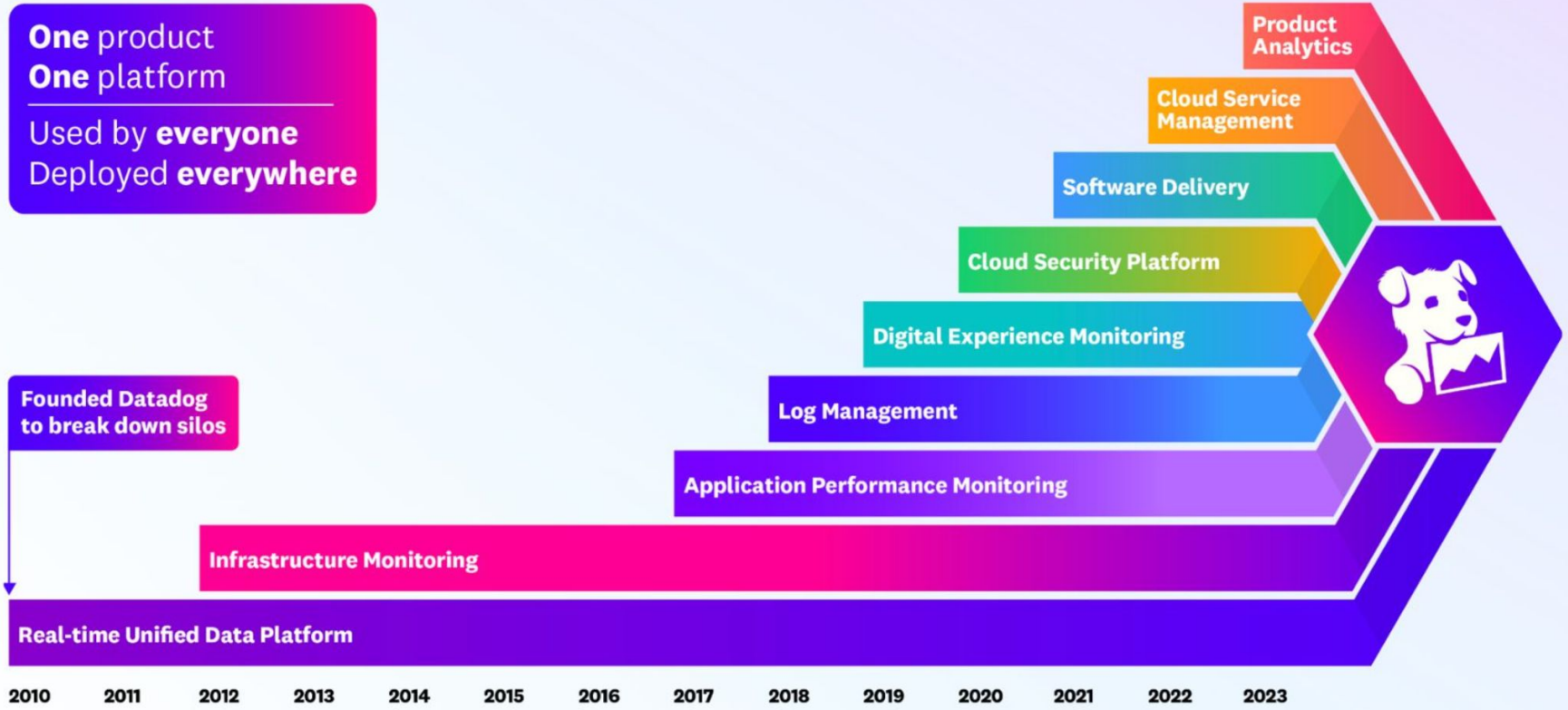
02 Why are we Interested in Datafusion?

03 What is Datafusion?

04 How are we using Datafusion?


DATADOG
Watchdog
Events
Dashboards
Infrastructure
Monitors
Metrics
Integrations
APM
Notebooks
Logs
Security
UX Monitoring
Shop.ist Timeboard
Edit Widgets
Search...
Saved Views
$Role *
$env staging
$az *
1h
Service Level Agreements
Shop.ist Homepage Uptime
Past 30 Days
99.69%
94% (1d 9h 49m) budget
Nov 24
Dec
Dec 8
Dec 15
Shop.ist Checkout Latency
Worst 4 groups
sorted by 7d status
Past 7 Days
aws:ap-northeast-1
99.99%
aws:eu-west-2
100.00%
Shop.ist Uptime - This Week
Previous week
99.77%
99% (1d 11h 27m) budget
Mon 9
Tue 10
Wed 11
Thu 12
Fri 13
Sat 14
Business KPIs
Count of Checkouts Completed
% of Carts Abandoned
Avg Time To Checkout
RUM - Frontend Metrics
Page views
Load time end median, p75 and p90
Top URLs
@VIEW.URL_DETAILS.PA
/
/cart
/checkout
Performance
Shopist System Load by host
p95 latency per Merchant
All of your monitoring and security tools in a single, unified platform
Visit datadoghq.com

