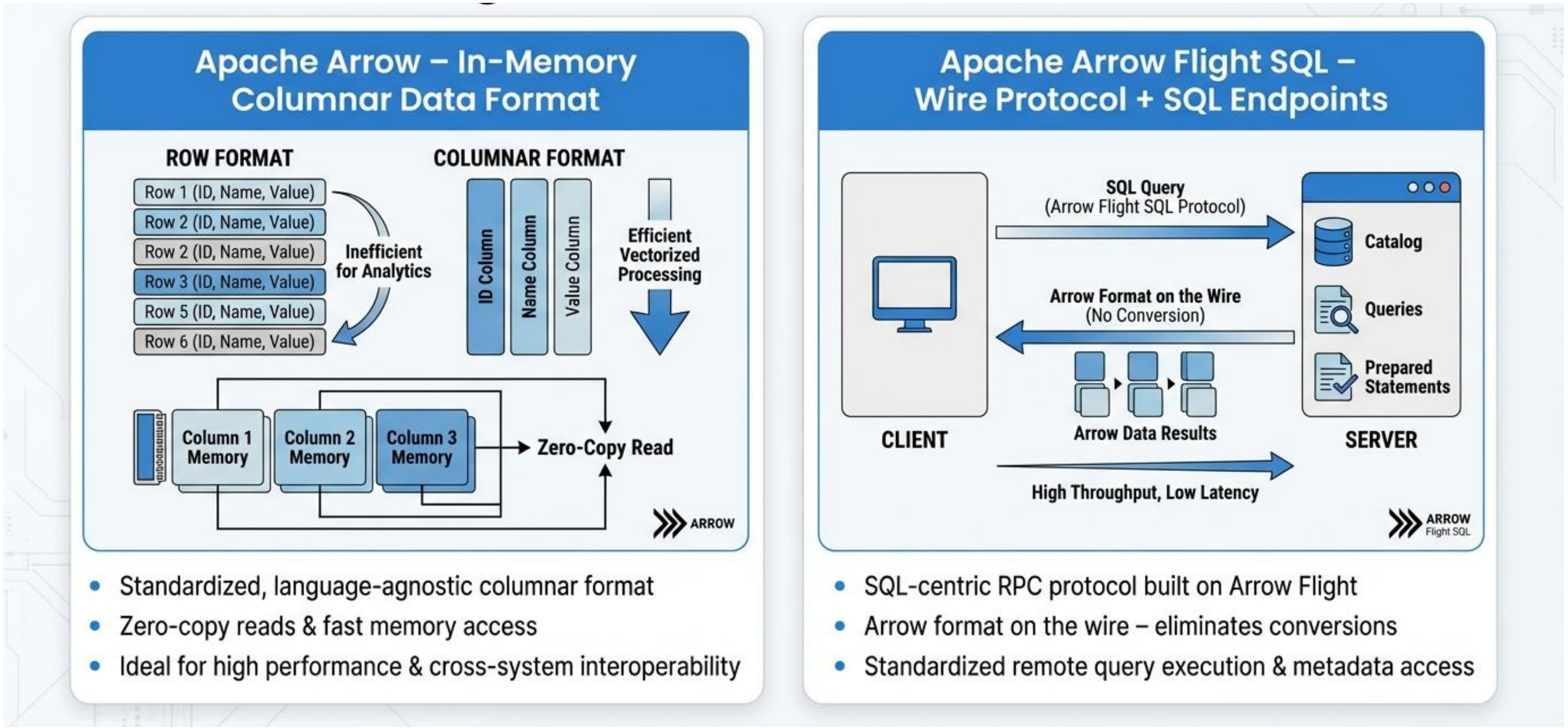
# Federating Databases with Apache DataFusion
# -
# Open Query Planning and Arrow-Native Interoperability

