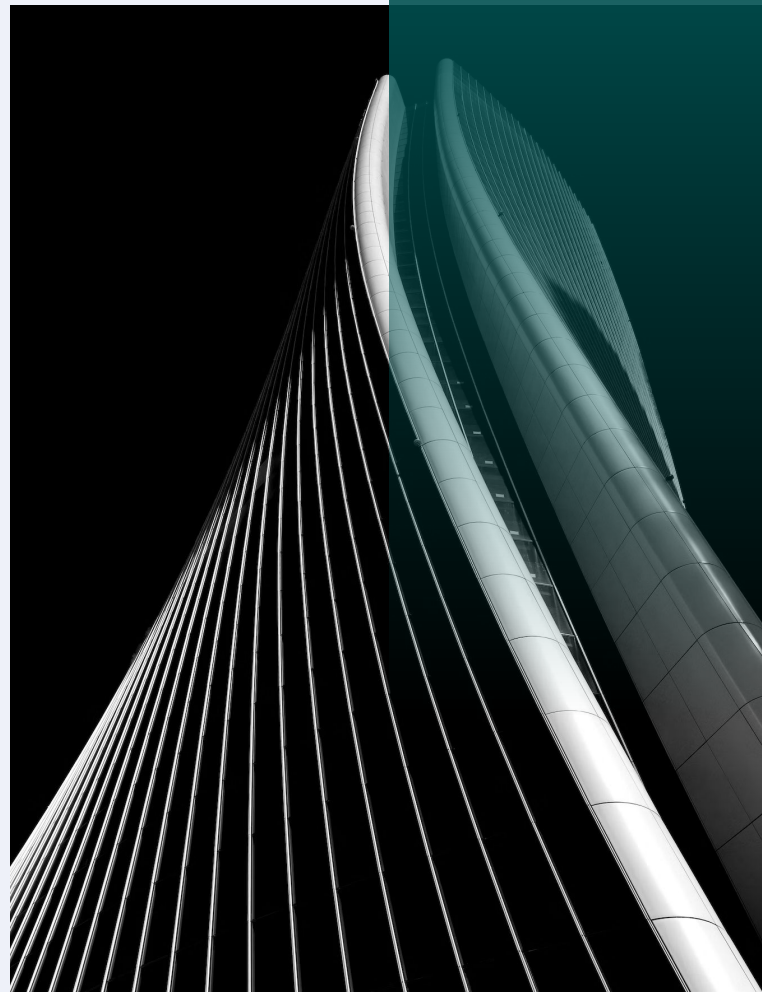



Deepyaman Datta

Unified Stream/Batch Execution with Ibis





Deepyaman Datta

Unified Stream/Batch Execution with Ibis



VOLTRON DATA

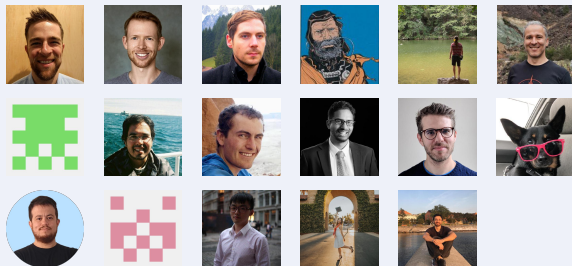
28 March, 2023

What is Ibis?

The portable Python dataframe library



3.9k stars
146 contributors



Have you ever...

...translated data analysis from pandas to PySpark?

...prototyped something in pandas, then thrown it over the wall to a data engineer?

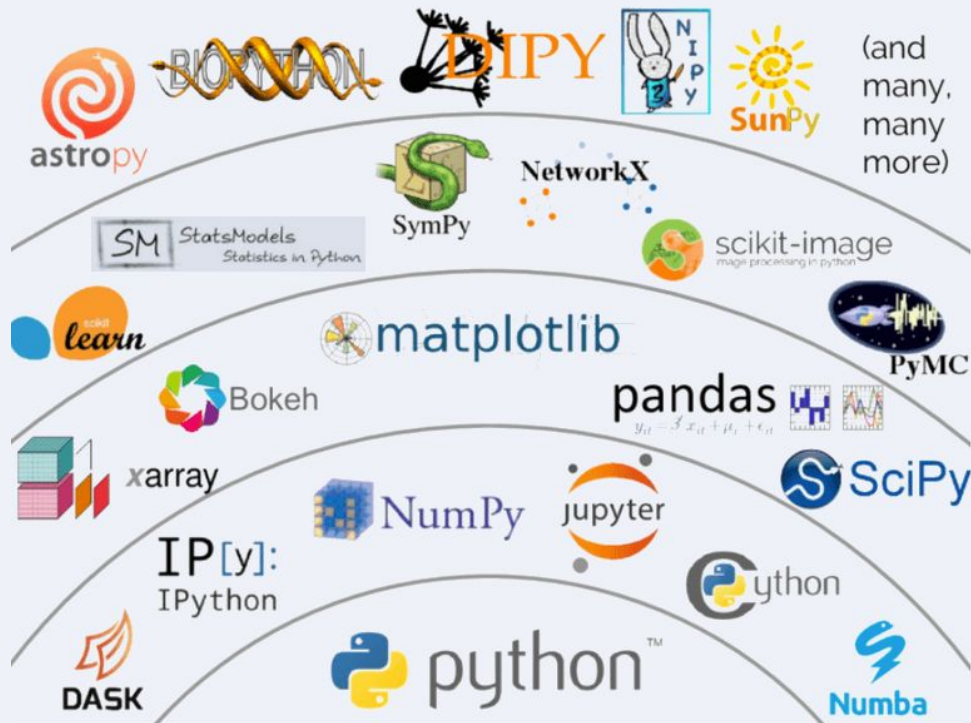
...received some pandas code that was thrown over a wall?

