# Ten years of building open source standards

From Parquet to Arrow to OpenLineage

**Julien Le Dem: Chief Architect at Astronomer**
**@J_**

# Agenda

## Chapters

## The birth of Parquet

## 15 years ago!

- First committership on Apache Pig
- Kept contributing
- User → Contributor → Committer → PMC member → PMC chair
- 2010: Read the Dremel paper

## Context for the inception of Parquet

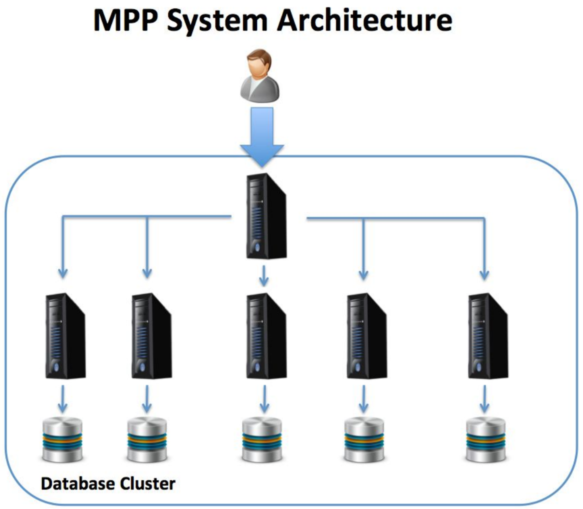
- Hadoop
  - Can store lots of data
  - Can process a lot of data
  - High latency
  - Cheap

- Vertica
  - Interactive queries
  - Not as scalable
  - Expensive

## Can we make **Hadoop** more like **Vertica**?

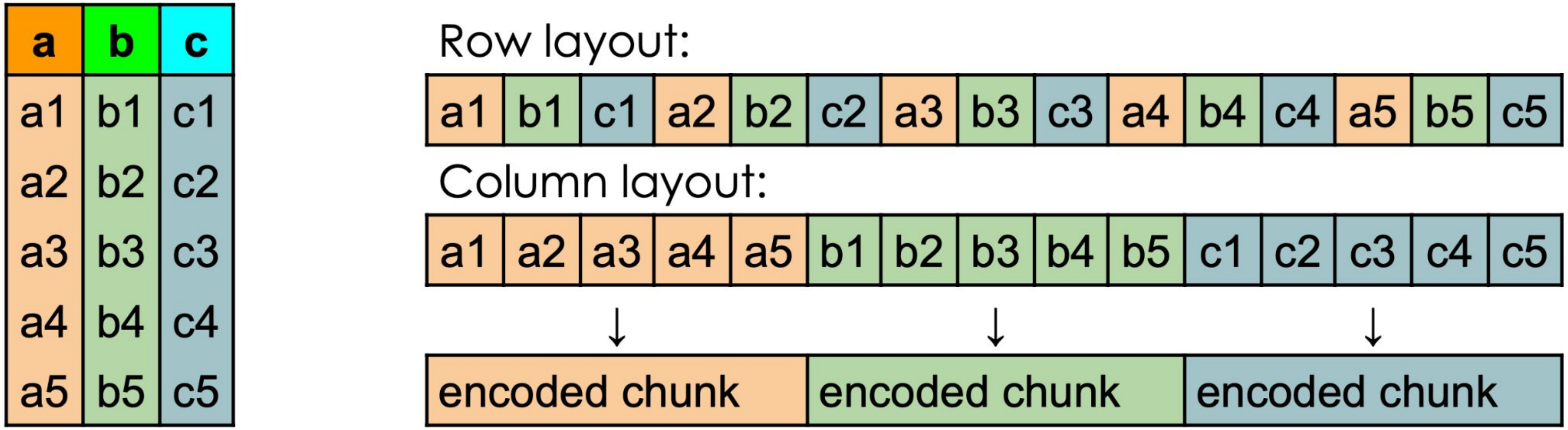
| hadoop | VERTICA |
|---|---|
| Map Reduce | Distributed Query engine |
| Distributed File System | Columnar Storage |

