influxdata®

# Building InfluxDB 3.0

## With Apache Arrow, DataFusion, Flight, Parquet

Andrew Lamb | Staff Engineer, InfluxData

influxdata®

**Andrew Lamb**

Staff Engineer
InfluxData

> ~~20~~ 21 😱years in enterprise software development

Oracle: Database (2 years)

DataPower: XSLT compiler (2 years)

Vertica: DB / Query Optimizer (6 years)

Nutonian/DataRobot: ML Startups (7 years)

InfluxData: InfluxDB 3.0, Arrow, DataFusion (4 years)

# Goals

Convince you, via example, that:

1. Databases of the future will be assembled from reusable components
2. This is not a terrible idea

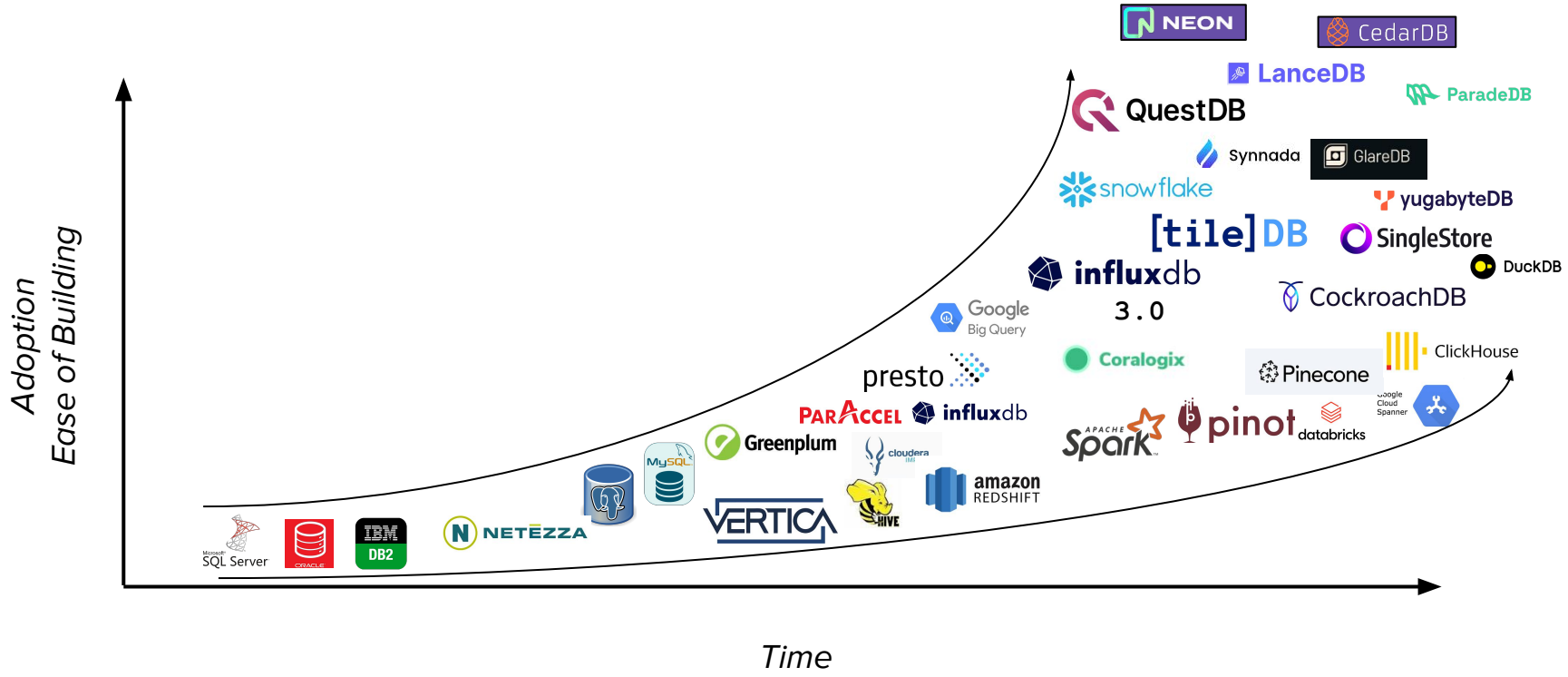
Talk outline:

- Database Implementation Trends
- Time Series Databases, need for (yet another) one
- FDAP: Flight, DataFusion, Arrow, Parquet: Rationale + Use InfluxDB 3.0

Blog: [Flight, DataFusion, Arrow, and Parquet: Using the FDAP Architecture to build InfluxDB 3.0]

# Thesis: Long Term Trends in Databases



influxdata®

# *“One Size Fits All”*

## *An Idea Whose Time Has Come and Gone*

*Stonebraker & Çetintemel (2005)*

Case Study: In which InfluxData decides to build a **new time series database**.

# What / Why of Time Series Databases

**Specialized for storing data with times (obviously).**

**Key Properties:**

1. **High volume, denormalized ingest**
    - eg. host=myhost123.example.com repeated over and over
2. **Low latency query after load:** milliseconds between ingest and query
3. **Schema on write:** new columns can appear at any time, backfills
4. **Rapid data value decay:** newest data is super important, falls off drastically

**Examples:**
Open ➔ InfluxDB, Timescale, Graphite Whisper, VictoriaMetrics
Closed ➔ Facebook Gorilla, Google Monarch, AWS Timestream, DataDog Husky

# InfluxDB 3.0 Requirements

- **Need:** No series cardinality limits:
  - **Tech:** TSM (LSM Tree / KV Store) ➔ Column Store
- **Need:** ‘Infinite’ Retention
  - **Tech:** TSM on locally attached disks ➔ object store (cheap!)
- **Need:** Elastic Scalability
  - **Tech:** Shared Nothing (local disk)➔ Disaggregated Storage (S3)
- **Need:** Ecosystem Compatibility
  - **Tech:** InfluxQL / custom APIs ➔ SQL + JDBC/ODBC

# So, let’s build a new Database

(How hard could that be, really?)

# It is hard (expensive) to build a new databases

Database company money raised

- Snowflake: $2B
- Databricks (Spark) $3.5B
- MongoDB: $311M
- SingleStore: $464.1M
- CockroachLabs (CockroachDB): $633.1M
- Pingcap (TiDB): $341.6M
- Elastic: $162M
- TimescaleDB: $181M
- DuckDB $? / MotherDuck: ~~$47.5M~~ $100M


Source: https://www.crunchbase.com

# “We can do it with the Apache Arrow Ecosystem”


Evan
(CEO)

Paul
(CTO)

influxdata® Products Developers Customers Resources Pricing Contact Us Log In Get InfluxDB

## Apache Arrow, Parquet, Flight and Their Ecosystem are a Game Changer for OLAP

By Paul Dix / Apr 16, 2020 / Community, Developer

Apache Arrow, a specification for an in-memory columnar data format, and associated projects: Parquet for compressed on-disk data, Flight for highly efficient RPC, and other projects for in-memory query processing will likely shape the future of OLAP and data warehousing systems. This will mostly be driven by the promise of interoperability between projects, paired with massive performance gains for pushing and pulling data in and out of big data systems. With object storage like S3 as the common data lake, OLAP projects need a common data API, which Parquet represents. For data science and query workloads, they need a common RPC that is optimized for pulling many millions of records to do more complex analytical and machine learning tasks.

In this post, I'll cover each of these areas and why I think the Apache Arrow umbrella of projects represents the common API around which current and future big data, OLAP, and data warehousing projects will collaborate and innovate. I'll conclude with some thoughts on where these projects are and where things might be going.

### Apache Arrow

Apache Arrow is an in-memory columnar data format. It is designed to take advantage of modern CPU architectures (like SIMD) to achieve fast performance on columnar data. It is ideal for vectorized analytical queries. The Arrow specification gives a standard memory layout for columnar and nested data that can be shared between processes and query processing libraries. It's the base level building block for working with in-memory (or MMAP'd) data that ties everything together. It's designed for zero-copy semantics to make moving data around as fast and efficient as possible.

### Persistence, bulk data and Parquet

Data is the API. I'm sure others said it before me, but I said it myself back in 2010 when talking about service-oriented design and building loosely coupled systems that interacted through message buses like RabbitMQ and later Kafka. What I meant then was that the data you passed through message queues served as the API for services that integrated with each other through those queues. And like APIs, that data needs to have a common serialization format and its schema should be versioned. More broadly, I was trying to highlight the importance of data interchange, which is as important for data processing and analytic systems as it is for services.

Categories

About Company
Community
Developer
- Community
- Flux
- InfluxData
- InfluxDB Templates
- InfluxDB
- Partners
- Kapacitor
- Release Notes
- Tech Tips
- Telegraf
- Chronograf

General
InfluxDB Cloud
InfluxDB Enterprise
InfluxDays
Newsroom
Trust
Tutorial
Use Case