Datadog Through The Years



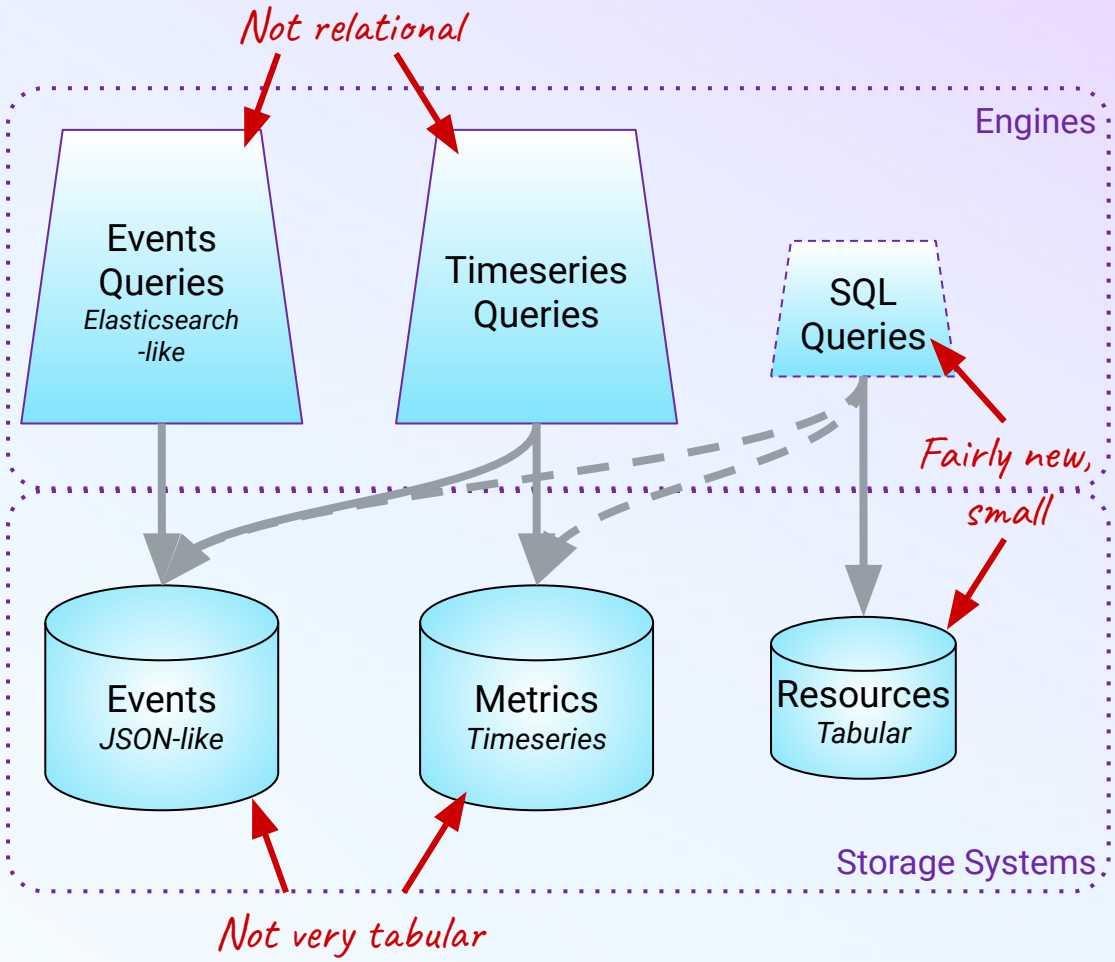
Our Current Tech Stack

Challenge

We have multiple non-relational engines from years of growth, acquisitions, and optimizations.