...used Parquet as a cross-language serialization format?

The PyData Stack

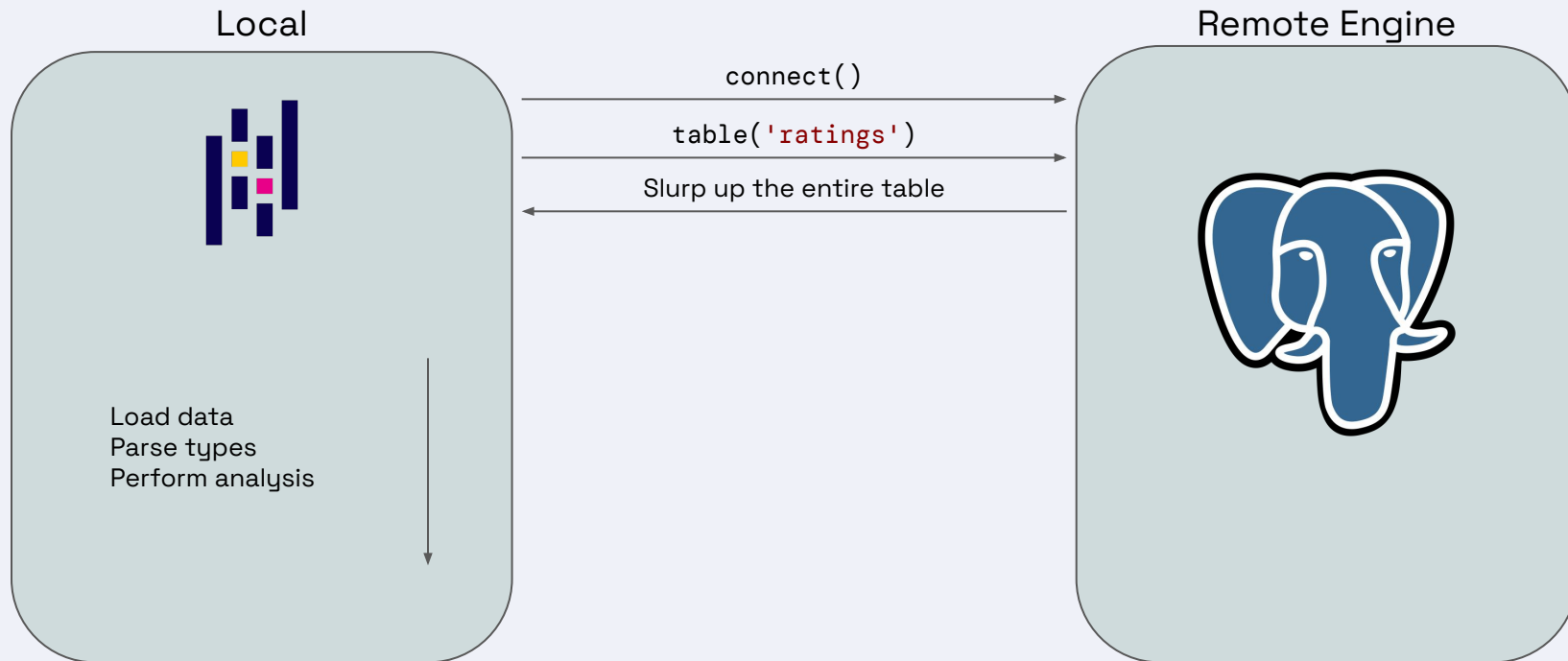
Data is local

Data fits in memory



Adapted from Jake VanderPlas, "The Unexpected Effectiveness of Python in Science", PyCon 2017