Michiel De Backker
@backkem

Ghasan Mohammad
@hozan23

TWINTAG

# At first; there was Apache Arrow.

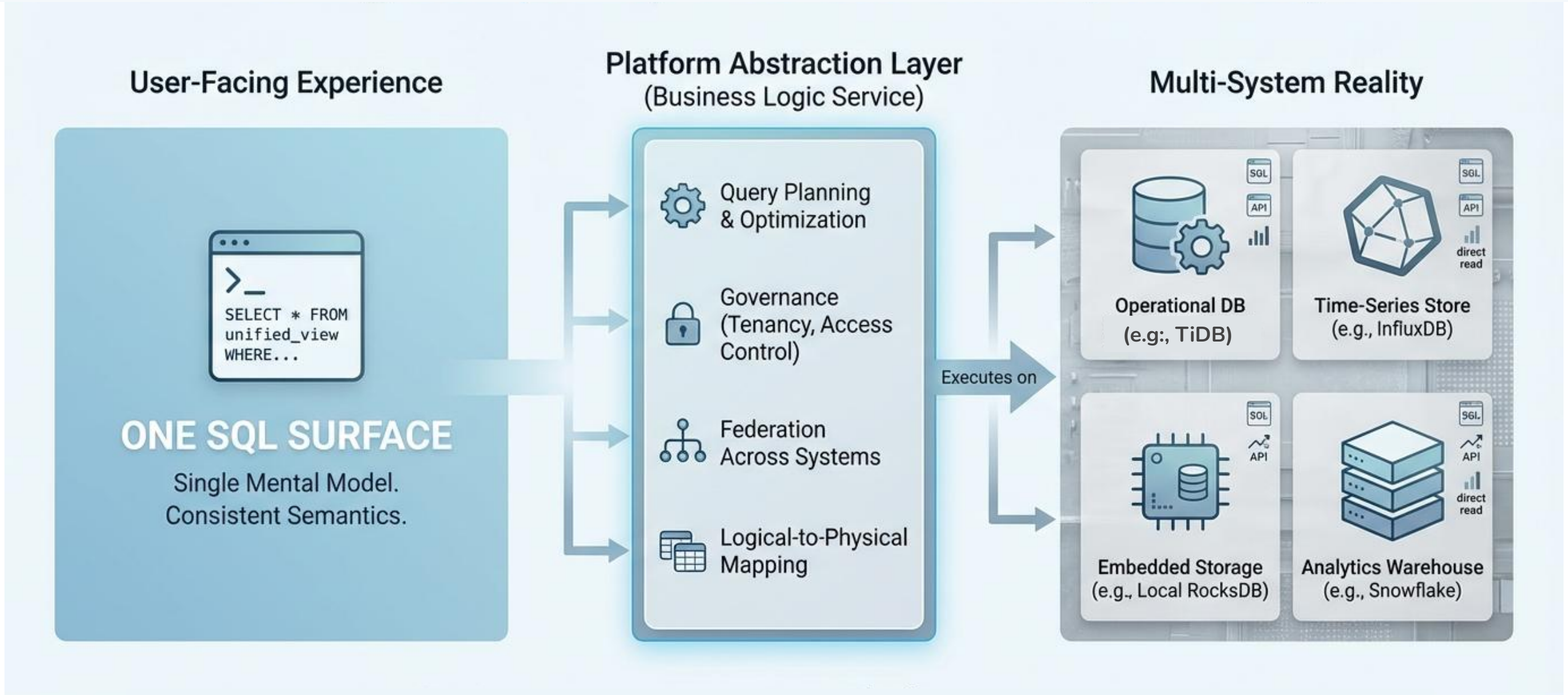


- Standardized, language-agnostic columnar format
- Zero-copy reads & fast memory access
- Ideal for high performance & cross-system interoperability

- SQL-centric RPC protocol built on Arrow Flight
- Arrow format on the wire – eliminates conversions
- Standardized remote query execution & metadata access

TWINTAG

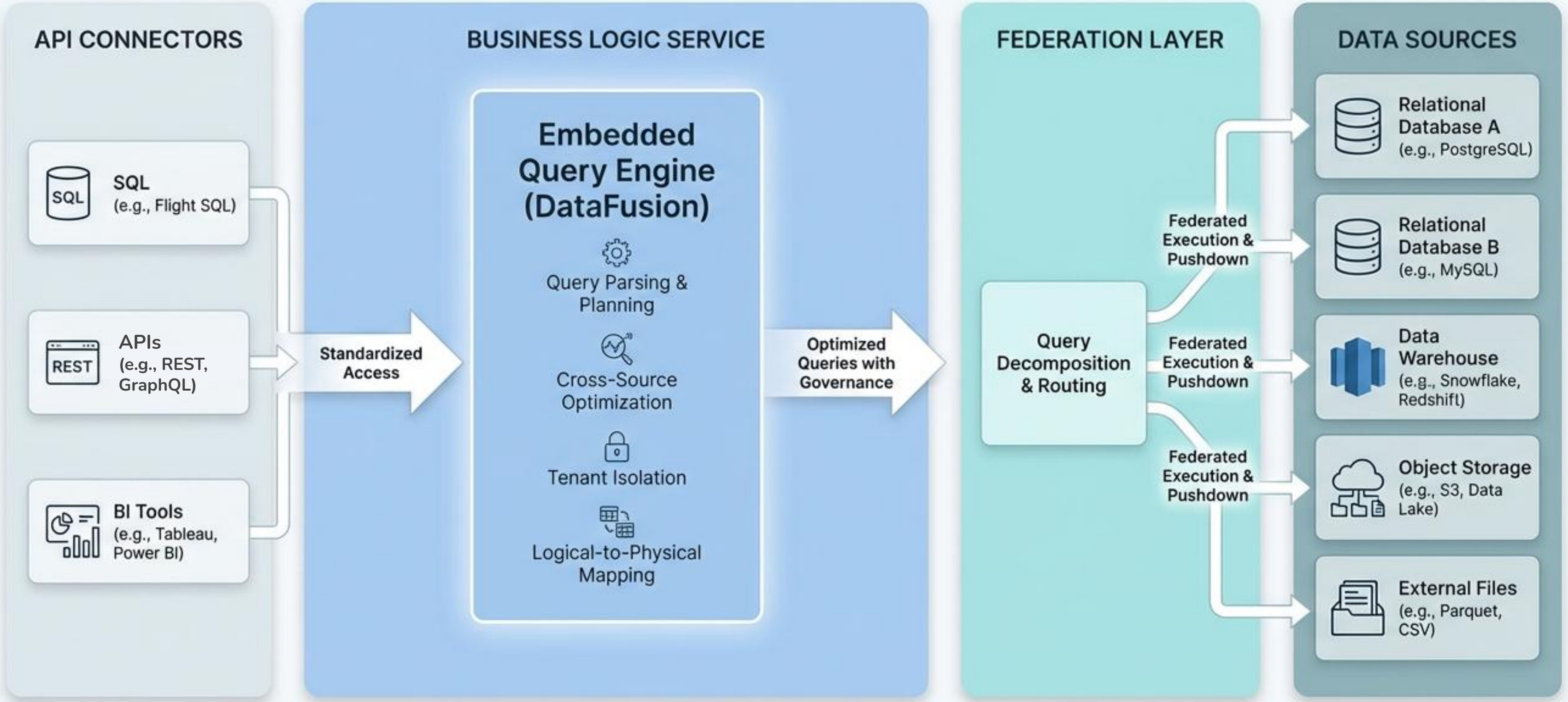
## Product design goal: User-Facing simplicity


User-Facing Experience
Platform Abstraction Layer
(Business Logic Service)
Multi-System Reality
>_
SELECT * FROM
unified_view
WHERE...
ONE SQL SURFACE
Single Mental Model.
Consistent Semantics.
Query Planning & Optimization
Governance (Tenancy, Access Control)
Federation Across Systems
Logical-to-Physical Mapping
Executes on
SQL
API
direct read
Operational DB (e.g:, TiDB)
Time-Series Store (e.g., InfluxDB)
Embedded Storage (e.g., Local RocksDB)
Analytics Warehouse (e.g., Snowflake)

