Serverless Data pipelines in Rust

Rust Vienna Meetup

January 2024

Background

- Michele Vigilante
- Data Engineer *(Radancy)*
- Previous work experience
  - Writing automation software in C/C++
  - Backend web dev in Java/Scala
  - Data engineering with Scala/Rust
- I like coffee ☕, video games 🎮 and cats 😺

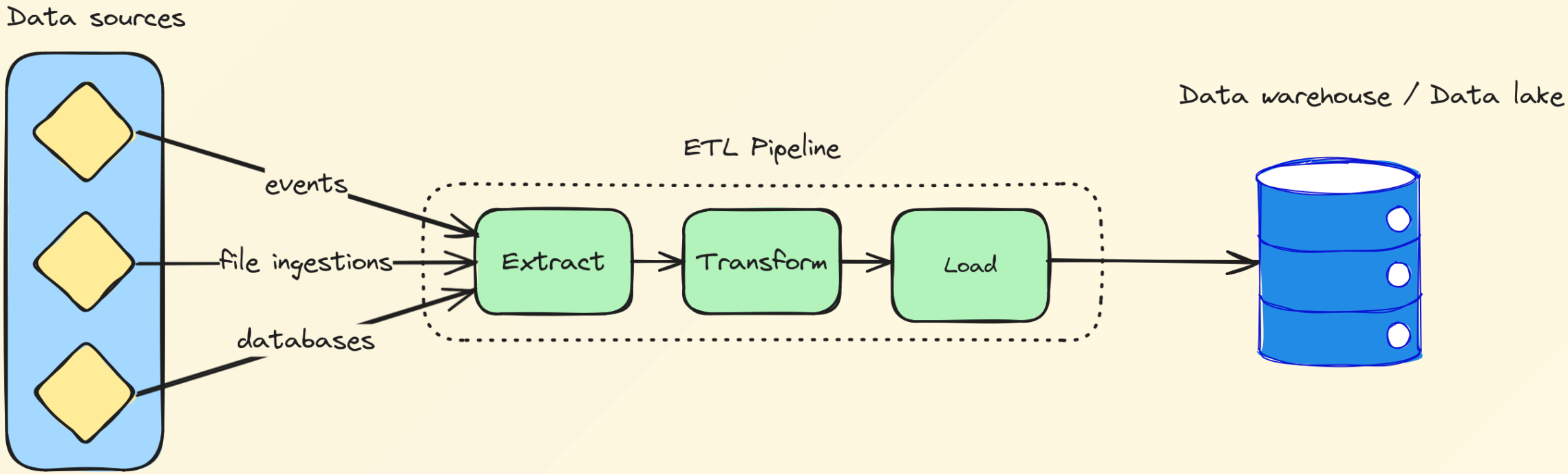
michelevig@protonmail.com

What is a data pipeline?

> A data pipeline is a set of data processing steps that transfer data from one system to another, transforming and consolidating it along the way. It's crucial in the era of big data for efficient data collection, storage, and analysis.