Try InfluxDB Cloud

The most powerful time series database as a service

https://www.influxdata.com/blog/apache-arrow-parquet-flight-and-their-ecosystem-are-a-game-changer-for-olap/

influxdata®

# Benefits of building on FDAP foundation

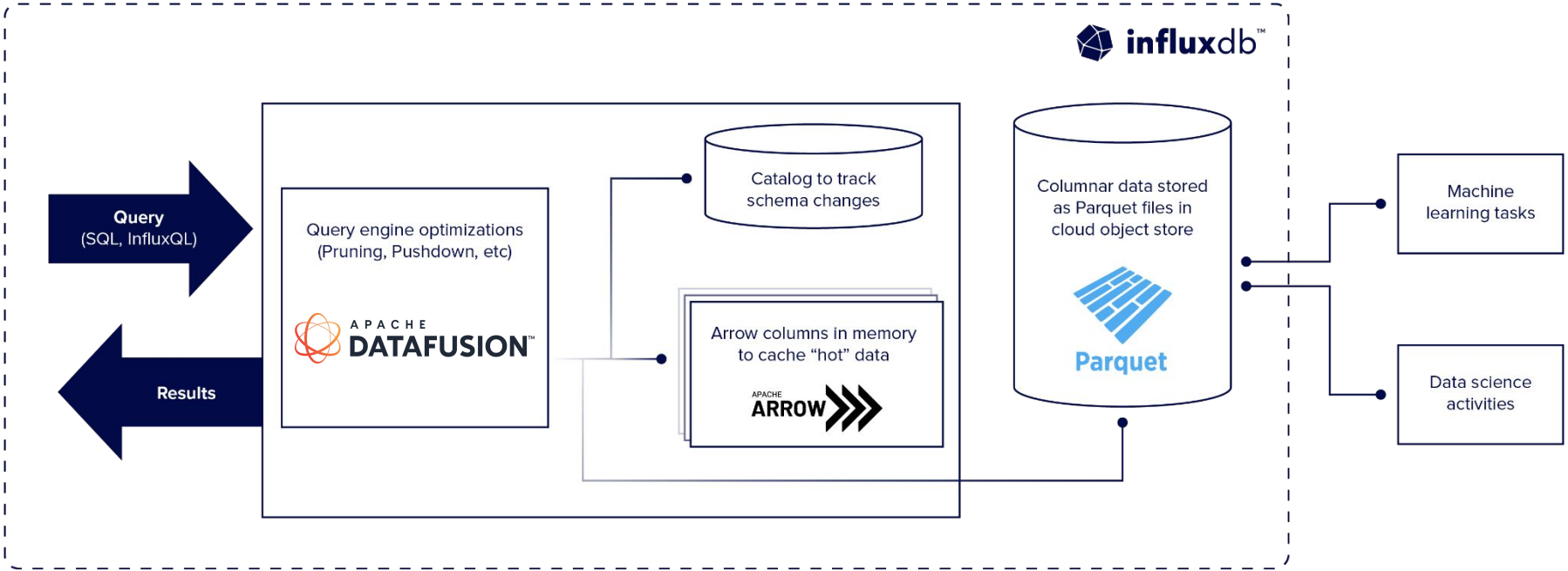
⇒ Innovation on Time Series, FDAP provides standard (lengthy to build) pieces

```
match tool_needed_for_database {
  File format (persistence) => Parquet,
  Columnar memory representation => Arrow Arrays,
  Operations (e.g. multiply, avg) => Compute Kernels,
  SQL + extensible query engine => Arrow DataFusion,
  Network transfer => Arrow Flight,
  JDBC/ODBC driver => Arrow FlightSQL,
}
```

# Let’s do it

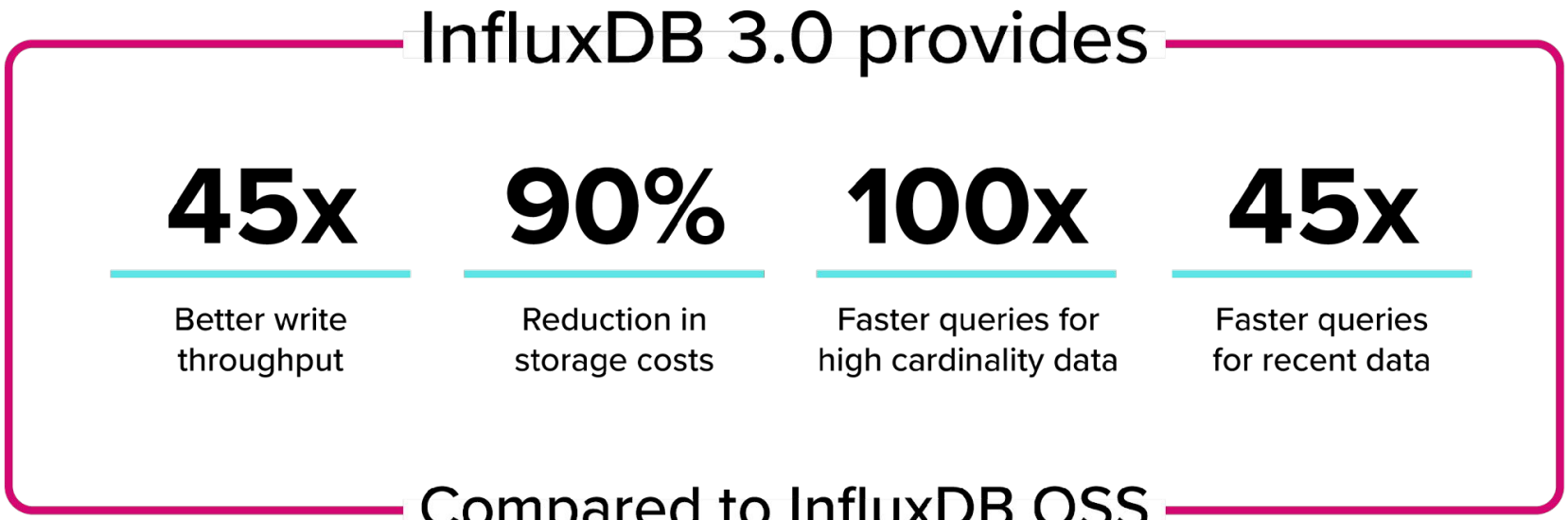
# InfluxDB 3.0 Architecture




 | 

influxdata®

# InfluxDB 3.0: Fast, Real Time & Cost Effective



influxdata®


Non profit governance of open source communities

“Community over Code” - The Apache Way

# Apache: Benefits for InfluxDB 3.0

- ⇒ Predictable Foundation
- **Stable License**: (ASL 20 years old) low risk of changes, (ahem OpenTofu)
- **Communication**: Predictable and open (if slow)
- **Multi-Vendor Participation**: Shared investment reduces individual risk
- **Long Term Maintenance**: Hedged against life changes, corporate strategy shifts,  VC funding cycles
- ⭐⭐⭐⭐⭐: Works far better than could be reasonably expected

Parquet

Columnar file format from 2013 (originally part of Hadoop ecosystem)

https://parquet.apache.org/

# Apache Parquet: Benefits for InfluxDB

⇒ Avoid Engineering overhead of building a custom file format + ecosystem

**Compression:** Works well with wide data variety (including time series)

**Performance:** Techniques such as projection, filter pushdown, late materialization

**Interoperability:** “Defacto” interchange format for analytics, immediate compatibility with 1000s of tools (Apache Iceberg makes it even easier)

# TSM (time series) vs Parquet Compression

| Format | File Count | Size (GB) |
|---|---|---|
| TSM | 483 | 591 |
| TSM (gzip) | 483 | 97 |
| Parquet | 246,140<br>(4K - 2.2GB) | 118 |

**Entire dataset (Parquet/TSM: 20%)**

| Format | File Count | Size (GB) |
|---|---|---|
| TSM | 1 | .110 |
| Parquet | 54<br>(4K - 7M) | .020 |

**File 1 (Parquet/TSM: 18%)**

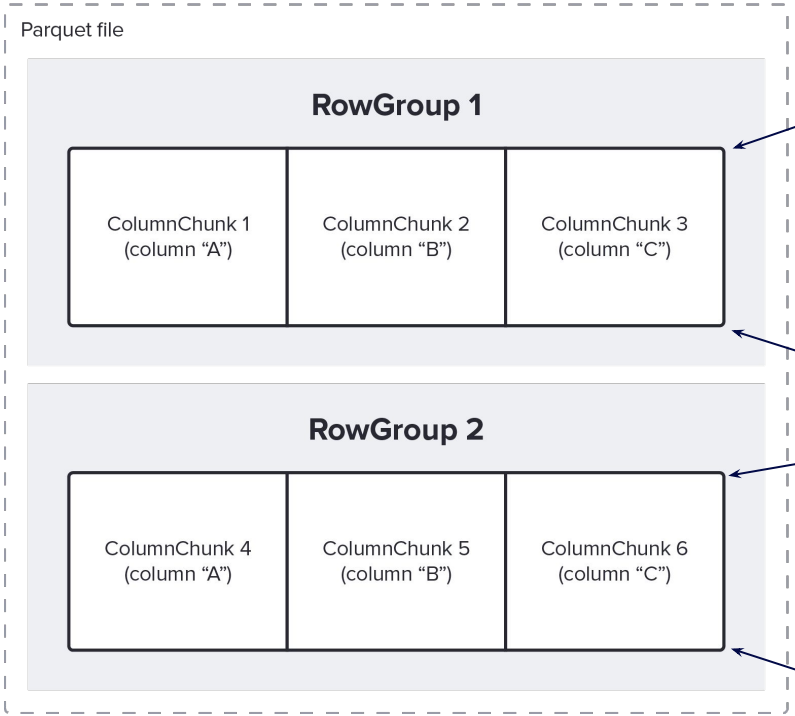
| Format | File Count | Size (GB) |
|---|---|---|
| TSM | 1 | 2 |
| Parquet | 974<br>(12K - 53M) | .226 |

**File 2 (Parquet/TSM 11%)**

influxdata®

# Parquet Organization




| A | B | C |
|---|---|---|
| 15 | Foo | 1/1/2023 |
| ••• | | |
| 11 | Bar | ..1/5/2023 |
| 50 | Baz | 1/1/2023 |
| ••• | | |
| 32 | Blarg | 1/6/2023 |