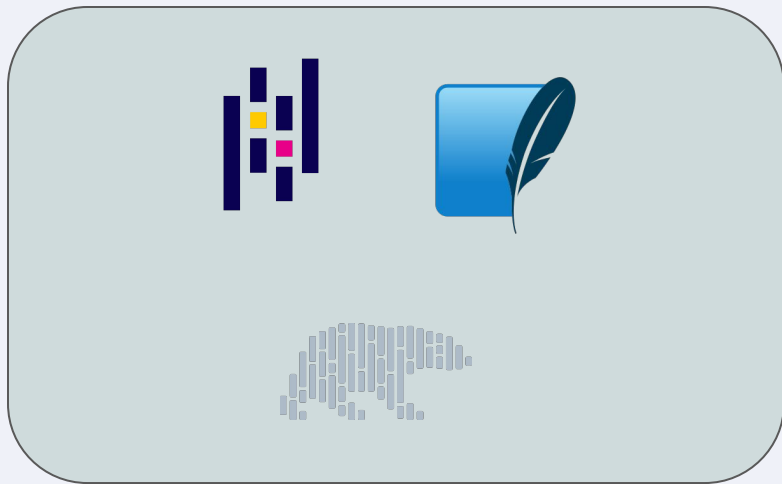
Local execution



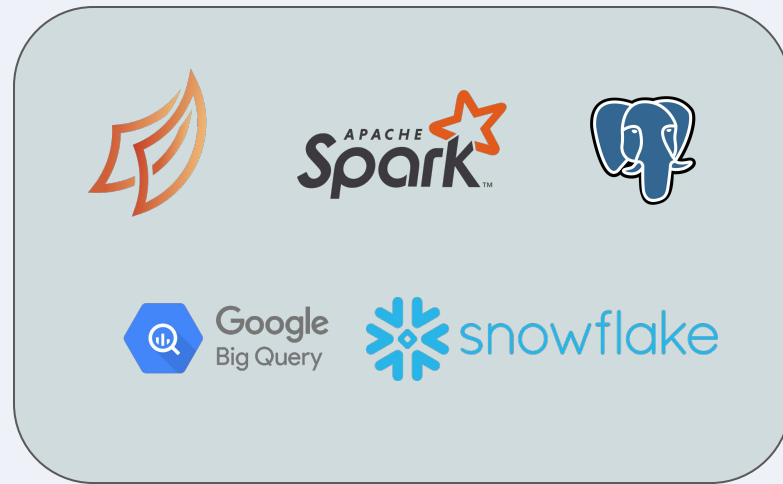
The PyData translation Problem

No one *wants* to write things twice, but...

Local / Dev



Distributed / Prod



**We need to talk
about SQL**

It's EVERYWHERE

It's between you and the data



SQL

Pros

Standardized[†]