- ChatGPT

Data Engineering in a nutshell



Programming Languages

SQL
python
Java
Scala

Clouds

aws
Azure

Big Data Frameworks

hadoop
Flink
Apache Spark
Apache Storm
Hive

ETL Orchestration

Apache nifi
AWS Data Pipeline
Azure Data Factory
Apache Airflow

Data Warehouses

amazon Redshift
snowflake
Azure Synapse Analytics
Google Big Query

DBs

MySQL
PostgreSQL
Oracle Database
mongoDB

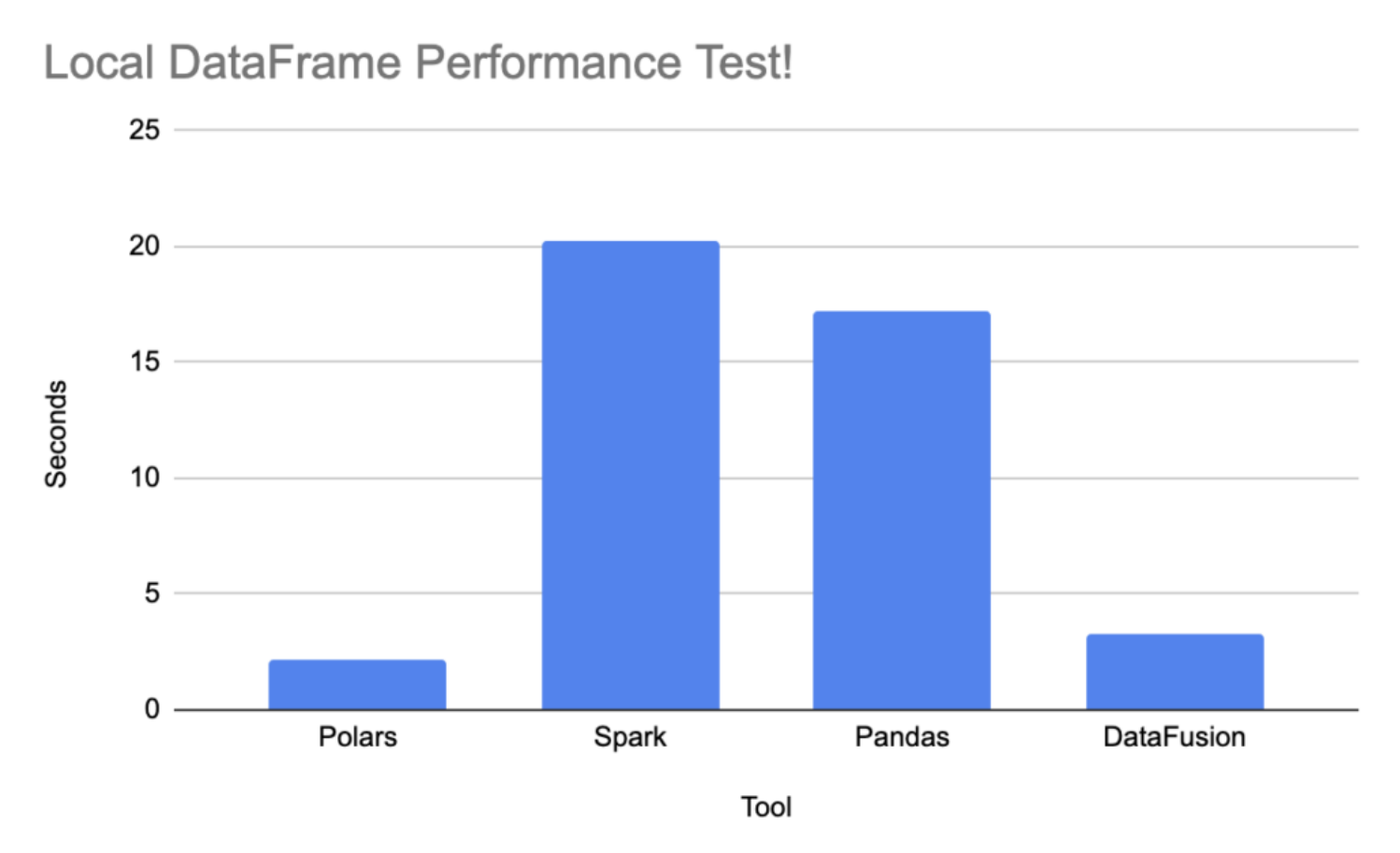
Why Rust?

- Good fit due to high performance & reliability
- Safety guarantees & error handling
- Amazing tooling & eco system
- Low cost of operation
- I want to 🙈

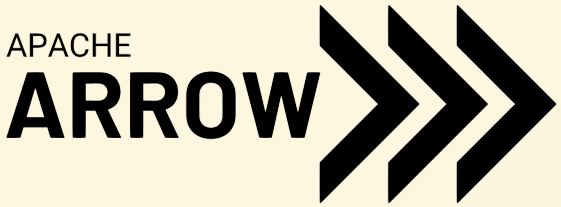
- Query engine / Data processing framework
- Authored by Andy Grove
  - Apache Arrow PMC chair
  - Currently at NVIDIA working on querying with GPUs
- Donated to the Apache Arrow project

Benchmarks

Dataset size: 5,485,922 records (~207 MB)



source: https://www.confessionsofadataguy.com/dataframe-showdown-polars-vs-spark-vs-pandas-vs-datafusion-guess-who-wins/



- Released in 2016
- Columnar data in-memory format
- Supports complex types like structs, maps, lists, etc.
- Arrow IPC Protocol
  - Enables interop with other arrow implementations

Object store

- Provides API for interacting with Object Stores using the `ObjectStore` trait
- Natively implemented
  - AWS S3
  - Google cloud storage
  - Azure Blob storage
  - Local file system
  - ...