# Paper reading

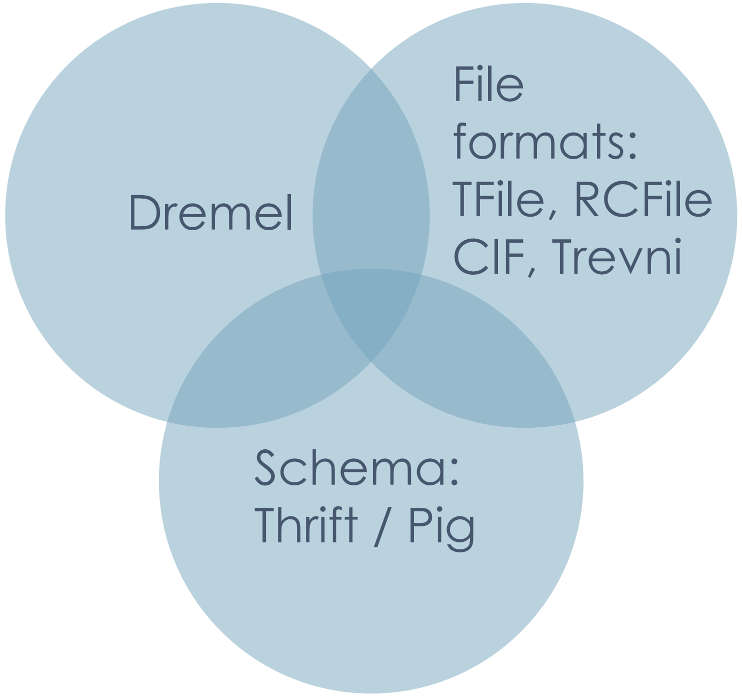


- Vertica / C-Store
- Dremel
- MonetDB

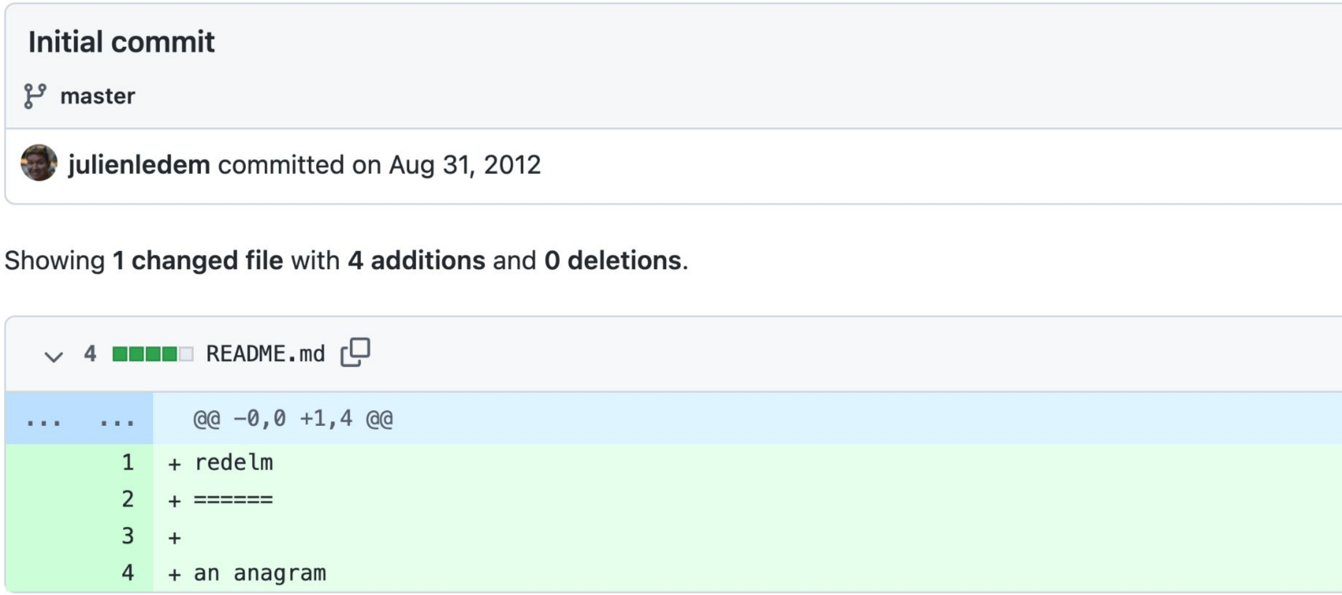
# Columnar layout



## Red Elm


Dremel
File formats: TFile, RCFile CIF, Trevni
Schema: Thrift / Pig

# First commit



**Initial commit**

master

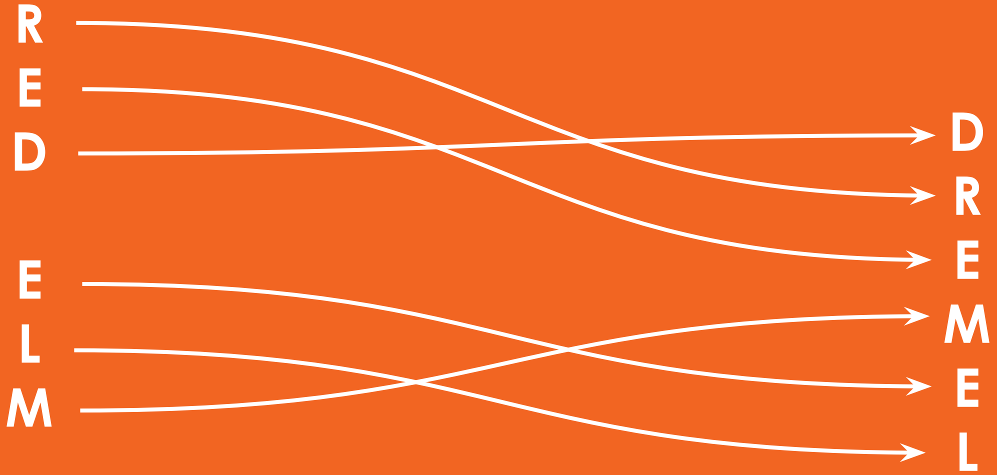
**julienledem** committed on Aug 31, 2012

Showing **1 changed file** with **4 additions** and **0 deletions**.

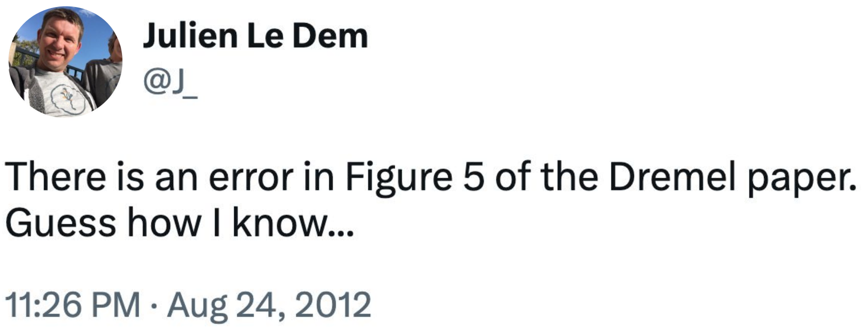
4 README.md

```
...  ...   @@ -0,0 +1,4 @@
        1  + redelm
        2  + ======
        3  +
        4  + an anagram
```

## That was quite ambitious



## Seeking partners

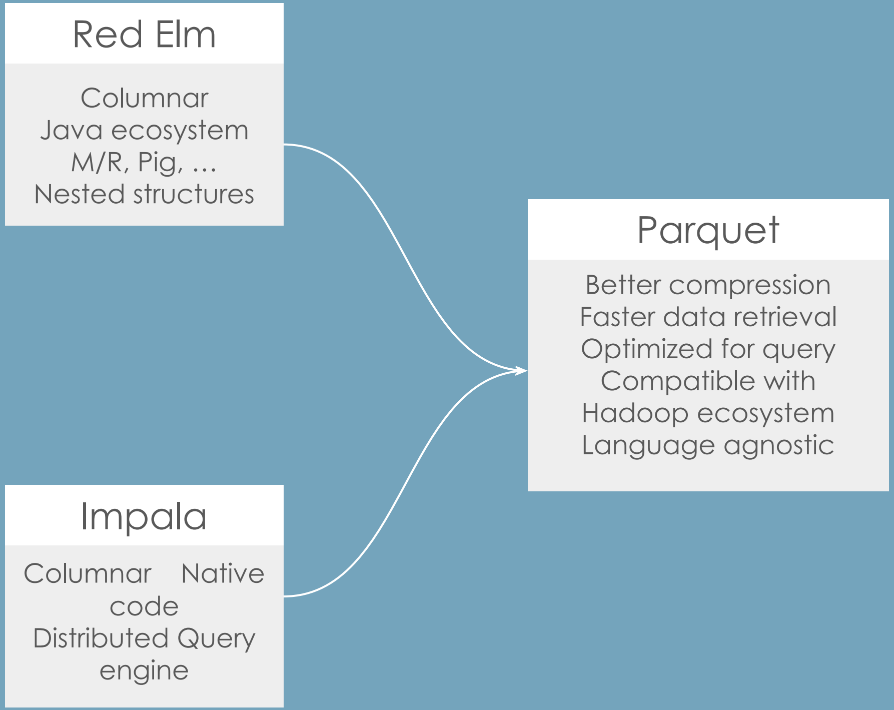


Julien Le Dem
@J_

There is an error in Figure 5 of the Dremel paper.
Guess how I know...

