Spark-less Data Stack in 2024

Background

- My name is Nok
- Software Engineer of Kedro (QuantumBlack, McKinsey), London
- Open source Python pipeline framework
- *Building Data Pipeline with Python* (PyConHK 2022)
- Anything data

kedro

https://github.com/kedro-org/kedro

Setting the scene

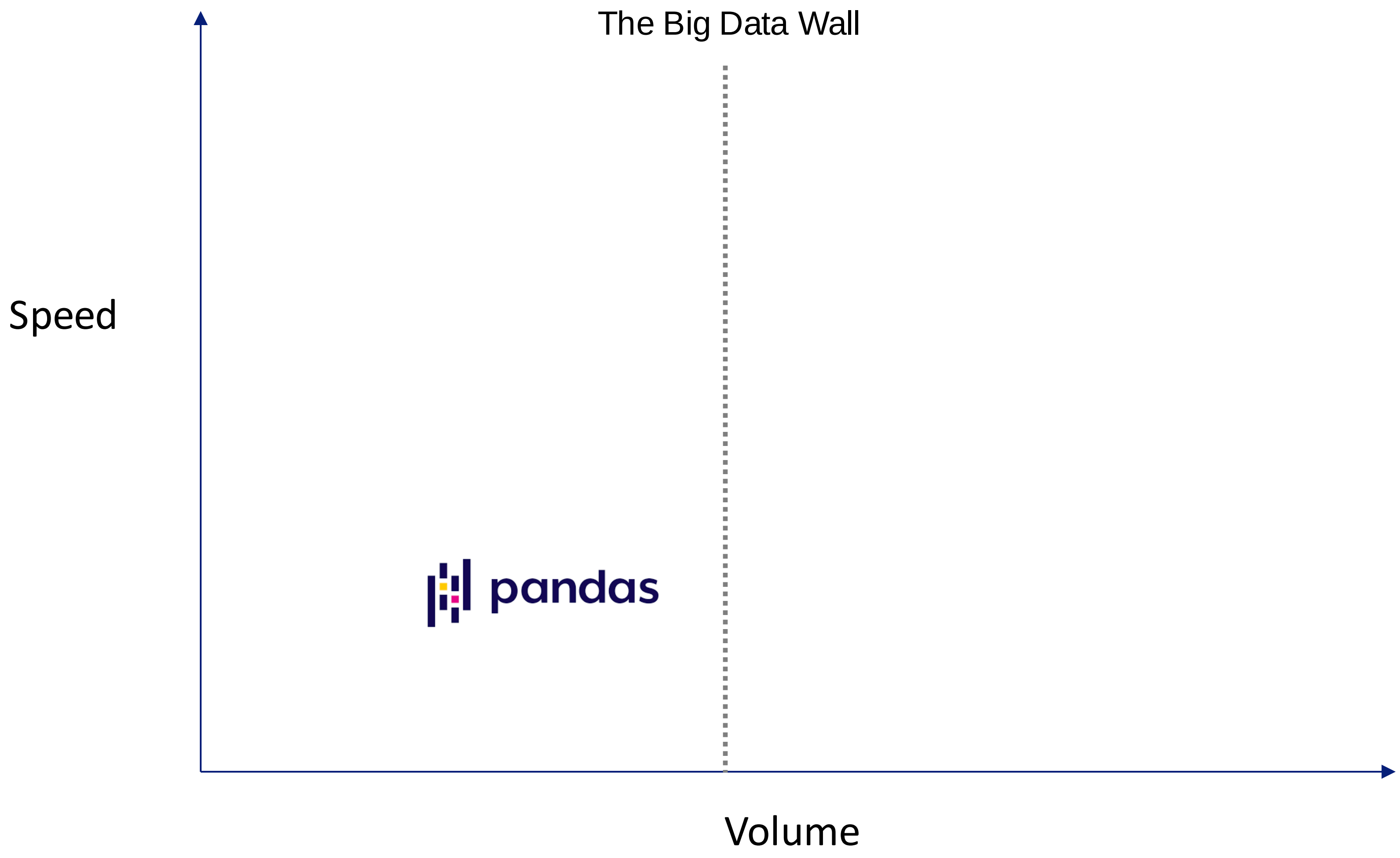
Have You Ever Experienced This?

- Handed off a data pipeline written in pandas (a.k.a. notebook) at the end of a project, that needed to be re-written in Spark for production?
- Taken over a Spark pipeline processing multiple CSV files for ad-hoc analysis?

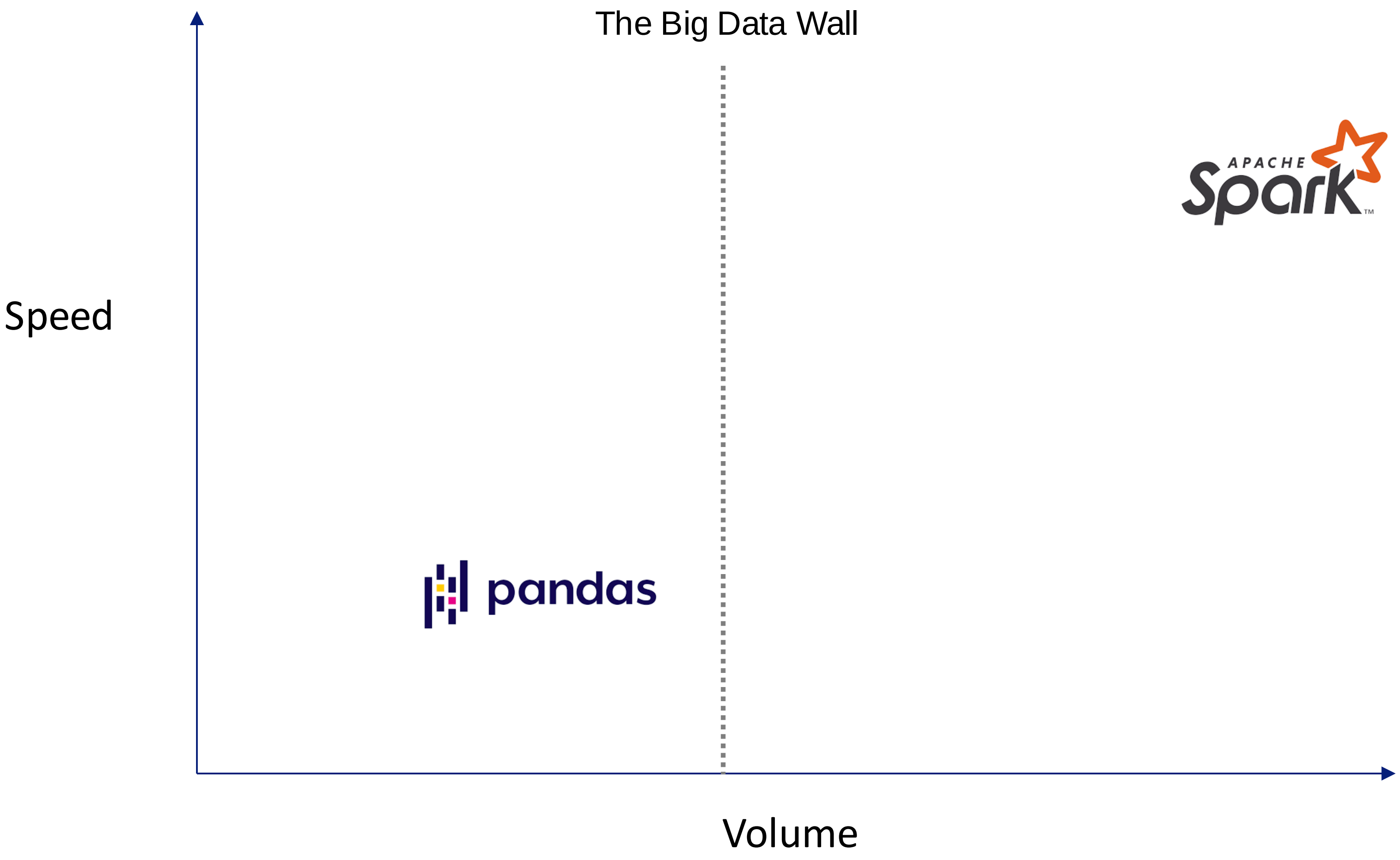
Why Spark is so popular?