**Customers see one system, Platform manages diverse storage**

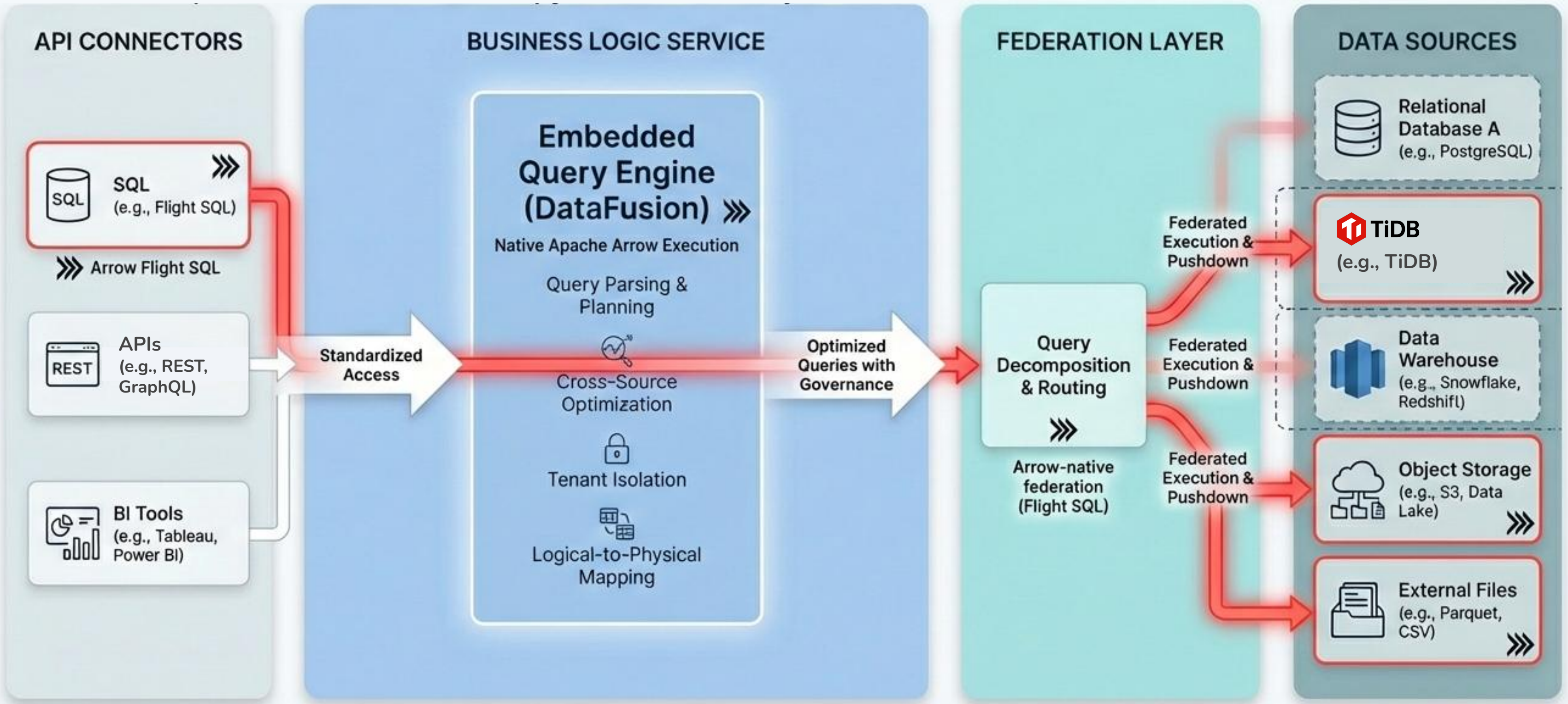
TWINTAG

# Shared Data Fabric Architecture



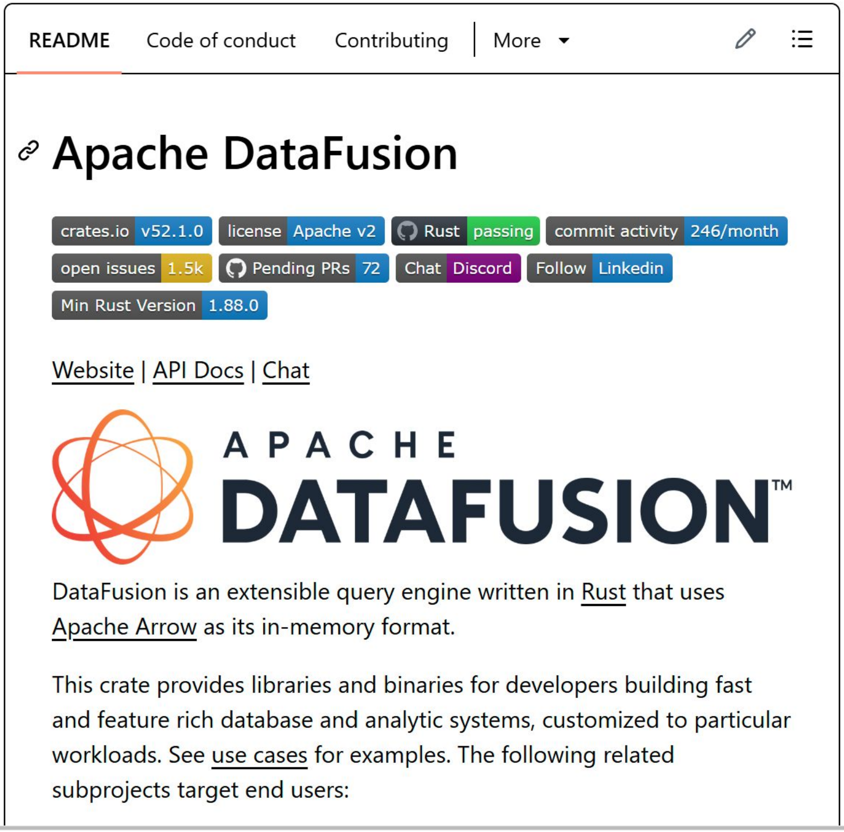
# Shared Data Fabric: Unifying Heterogeneous Sources

Using End-to-End Apache Arrow for efficient ('Zero Copy') data handling


API CONNECTORS
SQL (e.g., Flight SQL)
Arrow Flight SQL
APIs (e.g., REST, GraphQL)
BI Tools (e.g., Tableau, Power BI)
BUSINESS LOGIC SERVICE
Standardized Access
Embedded Query Engine (DataFusion)
Native Apache Arrow Execution
Query Parsing & Planning
Cross-Source Optimization
Tenant Isolation
Logical-to-Physical Mapping
Optimized Queries with Governance
FEDERATION LAYER
Query Decomposition & Routing
Arrow-native federation (Flight SQL)
Federated Execution & Pushdown
Federated Execution & Pushdown
Federated Execution & Pushdown
DATA SOURCES
Relational Database A (e.g., PostgreSQL)
TiDB (e.g., TiDB)
Data Warehouse (e.g., Snowflake, Redshifl)
Object Storage (e.g., S3, Data Lake)
External Files (e.g., Parquet, CSV)

🟥 Apache Arrow (In-memory / Flight SQL wire format)