DBMS vs Query engine

- Storage ← *Object store*
- Catalog
- Query Engine ← *Datafusion*
- User management & permissions
- Clustering
- etc.

When should I use datafusion?

- Medium data
  - too big for Excel
  - too small for Spark (big data)
- Slow moving data
  - for analytical workloads
  - "non realtime"

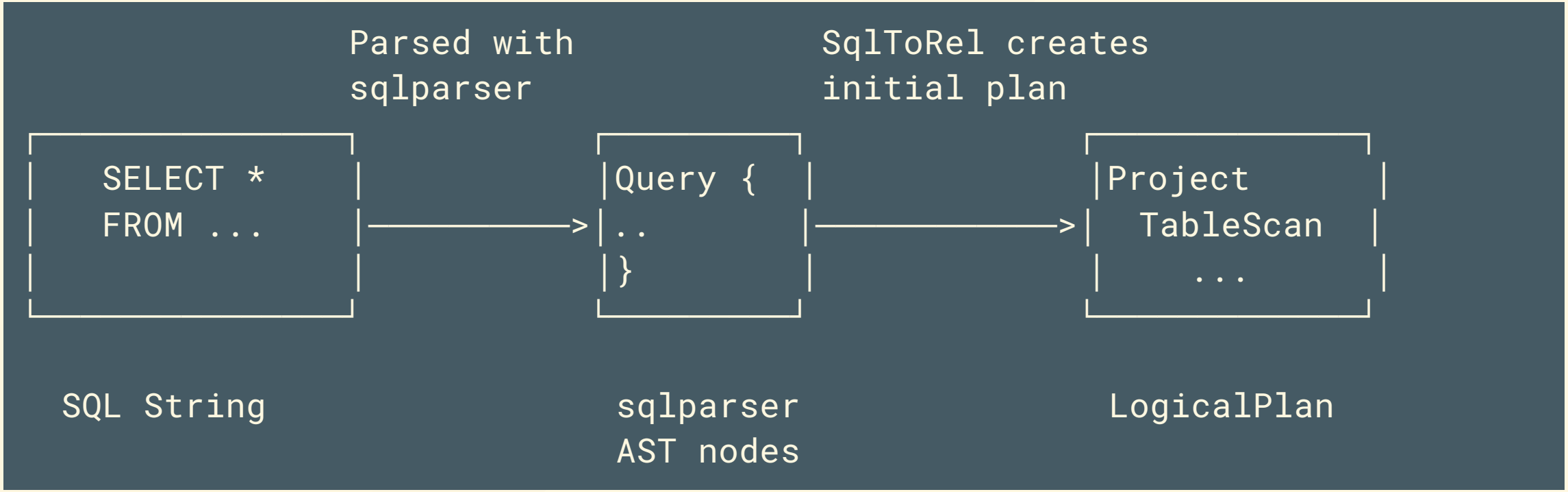
DataFrame example

```
let ctx = SessionContext::new();

// create the dataframe
let df = ctx.read_csv("tests/data/example.csv", CsvReadOptions::new()).await?;

// create a plan
let df = df.filter(col("a").lt_eq(col("b")))?
           .aggregate(vec![col("a")], vec![min(col("b"))])?
           .limit(0, Some(100))?;

// execute the plan
let results: Vec<RecordBatch> = df.collect().await?;
```