11:26 PM · Aug 24, 2012

Parquet



Red Elm

Columnar
Java ecosystem
M/R, Pig, …
Nested structures

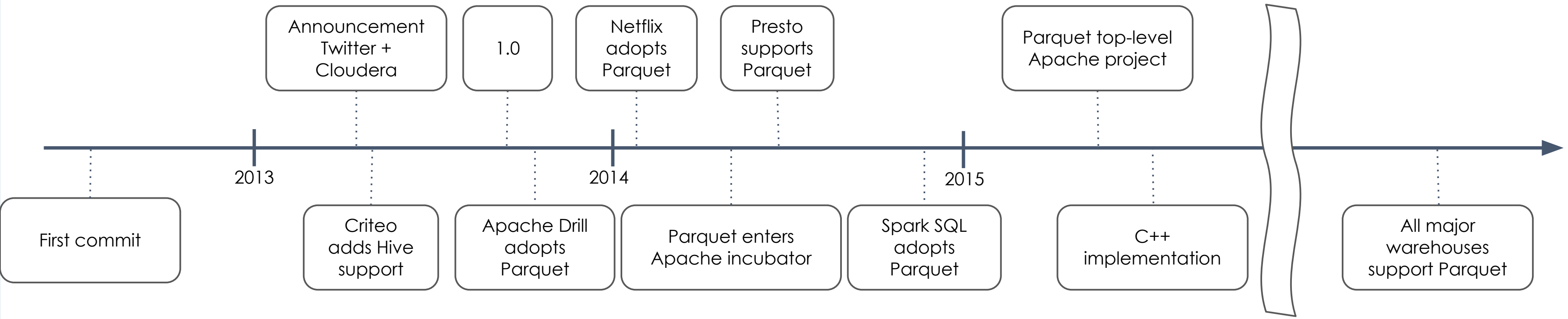
Impala

Columnar Native code
Distributed Query engine

Parquet

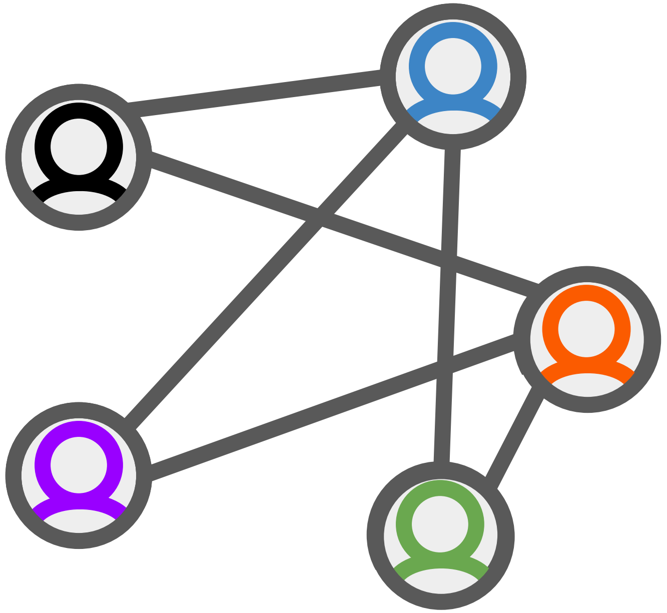
Better compression
Faster data retrieval
Optimized for query
Compatible with Hadoop ecosystem
Language agnostic

# Adoption


Announcement Twitter + Cloudera
1.0
Netflix adopts Parquet
Presto supports Parquet
Parquet top-level Apache project
2013
2014
2015
First commit
Criteo adds Hive support
Apache Drill adopts Parquet
Parquet enters Apache incubator
Spark SQL adopts Parquet
C++ implementation
All major warehouses support Parquet

## Lessons learned

Every contributor is a stakeholder



## The snowball effect

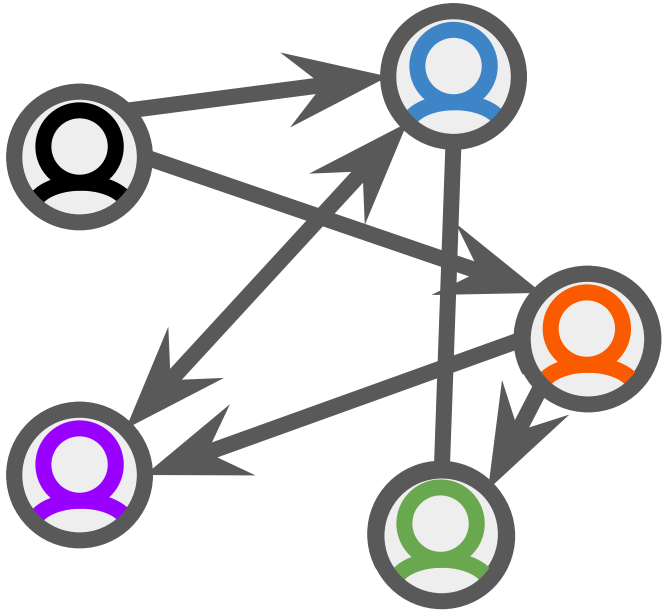
## Open source comes in all shapes and sizes

- Open source licenses
- Governance
- Foundations

II

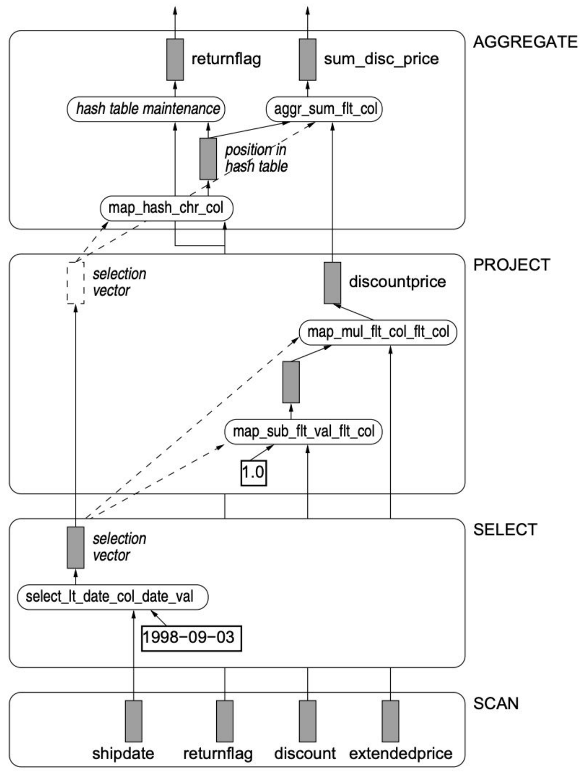
## From Parquet to Arrow

## Need for in-memory columnar format



Vectorization in query engines. Moving from row oriented to columnar

## MonetDB/X100

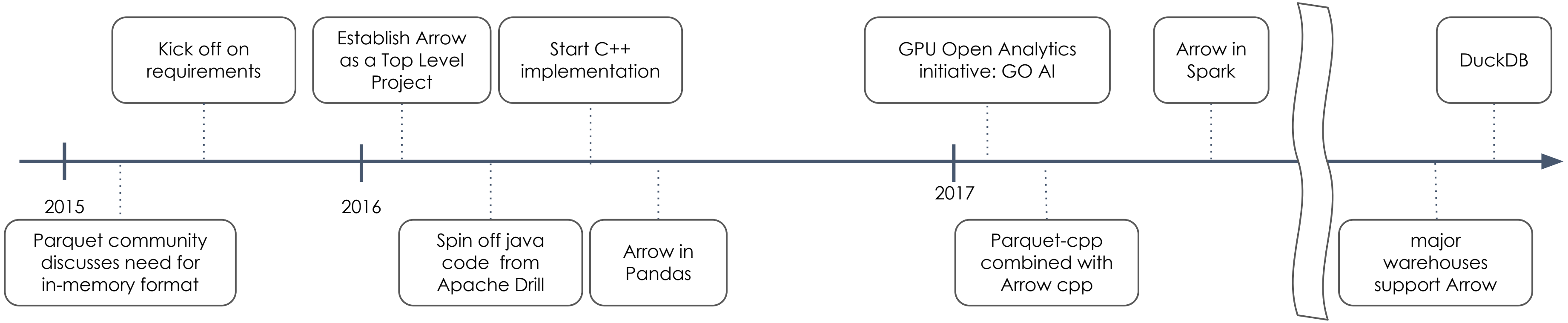


https://www.cidrdb.org/cidr2005/papers/P19.pdf

## Kick off: Consensus on initial requirements

| Arrow | Parquet |
|---|---|
| Fast in-memory processing: | Fast disk retrieval: |
| Vectorized execution | Projection/Predicate push down |
| Zero copy transfer | At-rest Compression |
| Cross language | Cross language |

# Arrow



Kick off on requirements
Establish Arrow as a Top Level Project
Start C++ implementation
GPU Open Analytics initiative: GO AI
Arrow in Spark
DuckDB
2015
2016
2017
Parquet community discusses need for in-memory format
Spin off java code from Apache Drill
Arrow in Pandas
Parquet-cpp combined with Arrow cpp
major warehouses support Arrow