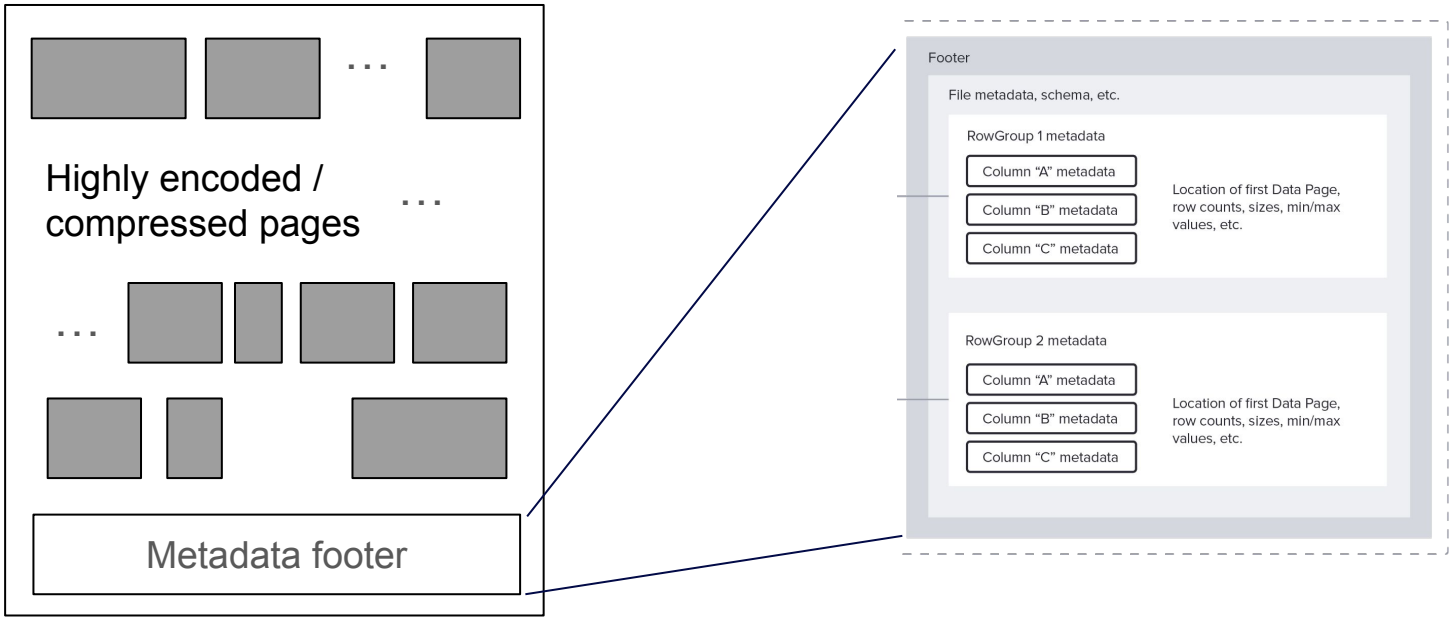
*(“PAX” in DB literature)*

Source: https://www.influxdata.com/blog/querying-parquet-millisecond-latency/

influxdata®

# Parquet Structure + Metadata




*Footer contains location of pages, and statistics such as min/max/count/nullcount.*

*(“Zone Maps”, “Small Materialized Aggregates” in DB literature)*

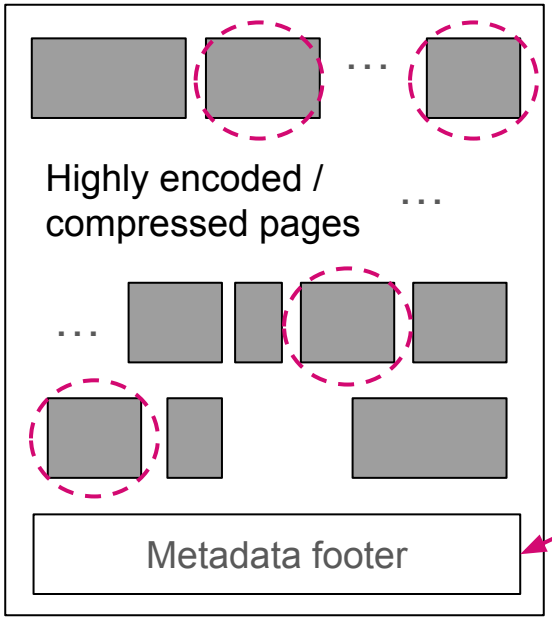
Source:
https://www.influxdata.com/blog/querying-parquet-millisecond-latency/




influxdata®

# Parquet Projection + Filter Pushdown

*2. Only read/decode necessary pages*




*1. Consult metadata*

Metadata + query to prune (skip) pages that aren't needed

```
SELECT A
...
WHERE C > 25
```

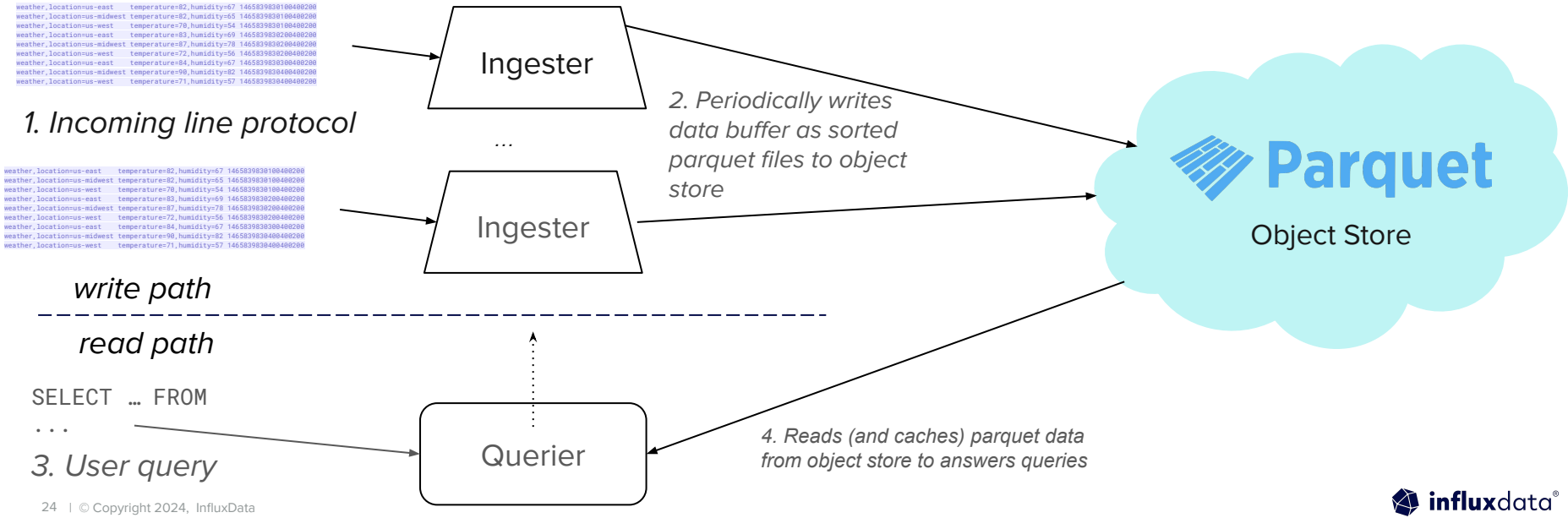
Source:
https://www.influxdata.com/blog/querying-parquet-millisecond-latency/

# How does InfluxDB 3.0 use Parquet?

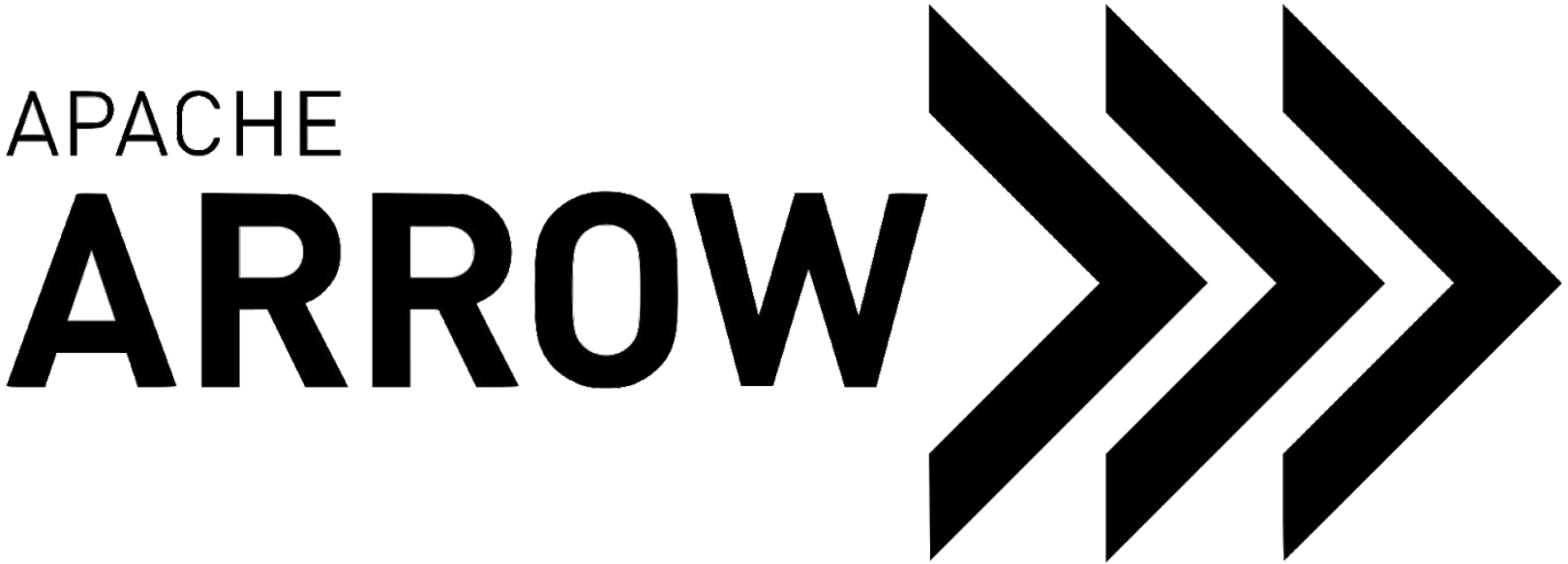
Durable Store: **All** data durably persisted to object store as parquet;

Query: Read from parquet files + latest unpersisted ingester data



influxdata®




*In memory format for fast vectorized processing*

https://arrow.apache.org/

# Arrow: Benefits for InfluxDB

⇒ Best practice for storing columnar data in memory

**Type System:** Full type support (integers, strings, timestamps, etc)

**Null Support:** Standard null bitmask representation + semantics

**Efficient Encodings:** Dictionary encoding for Strings

**Natural Integration:** DataFusion, parquet libraries, Arrow Flight

**Optimized Compute Kernels:** Fast vectorized kernels are well understood, but time consuming to implement, test and maintain