Concise*

[†]Kind of, but also not really

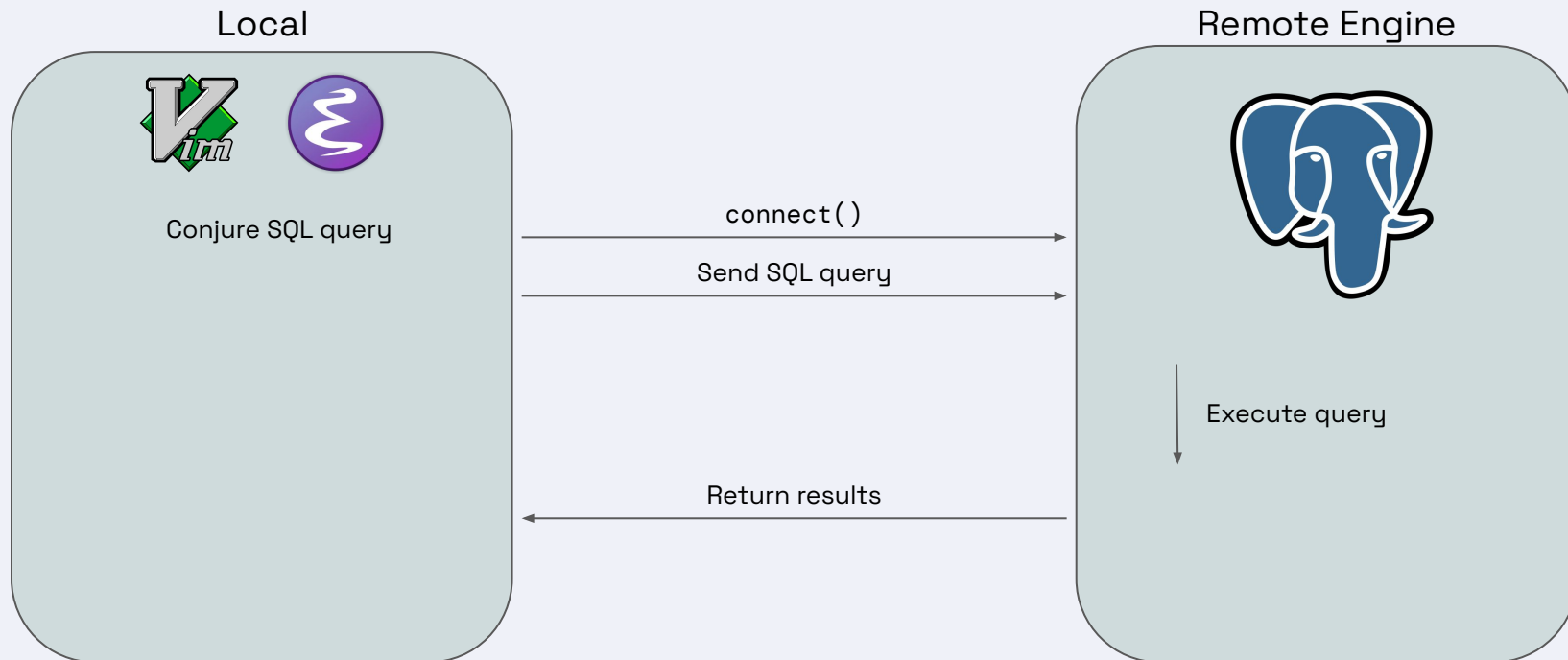
Cons

Effectively untestable

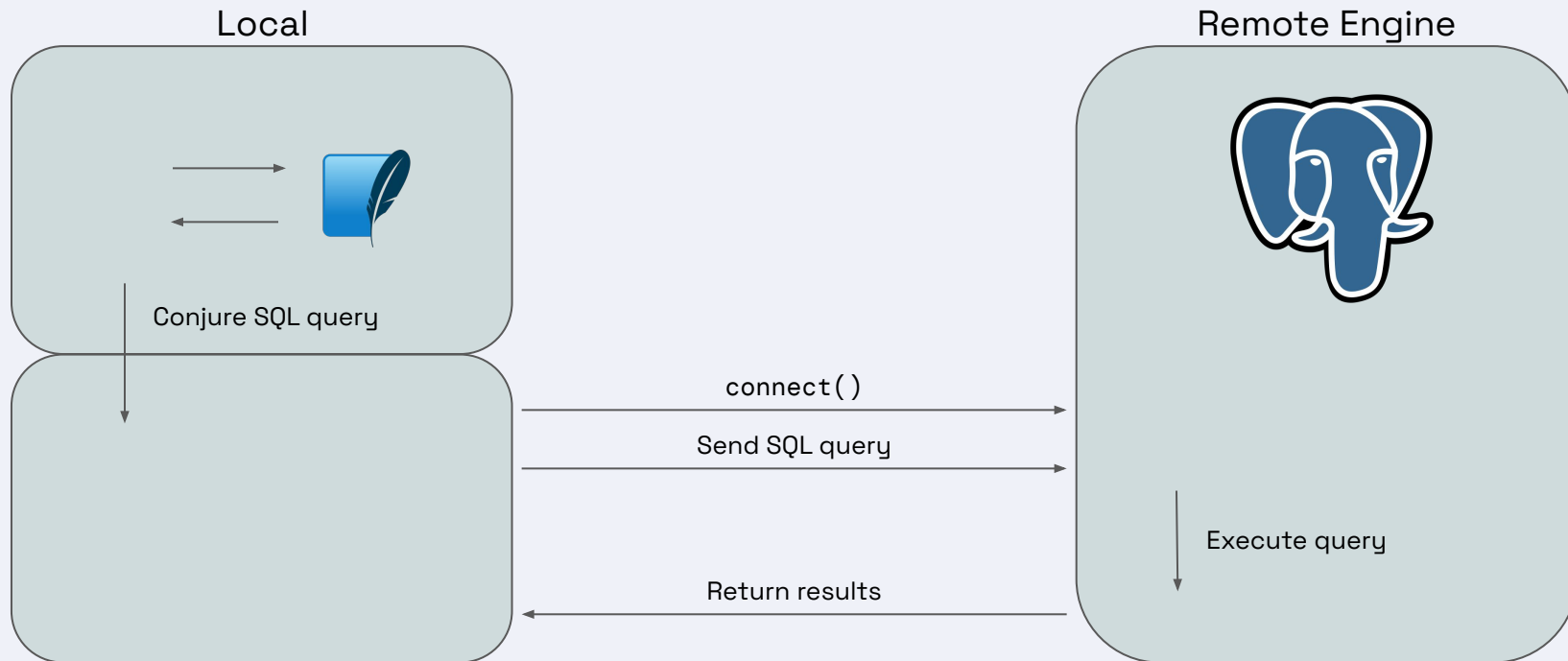
*Sometimes inscrutable

Fails at runtime

Remote execution (the good kind)