- Spark was designed for Big Data. (i.e. doesn’t fit on a single machine)
- Spark is a safe choice
- Spark is part of the infrastructure for data platforms, least resistance path

Pandas vs Spark



Pandas vs Spark




"IF YOOU HAVE A HAMMMER, EVERYTHING bds LIKE A NAIL"
SPARK
POLARS
PANDAS
"PANDAS
MEDIUM
POLADS
POLARS
POLARS
POLASK
PANDAS
PANDAS
MEDIUM
DASK
SPARK
SPARK
DUCKDB
"DUCKDB." DASK

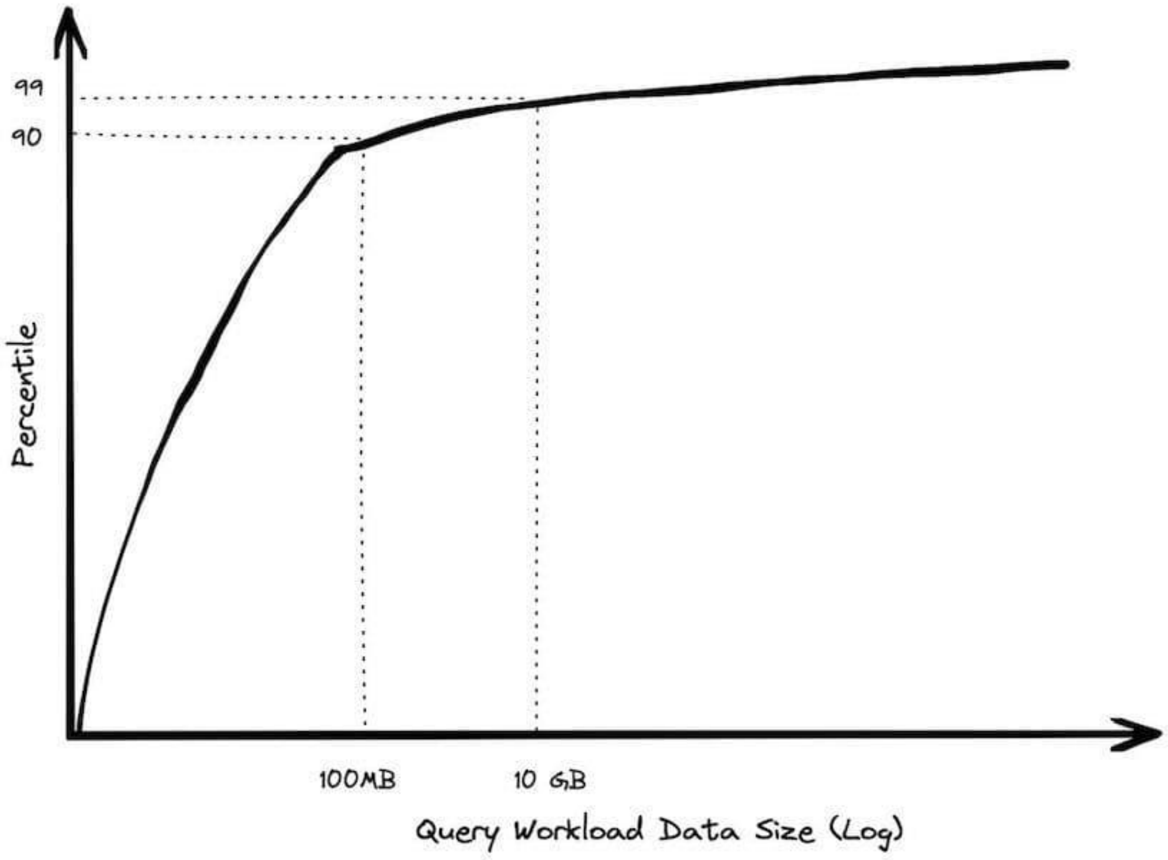
You may not really have a big data problem (Excel is not big data)

BIG DATA IS DEAD

2023/02/07 - 19 min read

BY JORDAN TIGANI

- Analysis of queries that runs on BigQuery



Source: https://motherduck.com/blog/big-data-is-dead/

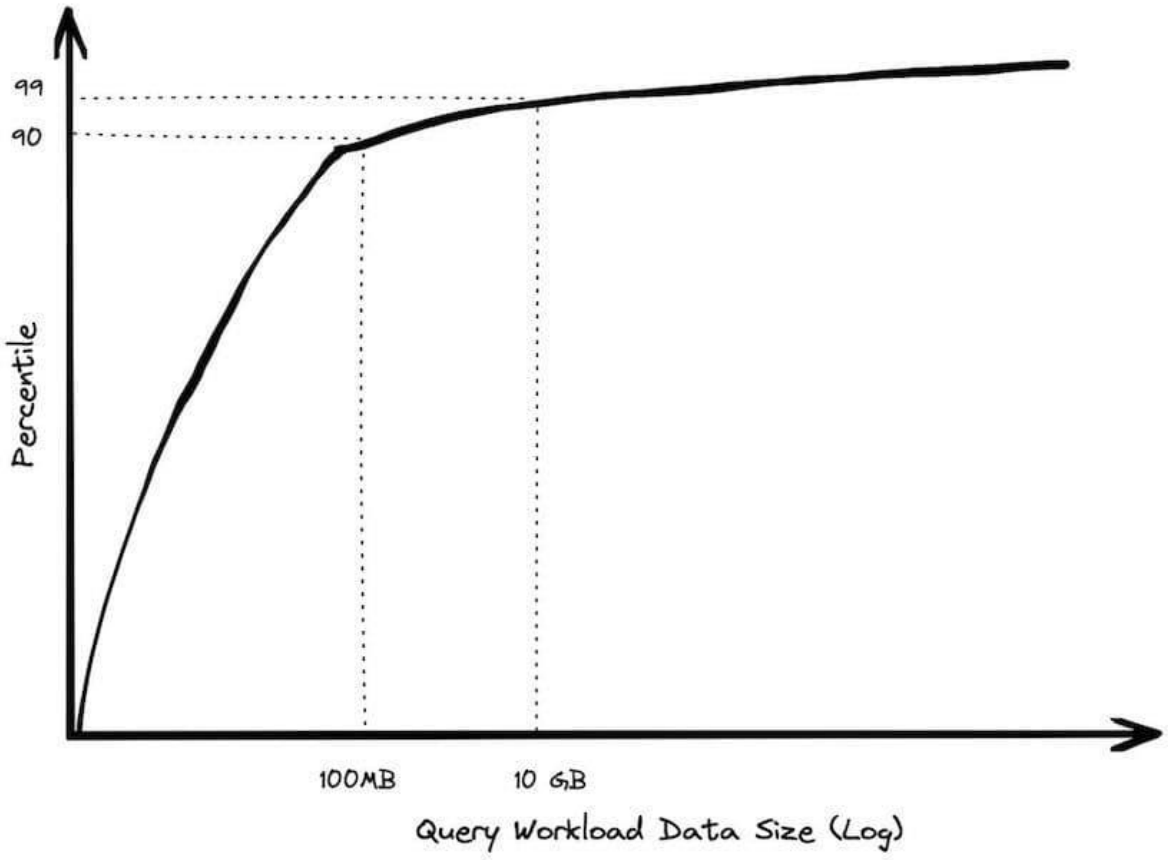
You may not really have a big data problem

BIG DATA IS DEAD

2023/02/07 - 19 min read

BY JORDAN TIGANI

- Analysis of queries that runs on BigQuery
    - Data is not increasing as fast as we expected
    - Storage size != Workload size



Source: https://motherduck.com/blog/big-data-is-dead/

Modern era – What has changed?

Separation of compute and storage

- Spark(2014), Parquet(2015), Arrow(2016)
- Cloud
- Spark reads Parquet stored in S3
- The same Parquet file can be read by Spark, DuckDB, pandas without moving the data
    - More tools are built