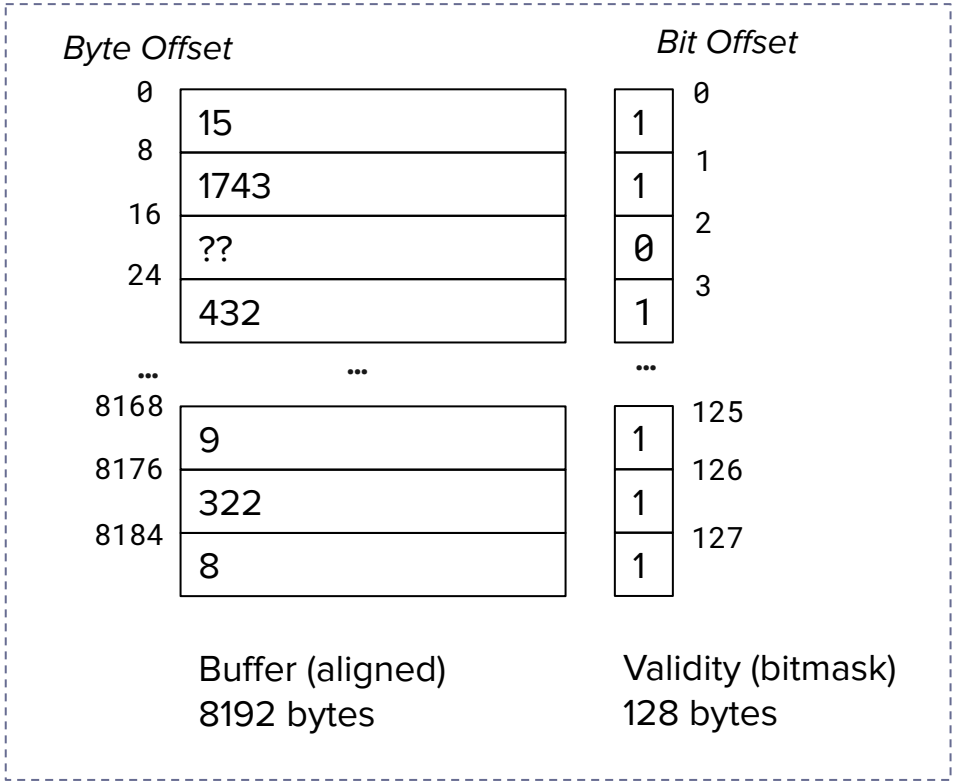
*Note: Arrow has (many) more features than this*

# Arrow Array: Int64Array

| |
|---|
| 15 |
| 1743 |
| NULL |
| 432 |
| .. |
| 9 |
| 322 |
| 8 |

**Logically 1024 8 byte integers**



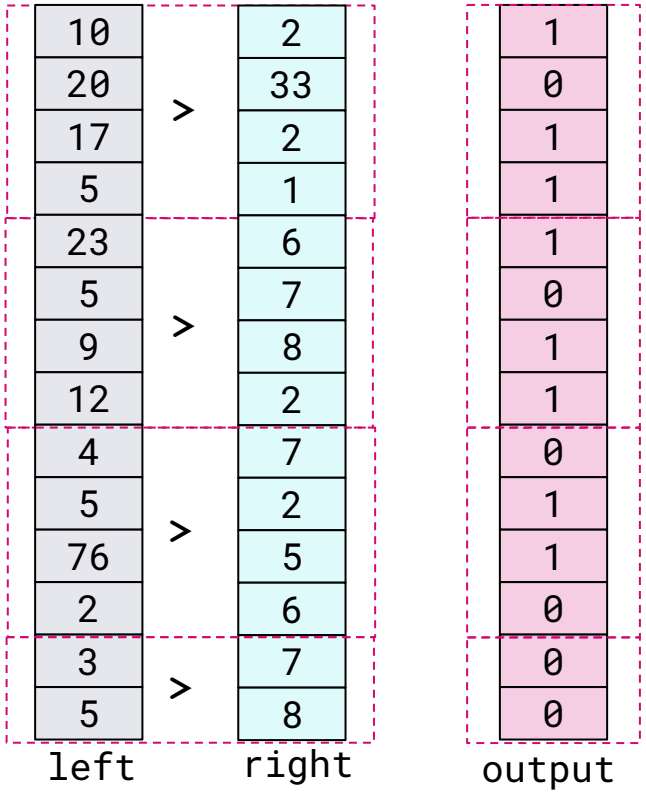

**Arrow Array**

Pretty much what you will find in every vectorized column store engine

influxdata®

# Compute Kernels





```
let output = gt(
  &left,
  &right
);
```

*The `gt` (greater than) kernel computes an output `BooleanArray` where each element is `left > right`*

*Kernels handle nulls (validity masks), optimizations for different data sizes, etc.*

*~50 different kernels, full list: [docs.rs page]*

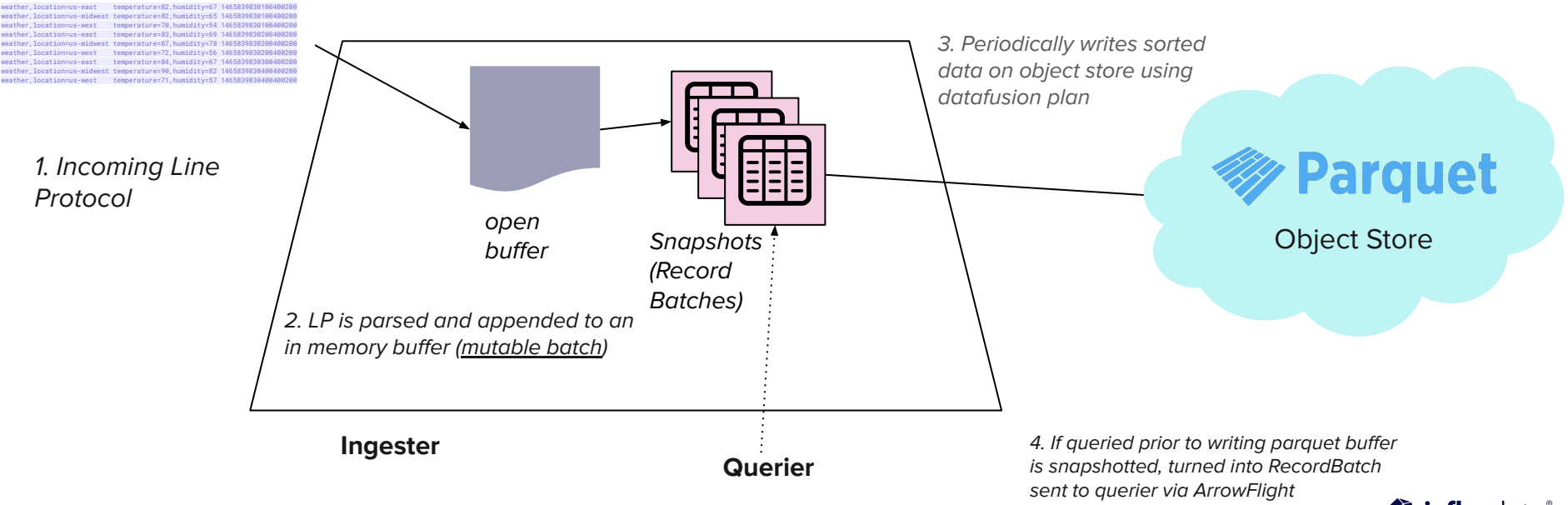



influxdata®

# How does InfluxDB 3.0 use Arrow: Ingester Buffers

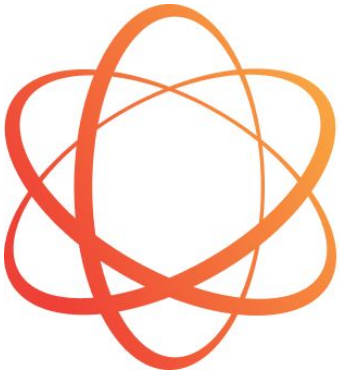
**Indirectly:** via DataFusion and Flight.

**Directly:** Ingest path, which parses input line protocol ➔ Arrow Arrays



influxdata®



# APACHE DATAFUSION™

*Highly customizable, fast query engine that uses Arrow*

https://github.com/apache/arrow-datafusion

# Arrow DataFusion: Benefits for InfluxDB

⇒ Fast full featured, extensible query engine

**All the OLAP buzzwords:** vectorized, columnar, multi-core, streaming, out of core, …

**“Full” SQL:** JOINs, date/time/timestamp functions, structured data, …

**Customizable:** Easily extend time series specific functions (e.g. date_bin gapfill, InfluxQL, …)

# DataFusion: Input / Output

```
SELECT status, COUNT(1)
FROM http_api_requests_total
WHERE path = '/api/v2/write'
GROUP BY status;
```