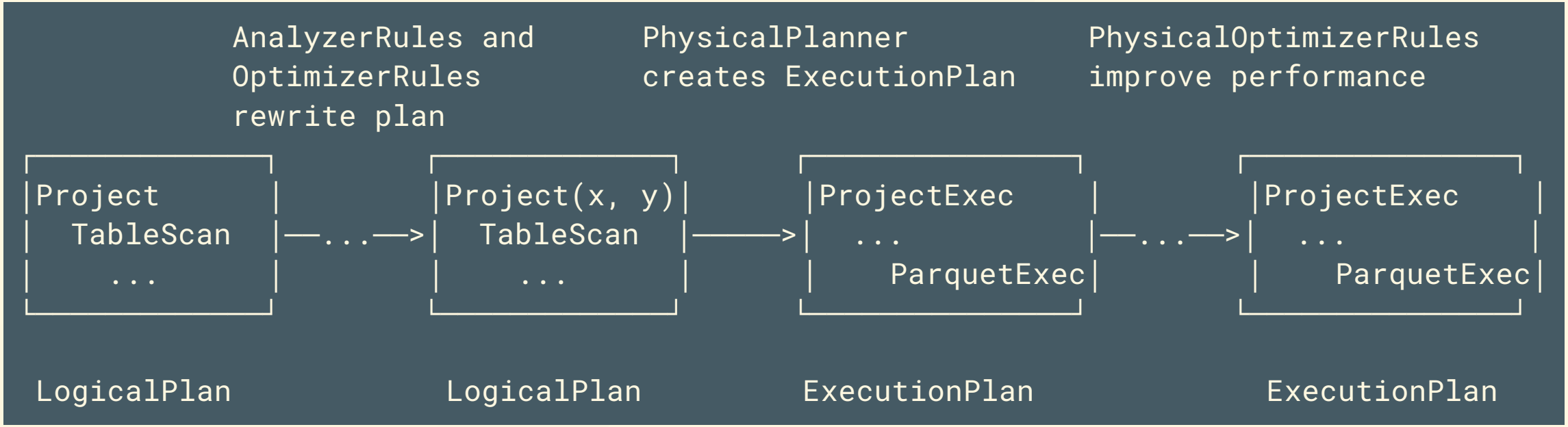
SQL Example

```
let ctx = SessionContext::new();

ctx.register_csv("example", "tests/data/example.csv", CsvReadOptions::new()).await?;

// create a plan
let df = ctx.sql("SELECT a, MIN(b) FROM example WHERE a <= b GROUP BY a LIMIT 100").await?;

// execute the plan
let results: Vec<RecordBatch> = df.collect().await?;
```

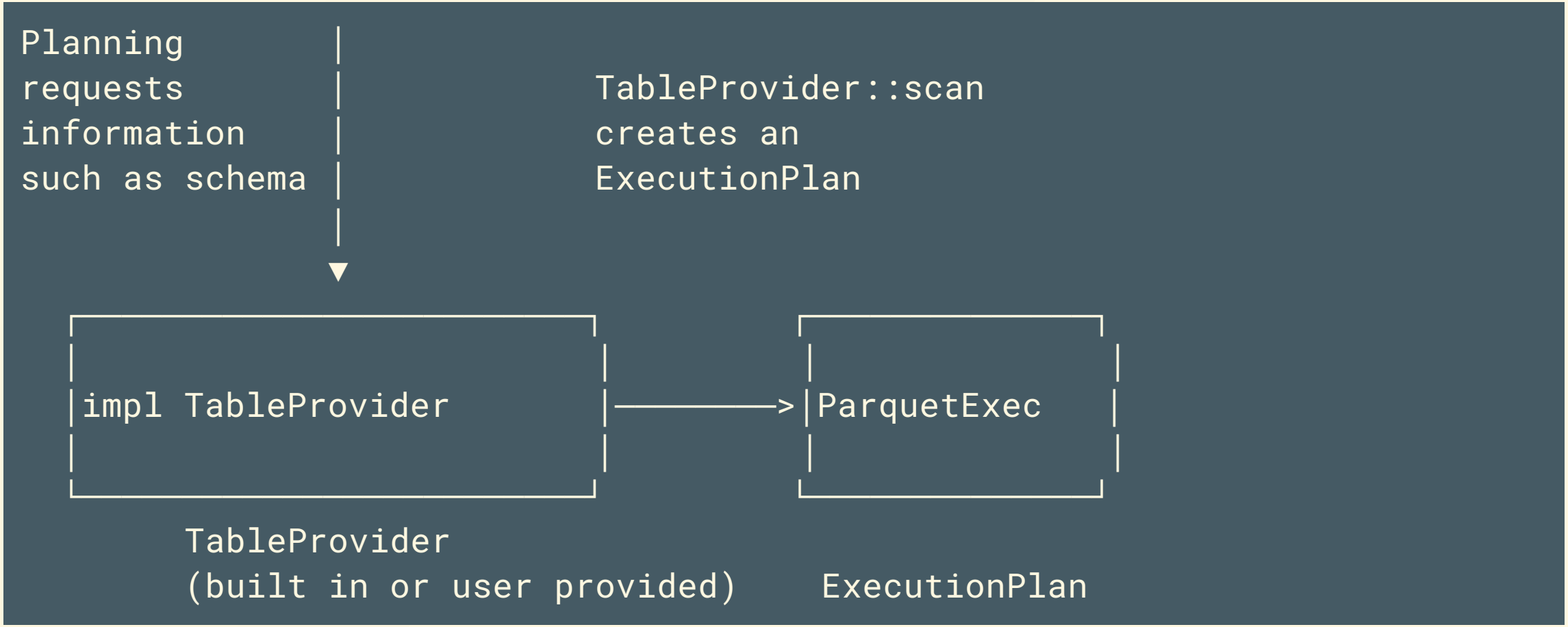
Datafusion SQL



Datafusion Query Plan



Data Sources



Let's get practical!