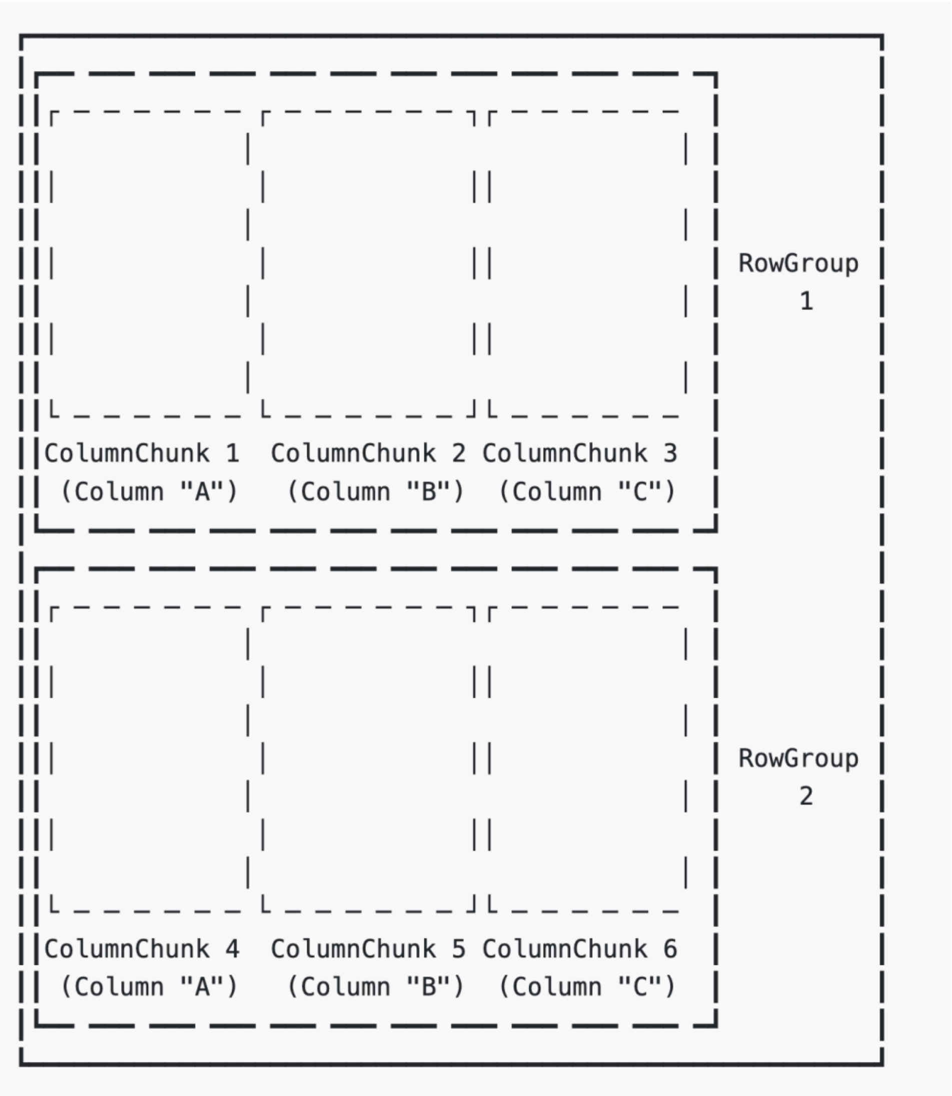
Modern columnar storage are efficient

PUBLISHED 26 Dec 2022
BY tustvold and alamb

Querying Parquet with Millisecond Latency

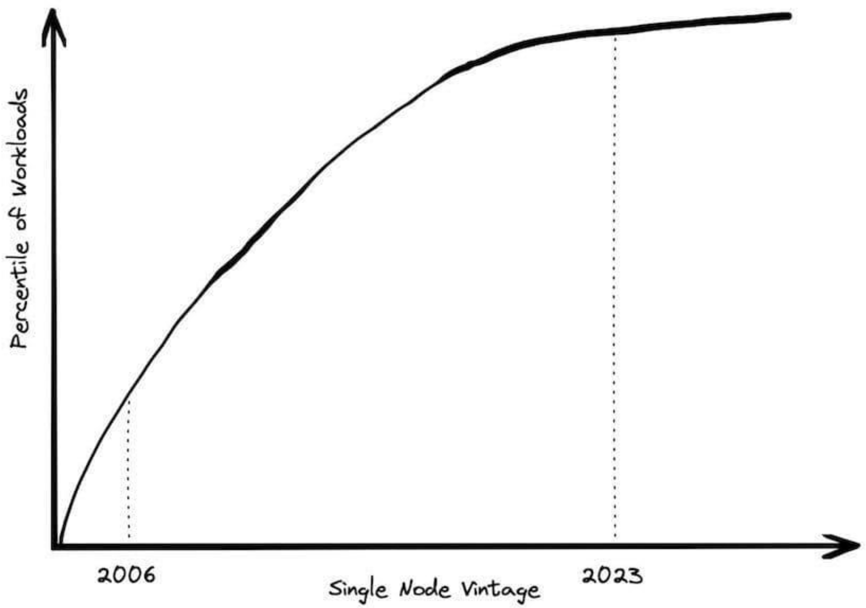
“… querying Parquet files — be it on local disk or remote object storage. It is able to query GBs of Parquet in a matter of milliseconds.”

- Full table scan are rare
- Subset of columns
- Filter by time

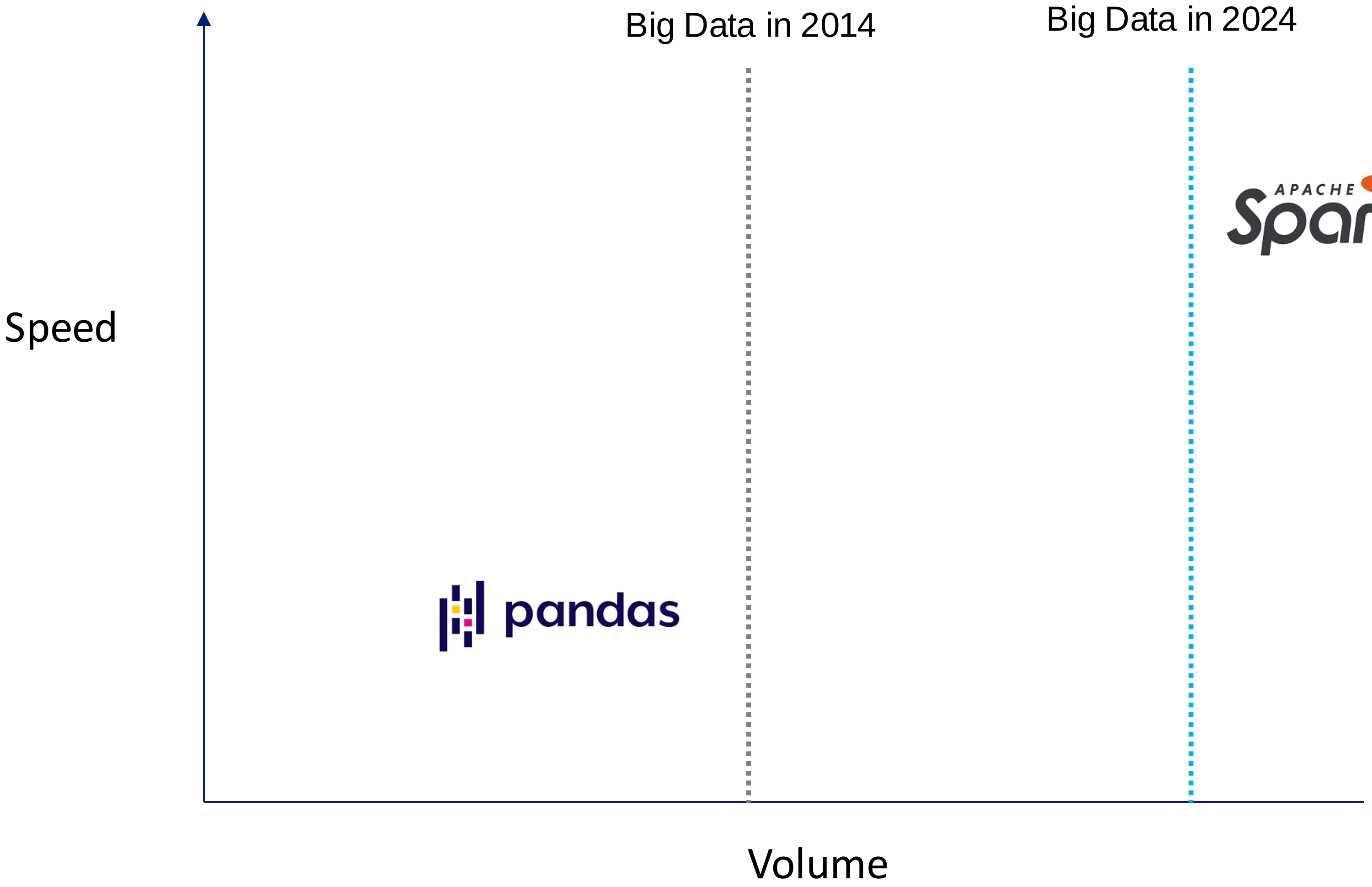


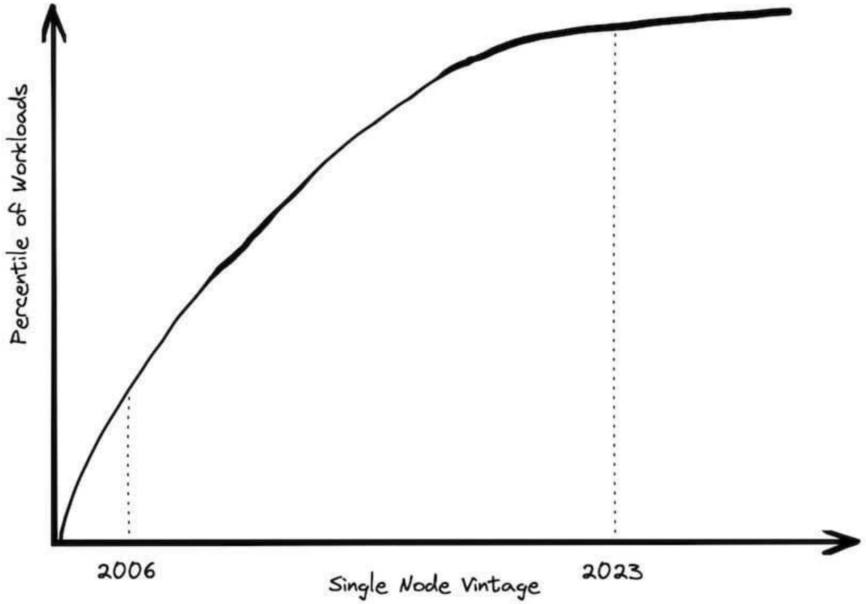
Source: https://arrow.apache.org/blog/2022/12/26/querying-parquet-with-millisecond-latency/