Goal

Enable more complex queries from multiple data sources

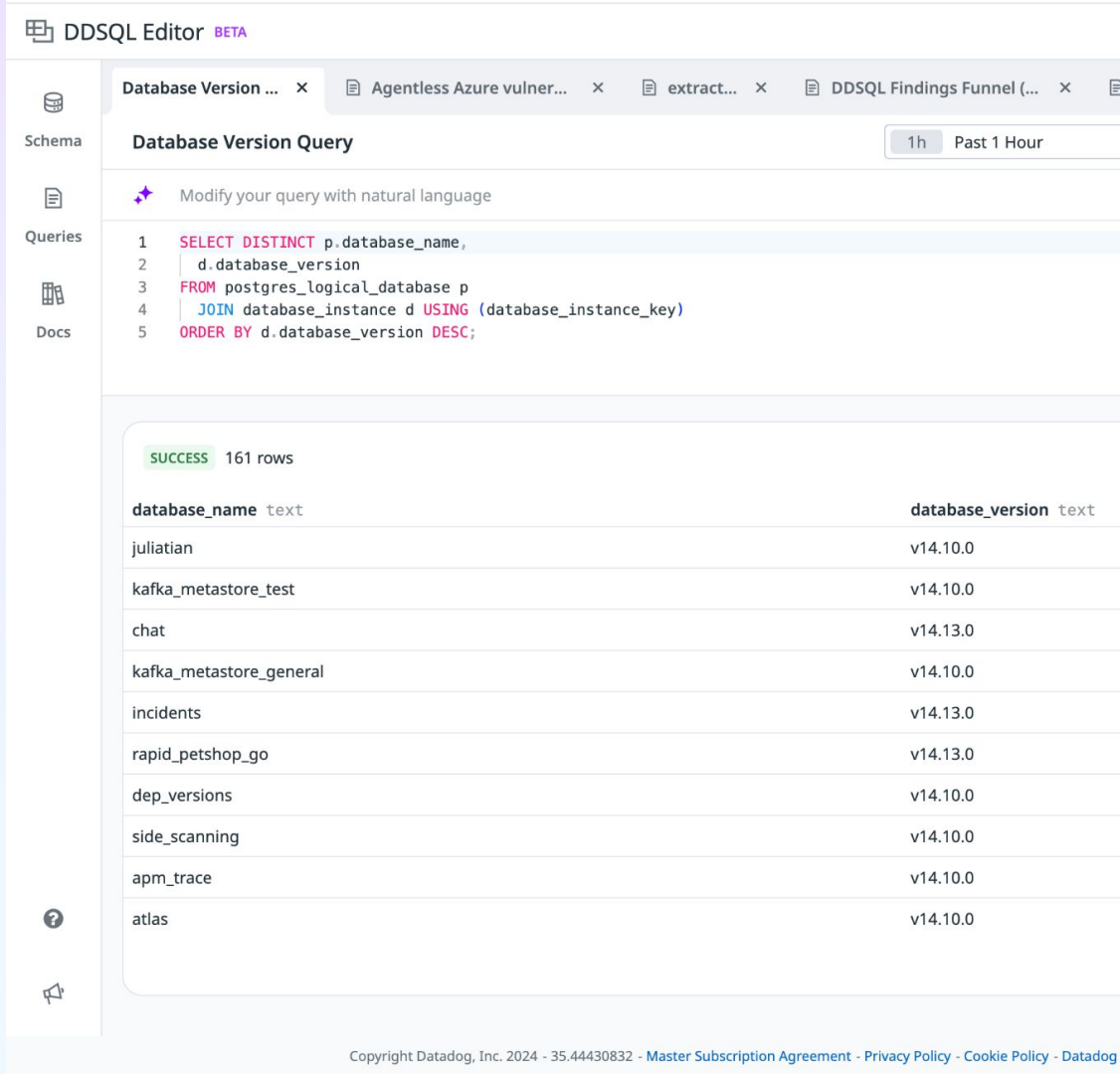


DDSQL Editor

Single access point

SQL style syntax

Joins across data stores

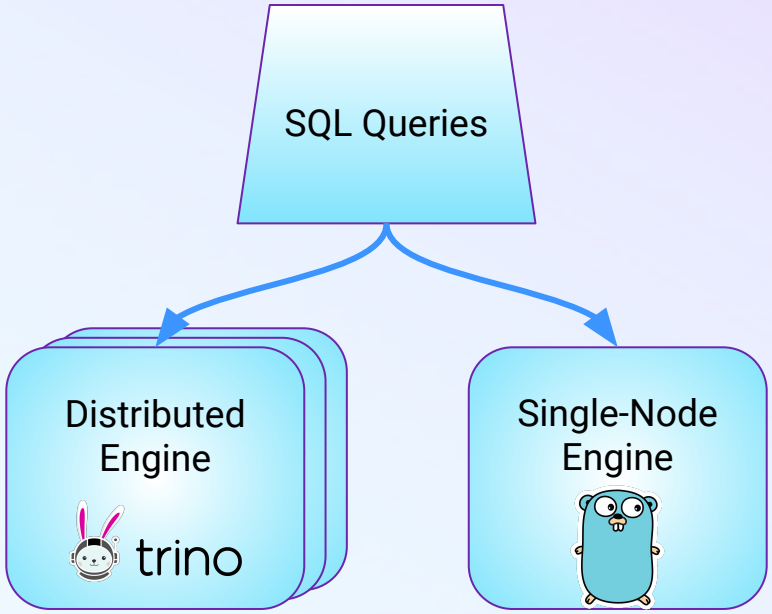


Cross Product Queries

Today, we use the distributed SQL engine Apache Trino; a tool designed to efficiently query vast amounts of data.

Trino was not designed for high throughput, real-time queries on small datasets

For these queries, we have our own execution engine. But building out complete functionality has had ups and downs.



98%

Of our cross product queries could be executed in single node.

DATADOG

We're looking to Datafusion for high throughput, low-cost queries.

What is

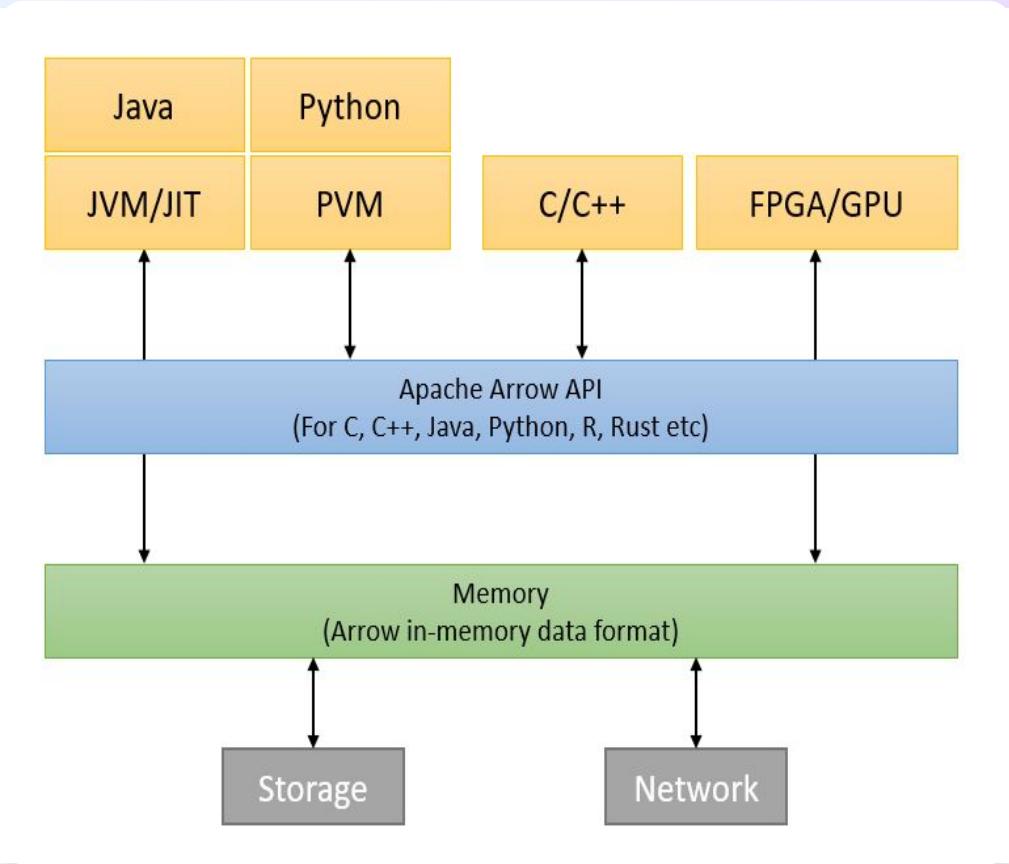
> DataFusion is an **extensible query engine** written in Rust that uses **Apache Arrow** as its in-memory format. DataFusion's target users are developers **building fast and feature rich** database and analytic systems, **customized** to particular workloads.