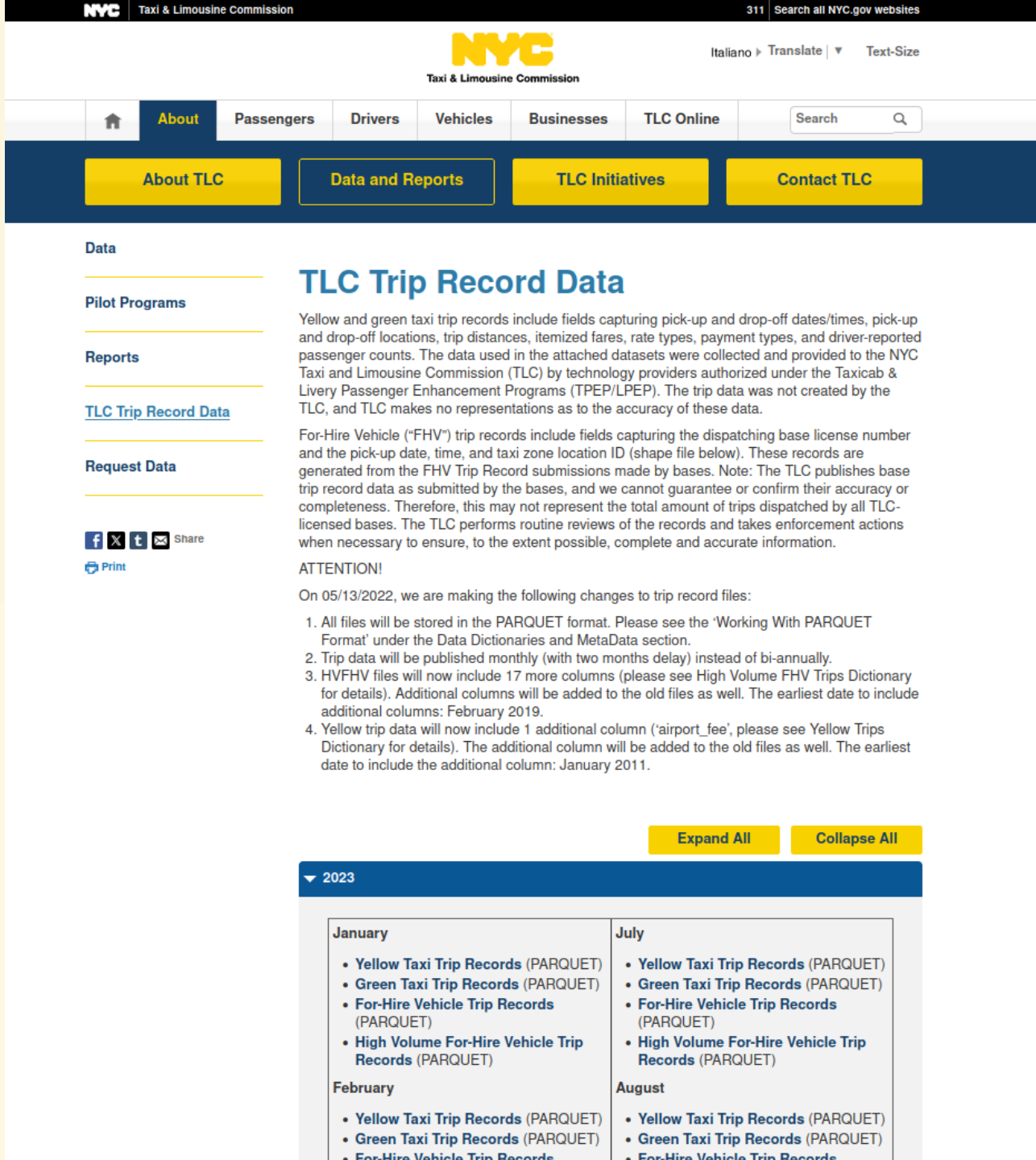
Story time

You're working for a company that analyzes city traffic. You're a data engineer and you receive a request to help the analytics team to look into NYC Taxi trip record data.