***SQL Query***

```
ctx.read_table("http"
)?
  .filter(...)?
  .aggregate(..)?;
```

***DataFrame***

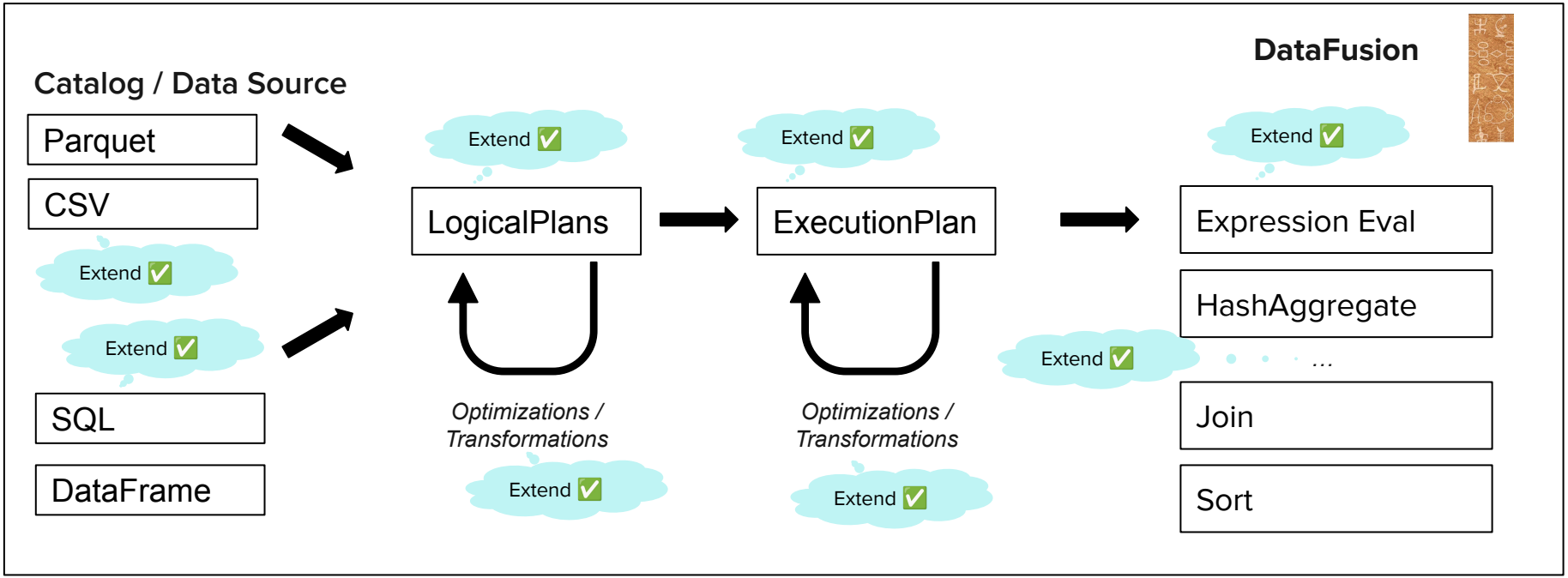
*Catalog information: tables, schemas, etc*

***Data Batches***


***Data Batches***

# DataFusion: Totally Customizable Architecture




Query FrontEnds

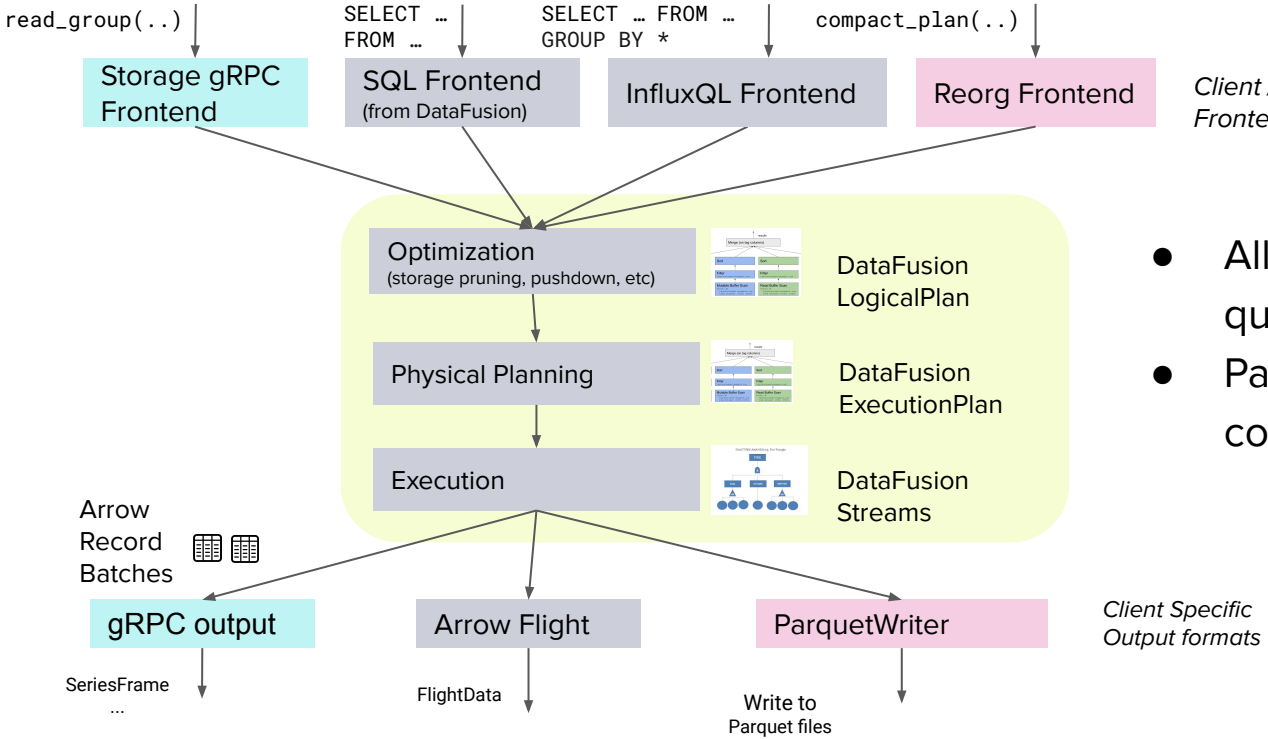
Plan Representations (DataFlow Graphs)

**Optimized Execution Operators (Arrow Based)**

influxdata®

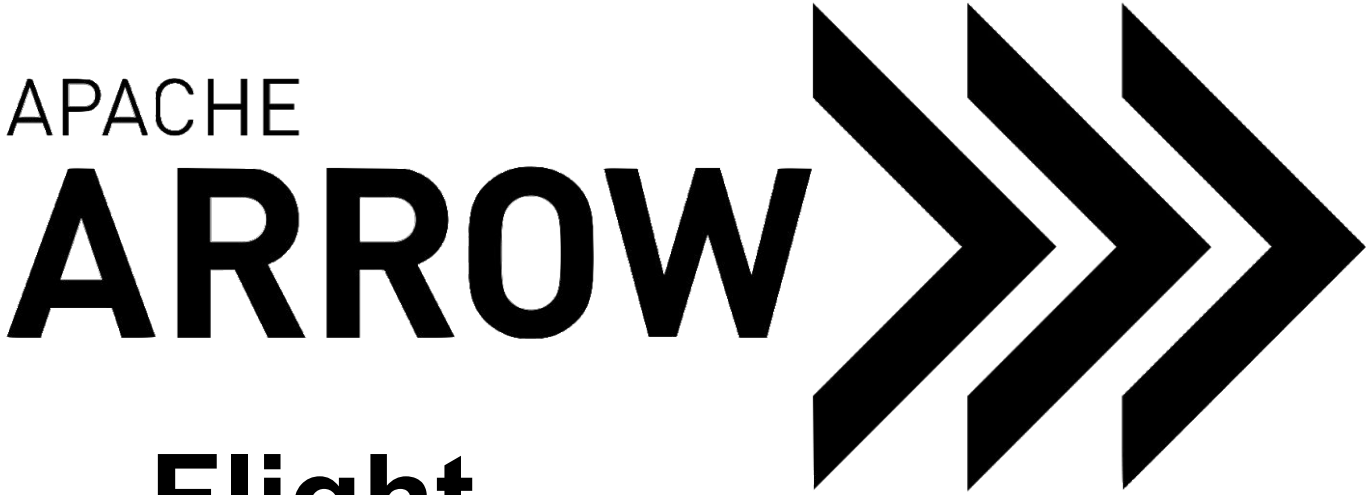
# How InfluxDB 3.0 uses DataFusion




- All queries (3 different query languages)
- Parquet creation + compaction

influxdata®




*Efficiently send columnar data (as Arrow Arrays) over the network*

https://arrow.apache.org/docs/format/Flight.html

# Arrow Flight: Benefits for InfluxDB

⇒ Efficient network transfer; wide client support

**Network Efficient:** columnar format, high bandwidth transfer

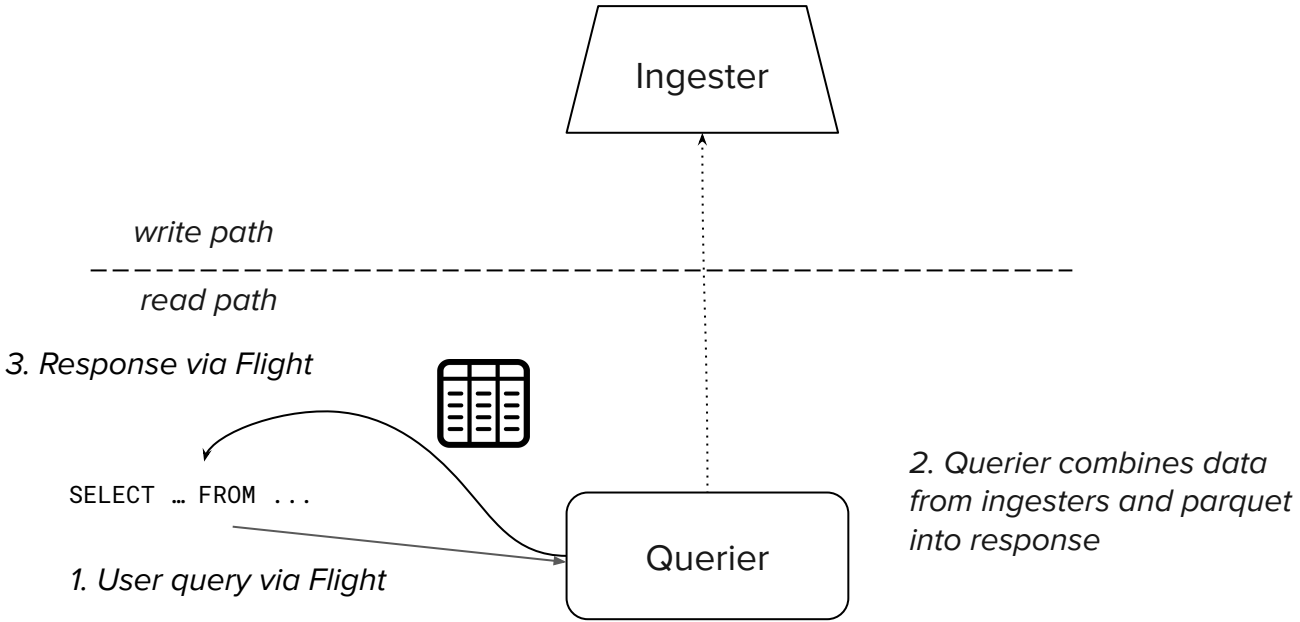
**CPU Efficient:** “zero copy” Serialization / Deserialization