Remote execution (the good kind)



Problem solved

NARRATOR: It wasn't

The translation problem

The SQL standard is a standard, but how standard are standards?

tconst	averageRating	numVotes
string	string	string
tt0000001	5.7	1919\n
tt0000002	5.8	260\n
tt0000003	6.5	1726\n
tt0000004	5.6	173\n
tt0000005	6.2	2541\n
tt0000006	5.1	175\n
tt0000007	5.4	797\n
tt0000008	5.4	2061\n
tt0000009	5.2	200\n
tt0000010	6.9	6949\n
...



tconst	avg_rating	num_votes
string	float64	int64
tt0000001	5.7	1919
tt0000002	5.8	260
tt0000003	6.5	1726
tt0000004	5.6	173
tt0000005	6.2	2541
tt0000006	5.1	175
tt0000007	5.4	797
tt0000008	5.4	2061
tt0000009	5.2	200
tt0000010	6.9	6949
...

The translation problem

SQLite

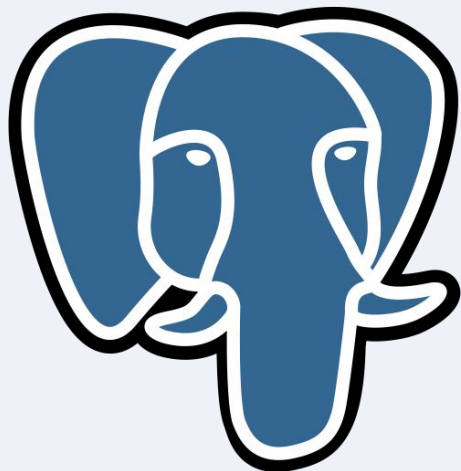
```
SELECT
  tconst,
  CAST(averageRating AS REAL(53)) AS avg_rating,
  CAST(numVotes AS INTEGER) AS num_votes
FROM ratings
```

PostgreSQL

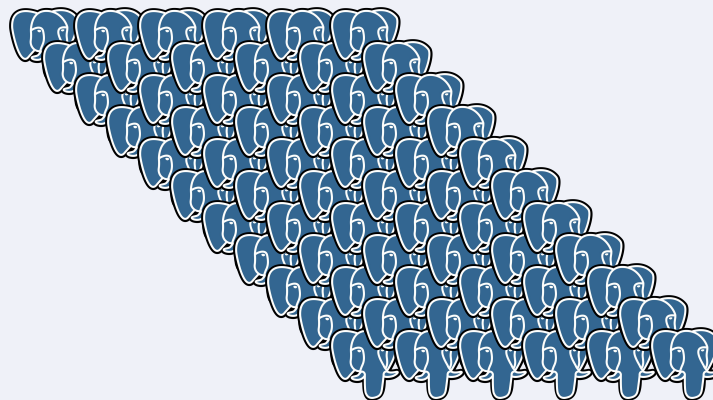
```
SELECT
  tconst,
  CAST(averageRating AS DOUBLE PRECISION) AS avg_rating,
  CAST(numVotes AS BIGINT) AS num_votes
FROM ratings
```

The parametrization problem

One big query?



Or many small(er) queries?



What's ~~the~~ a solution?

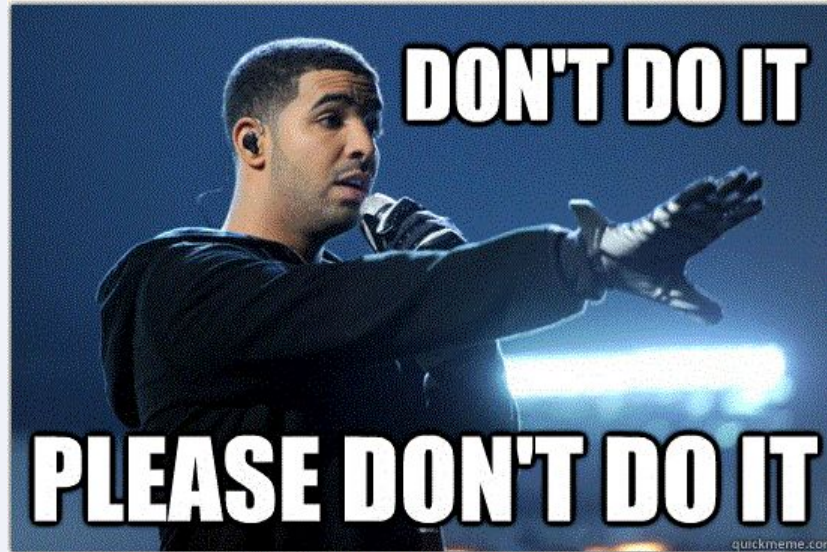
Presented with:

- The translation problem
- The parameterization problem
- Want to use Python
- Don't want to write a bunch of SQL

“

I know! I'll generate strings.”

- **Everyone, at some point in their lives**



For a more nuanced treatment of the topic, see [Gil Forsyth's talk at PyData NYC](#) or [Phillip Cloud's presentation at Trino Fest](#)

What if...

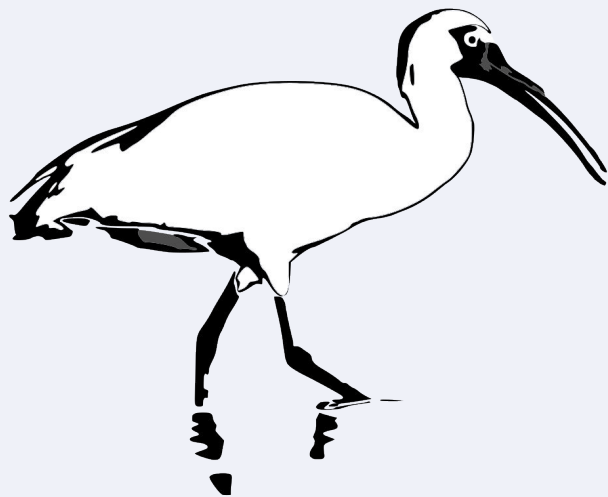
...instead of generating strings, you could write concise Python code that type checks and *eventually* generates strings?

Enter the Ibis

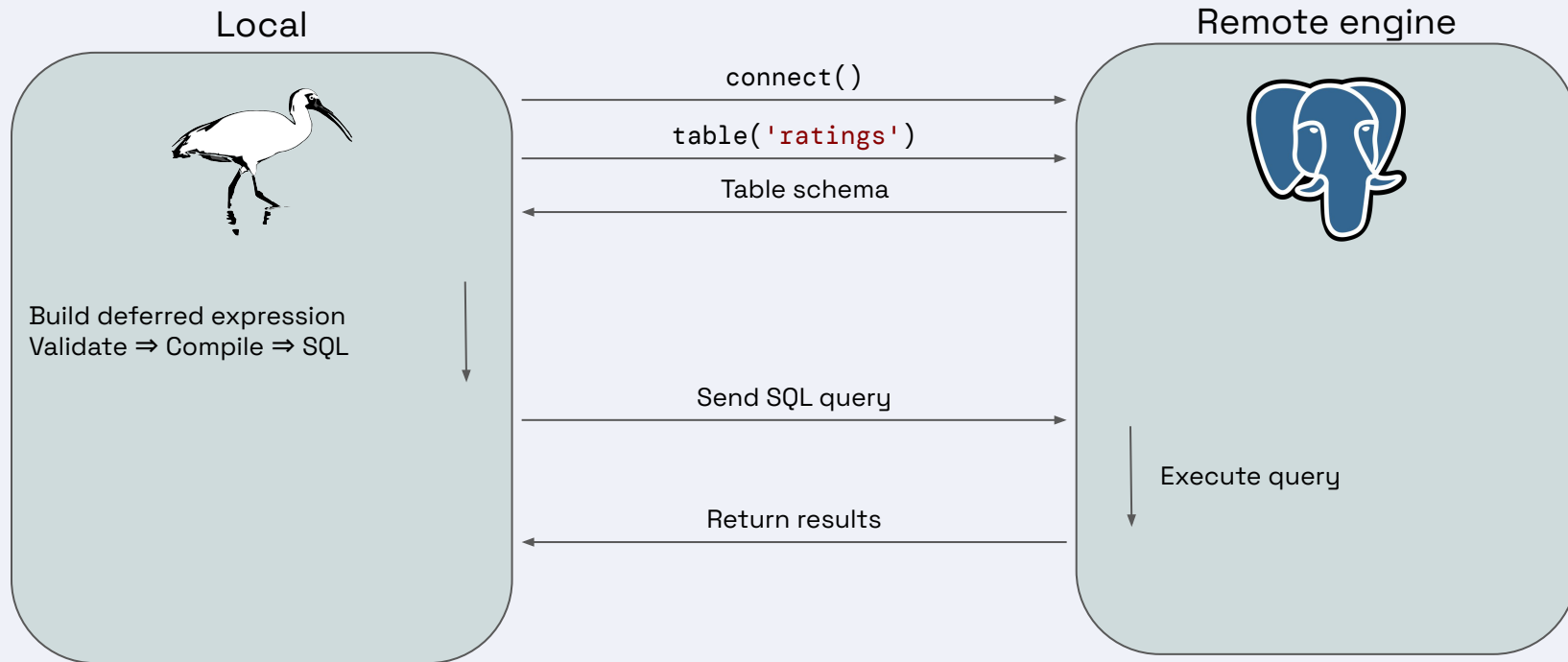
A lightweight Python library
for data wrangling