The definition of Big Data changes

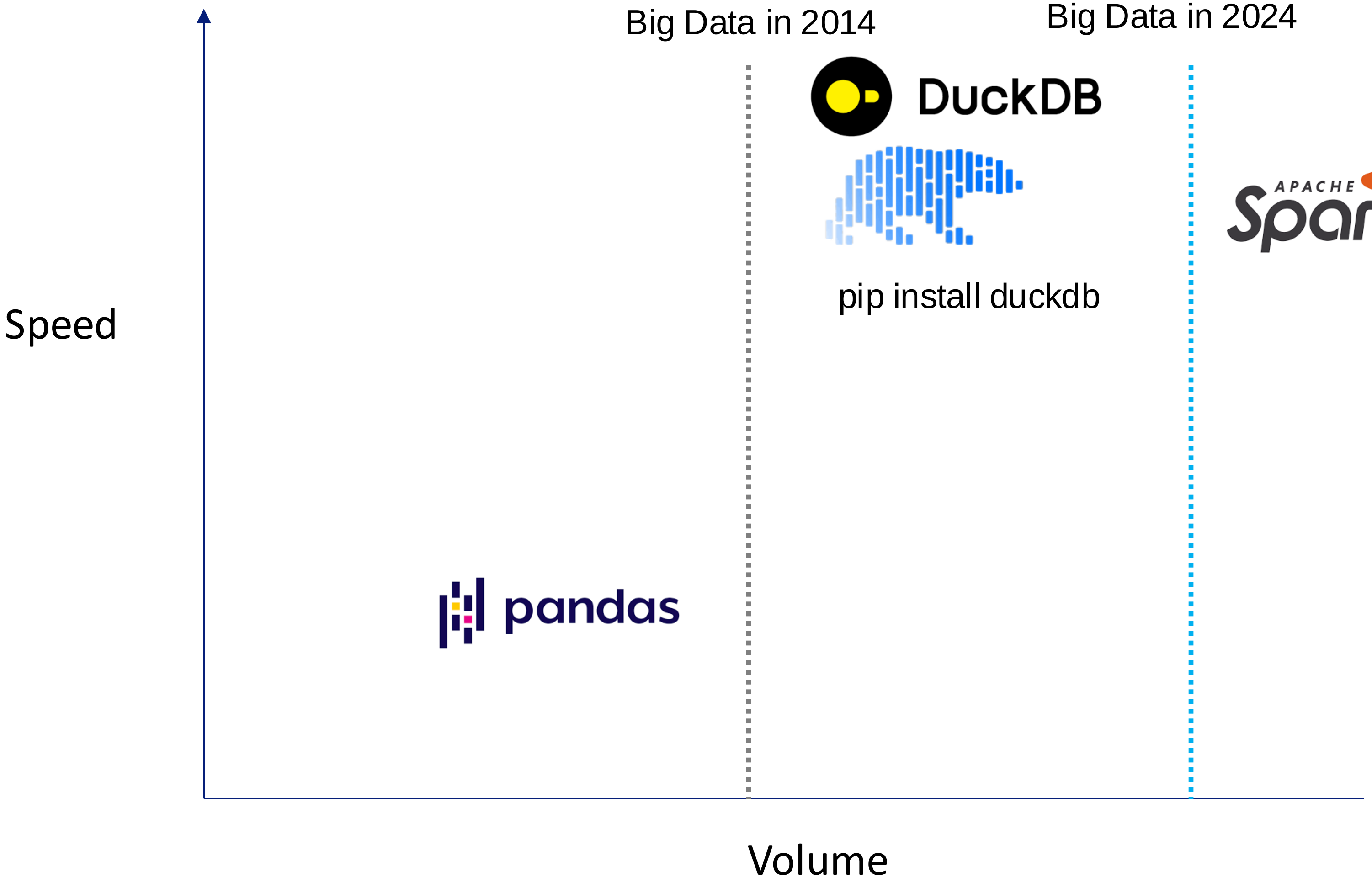


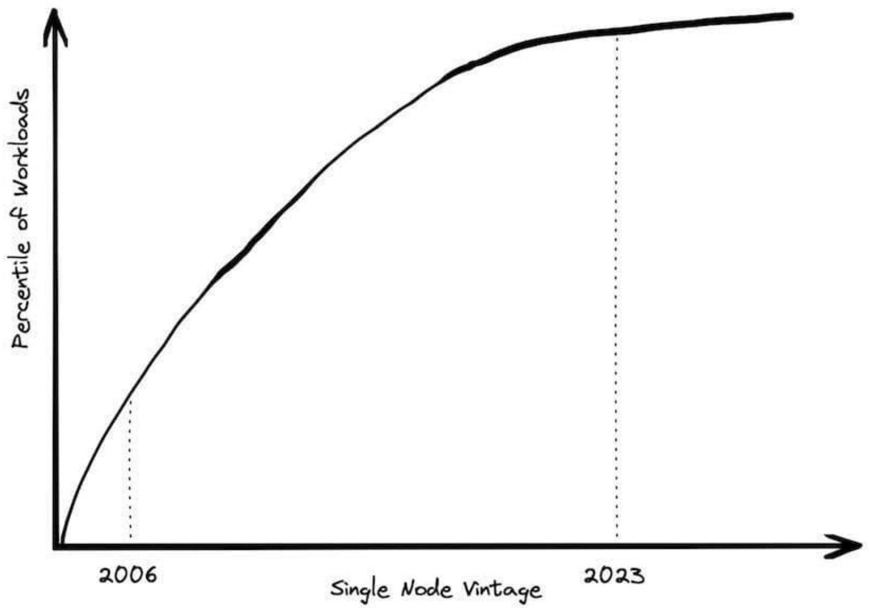
The definition of Big Data shifts





More prominent single-core player appears



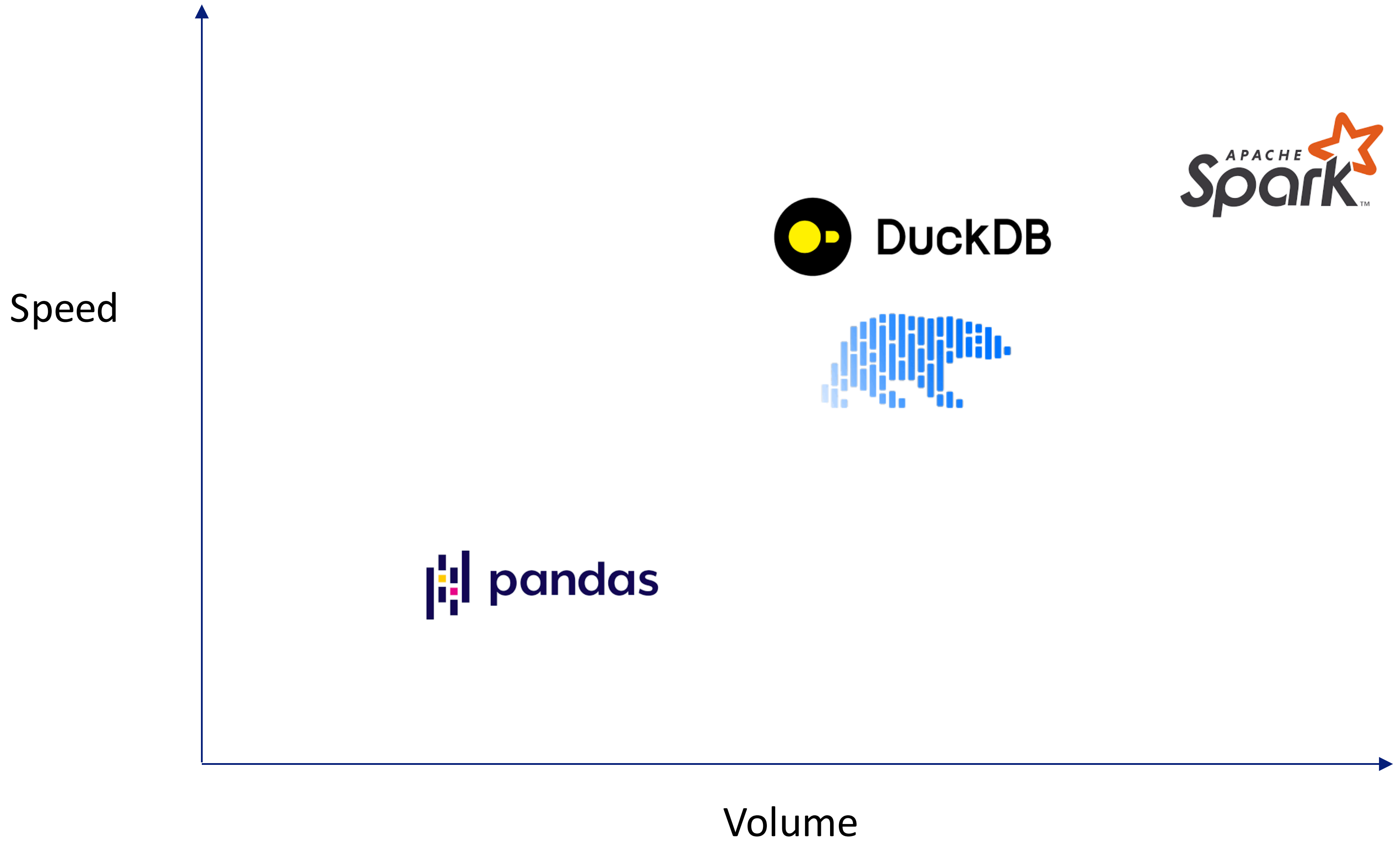


The Modern Era – What has changed?

- Compute and storage are de-coupled
  - Spark reads Parquet stored in S3
- More flexibility to choose the compute engine without moving data
- Single node processing becomes much more powerful
- Data validation (running unit test on data) is a norm (*great-expectations*, *pandera* etc)
  - Spark is SLOW for small workload

The Modern Era – What has not changed?

- SQL is not going anywhere, it will outlive all of us.
- Performance is not the only concern. How to play well with existing systems?