## DuckDB

“SQLite for OLAP”

“Local mode for your DWH”

“Your big data fits in memory”

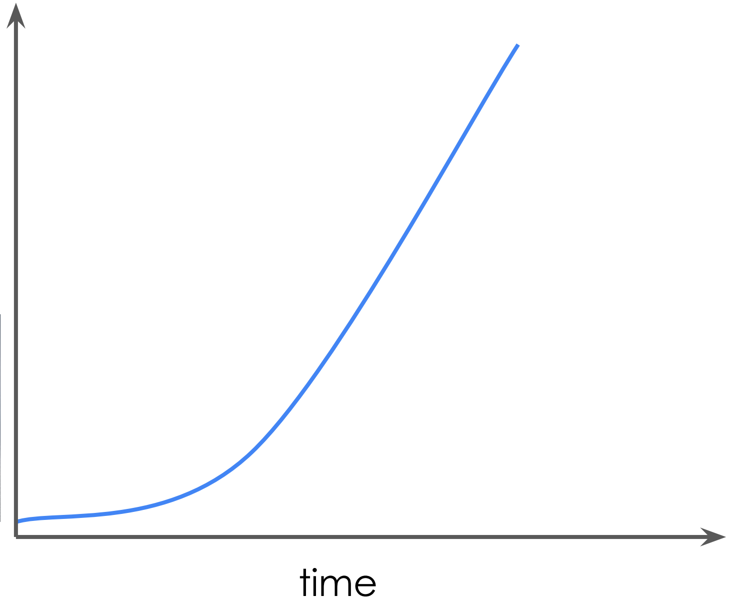
**DuckDB**

Interfaces to Parquet/Arrow

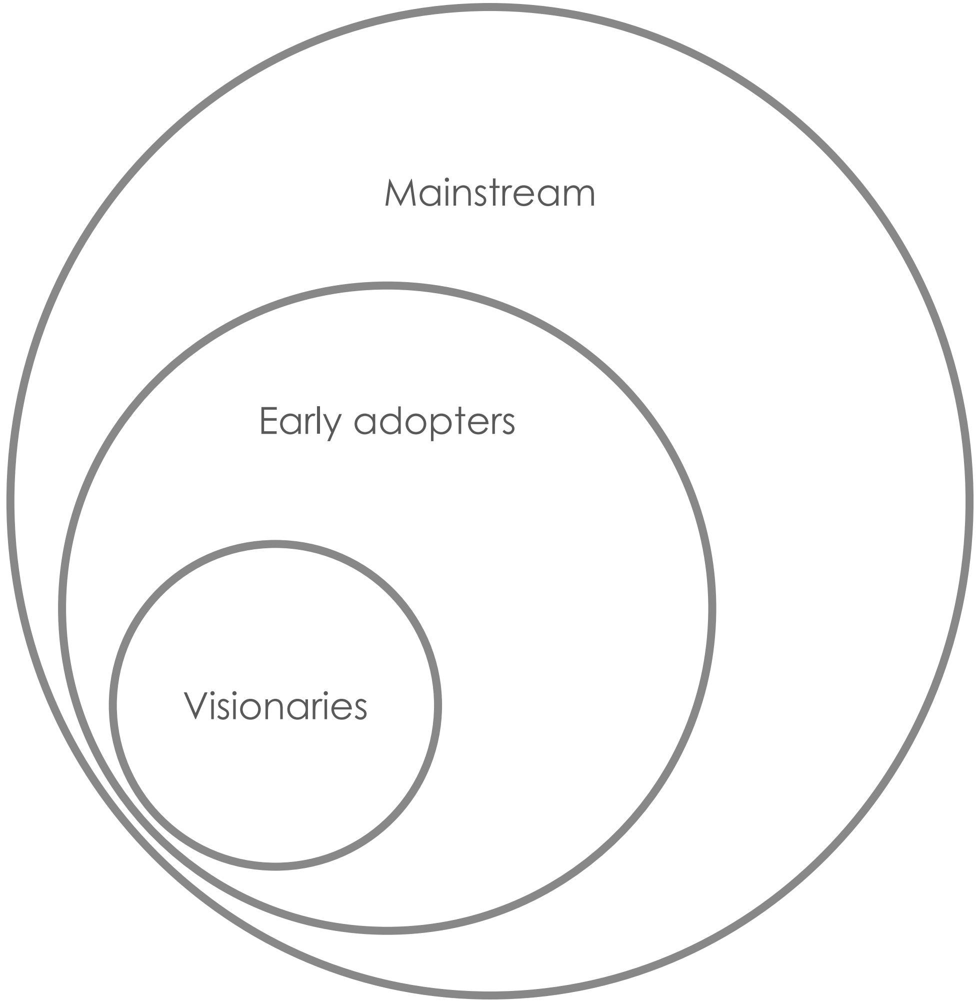
Massively parallel on single machine

## Lessons learned

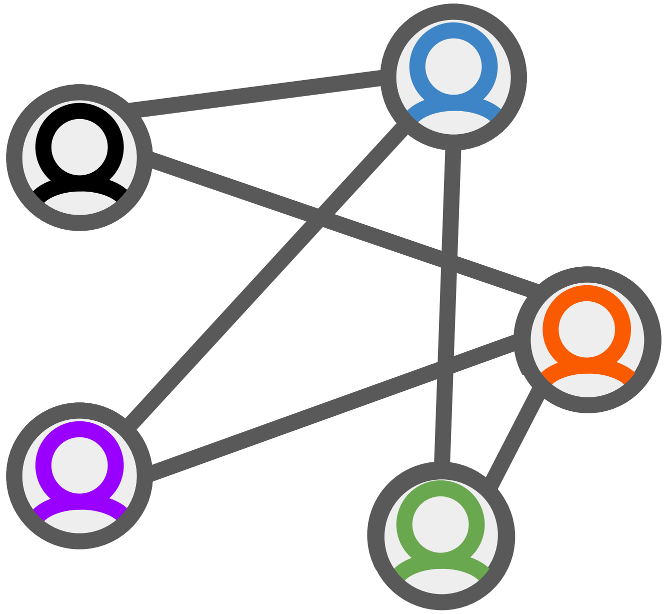
## Bootstrap community with an initial spec