datafusion.apache.org

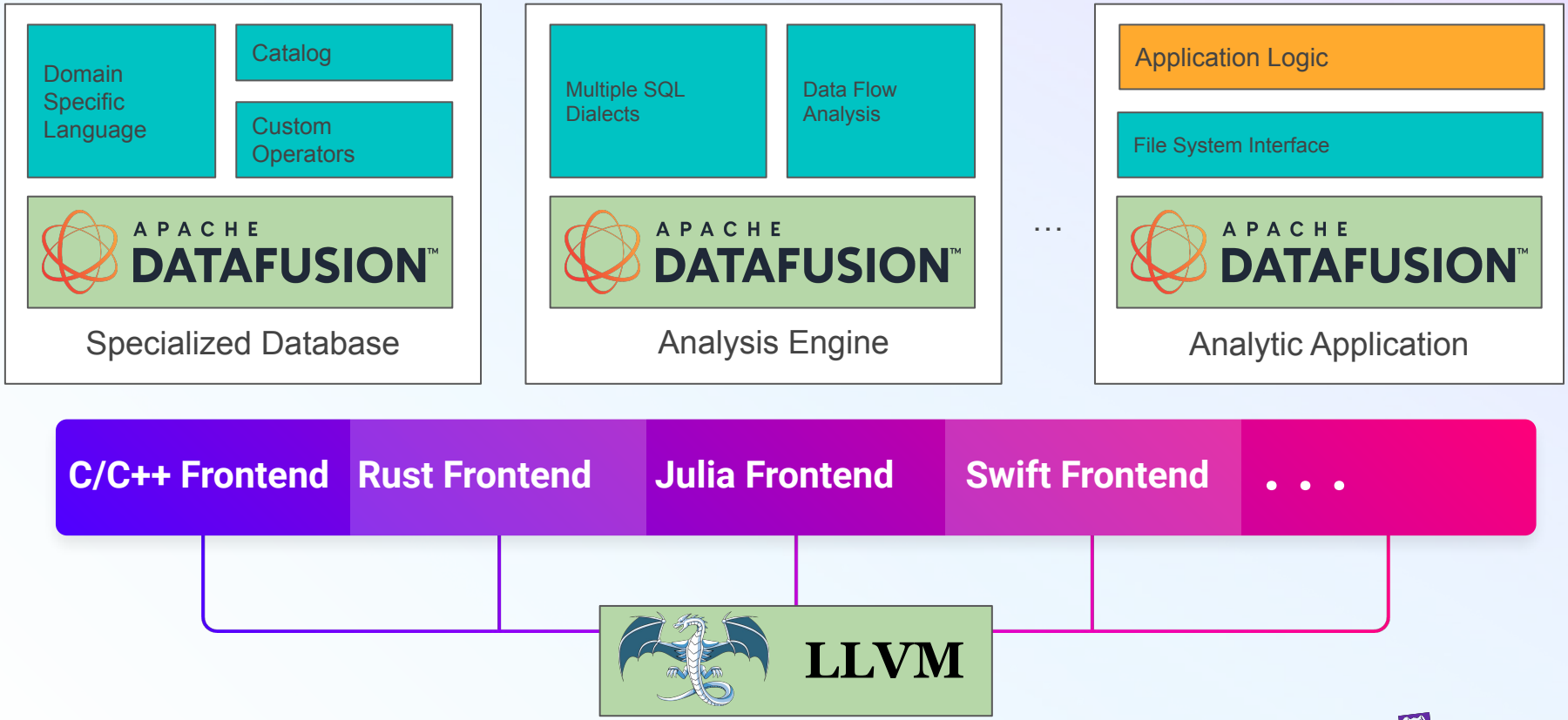
Apache Arrow

- Language-independent
- columnar memory format
- Efficient over the wire communication (Flight)
- Built for efficient analytic operations on modern hardware (SimD)

Libraries are available for C, C++, C#, Go, Java, JavaScript, Julia, MATLAB, Python, R, Ruby, and Rust



Building fast and feature rich database and analytic systems: DataFusion is LLVM for Databases



Query Engine Extensibility

Relational

- SQL Operations
- DataFrame Operations
- Multiple API's

Composable

- Parsing
- Query-planning
- UDF's
- Table Providers

Performant

- Columnar
- Vectorized
- Streaming
- Multi Threaded
- No Garbage Collector

DATADOG

Customized to Particular Workloads

Domain-Specific Database Systems

- Time series databases (e.g. InfluxDB 3.0 and Coralogix)
- Streaming SQL platforms (e.g. Synnada and Arroyo).