## DataFusion: Query engine shipped as a library



# Apache DataFusion

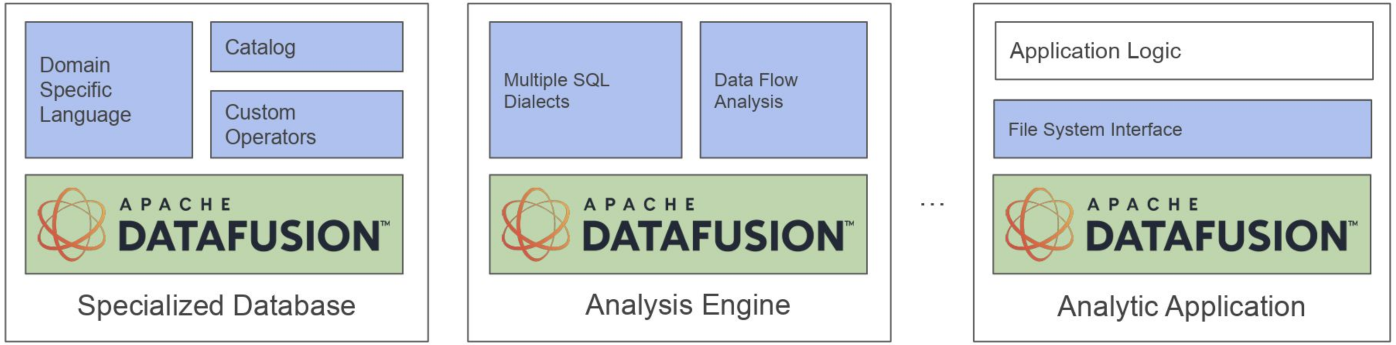
Website | API Docs | Chat

DataFusion is an extensible query engine written in Rust that uses Apache Arrow as its in-memory format.

This crate provides libraries and binaries for developers building fast and feature rich database and analytic systems, customized to particular workloads. See use cases for examples. The following related subprojects target end users:

## "DataFusion is LLMV for Databases."

— Andrew Lamb, Apache {DataFusion, Arrow} PMC[1]

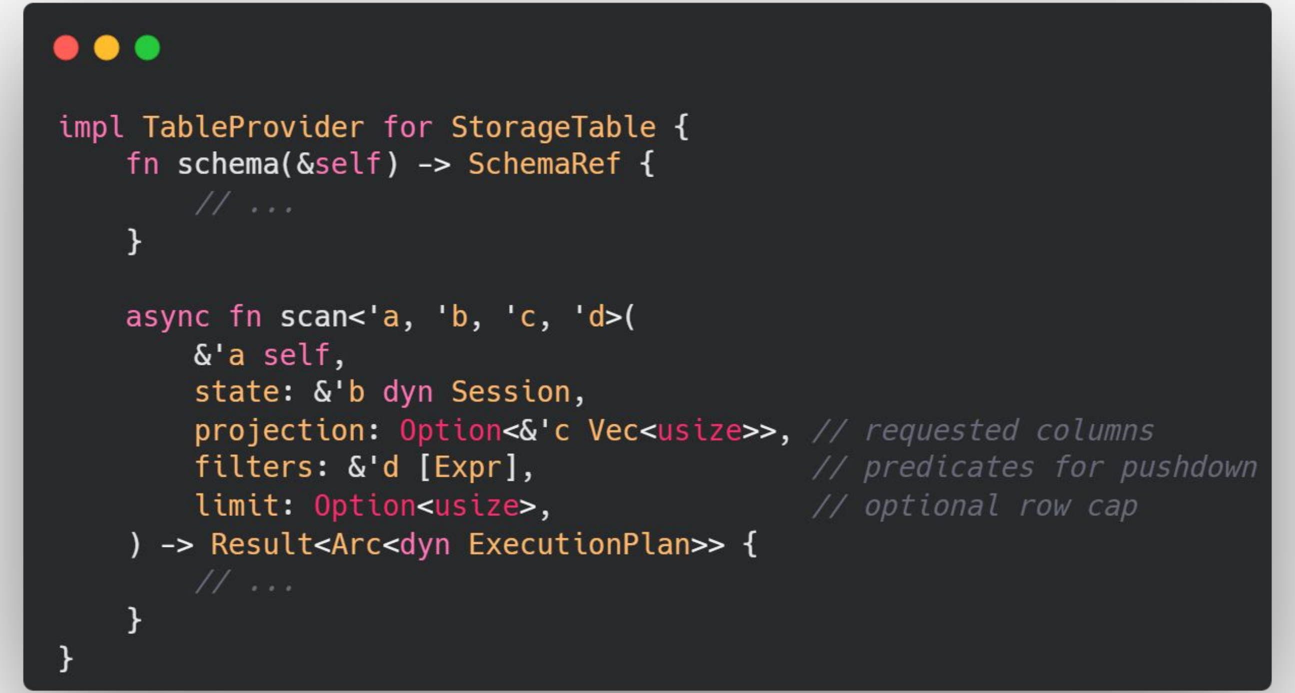


DataFusion enables innovation in data intensive systems

- High quality reusable SQL planner, optimizer, function library, vectorized operators, etc
- Focus on language design, data management, use case specific features

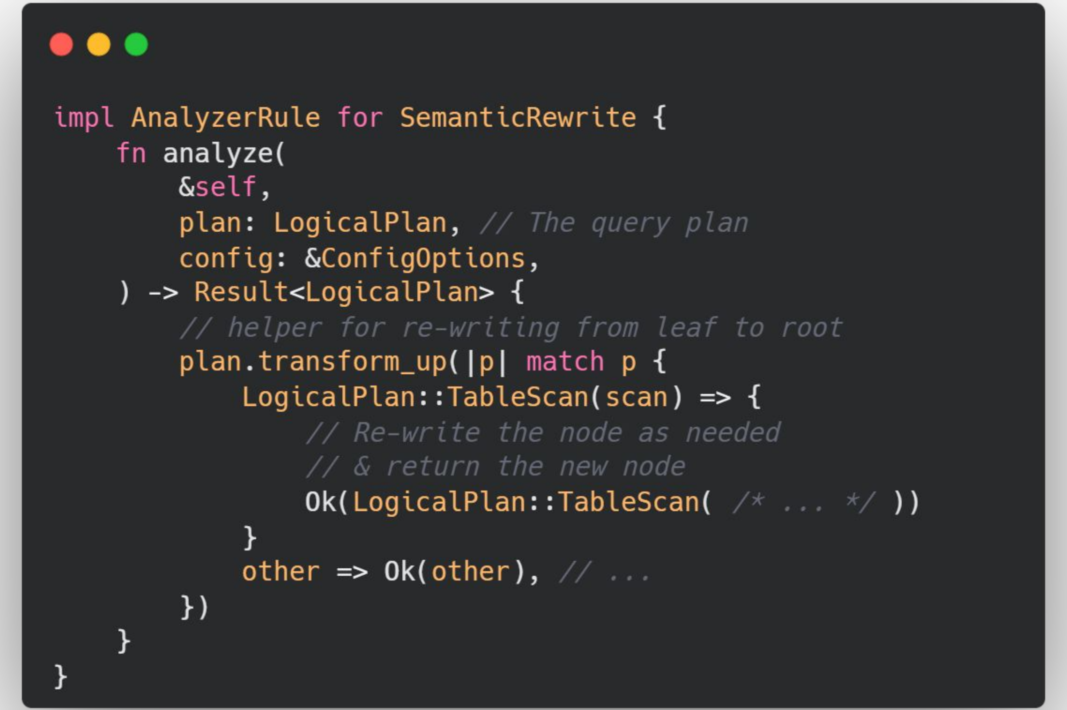
[1] A. Lamb, "Apache DataFusion: Design choices when building modern analytic systems," Boston University Data Systems Seminar, Boston, MA, USA, 28-Oct-2024. [Online]. Available: https://midas.bu.edu/assets/slides/andrew_lamb_slides.pdf