**Pre Existing Clients:** Minimal effort to make InfluxDB 3.0 clients (just use Arrow Flight clients)

# How does InfluxDB 3.0 use Flight? Native Query Protocol

1. Client ↔ Querier

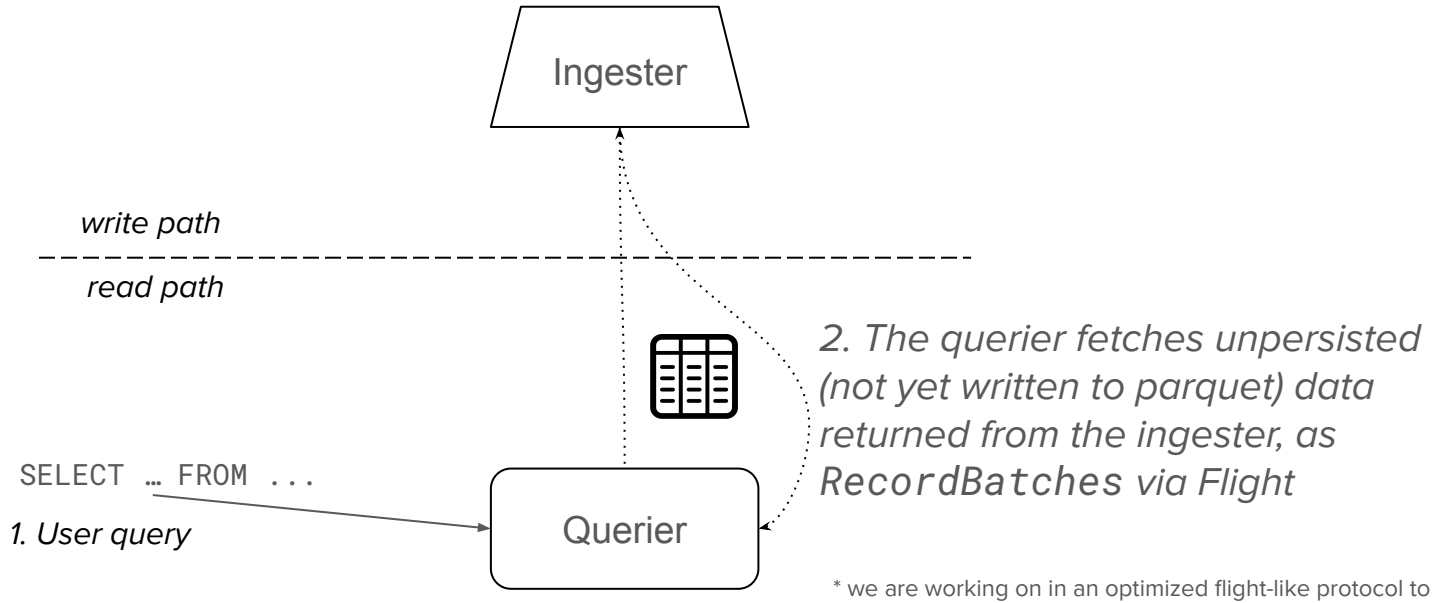


influxdata®

# How does InfluxDB 3.0 use Flight? Ingester ↔ Querier

1. Ingester and Querier's internal protocol is built on Flight*




* we are working on in an optimized flight-like protocol to reduce latency

influxdata®


*Execute SQL queries and return results with a standard (non custom) client*

https://arrow.apache.org/docs/format/FlightSql.html

# Arrow FlightSQL: Benefits for InfluxDB

⇒ Access to SQL ecosystem, without implementing our own drivers/connectors

**Prepackaged Client Libraries:** JDBC/ODBC/Ecosystem drivers

**Pre-packaged Integrations:** Many systems already read from FlightSQL

**Community Leverage:** E.g. FlightSQL Grafana plugin, others contribute

# Apache Arrow FlightSQL

- Send SQL queries, receive Responses as Arrow Arrays
- Has clients in many languages / APIs (JDBC, python DB API, etc)

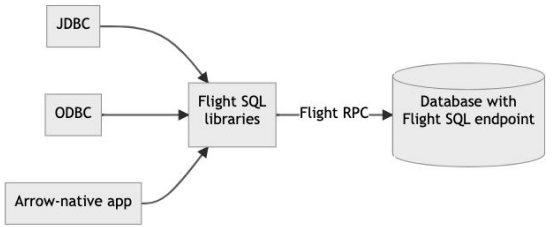
## Introducing Apache Arrow Flight SQL: Accelerating Database Access

PUBLISHED 16 Feb 2022
BY José Almeida, James Duong, Vinicius Fraga, Juscelino Junior, David Li, Kyle Porter, Rafael Telles

We would like to introduce Flight SQL, a new client-server protocol developed by the Apache Arrow community for interacting with SQL databases that makes use of the Arrow in-memory columnar format and the Flight RPC framework.

Flight SQL aims to provide broadly similar functionality to existing APIs like JDBC and ODBC, including executing queries; creating prepared statements; and fetching metadata about the supported SQL dialect, available types, defined tables, and so on. By building on Apache Arrow, however, Flight SQL makes it easy for clients to talk to Arrow-native databases without converting data. And by using Flight, it provides an efficient implementation of a wire format that supports features like encryption and authentication out of the box, while allowing for further optimizations like parallel data access.

While it can be directly used for database access, it is not a direct replacement for JDBC/ODBC. Instead, Flight SQL serves as a concrete wire protocol/driver implementation that can support a JDBC/ODBC driver and reduces implementation burden on databases.