## Find like-minded people who will drive the vision



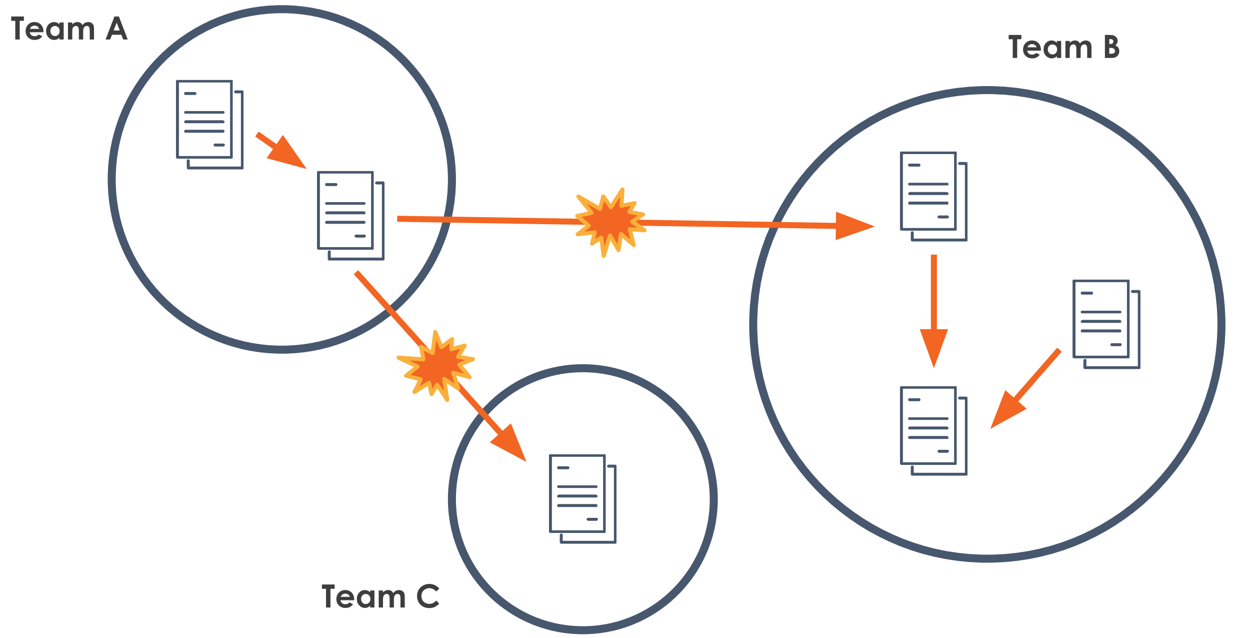
It's about the connections we built along the way.