## Extending DataFusion: The TableProvider



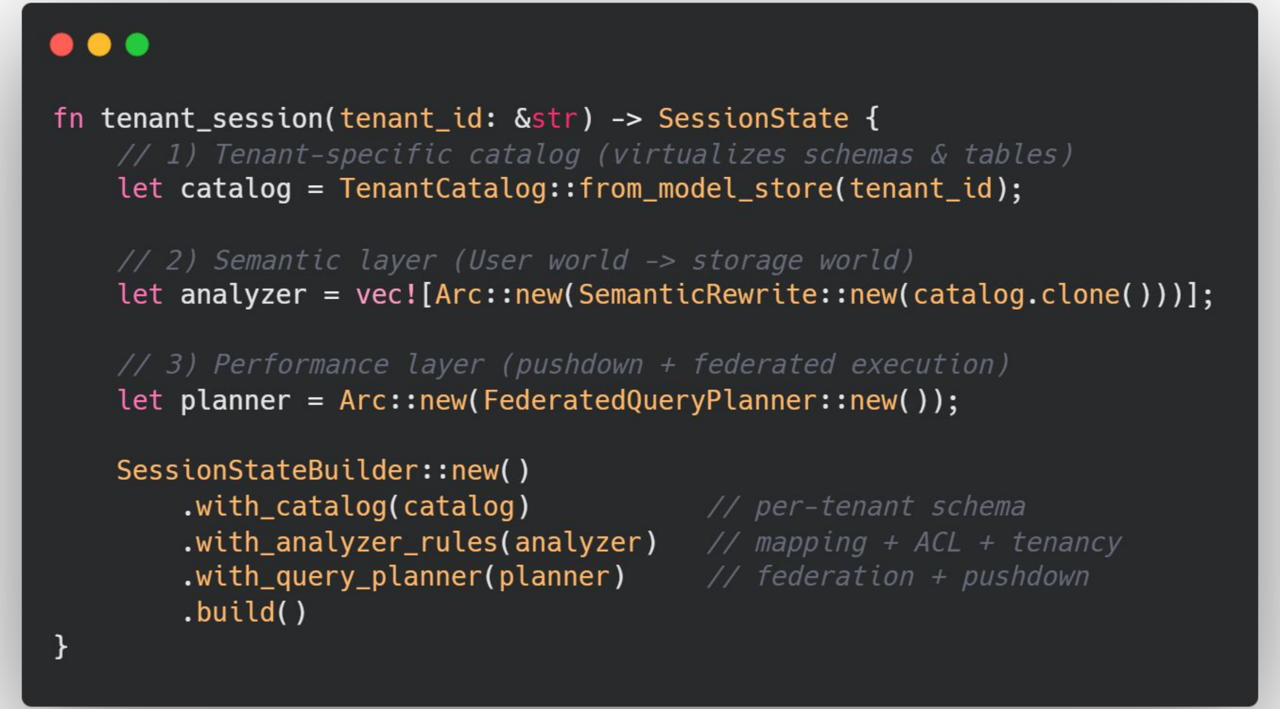
```
impl TableProvider for StorageTable {
    fn schema(&self) -> SchemaRef {
        // ...
    }

    async fn scan<'a, 'b, 'c, 'd>(
        &'a self,
        state: &'b dyn Session,
        projection: Option<&'c Vec<usize>>, // requested columns
        filters: &'d [Expr],                // predicates for pushdown
        limit: Option<usize>,               // optional row cap
    ) -> Result<Arc<dyn ExecutionPlan>> {
        // ...
    }
}
```

## Extending DataFusion: Analyser / Optimizer rules



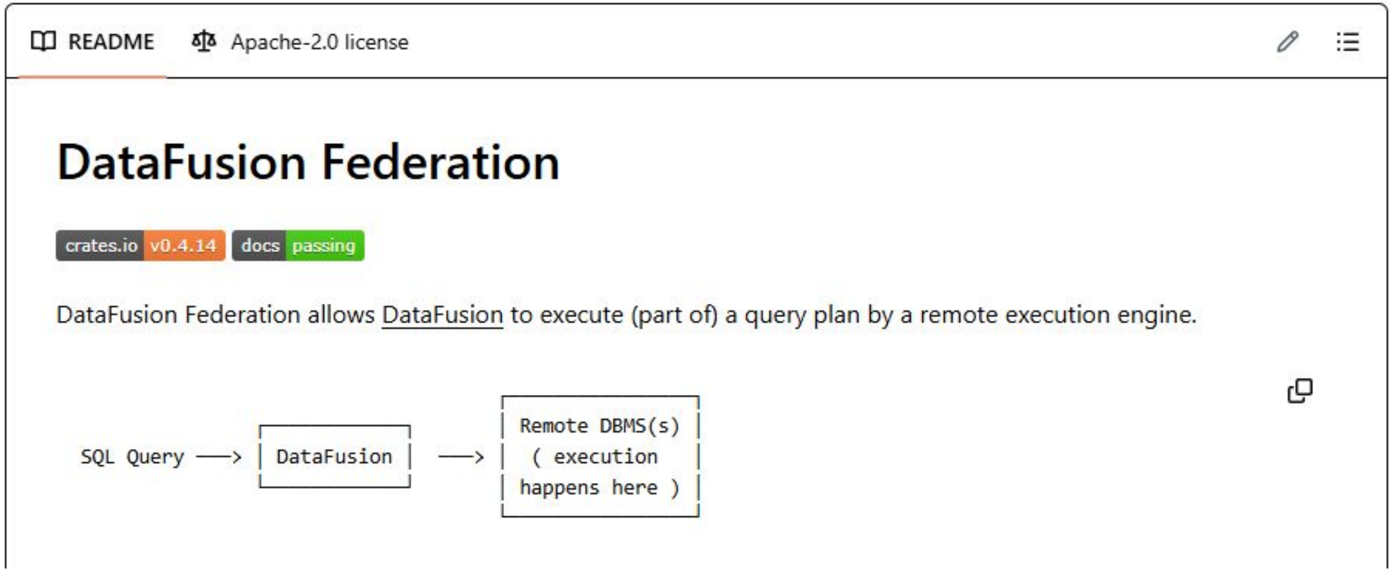
```
impl AnalyzerRule for SemanticRewrite {
    fn analyze(
        &self,
        plan: LogicalPlan, // The query plan
        config: &ConfigOptions,
    ) -> Result<LogicalPlan> {
        // helper for re-writing from leaf to root
        plan.transform_up(|p| match p {
            LogicalPlan::TableScan(scan) => {
                // Re-write the node as needed
                // & return the new node
                Ok(LogicalPlan::TableScan( /* ... */ ))
            }
            other => Ok(other), // ...
        })
    }
}
```