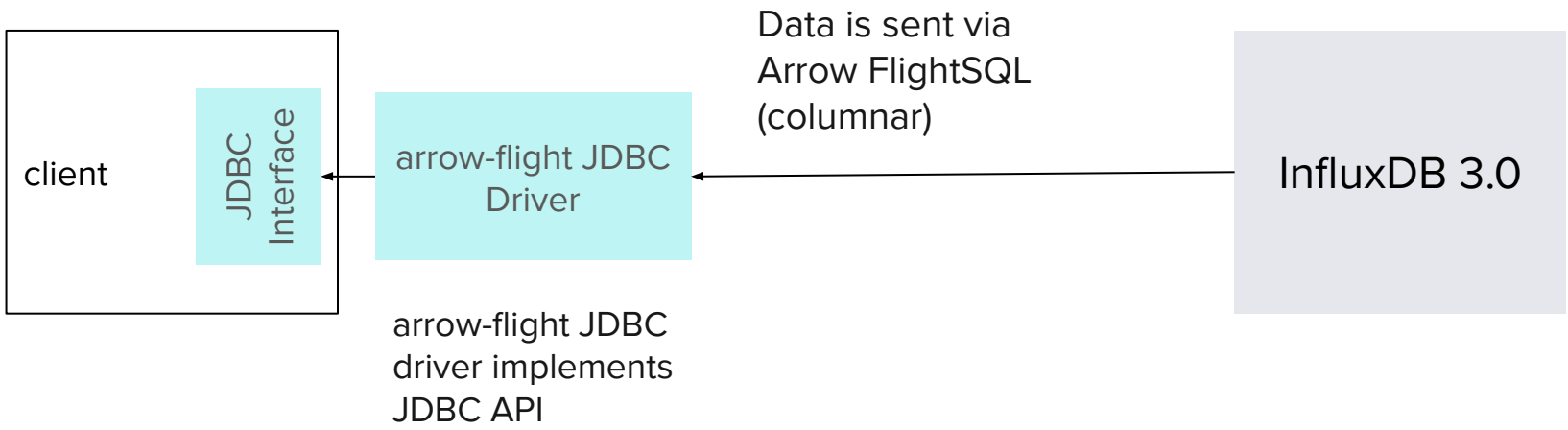

Read More:Expanding Arrow's Reach with a JDBC Driver for Arrow Flight SQL

https://arrow.apache.org/blog/2022/02/16/introducing-arrow-flight-sql/

influxdata®

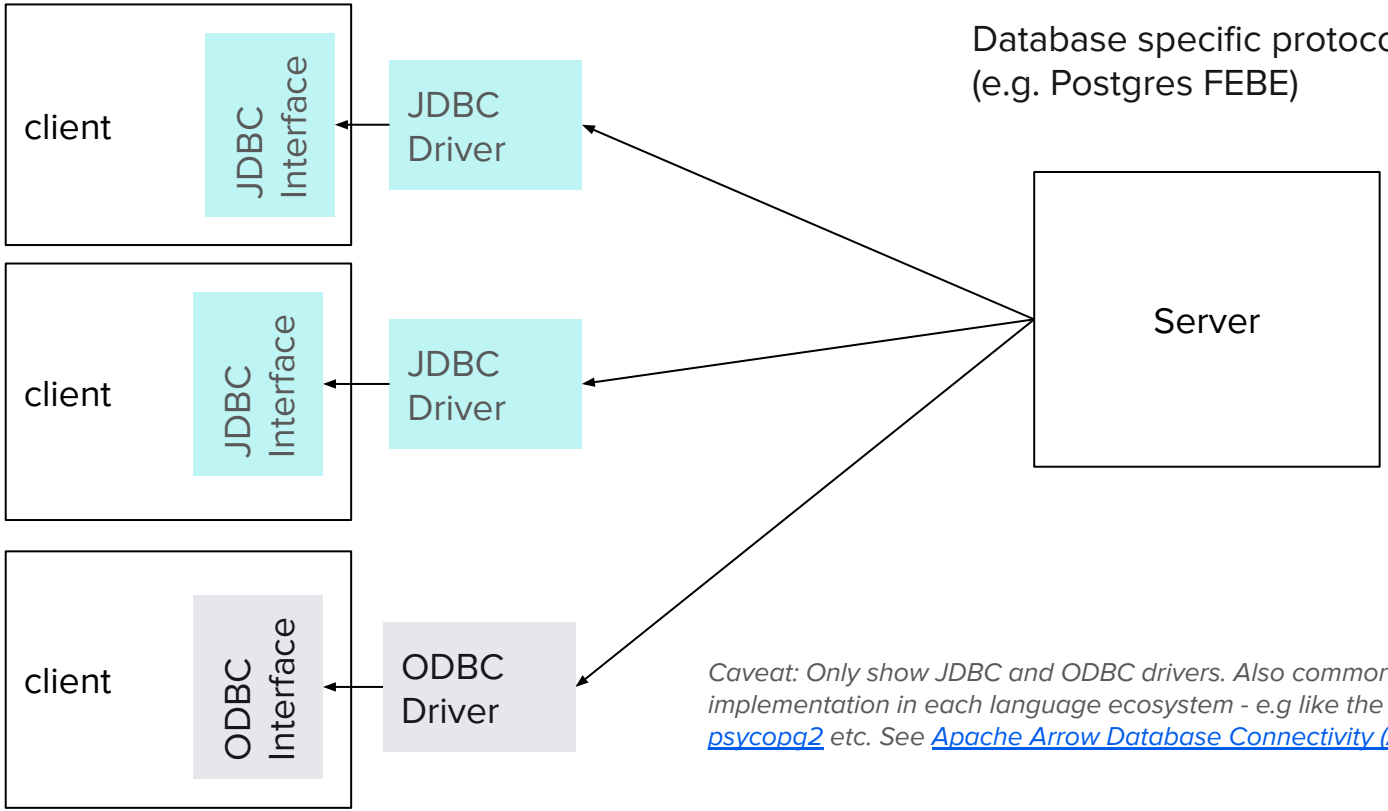
# How InfluxDB 3.0 uses FlightSQL




✅ Access to the ecosystem without having to implement JDBC, ODBC, … 😅

# Before Flight SQL



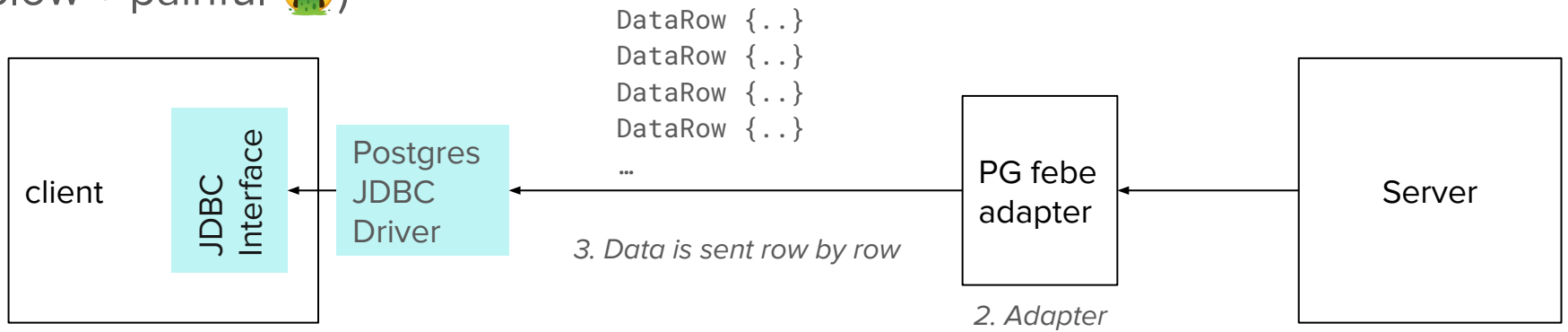

*Caveat: Only show JDBC and ODBC drivers. Also common is a native implementation in each language ecosystem - e.g like the Python DB API 20 and psycopg2 etc. See Apache Arrow Database Connectivity (ADBC) for more details*




influxdata®

# Alternate Strategy: Use Posgres FEBE

(Slow + painful 🤮)




```
Client: Prepare query:"SELECT * FROM foo WHERE a = $1;" name:""
Client: Bind name:"" parameters:[{format:"text", value:"42"}]
… (steps elided) …
Server: RowDescription fields:[{name:"a", type:"int", format:"text"}, …]
Server: DataRow fields:[{data:"42"}, {data: "Hunter Valley"} …]
Server: DataRow fields:[{data:"12"}, {data: "Merrimack Valley"} …]
… (lots ros for each data)
Server: DataRow fields:[{data:"321"}, {data: "Charles River Watershed"} …]
… (steps elided) …
```

It also turns out clients using the postgres driver tend to try and query the postgres metadata tables 😱




influxdata®

# Conclusion

# Conclusion

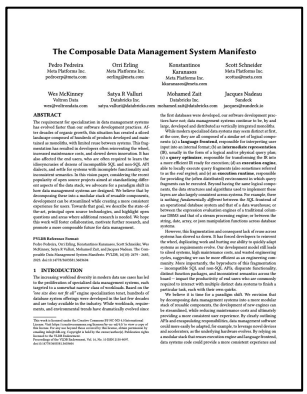
- Building Databases from scratch is hard (and expensive 💸)
- You don’t have to anymore 🎉
- We built InfluxDB 3.0 using Apache **F**light, **D**ataFusion, **A**rrow, and **P**arquet,  and it was awesome
- ⭐⭐⭐⭐⭐: Highly recommended for your next projects

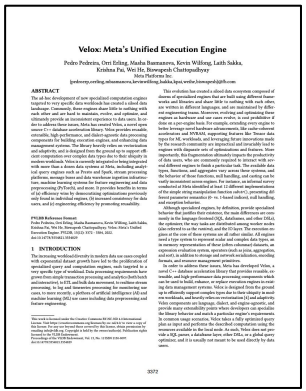
# Related Work


The Modern Data Architecture
The Deconstructed Database

The Modern Data Architecture: The Deconstructed Database (USENIX ;login: Winter 2018)


The Composable Data Management System Manifesto

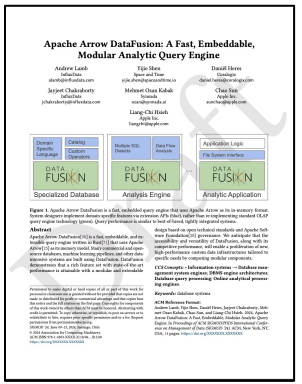
The Composable Data Management System Manifesto (VLDB 2023)


Velox: Meta's Unified Execution Engine

Velox: Meta's Unified Execution Engine (VLDB 2022)


A Deep Dive into Common Open Formats for Analytical DBMSs

A Deep Dive into Common Open Formats for Analytical DBMSs (VLDB 2023)

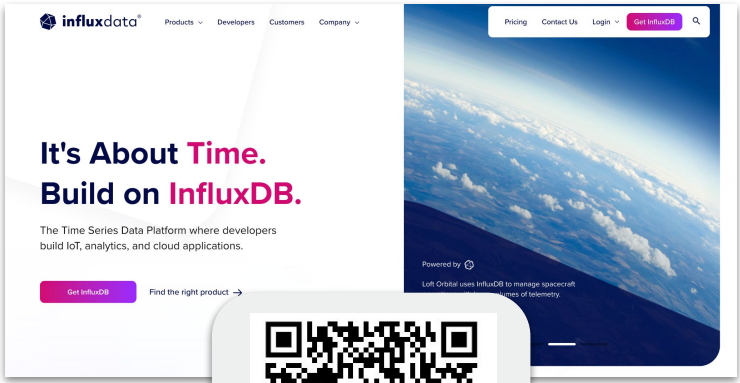

Apache Arrow DataFusion: A Fast, Embeddable, Modular Analytic Query Engine

Apache Arrow DataFusion: A Fast, Embeddable, Modular Analytic Query Engine (To Appear SIGMOD 2024)