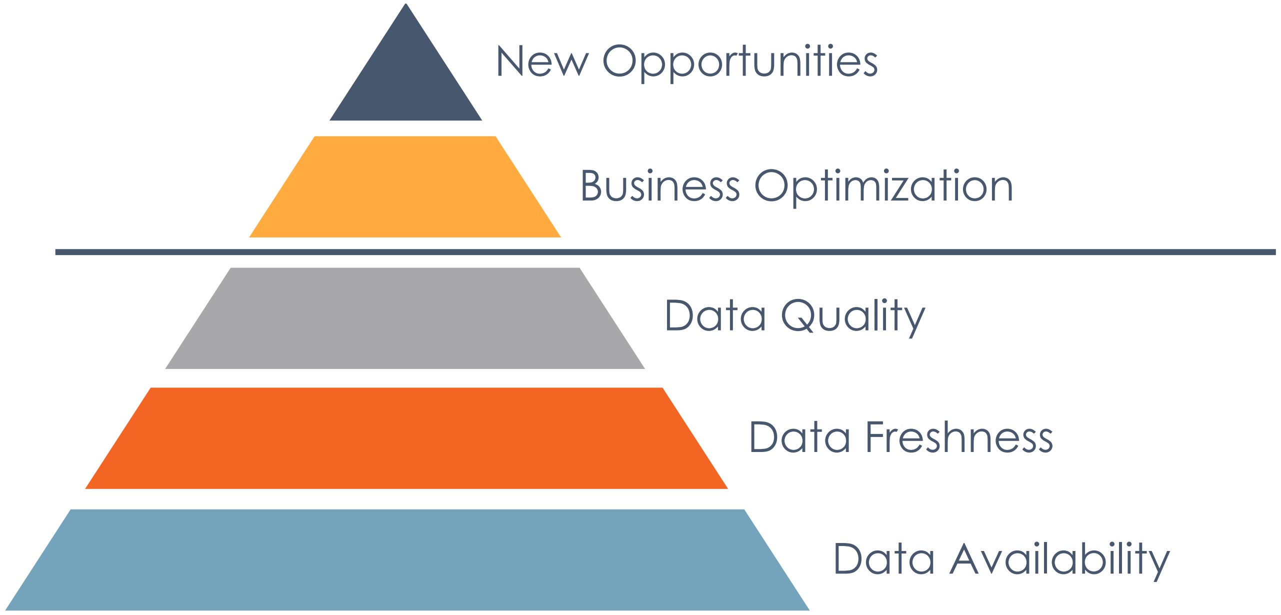
III

## Onwards: OpenLineage

# Building a healthy data ecosystem



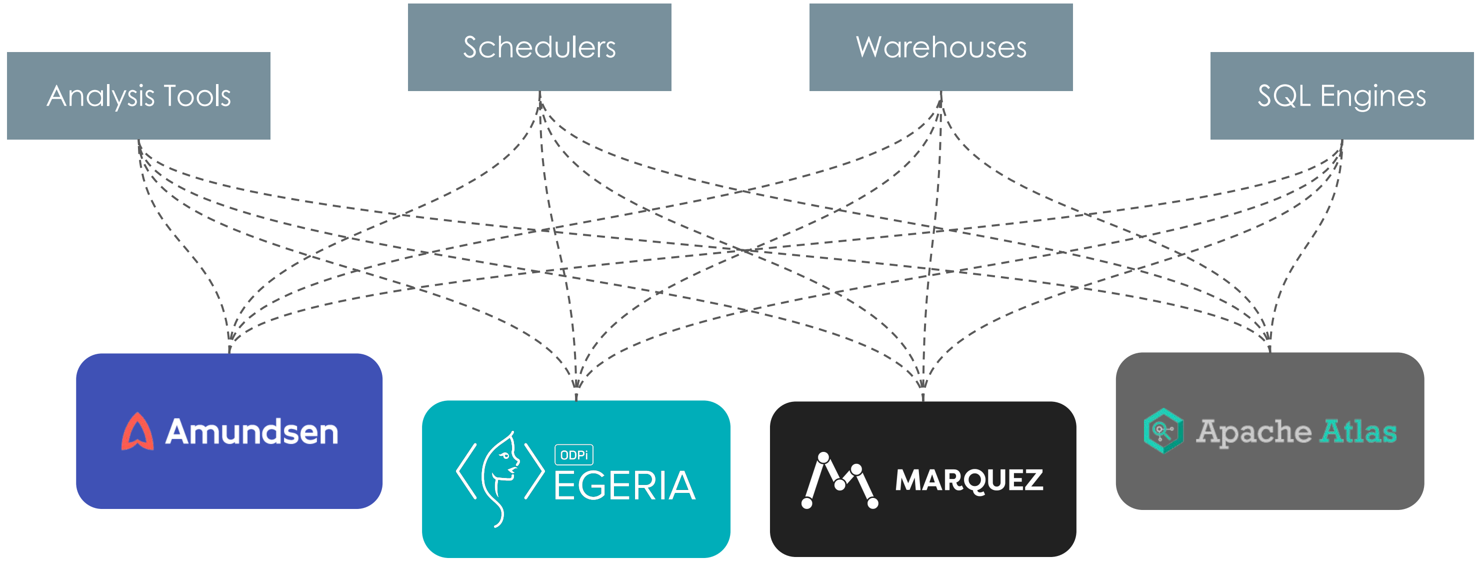
## ~~Maslow's~~ Data hierarchy of needs



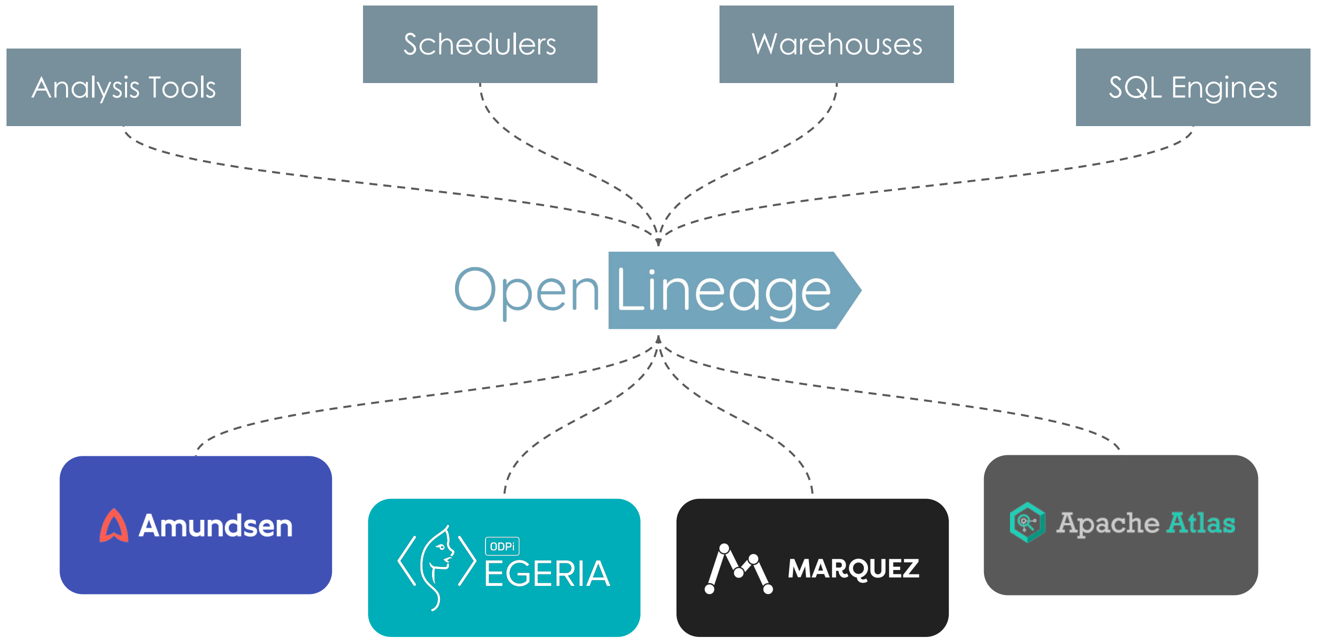
## Marquez: open source metadata

- Missing piece in data ecosystem
- Build a map of all datasets and transformations
- Paved the way to solve data reliability