## Extending DataFusion: SessionState



```rust
fn tenant_session(tenant_id: &str) -> SessionState {
    // 1) Tenant-specific catalog (virtualizes schemas & tables)
    let catalog = TenantCatalog::from_model_store(tenant_id);

    // 2) Semantic layer (User world -> storage world)
    let analyzer = vec![Arc::new(SemanticRewrite::new(catalog.clone()))];

    // 3) Performance layer (pushdown + federated execution)
    let planner = Arc::new(FederatedQueryPlanner::new());

    SessionStateBuilder::new()
        .with_catalog(catalog)            // per-tenant schema
        .with_analyzer_rules(analyzer)    // mapping + ACL + tenancy
        .with_query_planner(planner)      // federation + pushdown
        .build()
}
```

## DataFusion Federation: Remote Query Execution



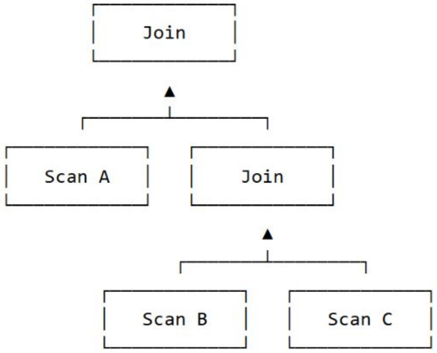
# DataFusion Federation

DataFusion Federation allows DataFusion to execute (part of) a query plan by a remote execution engine.

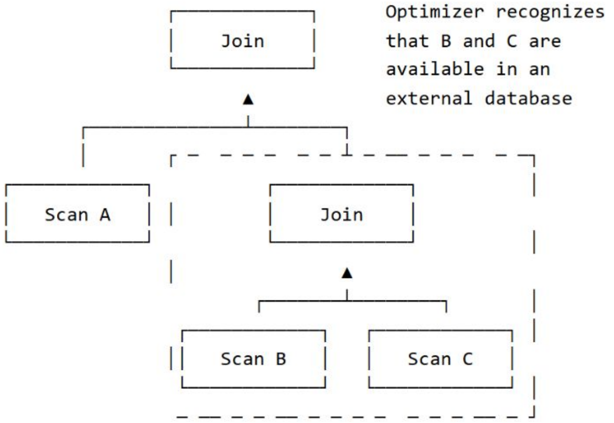
```
                                          ┌─────────────────┐
                  ┌──────────────┐        │ Remote DBMS(s)  │
SQL Query ───>    │  DataFusion  │  ───>  │  ( execution    │
                  └──────────────┘        │ happens here )  │
                                          └─────────────────┘
```

# DataFusion Federation: Sub-plan identification

Say you have a query plan as follows:



DataFusion Federation will identify the largest possible sub-plans that can be executed by an external database:[1]



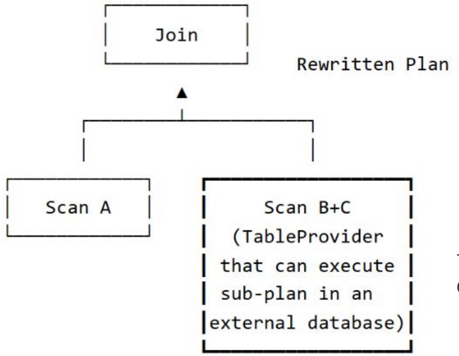
Join
Optimizer recognizes that B and C are available in an external database
Scan A
Join
Scan B
Scan C

[1] Requires TableProvider extension. A large collection is available at https://github.com/datafusion-contrib/datafusion-table-providers

TWINTAG

# DataFusion Federation: ✂️sub-plan(s)

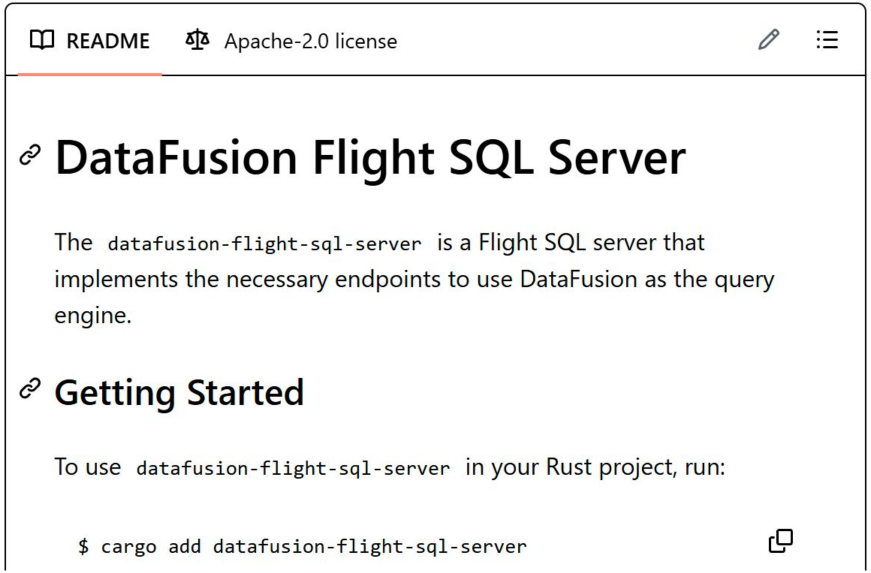
The sub-plans are cut out and replaced by an opaque federation node in the plan:



Join
Rewritten Plan
Scan A
Scan B+C (TableProvider that can execute sub-plan in an external database)

→ During execution the sub-plan is serialized to SQL (or other format) for remote execution.