Run-times for specialized query front-ends

- Comet for Apache Spark
- Seafowl for PostgreSQL
- Vega
- InfluxQL

SQL analysis tools

- Dask-sql
- SDF

Table formats

Rust implementations of…

- Delta Lake
- Apache Iceberg
- Lance

Today

Segmented Query Stack

Proprietary memory layouts increase conversion overhead

Different execution engines lead to inconsistent user experiences

Multiple query representations limits interoperability

Goal

Composable Data Systems

Apache Arrow minimizes conversion + marshaling costs

Datafusion's extensibility let it match user's expected behavior

Substrait can be used across our downstream data sources

How we are using Datafusion at Datadog

Event Reads

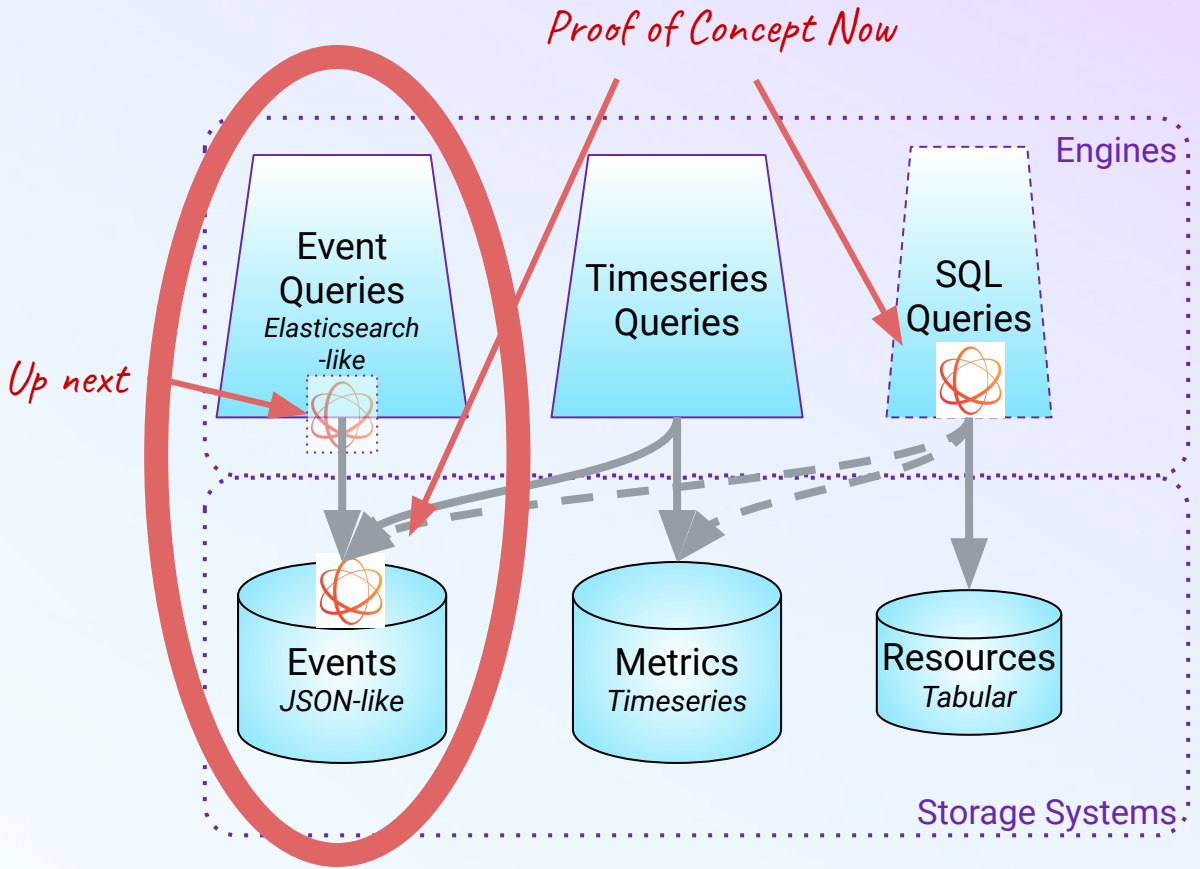
Filter and project on read

Event Queries

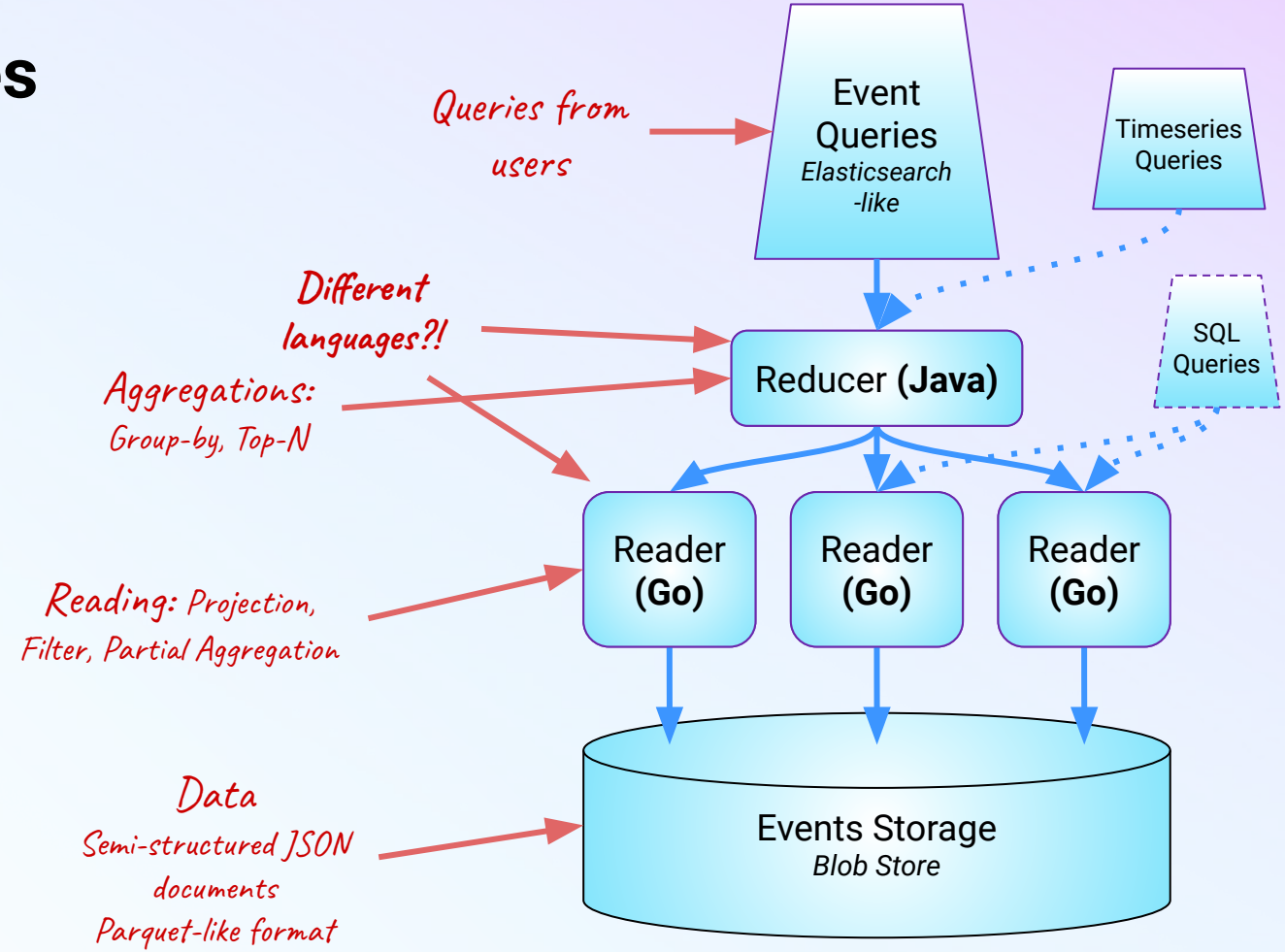
Aggregation, sort, limit

SQL Queries

JOINs, subqueries, CTEs



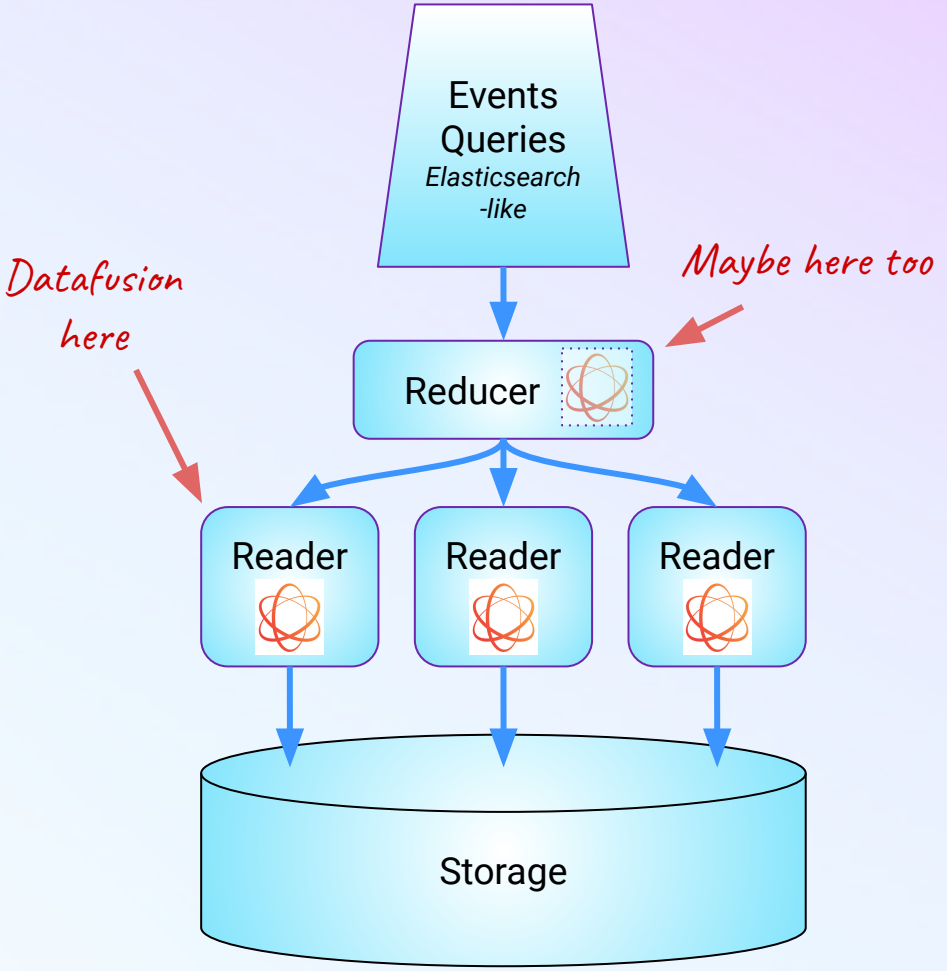
Events Queries



Goal

Datafusion as a shared common engine across services

- Out of the box industry-supported “framework”
- Consolidate contracts and behaviors
  - Single IR, single engine, single format
- Free time to focus on higher level problems
  - Text search, interactive queries, approximate operators, shuffling, etc