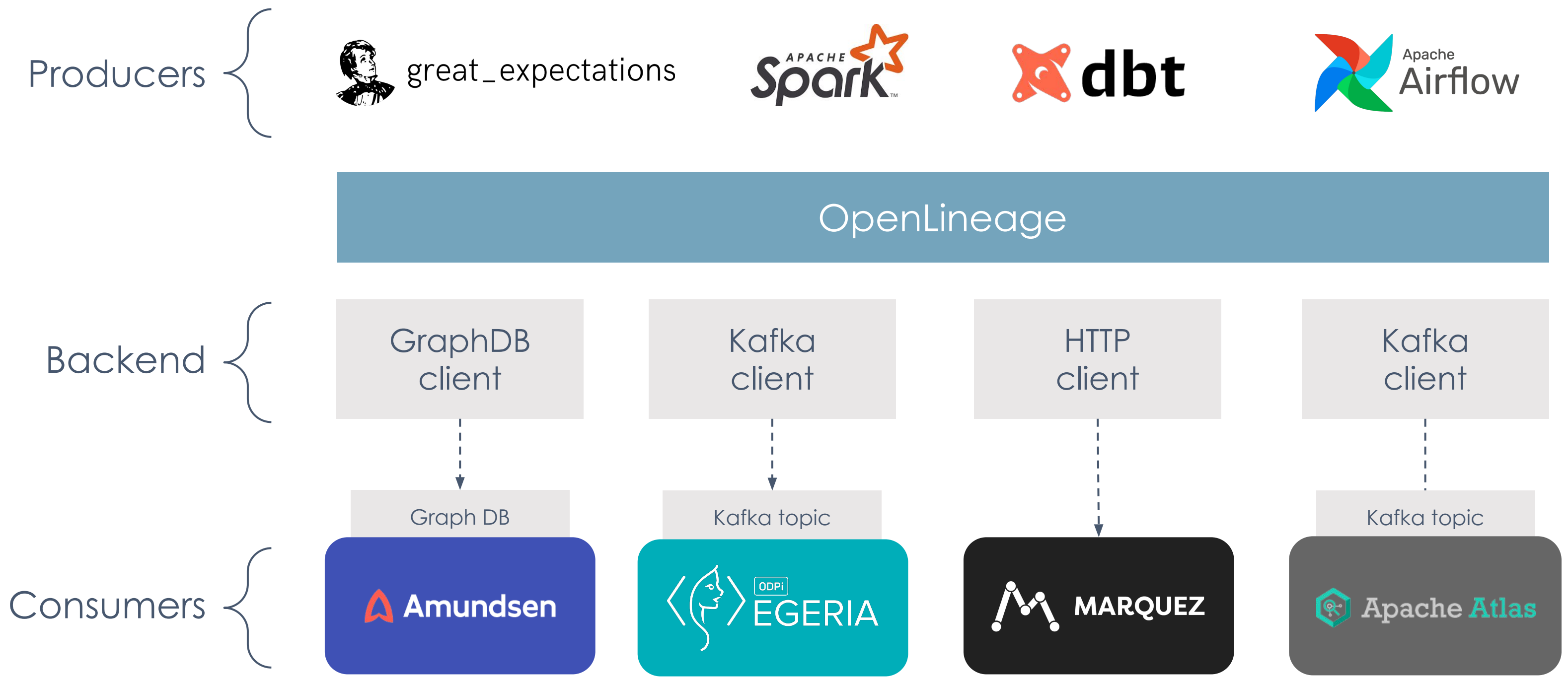
# Before OpenLineage



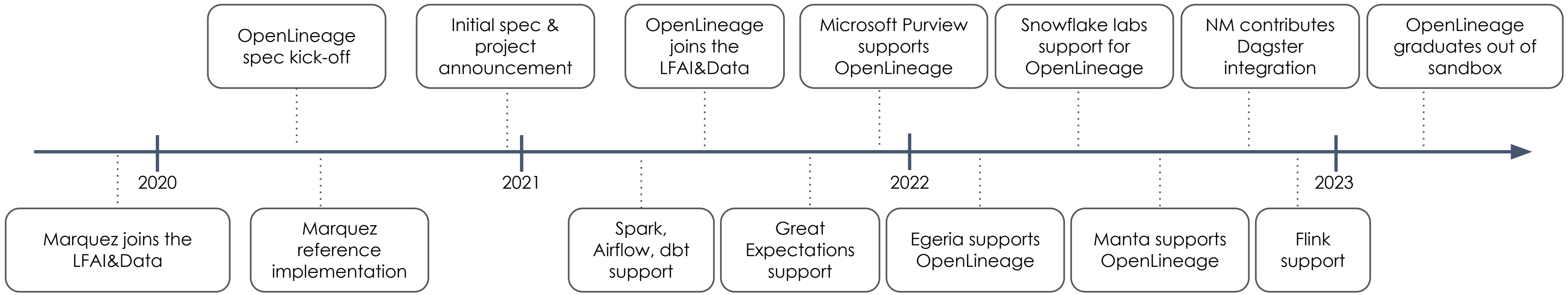
## With OpenLineage



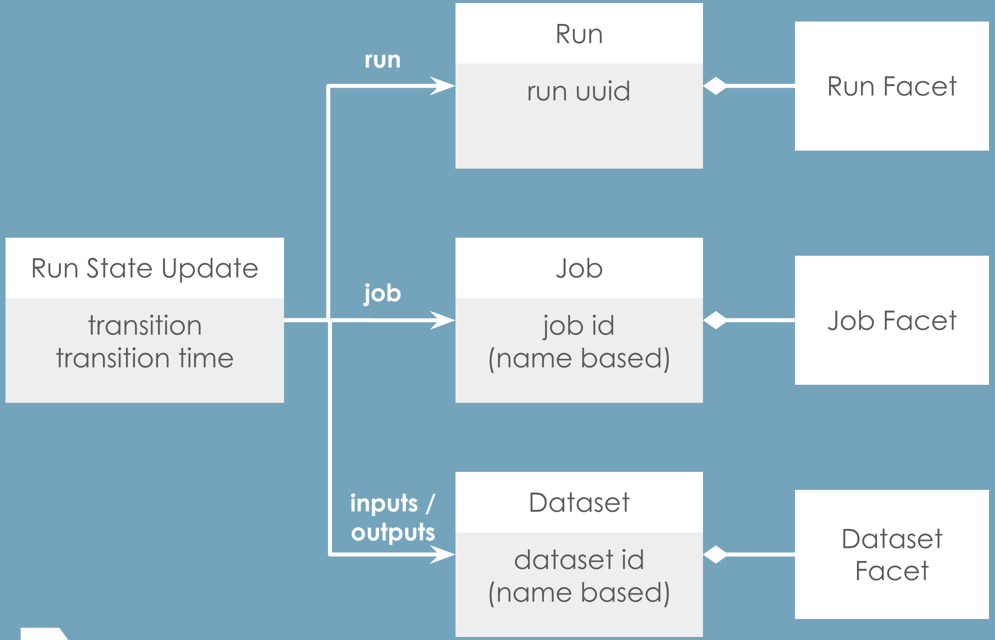
# Where OpenLineage potentially fits



# OpenLineage


Marquez joins the LFAI&Data
2020
OpenLineage spec kick-off
Marquez reference implementation
Initial spec & project announcement
2021
Spark, Airflow, dbt support
OpenLineage joins the LFAI&Data
Great Expectations support
Microsoft Purview supports OpenLineage
2022
Egeria supports OpenLineage
Snowflake labs support for OpenLineage
Manta supports OpenLineage
NM contributes Dagster integration
Flink support
2023
OpenLineage graduates out of sandbox

# Data model



Built around core entities:
Datasets, Jobs, and Runs

Defined as a JSON
Schema spec

Consistent naming for:
Jobs (*scheduler.job.task*)
Datasets (*instance.schema.table)*

## Facet examples

Dataset:

- Stats
- Schema
- Version

Job:

- Source code
- Dependencies
- Source control
- Query plan

Run:

- Scheduled time
- Batch ID
- Query profile
- Params

# Building custom facets

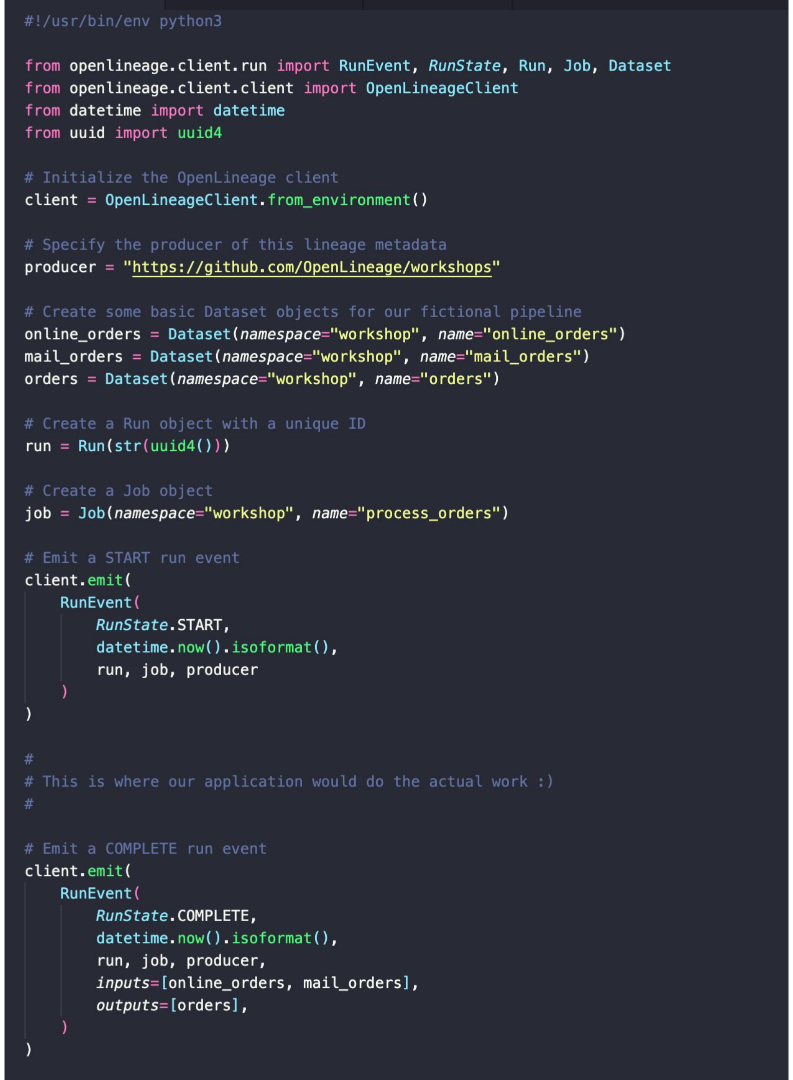
- Custom facets must use a distinct prefix named after the project defining them to avoid collision with standard facets defined in the OpenLineage.json spec

- Custom facets must follow the pattern:

  **{prefix}{name}{entity}Facet**

  Example: BigQueryStatisticsJobFacet

# Using the Python client



```python
#!/usr/bin/env python3

from openlineage.client.run import RunEvent, RunState, Run, Job, Dataset
from openlineage.client.client import OpenLineageClient
from datetime import datetime
from uuid import uuid4

# Initialize the OpenLineage client
client = OpenLineageClient.from_environment()

# Specify the producer of this lineage metadata
producer = "https://github.com/OpenLineage/workshops"

# Create some basic Dataset objects for our fictional pipeline
online_orders = Dataset(namespace="workshop", name="online_orders")
mail_orders = Dataset(namespace="workshop", name="mail_orders")
orders = Dataset(namespace="workshop", name="orders")

# Create a Run object with a unique ID
run = Run(str(uuid4()))

# Create a Job object
job = Job(namespace="workshop", name="process_orders")

# Emit a START run event
client.emit(
    RunEvent(
        RunState.START,
        datetime.now().isoformat(),
        run, job, producer
    )
)

#
# This is where our application would do the actual work :)
#

# Emit a COMPLETE run event
client.emit(
    RunEvent(
        RunState.COMPLETE,
        datetime.now().isoformat(),
        run, job, producer,
        inputs=[online_orders, mail_orders],
        outputs=[orders],
    )
)
```

## Resources

- OpenLineage
  - OpenLineage.md - the OpenLineage specification
  - Python and Java clients
  - Existing integrations
  - NAMING.md - naming conventions for Jobs and Datasets