Provides:

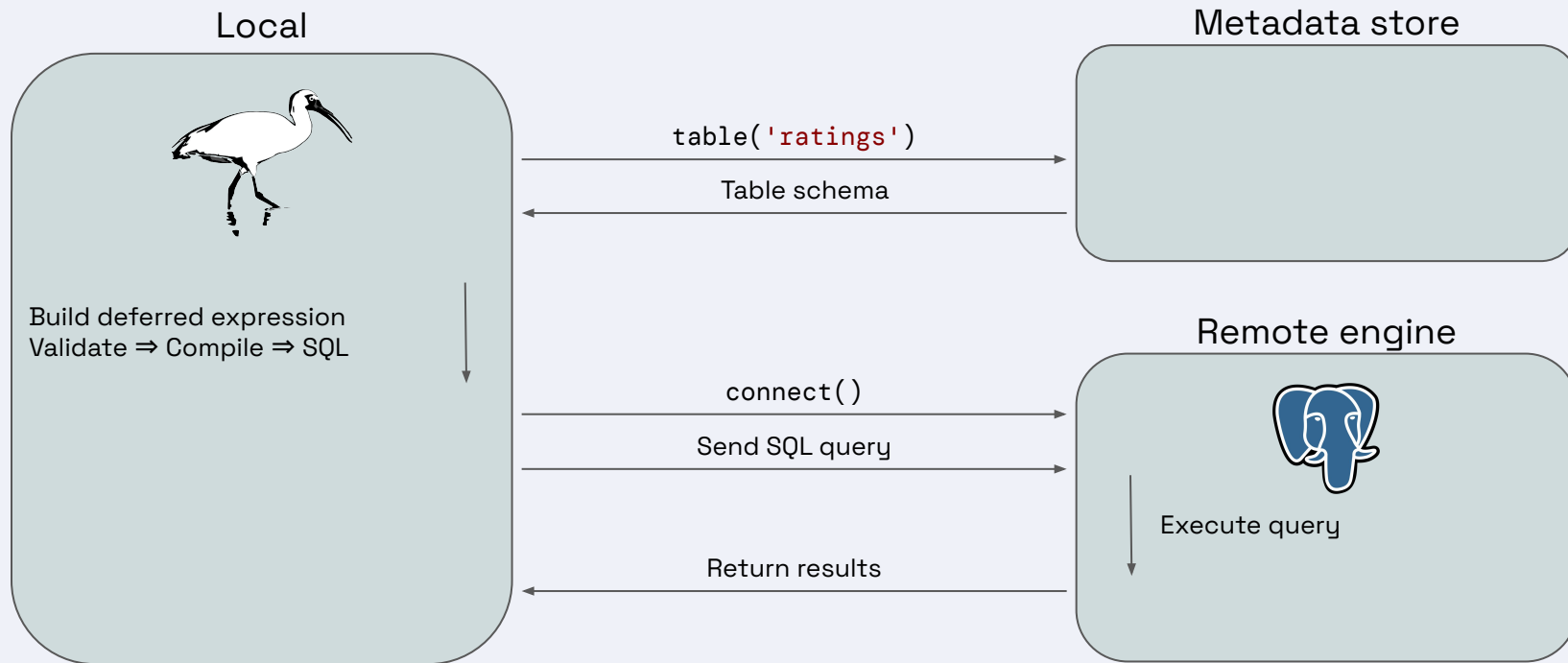
- Pythonic dataframe API
- Interfaces to 20+ query languages
- Deferred execution model