Events Queries and Datafusion

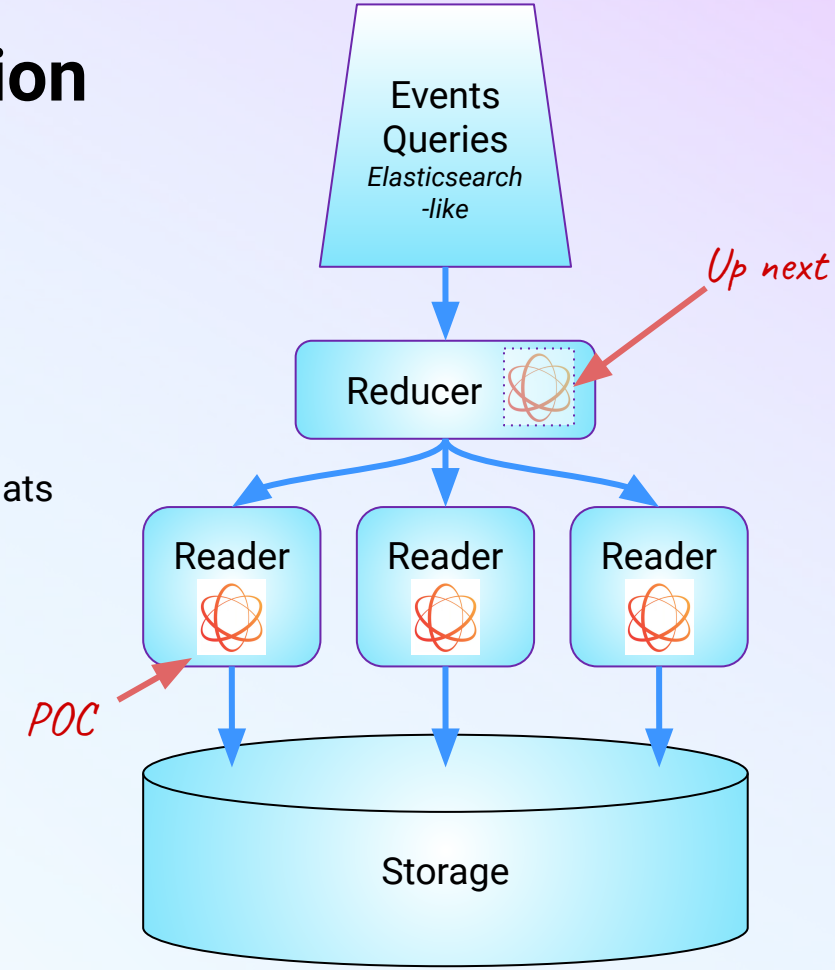
How is it going?

Challenge: Matching Behaviors

- Requires flexibility: extensions to match existing functionality
  - Custom coercions, approximations, exotic operators
- Efficient integration with our various storage formats
  - e.g. late materialization / Arrow translation: not all our encodings are cheaply translatable

Status

- Currently **shadowing** queries on readers
- Seeing **performance improvements** and discovering bugs in our existing engines
- **Contributing back** to community: Decoupling logical/physical types