The SQL world

BigQuery

MySQL

ORACLE

Unique Challenges

- Using the right tool for the right workload

Unique Challenges

- Using the right tool for the right workload
- Inherited problems with SQL

SQL

Pros	Cons
Standardized†	Effectively untestable
Concise*	*Sometimes inscrutable
	Fails at runtime

†Kind of, but also not really

Source: [Unified Stream/Batch Execution with Ibis]

SQL

Pros

Standardiz

Concise*

†Kind of, but also

```
CREATE PROCEDURE [unit_test_sales].[test calculation_of_the_sales_amount]
AS
BEGIN
    DECLARE @random_birthdate AS DATE

    EXEC tSQLt.FakeFunction 'sales.calculating_age'
        , 'unit_test_sales.Fake_calculating_age';

    EXEC tSQLt.FakeTable 'sales.sales_transactions';

    EXEC tSQLt.FakeTable 'sales.customer';

    INSERT INTO sales.customer (customer_id, customer_name)
    VALUES (1, 'Unit Test Customer')

    INSERT INTO sales.sales_transactions
    VALUES (1,1,10)

    INSERT INTO sales.sales_transactions
    VALUES (1, 1, 20)

    DROP TABLE IF EXISTS unit_test_sales.expected
    DROP TABLE IF EXISTS unit_test_sales.actual
    CREATE TABLE unit_test_sales.expected (sales_amount INT)

    INSERT INTO unit_test_sales.expected
    VALUES (13)

    CREATE TABLE unit_test_sales.actual (sales_amount INT)

    INSERT INTO unit_test_sales.actual
    EXECUTE sales.calculate_customer_sales_amount @customer_name = 'Unit Test Customer' , @cu
stomer_birth_date = @random_birthdate

    EXEC tSQLt.AssertEqualsTable 'unit_test_sales.expected'
        , 'unit_test_sales.actual';
END
```

Source: Unified Stream/Batch Execution with Ibis

Unique Challenge

- Using the right tool for the right workload
- Inherited problems with SQL
  - Standardization
  - Hard to test
- Explosion of interface, learning curve
  - Postgres
  - Pandas
  - DuckDB
  - PySpark

SQLGlot

SQLFrame

Ibis

the portable Python dataframe library

SQLFrame

SQLFrame – PySpark drop-in replacement

```
# Top Supplier query
from sqlframe import activate
from pyspark.sql.functions import col
from pyspark.sql import functions as F

from pyspark.sql import SparkSession

activate(engine="duckdb")
# Create a SparkSession object
session = SparkSession.builder.master("local").getOrCreate()
session
```

```
<sqlframe.duckdb.session.DuckDBSession at 0xffff7c2e6540>
```

Drop in replacement for Spark

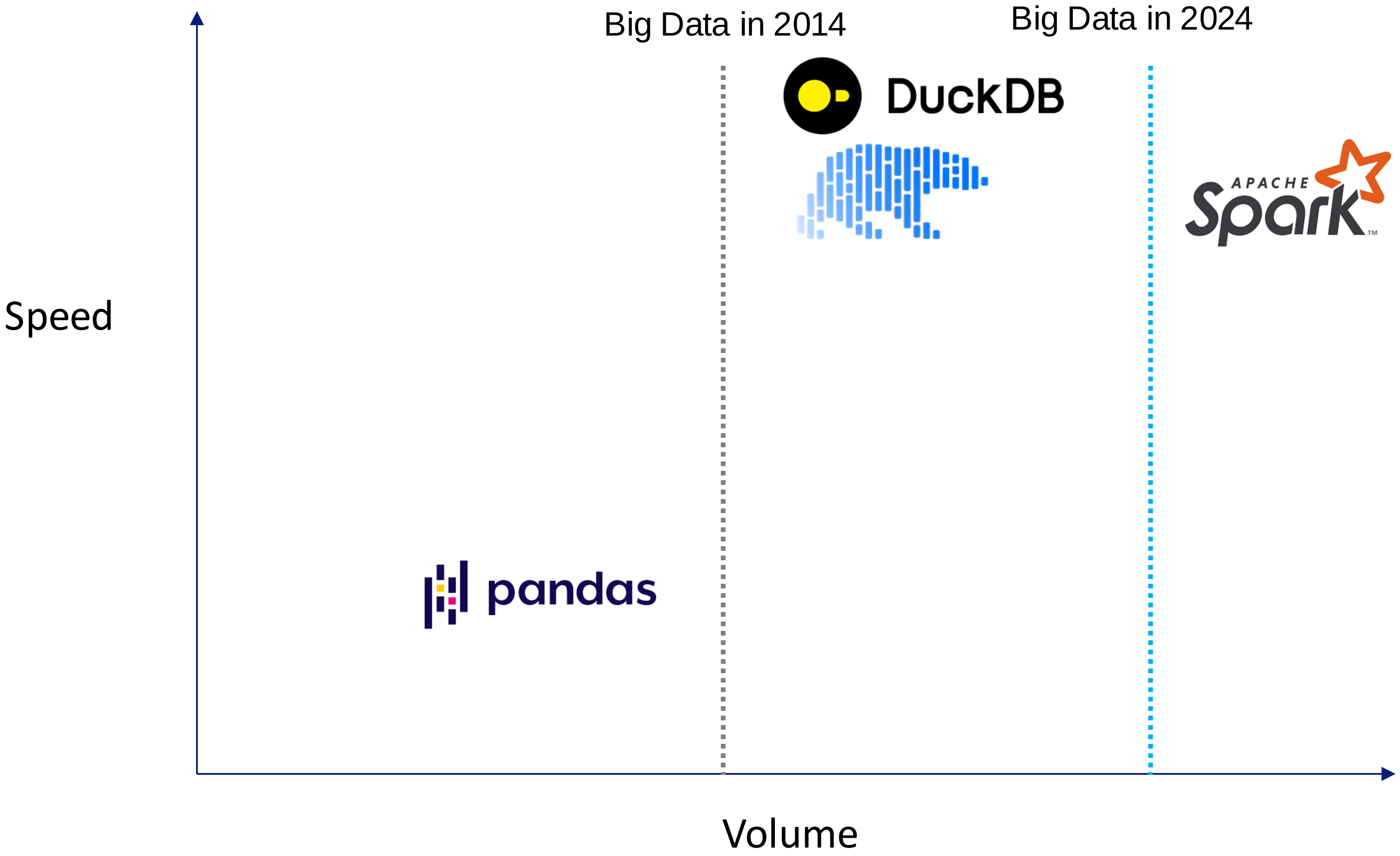
```
%%time
lineitem = session.read.parquet("data/lineitem.parquet")
supplier = session.read.parquet("data/supplier.parquet")

# Step 1: Create revenue equivalent in PySpark
revenue = (
    lineitem.filter(
        (col("l_shipdate") >= "1996-01-01") & (col("l_shipdate") < "1996-04-01")
    )
    .groupBy("l_suppkey")
    .agg(F.sum(col("l_extendedprice") * (1 - col("l_discount"))).alias("total_revenue"))
)
revenue.show()

max_revenue = revenue.agg(F.max("total_revenue")).first()[0]

result = (
    supplier.join(revenue, supplier.s_suppkey == revenue.l_suppkey)
    .filter(revenue.total_revenue == max_revenue)
    .select(
        supplier.s_suppkey,
        supplier.s_name,
        supplier.s_address,
        supplier.s_phone,
        revenue.total_revenue,
    )
    .orderBy(supplier.s_suppkey)
)

# Show the final result
result.show()
```

```
CPU times: user 7.28 s, sys: 985 ms, total: 8.27 s
Wall time: 761 ms
```