- Serve the analytics team with the following data
  - Average trip duration, average trip cost, etc.
  - Location based data tracking number of pickups and dropoffs
- Process all data from 2014 to 2023
  - Aggregate by month

Task #1: Get the data

- Data hosted on nyc.gov site
- Stored in parquet
- Predictable URL format (hosted on couldfront)



Easy...

```
# Define the input file containing the URLs
input_file = 'nyc_taxi_data/download_urls.txt'
pattern = r'yellow_tripdata_(\d{4})-(\d{2})\.parquet'

# Read the URLs from the input file
with open(input_file, 'r') as f:
    urls = f.readlines()

# Iterate through each URL and download the corresponding Parquet file
for url in urls:
    url = url.strip()
    try:
        response = requests.get(url, stream=True)
        if response.status_code == 200:
          # Store the file for further processing
          ...
```

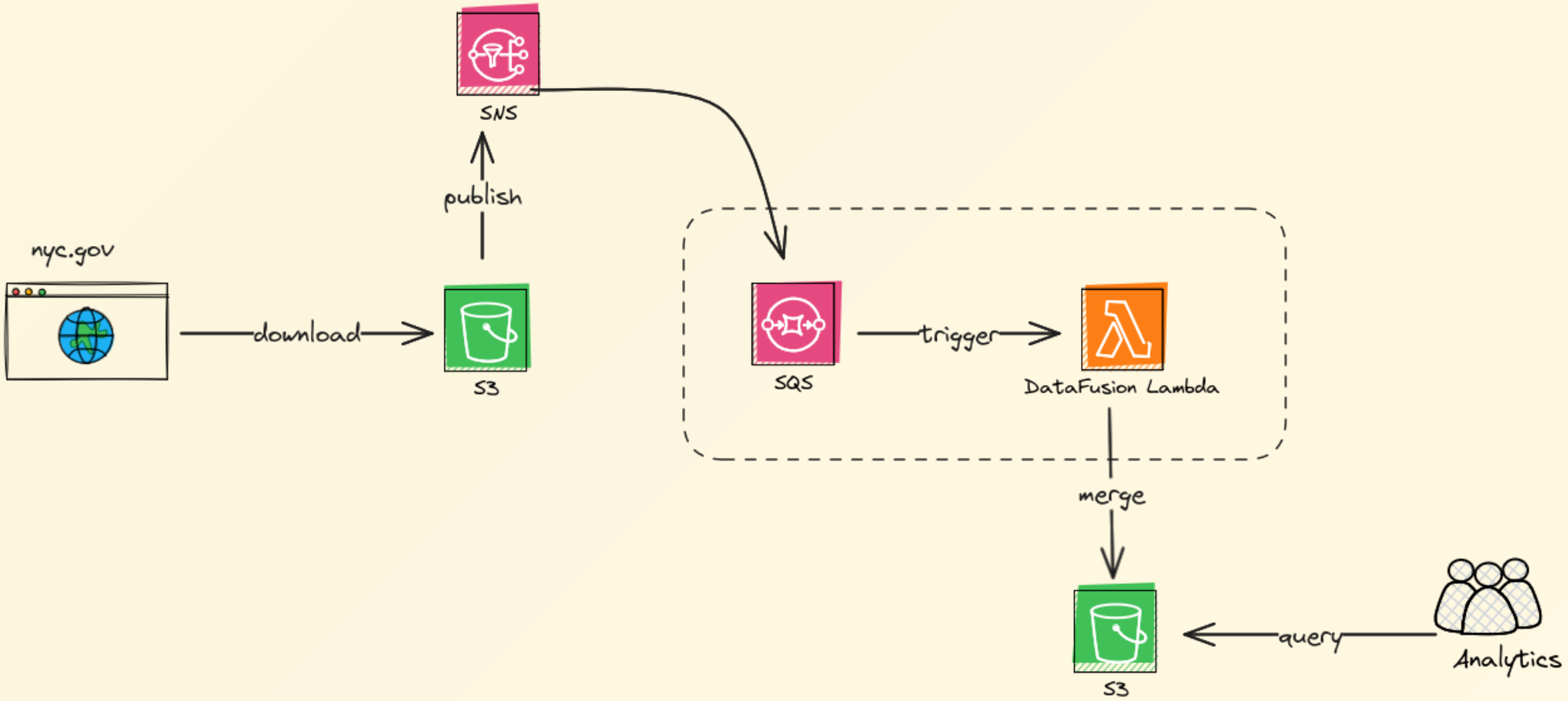
But...