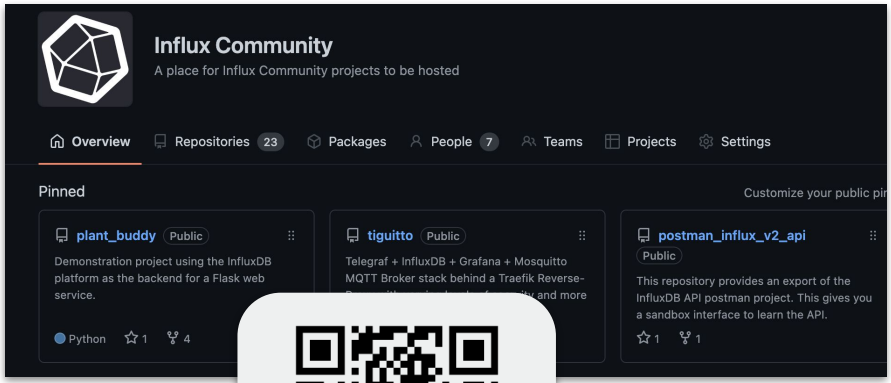


influxdata®

# Try It Yourself




https://www.influxdata.com




https://github.com/InfluxCommunity

 | 

influxdata®

influxdata®

THANK YOU

# Backup Slides

# Thank you!

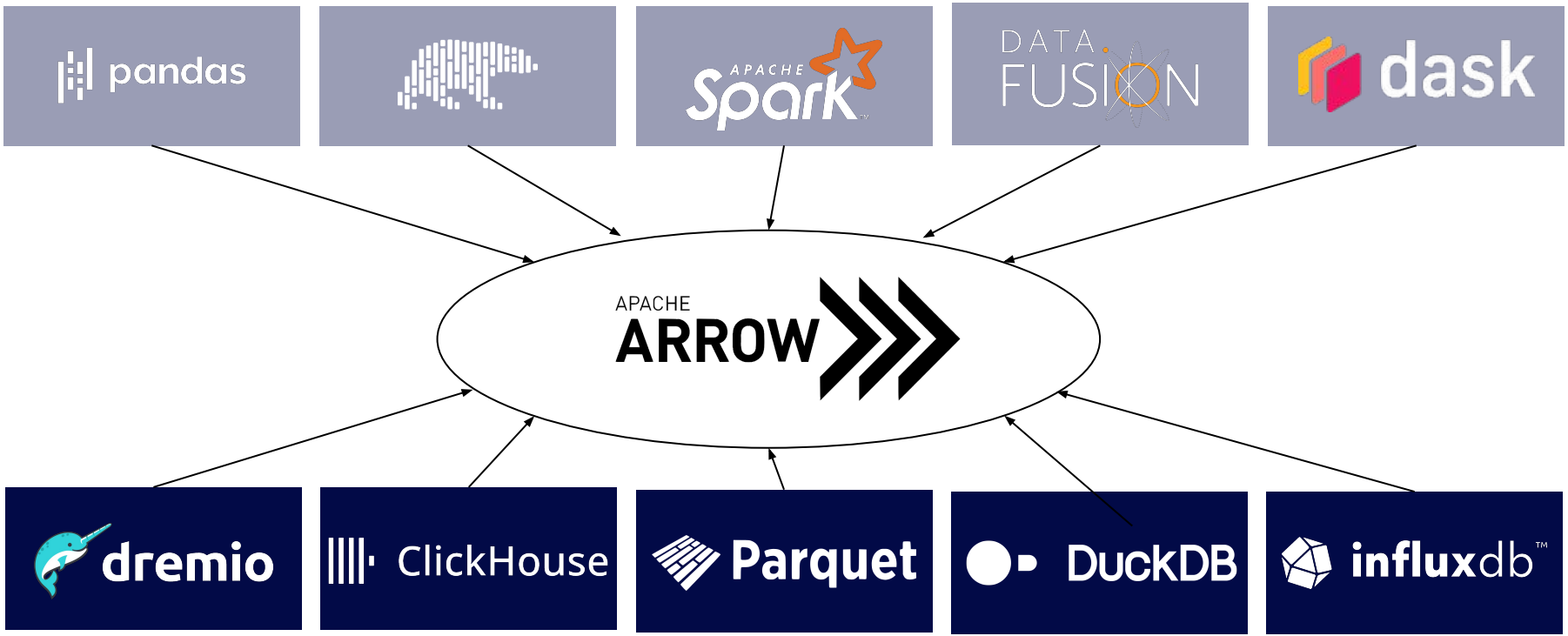
Questions / Discussion

Find out more:

- Flight, DataFusion, Arrow, and Parquet: Using the FDAP Architecture to build InfluxDB 3.0
- Apache Arrow: https://arrow.apache.org/
- Apache DataFusion: https://arrow.apache.org/datafusion/
- Apache Parquet: https://parquet.apache.org/

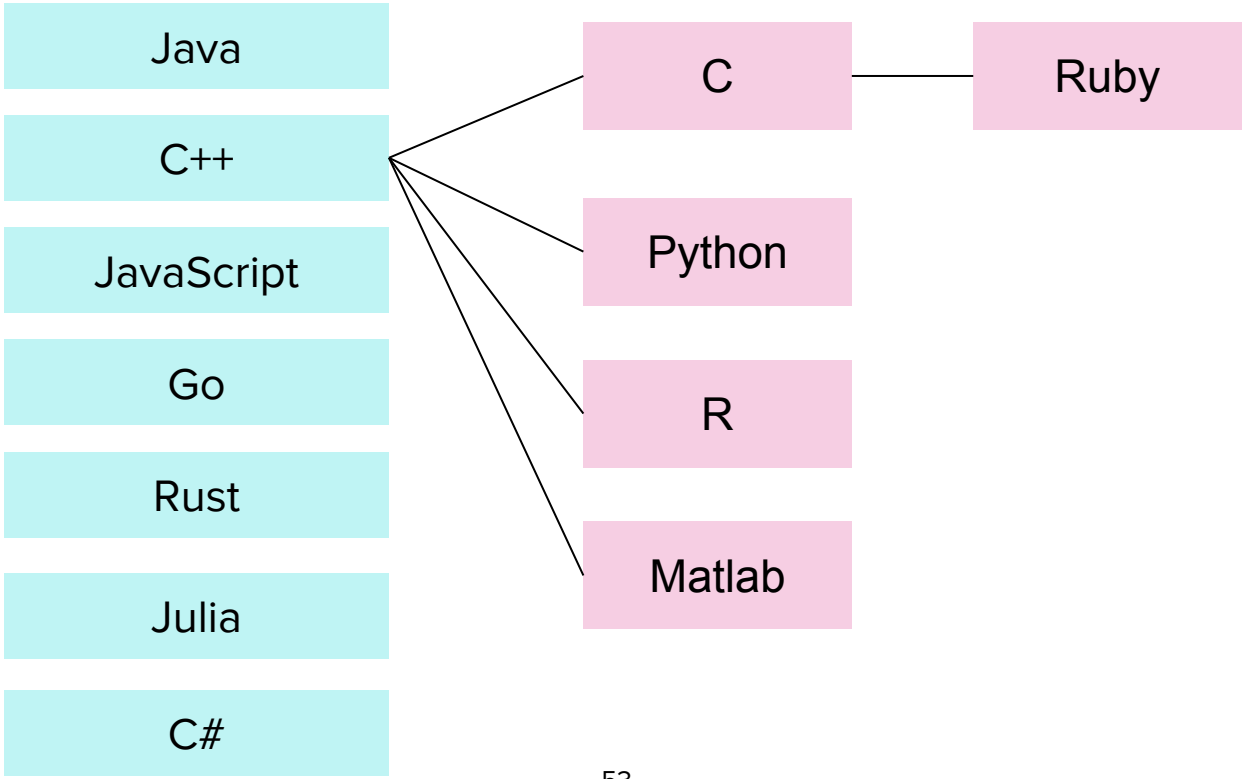
 

# Defragmenting Data Access Across Systems

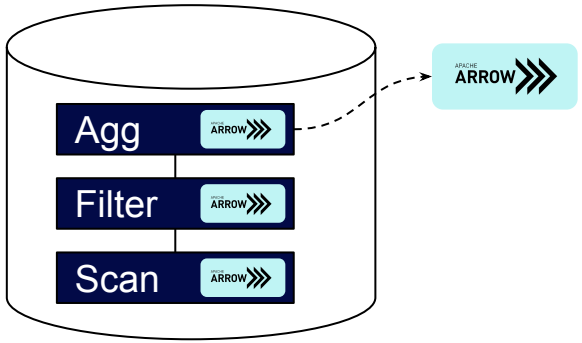


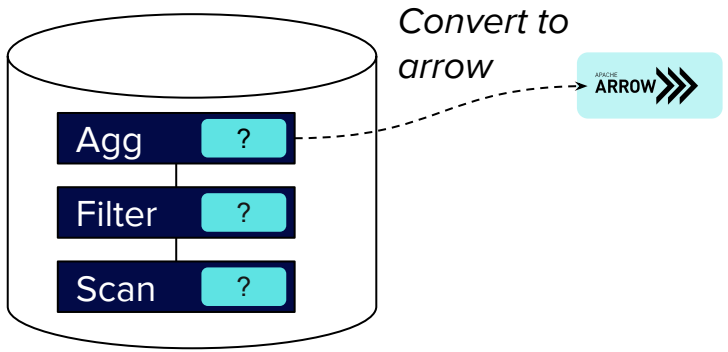

 

# Integration: Arrow Language Implementations



# Why Arrow Internally (and not just at interface)?




**Option 1:** Use Arrow Internally (DataFusion, pola.rs, Acero)

**Pros**: Fast interchange, reuse Arrow libraries

**Cons**: Constrained(*) to Arrow

**Option 2:** Use specialized structures internally, convert to Arrow at edges (Velox, DuckDB)

**Pros**: Can use specialized structures

**Cons**: Maintain specialized code

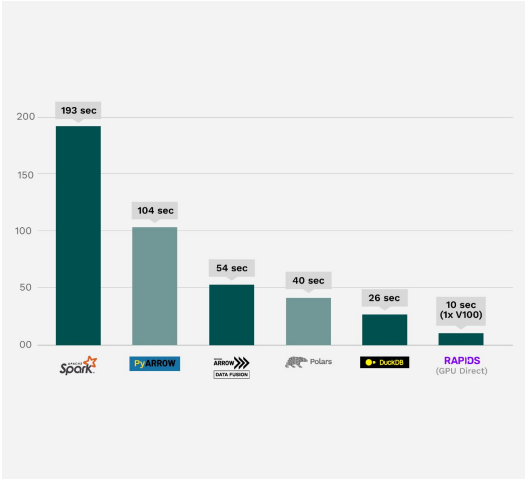
influxdata®

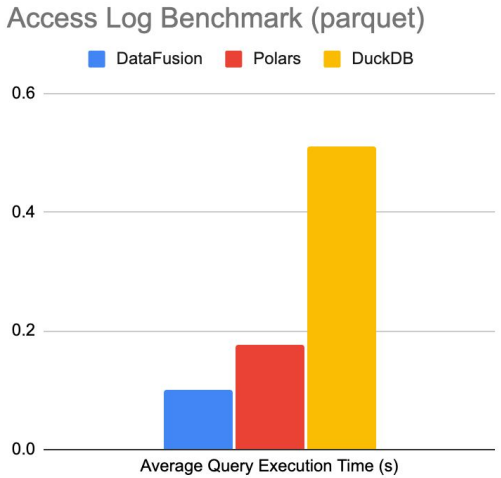
# Why Arrow Internally (and not just at interface)?

**Theory:** Using Arrow is "good enough" compared to specialized structures

Pooled open source development ➔ invest heavily in optimized Arrow kernels

So far results are encouraging




https://github.com/tustvold/access-log-bench

**Good:** Sorting, Filtering, Projection, Parquet

**Could Improve:** Grouping, Joining

https://voltrondata.com/resources/speeds-and-feeds-hardware-and-software-matter




influxdata®