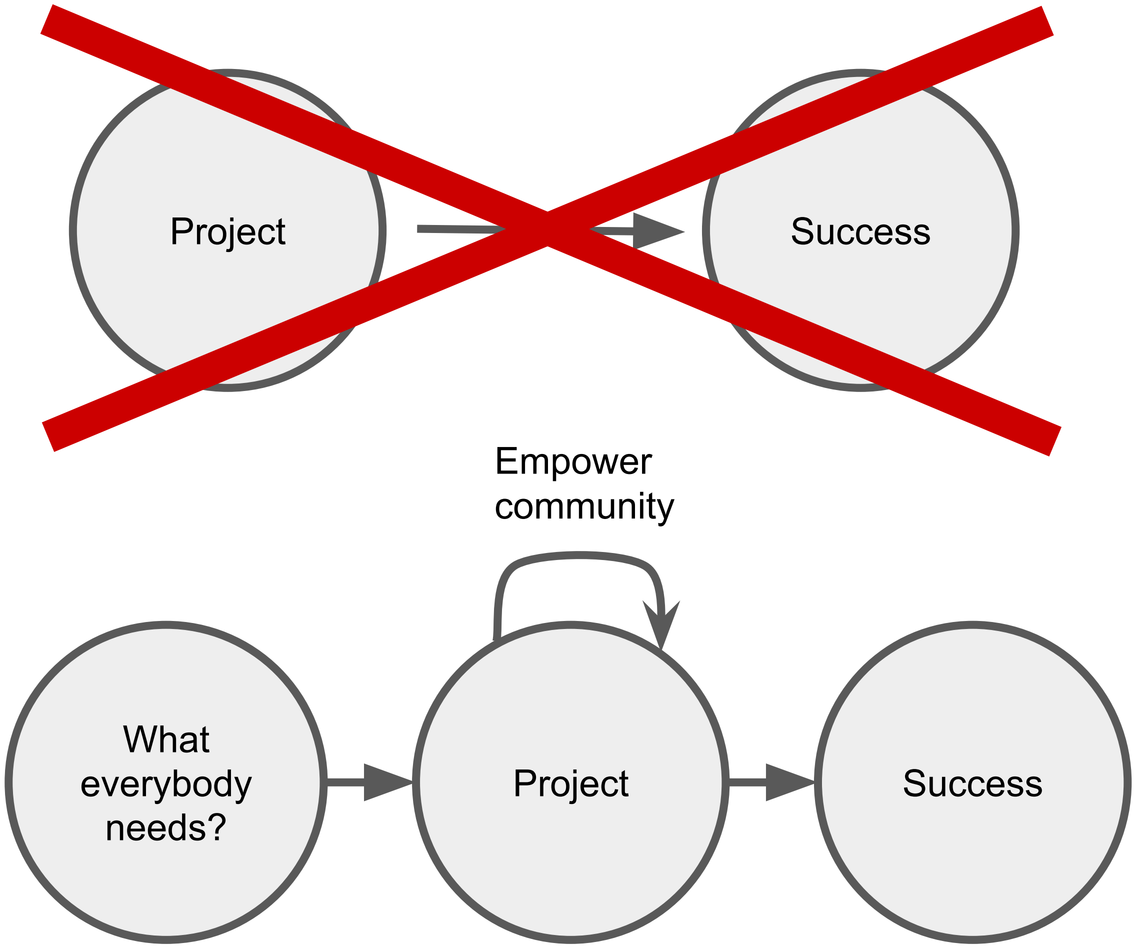
- Blogs / tutorials
  - Getting Started: https://openlineage.io/getting-started/
  - The lineage API: https://openlineage.io/blog/explore-lineage-api/
  - Facets: https://openlineage.io/blog/dataquality_expectations_facet/
  - Spark example: https://openlineage.io/blog/openlineage-spark/

# OMG the possibilities are endless

Dependency tracing
Root cause identification
Issue prioritization
Impact mapping
Precision backfills
Anomaly detection
Change management
Historical analysis
Automated audits

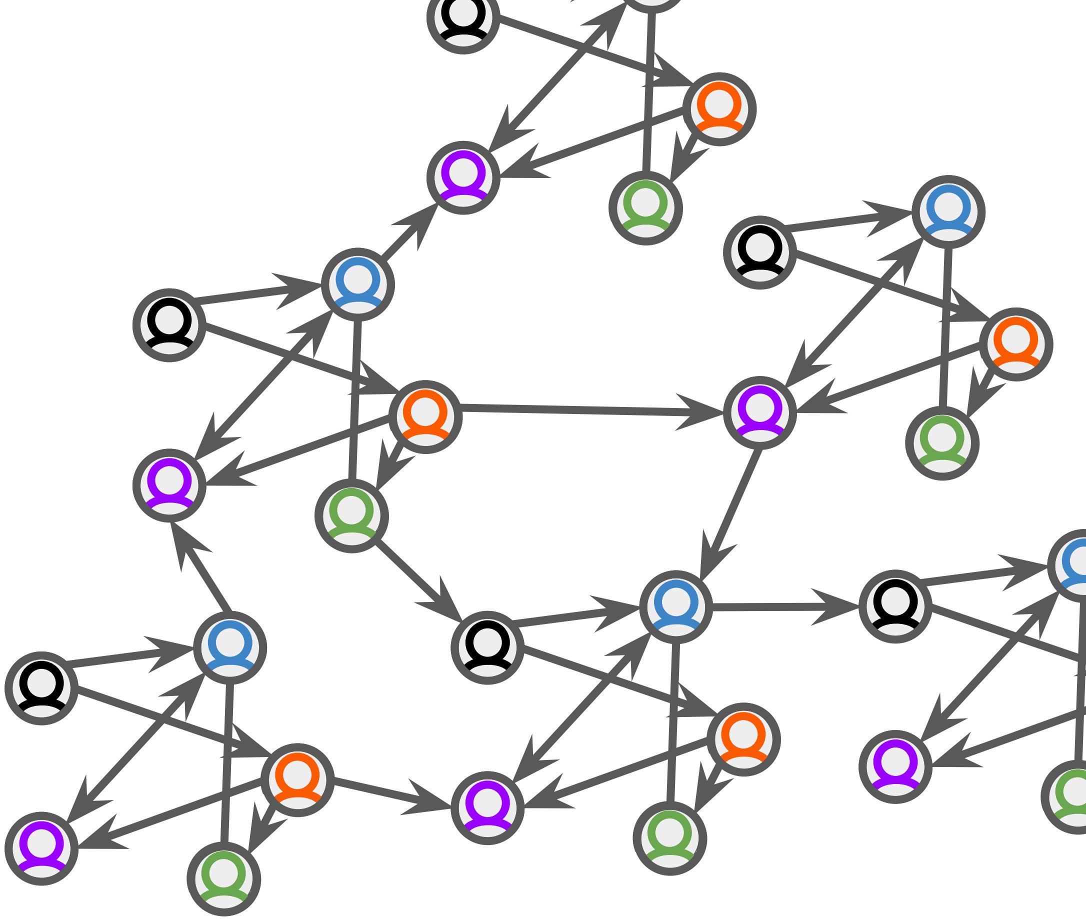
## Lessons learned

You don't start a project and then find a way to make it successful.



## Stone Soup, a fable about community

## Align incentives to build a network effect



## In summary: Lessons learned

- Every contributor is a **stakeholder**.
- The **Snowball** effect.
- Open Source comes in all **shapes** and sizes.
- **Bootstrap** community with an initial spec.
- **Collaborate** with trail blazers.
- It's about the **connections** we built along the way.
- Find what everybody **needs** and fill that need.
- The stone **soup**.
- **Align** incentives to build a network effect.

Thanks :)