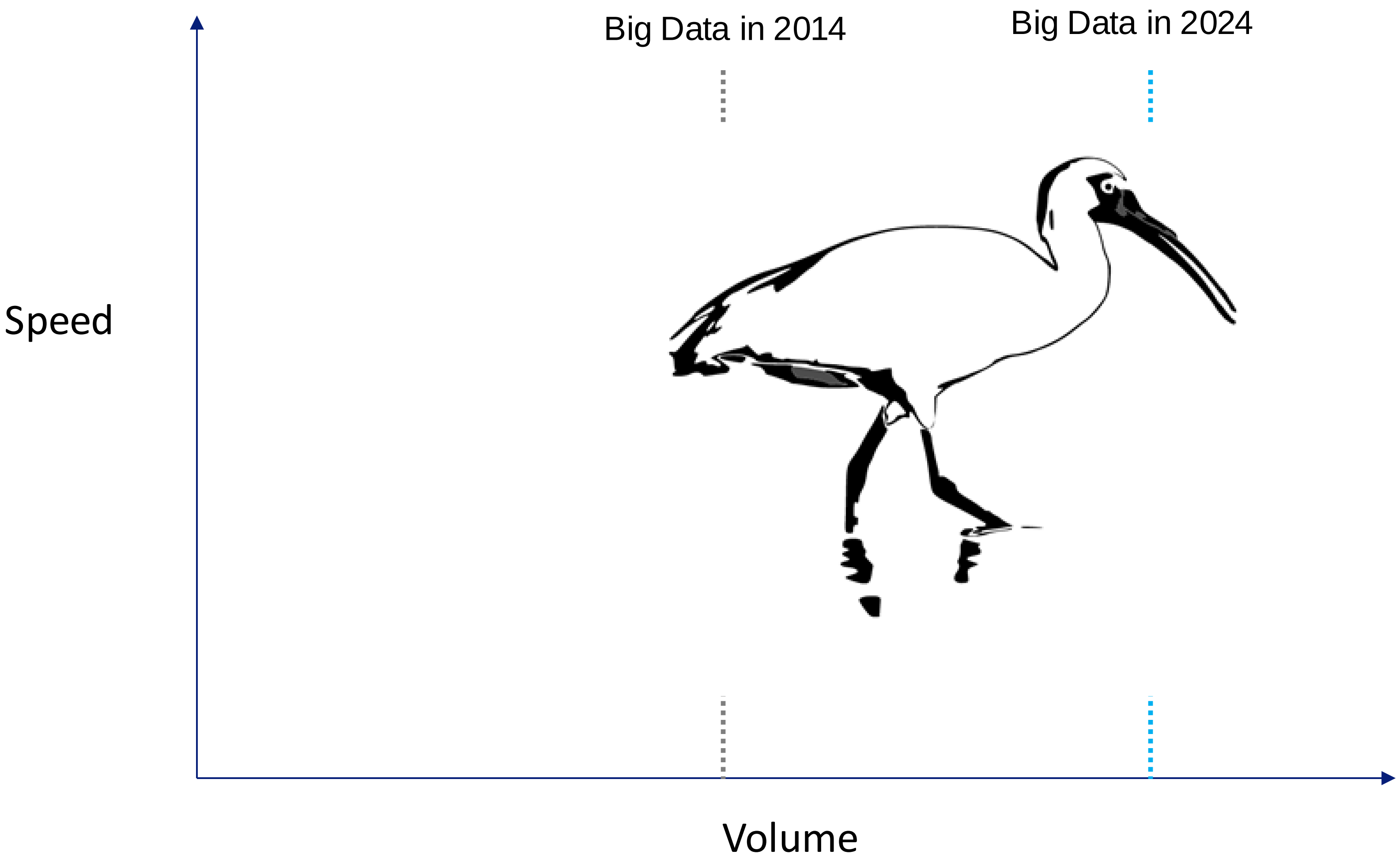
Ibis

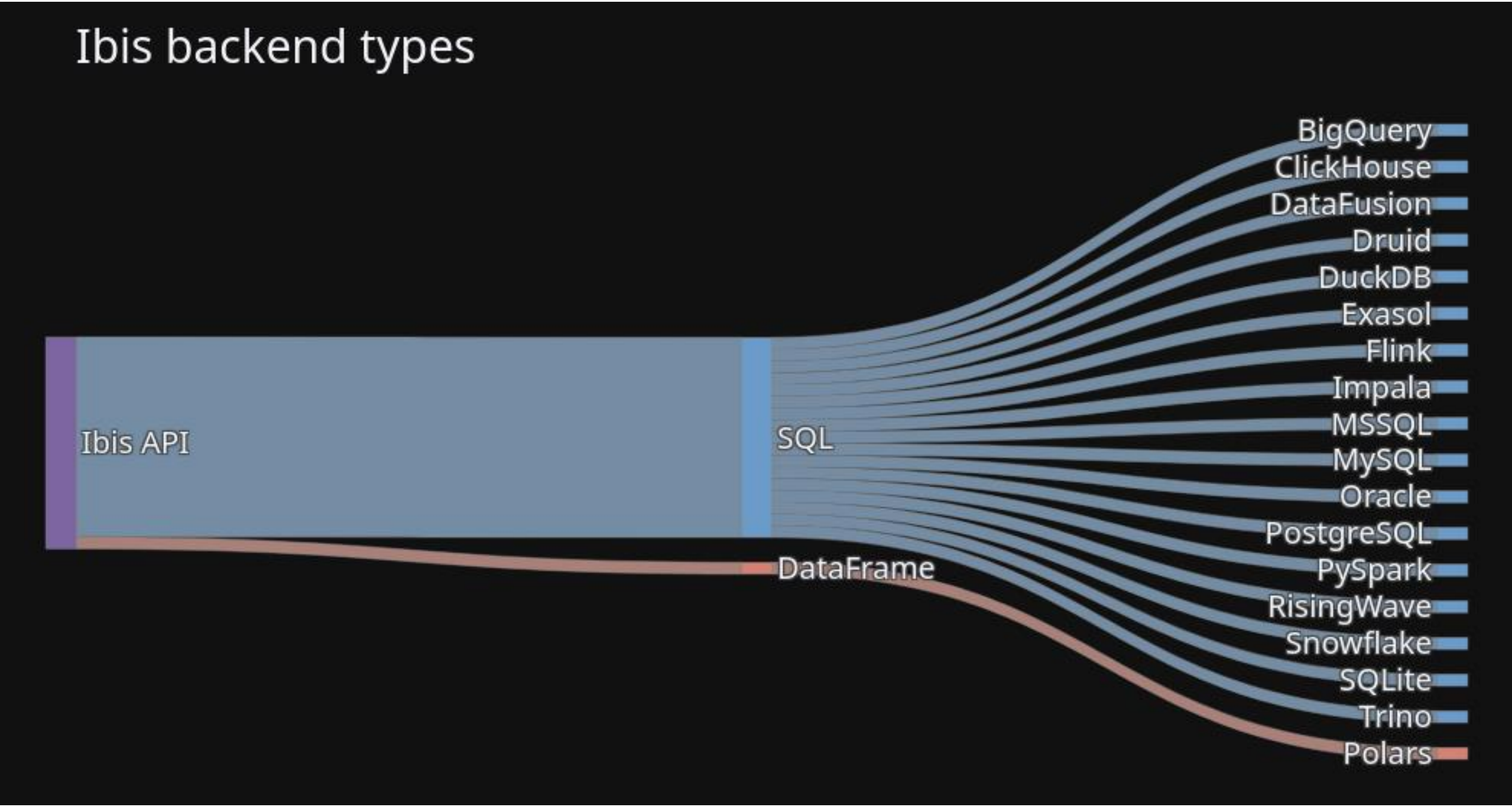
the portable Python dataframe library

Ibis is the frontend



Write Once, Run Anywhere




Ibis backend types
Ibis API
SQL
DataFrame
BigQuery
ClickHouse
DataFusion
Druid
DuckDB
Exasol
Flink
Impala
MSSQL
MySQL
Oracle
PostgreSQL
PySpark
RisingWave
Snowflake
SQLite
Trino
Polars

Tabular Data

Input

Filter height >100 and sort by mass

name	height	mass
string	int64	float64
Luke Skywalker	172	77.0
C-3PO	167	75.0
R2-D2	96	32.0
Darth Vader	202	136.0
Leia Organa	150	49.0

Output

name	height	mass
string	int64	float64
Leia Organa	150	49.0
C-3PO	167	75.0
Luke Skywalker	172	77.0
Darth Vader	202	136.0

Source: Scipy 2024 Ibis: because SQL is everywhere and so is Python

Query tabular data

```
df[df.height > 100].sort_values("mass")
```

```
df.filter(pl.col("height") > 100).sort(pl.col("mass"))
```

```
df.filter(df.height > 100).orderBy(df.mass).show()
```

Source: Scipy 2024 Ibis: because SQL is everywhere and so is Python

Ibis equivalent syntax

```
df.filter(df.height > 100).order_by(df.mass)
```

Source: Scipy 2024 Ibis: because SQL is everywhere and so is Python

SQL ain't standard

```sql
SELECT SUM(CAST(CONTAINS(LOWER("name"), 'darth') AS INT)) FROM starwars
```

```sql
SELECT SUM(CAST(STRPOS(LOWER("name"), 'darth') > 0 AS INT)) FROM "starwars"
```

```sql
SELECT SUM(CAST(STRPOS(LOWER(`name`), 'darth') > 0 AS INT64)) FROM `starwars`
```

```sql
SELECT SUM(IIF(CONTAINS(LOWER([name]), 'darth'), 1, 0)) FROM [starwars]
```

Source: Scipy 2024 Ibis: because SQL is everywhere and so is Python

You *could* do this...

```
SELECT
  {{ var.sum }}(
    {% if var.contains == 'strpos' %}
      CAST(
        {{ var.contains }}(LOWER({{ var.quote }}{{ var.name }}{{ var.quote }}), 'darth'){{
var.contains_suffix }} AS {{ var.cast_type }}
      )
    {% elif var.contains == 'CONTAINS' and var.quote == '[' %}
      IIF({{ var.contains }}(LOWER({{ var.quote }}{{ var.name }}{{ ']' }}), 'darth'), 1, 0)
    {% else %}
      CAST(
        {{ var.contains }}(LOWER({{ var.quote }}{{ var.name }}{{ var.quote }}), 'darth') AS {{
var.cast_type }}
      )
    {% endif %}
  )
FROM
  {{ var.quote }}{{ var.table }}{{ var.quote }}
```

Source: Scipy 2024 Ibis: because SQL is everywhere and so is Python

Ibis will do this for you

```
starwars.name.lower().contains("darth").sum()
```

DuckDB

Postgres

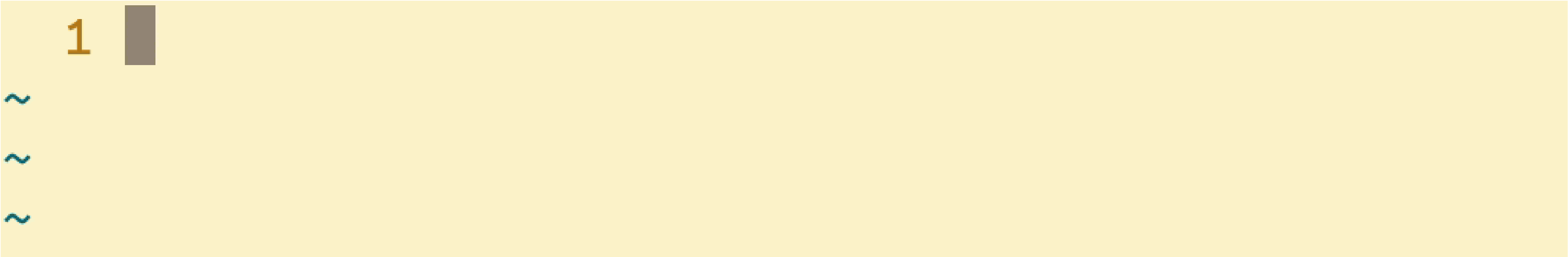
BigQuery

Snowflake

A lot of

Source: Scipy 2024 Ibis: because SQL is everywhere and so is Python

Building Queries



Source: Scipy 2024 Ibis: because SQL is everywhere and so is Python

Building Queries

```
1 _
~
~
~
~
```

```
1 █
~
~
~
```

```python
proj = t.select("two", "one")
```

```sql
SELECT
  "t0"."two",
  "t0"."one"
FROM "my_data" AS "t0"
```