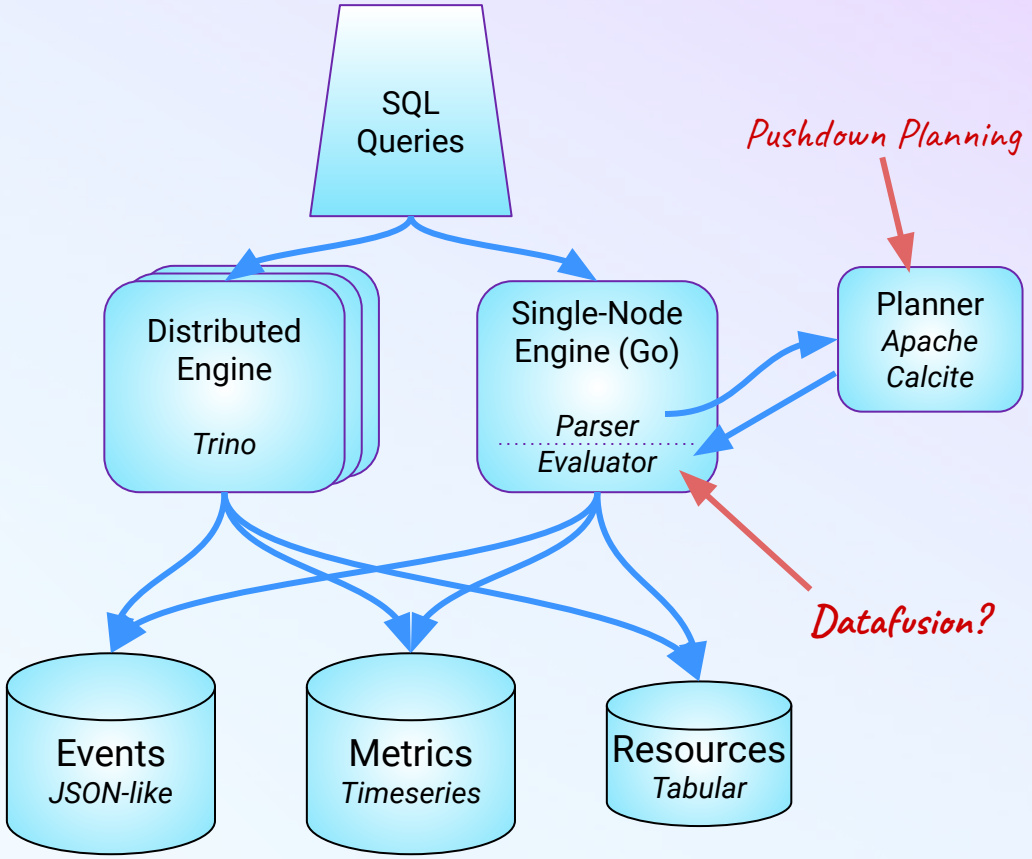
SQL Queries

Distributed Engine

For large operations

Single Node Engine

For high-throughput,
Low-cost queries



SQL Queries and Datafusion

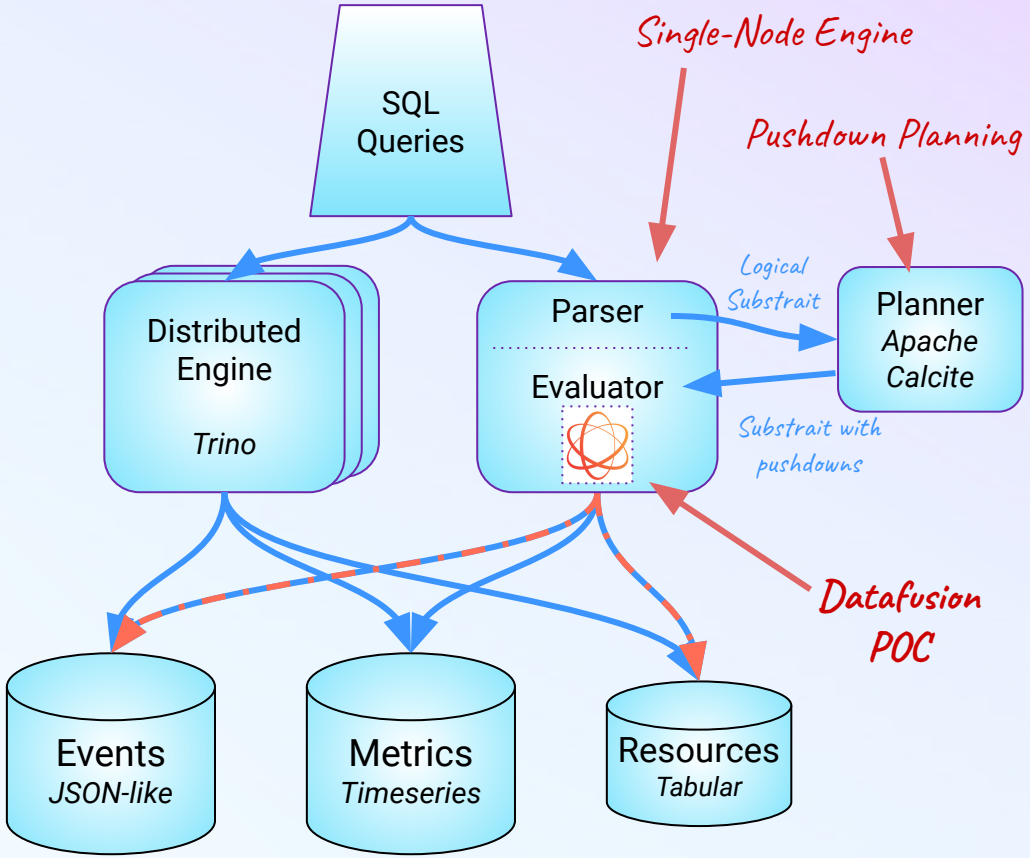
How is it going?

Challenges

- Our own **SQL dialect**
  - ⇒ Separate parser
- Extension **types**
  - String sets (for tags), timeseries
- Efficiency requires **pushdown** operations
  - ⇒ Separate planner
- Different **downstreams** to integrate with
  - ⇒ Substrait (Contributions)

Status

- **Proof of concept**
  - Embedded library in Go service
  - Planner integration via substrait
- **Basic SQL operations** functional
- Events and Resources as **sources** (no Metrics)



Ongoing Work

Integrating and Extending via Substrait
Using Substrait requires many extensions

Bridging Rust to Go
Embedding eases the transition, but FFI work is finicky

Planner Integration
Calcite plans ≠ Datafusion plans

Consolidation
Standardizing across our different engines:
Substrait extensions, connectors, function behavior, …

TODAY

Specialized Engines

with converging functionality

Duplication of effort

(Reimplementation)

Custom IR / Formats

Different protobufs for each source

Behavior discrepancies

Re-inventing commodity operations

VISION

Convergence

of implementation and behavior

Library Reuse

Datafusion for same functions, different places

Adaptability

Add or replace components, sources

Simplify Contracts

Substrait for plans

Arrow for data

DATADOG

Thank you!