## DataFusion over the wire!



### DataFusion Flight SQL Server

The `datafusion-flight-sql-server` is a Flight SQL server that implements the necessary endpoints to use DataFusion as the query engine.

#### Getting Started

To use `datafusion-flight-sql-server` in your Rust project, run:

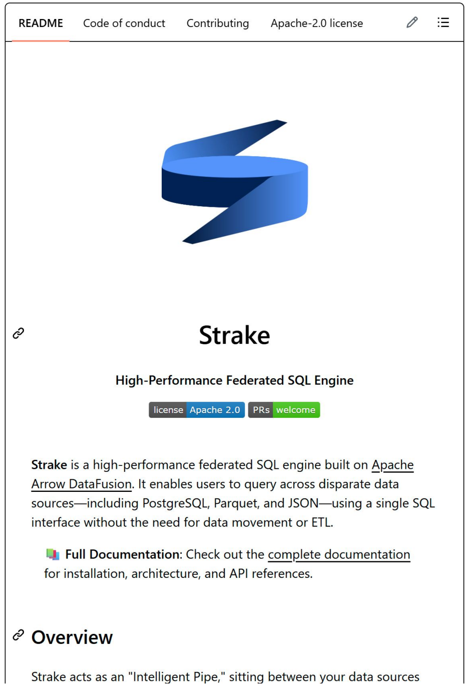
```
$ cargo add datafusion-flight-sql-server
```

Expose DF to other services

- Stateless (no connection)
- Basic/Bearer auth

TWINTAG

## Build on top!



README Code of conduct Contributing Apache-2.0 license

### Strake

**High-Performance Federated SQL Engine**

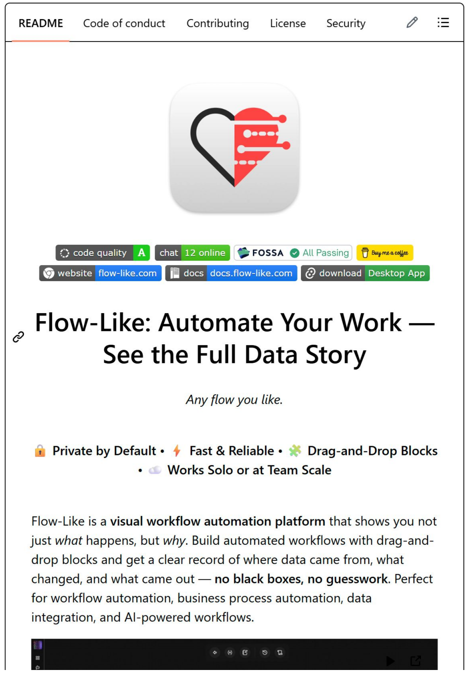
license Apache 2.0 PRs welcome

**Strake** is a high-performance federated SQL engine built on Apache Arrow DataFusion. It enables users to query across disparate data sources—including PostgreSQL, Parquet, and JSON—using a single SQL interface without the need for data movement or ETL.

> 📚 **Full Documentation**: Check out the complete documentation for installation, architecture, and API references.

### Overview

Strake acts as an "Intelligent Pipe," sitting between your data sources



README Code of conduct Contributing License Security

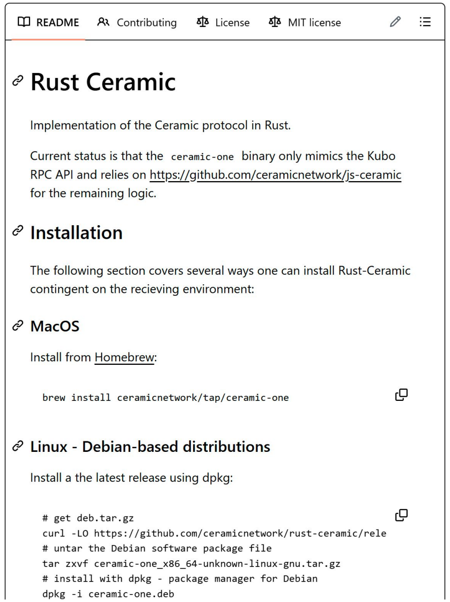
code quality A chat 12 online FOSSA All Passing Buy me a coffee
website flow-like.com docs docs.flow-like.com download Desktop App

### Flow-Like: Automate Your Work — See the Full Data Story

*Any flow you like.*

🔒 **Private by Default** • ⚡ **Fast & Reliable** • 🧩 **Drag-and-Drop Blocks** • ☁️ **Works Solo or at Team Scale**

Flow-Like is a **visual workflow automation platform** that shows you not just *what* happens, but *why*. Build automated workflows with drag-and-drop blocks and get a clear record of where data came from, what changed, and what came out — **no black boxes, no guesswork**. Perfect for workflow automation, business process automation, data integration, and AI-powered workflows.



README Contributing License MIT license

### Rust Ceramic

Implementation of the Ceramic protocol in Rust.

Current status is that the `ceramic-one` binary only mimics the Kubo RPC API and relies on https://github.com/ceramicnetwork/js-ceramic for the remaining logic.

### Installation

The following section covers several ways one can install Rust-Ceramic contingent on the recieving environment:

#### MacOS

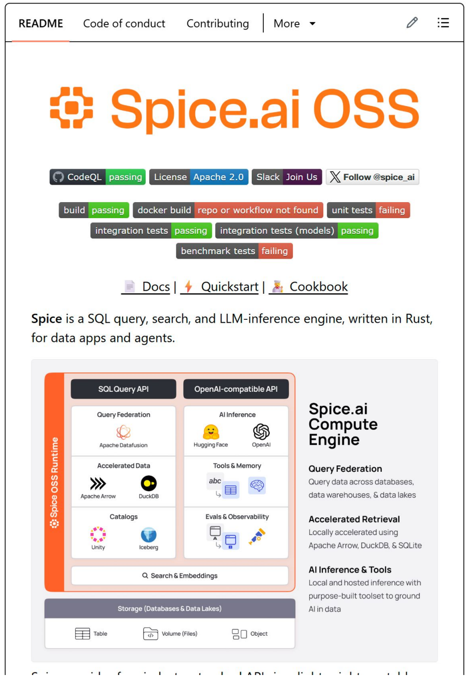
Install from Homebrew:

```
brew install ceramicnetwork/tap/ceramic-one
```