- Dataset is quite large (10GB) → need to preaggregate
- Data needs to be interpreted → translate into our domain

Task #2: Build a pipeline

- You know rust
- Limited resources
- Needs to be reliable
- Compatible with analytics tooling

Architecture



Extract...

```
let input_date = NaiveDate::from_ymd_opt(2022, 01, 01).unwrap();
let input_path = format!(
    "./nyc_taxi_data/raw_data/{year}/yellow_tripdata_{year}-{month}.parquet",
    year = input_date.format("%Y"),
    month = input_date.format("%m")
);

// Load data frame
let ctx = SessionContext::new();

let df = ctx
    .read_parquet(input_path, ParquetReadOptions::default())
    .await?;
```

Transform - select data...

```
let df = df
    .select(vec![
        col("tpep_pickup_datetime"),
        col("tpep_dropoff_datetime"),
        col(r#""PULocationID""#).alias("pickup_location_id"),
        col(r#""DOLocationID""#).alias("dropoff_location_id"),
        col("trip_distance"),
        col("passenger_count").alias("passenger_count"),
        col("total_amount"),
        col("tip_amount"),
    ])?
    ...
    .with_column(
        "trip_duration_minutes",
            col("tpep_dropoff_datetime") - col("tpep_pickup_datetime"),
    )?;
```

Transform - aggregate data...

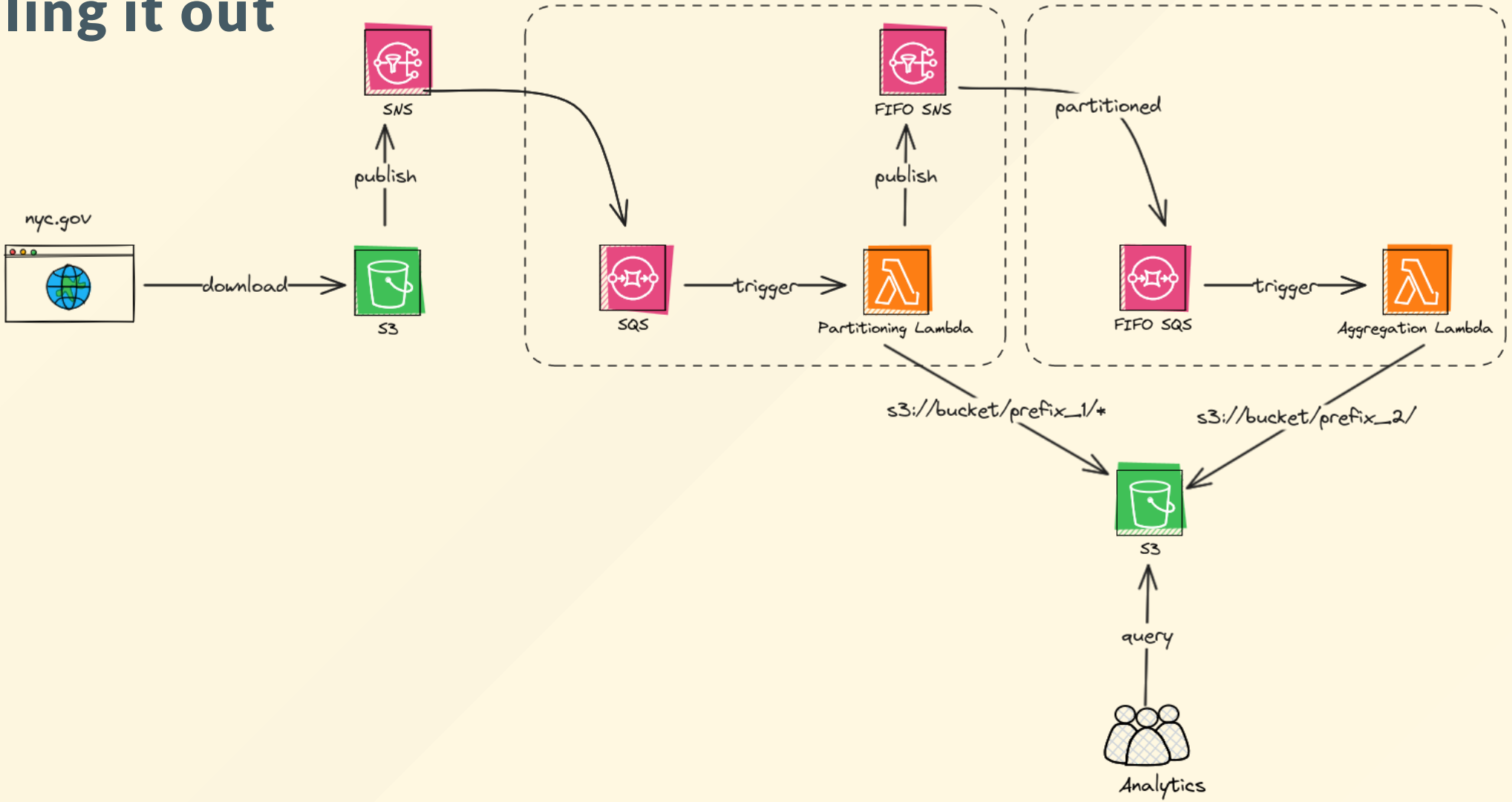
```
let trip_stats = df
    .aggregate(
        vec![col("year"), col("month_start")],
        vec![
            count(lit(1_i32)).alias("trip_count"),
            avg(col("passenger_count")).alias("avg_passengers"),
            avg(col("trip_distance")).alias("avg_trip_distance_miles"),
            avg(col("trip_duration_minutes")).alias("avg_trip_duration_minutes"),
            avg(col("total_amount")).alias("avg_amount"),
            avg(col("tip_amount")).alias("avg_tip"),
        ],
    )?
    .sort(vec![col("month_start").sort(true, false)])?
    .cache()
    .await?;
```