Source: Scipy 2024 Ibis: because SQL is everywhere and so is Python

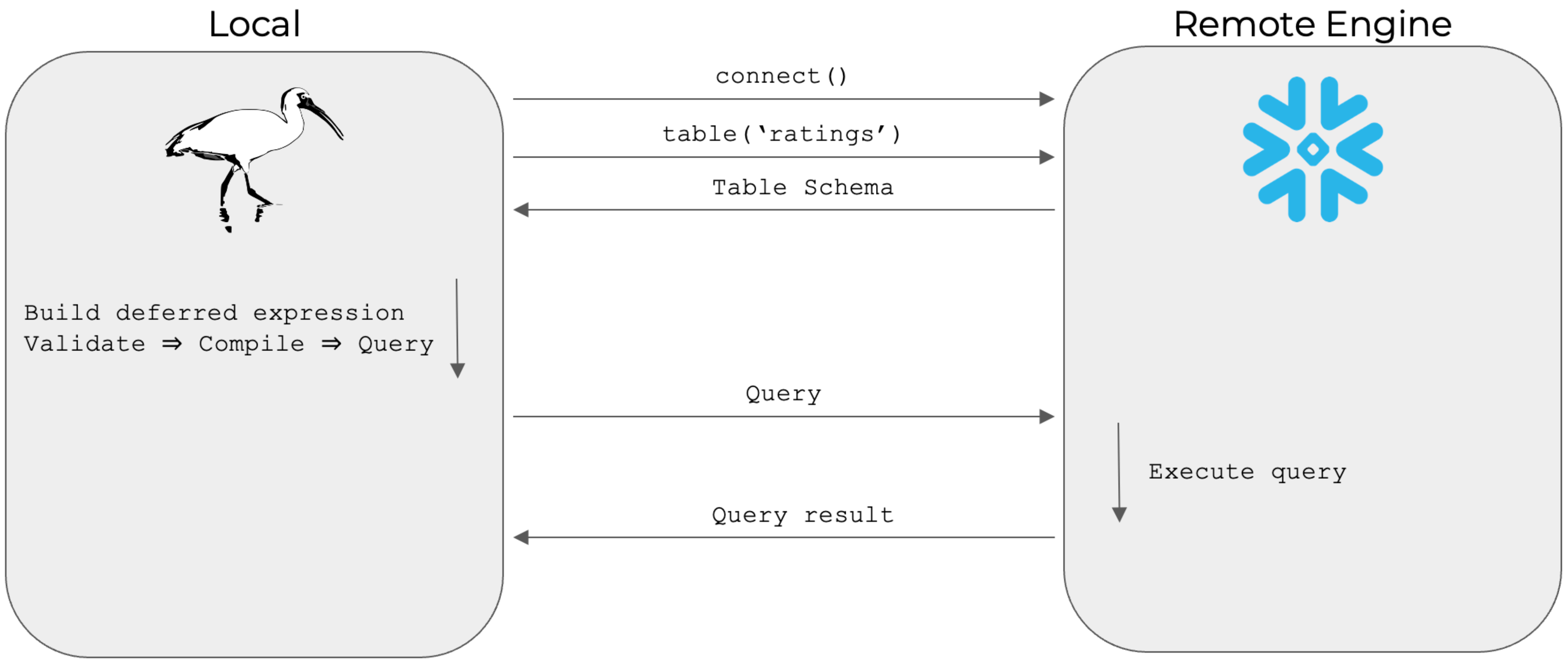
Mix SQL with expression

```python
revenue = con.sql("""
    select
        l_suppkey as supplier_no,
        sum(l_extendedprice * (1 - l_discount)) as total_revenue
    from
        lineitem
    where
        l_shipdate >= date '1996-01-01'
        and l_shipdate < date '1996-01-01' + interval '3' month
    group by
        l_suppkey
        """).alias("revenue")
```

```python
revenue.filter(revenue.total_revenue < 1000000)
```

supplier_no	total_revenue
int64	decimal(38, 4)
3701	999540.0360
6836	945408.3378
588	984253.3666
7300	736906.9482
5091	908724.7276
3636	869185.6532
4646	905186.8186
7820	838587.9808
5007	873000.0130
959	849076.6788
…	…

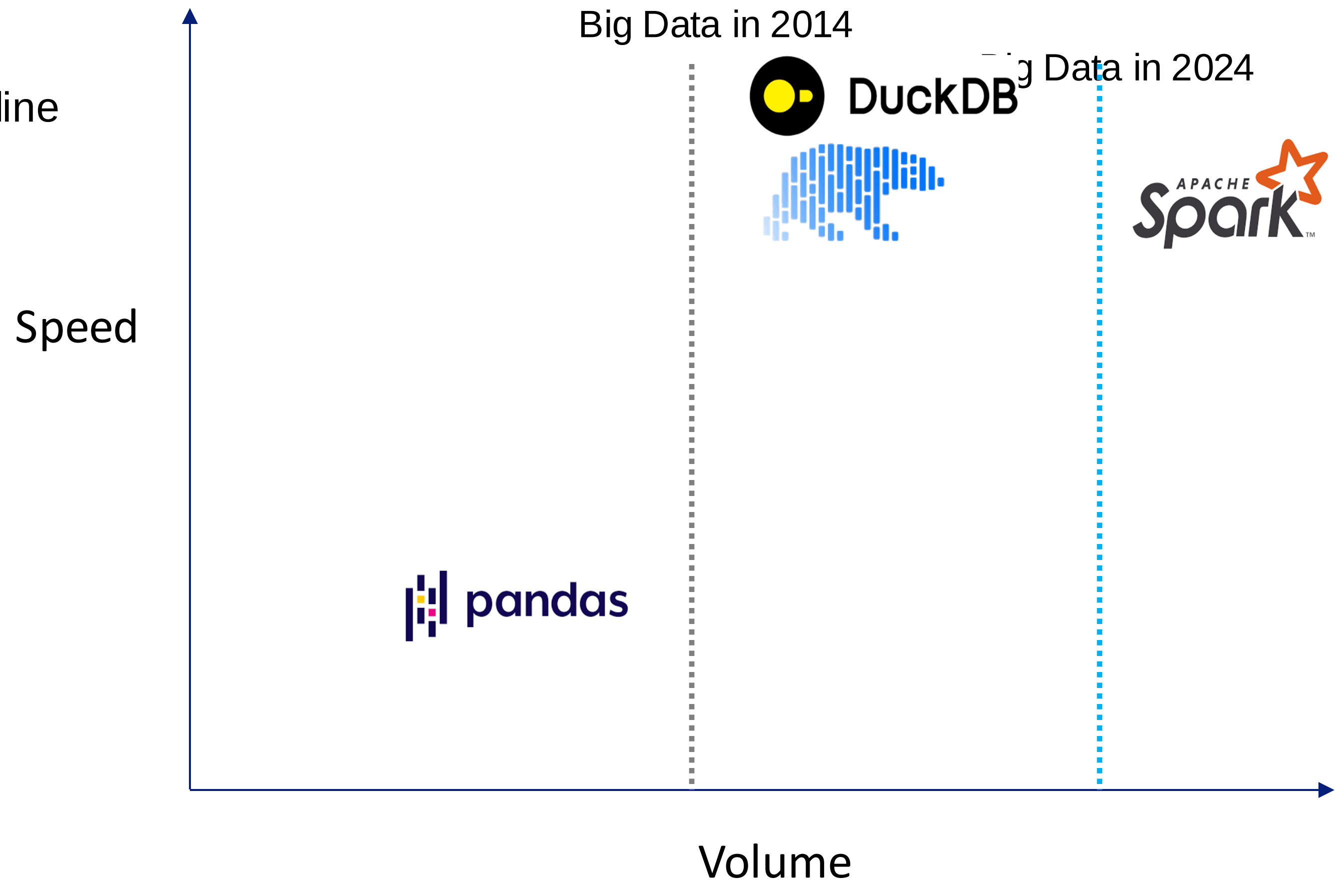
Ibis is mostly a “frontend” - Remote processing of remote data



Source: Scipy 2024 Ibis: because SQL is everywhere and so is Python

Conclusion

- Choose the right tool for the right workload
  - DuckDB, polars are very easy to deploy
- Is it necessary to run on Spark?
- ibis can help if you want to make sure your pipeline is engine agnostic, and flexible to scale.



SQL Translation

SQLGlot

```
sql = "SELECT CAST(a AS INTEGER), CAST(b as REAL(53)) from c where d>1 limit 5"
```

Oracle

?