#### Linux - Debian-based distributions

Install a the latest release using dpkg:

```
# get deb.tar.gz
curl -LO https://github.com/ceramicnetwork/rust-ceramic/rele
# untar the Debian software package file
tar zxvf ceramic-one_x86_64-unknown-linux-gnu.tar.gz
# install with dpkg - package manager for Debian
dpkg -i ceramic-one.deb
```



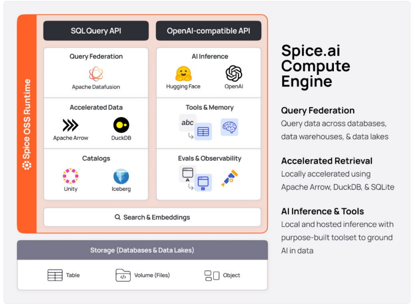
README Code of conduct Contributing More

### Spice.ai OSS

CodeQL passing License Apache 2.0 Slack Join Us Follow @spice_ai
build passing docker build repo or workflow not found unit tests failing
integration tests passing integration tests (models) passing
benchmark tests failing

📄 Docs | ⚡ Quickstart | 🧑‍🍳 Cookbook

**Spice** is a SQL query, search, and LLM-inference engine, written in Rust, for data apps and agents.



## It's dangerous to go alone! Take this.

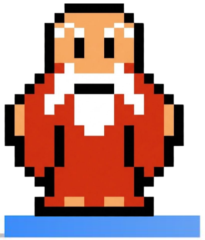
github.com/apache/datafusion

DataFusion extensions

- datafusion-contrib/datafusion-federation
- datafusion-contrib/datafusion-table-providers (by @spiceai)
- datafusion-contrib/datafusion-flight-sql-server
- github.com/pingcap/tidb/pull/65422

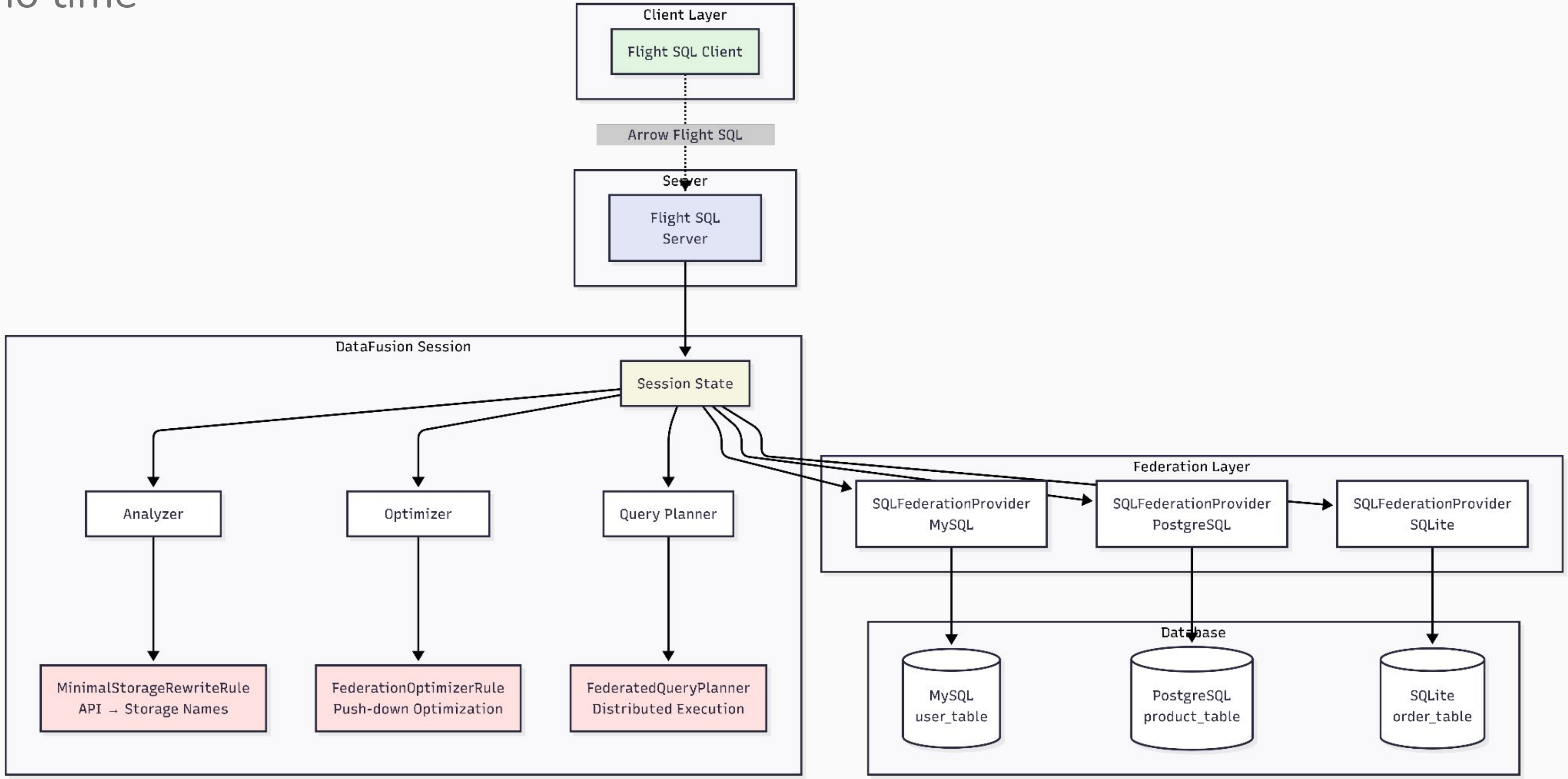
Demo coming right up!

- github.com/twintag/fosdemdemo2026



TWINTAG

# Demo time


Client Layer
Flight SQL Client
Arrow Flight SQL
Server
Flight SQL Server
DataFusion Session
Session State
Analyzer
Optimizer
Query Planner
MinimalStorageRewriteRule API → Storage Names
FederationOptimizerRule Push-down Optimization
FederatedQueryPlanner Distributed Execution
Federation Layer
SQLFederationProvider MySQL
SQLFederationProvider PostgreSQL
SQLFederationProvider SQLite
Database
MySQL user_table
PostgreSQL product_table
SQLite order_table

TWINTAG