Load...

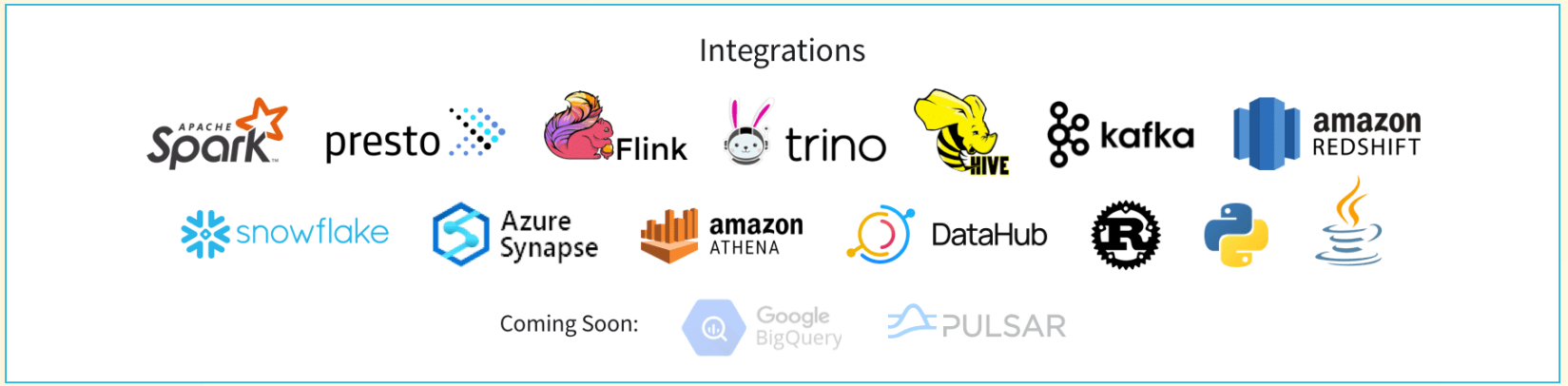
```
trip_stats
    .write_parquet("./trip_stats/", DataFrameWriteOptions::default(), None)
    .await?;
```


WE DID IT!
WOOHOO

Scaling it out



- Part of linux foundation since 2019
- Effectively makes file storage systems ACID compliant
- Rust implementation is fairly mature
- Datafusion integration!



Links

- Andrew Lamb - DataFusion and Arrow
- Jorge C Leitao - From bits to Data Frames
- ClickBench - https://benchmark.clickhouse.com/