SQL Translation

SQLGlot

```
sql = "SELECT CAST(a AS INTEGER), CAST(b as REAL(53)) from c where d>1 limit 5"
```

from sqlglot import transpile

Oracle

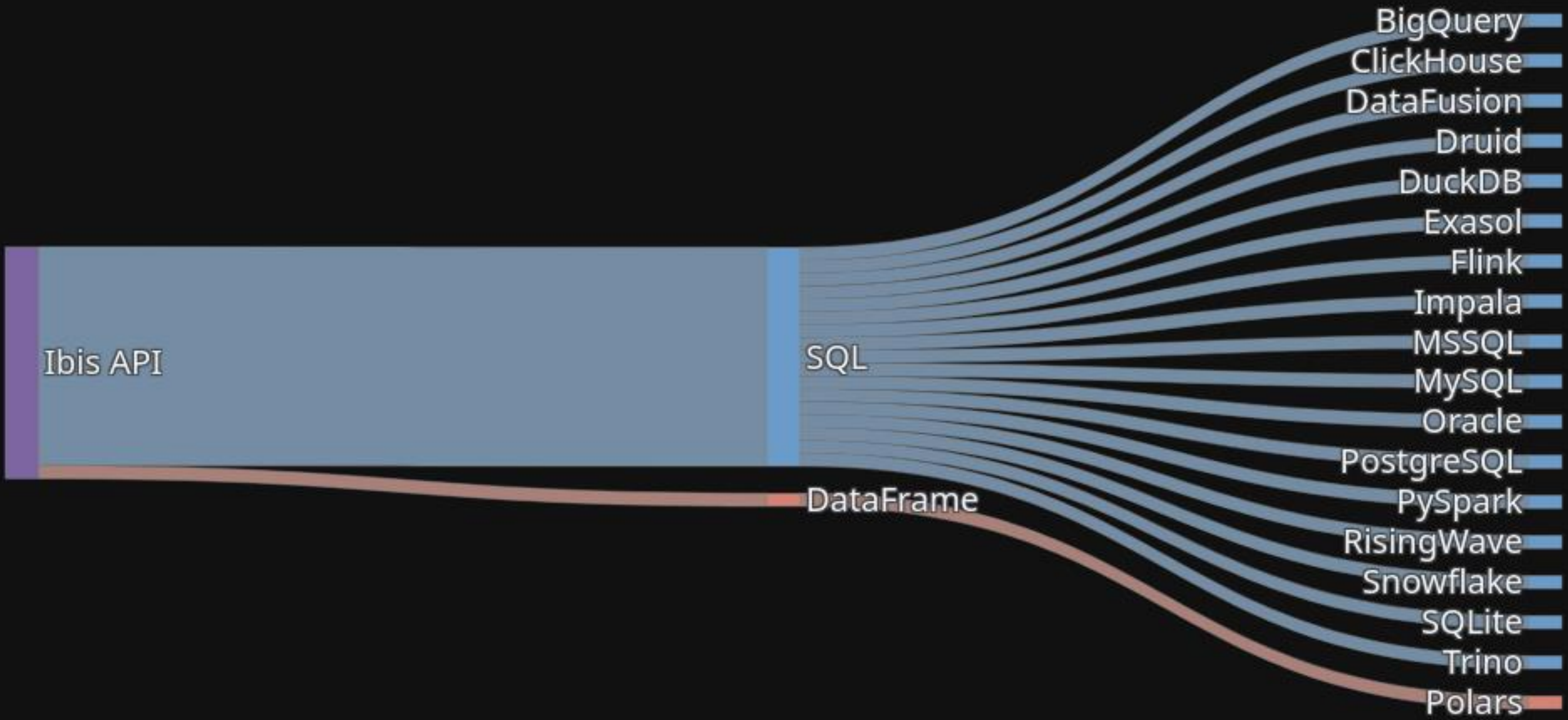
```
transpile(sql, write="oracle")
✓ 0.0s                                                        Python
['SELECT CAST(a AS NUMBER), CAST(b AS FLOAT(53)) FROM c WHERE d > 1 FETCH FIRST 5 ROWS ONLY']
```

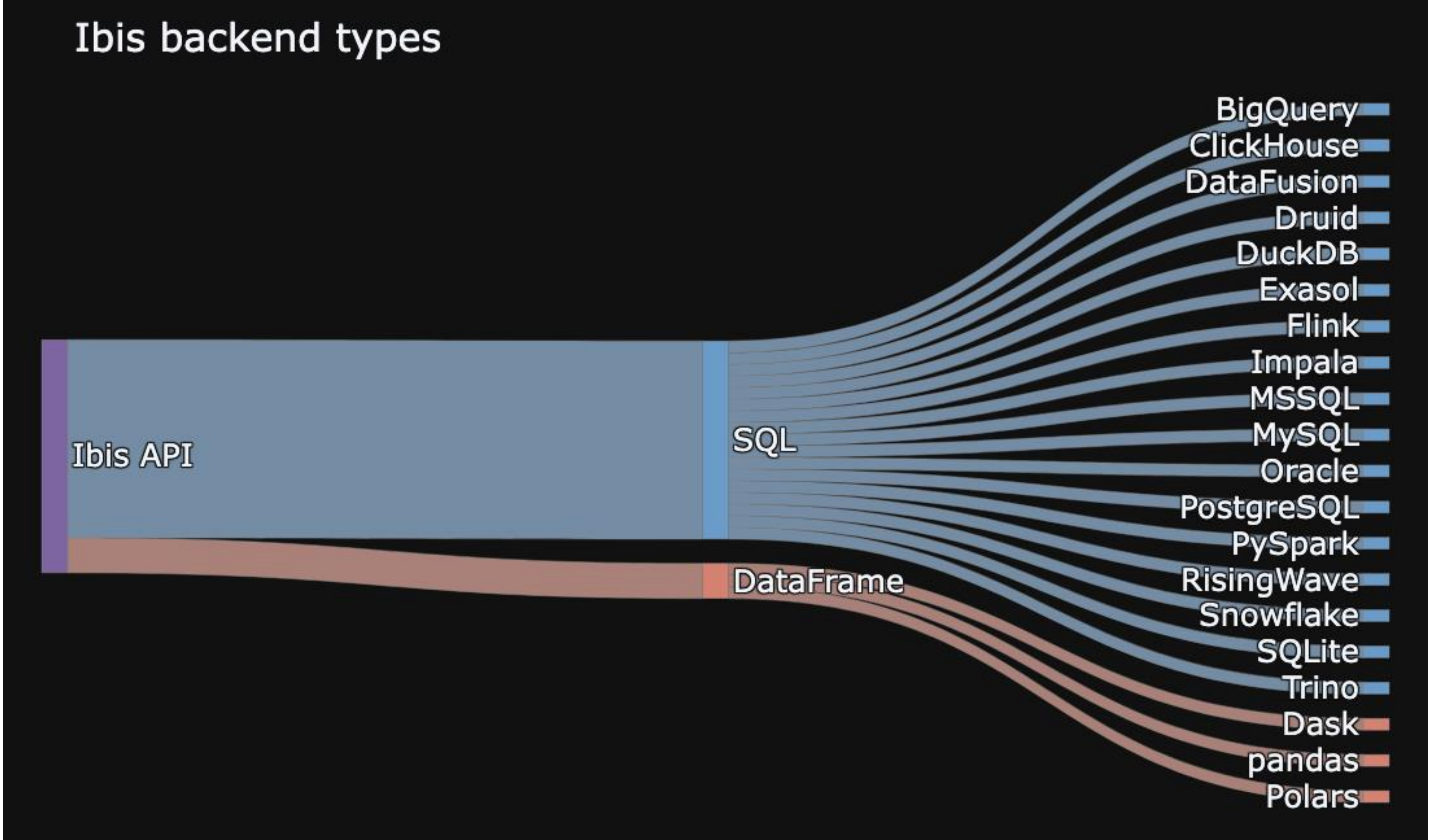
Reference

- Github Repo: https://github.com/noklam/pyconhk-2024
- Scipy 2024 Ibis: because SQL is everywhere and so is Python: https://cfp.scipy.org/2024/talk/CKVVA3/
- [Unified Stream/Batch Execution with Ibis]
- Big Data is Dead: https://motherduck.com/blog/big-data-is-dead/

Thank you

Ibis backend types




Ibis backend types
Ibis API
SQL
DataFrame
BigQuery
ClickHouse
DataFusion
Druid
DuckDB
Exasol
Flink
Impala
MSSQL
MySQL
Oracle
PostgreSQL
PySpark
RisingWave
Snowflake
SQLite
Trino
Dask
pandas
Polars

Dataframe?

- DataFrame is more powerful and expressive than SQL (string)
  - Support from programming language
    - Autocompletion
    - If-else control flow
    - Debuggable (put a breakpoint)
  - Most non-SQL engine are dataframe base:
    - Pandas
    - Spark
    - Polars

Dataframe is debuggable

- Multiple CTEs
  - Caching CTE as temp table

Questions?

https://ibis-project.org/

```
pip install ibis-framework
pip install ibis-framework[{backend}]

conda install -c conda-forge ibis-framework
                             ibis-bigquery
                             ibis-clickhouse
                             ibis-dask
                             ibis-datafusion
                             ibis-druid
                             ibis-duckdb
                             ibis-exasol
                             ibis-flink
                             ibis-impala
                             ibis-mssql
                             ibis-mysql
                             ibis-oracle
                             ibis-polars
                             ibis-postgres
                             ibis-pyspark
                             ibis-risingwave
                             ibis-snowflake
                             ibis-sqlite
                             ibis-trino
```

DuckDB:

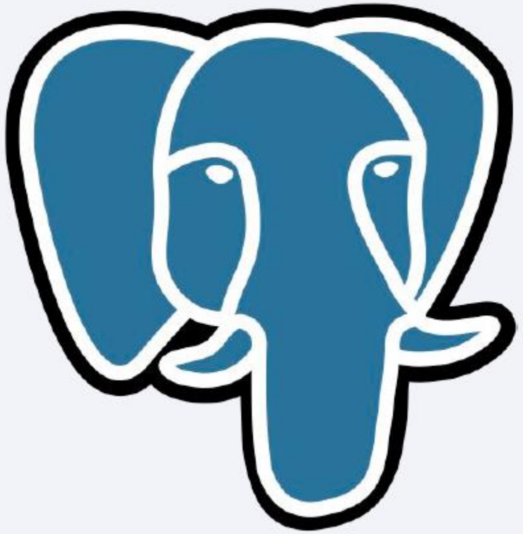
300 million rows.

40 seconds.

No supercomputer needed.

The parametrization problem

One big query?



Or many small(er) queries?