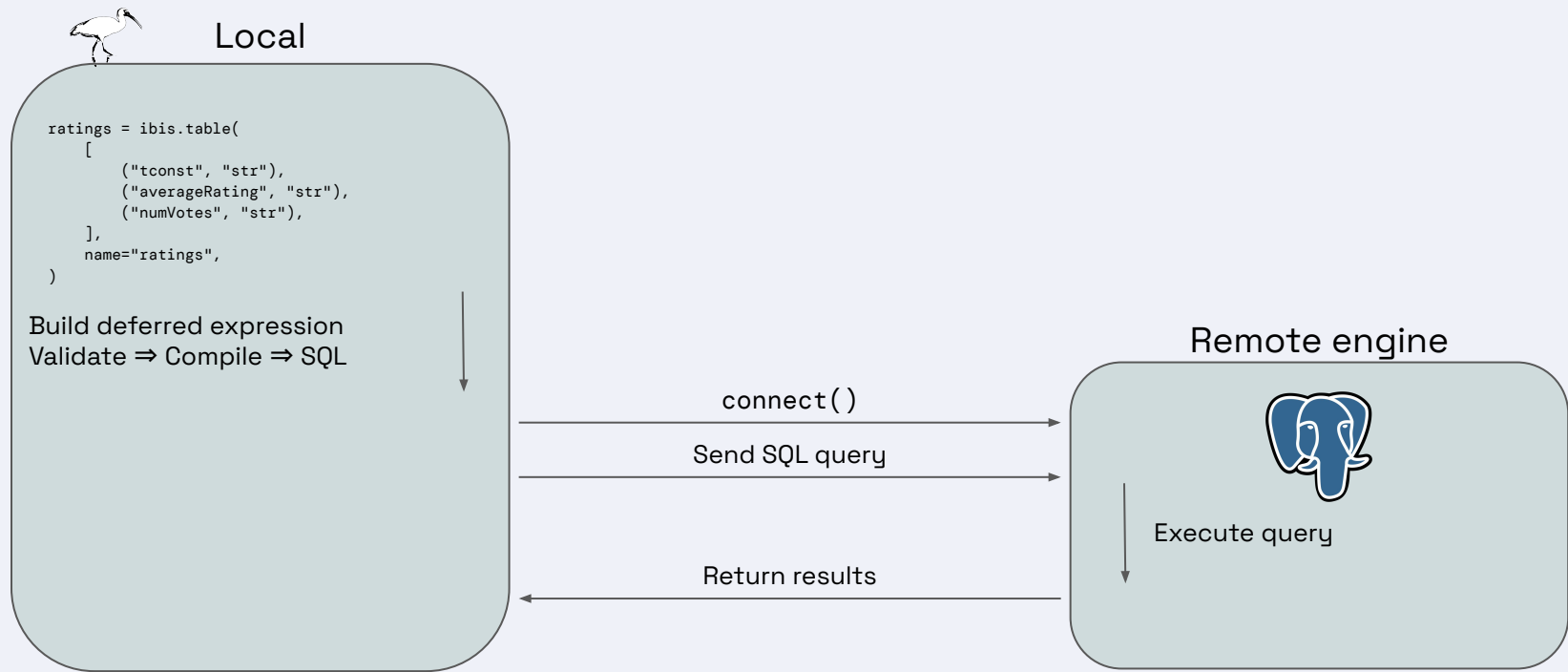
Deferred execution



Deferred execution



Deferred execution



“

I thought this talk was about streaming?”

Have you ever...

...translated data analysis from pandas to (Py)Flink?

...prototyped something in pandas, then thrown it over the wall to a data engineer?

...duplicated the same logic in batch and streaming contexts?

...backfilled a streaming feature on a batch backing table?

Streaming is different

Things that don't exist or aren't as important in batch:

- Streaming sources (append-only logs and changelogs) and sinks
- Time semantics (event time vs. processing time, watermarks)
- Table-valued functions (TVFs)
- Temporal (as-of) joins

This example uses mock payments data. The `payment_msg` Kafka topic contains messages in the following format:

```
{
  "createTime": "2023-09-20 22:19:02.224",
  "orderId": 1695248388,
  "payAmount": 88694.71922270155,
  "payPlatform": 0,
  "provinceId": 6
}
```

```
[ ]: from itertools import islice

from kafka import KafkaConsumer

consumer = KafkaConsumer("payment_msg")
for msg in islice(consumer, 3):
    print(msg)
```

```
[ ]: import json

import pandas as pd

payments_df = pd.DataFrame(json.loads(row.value) for row in islice(consumer, 100))
payments_df["createTime"] = pd.to_datetime(payments_df["createTime"])
payments_df
```

```
[ ]: import ibis

con = ibis.duckdb.connect()
t = con.create_table("payments", payments_df)
t
```

```
[ ]: agged = t.select(
    "provinceId",
    pay_amount=t.payAmount.sum().over(
        range=(-ibis.interval(seconds=10), 0),
        group_by=t.provinceId,
        order_by=t.createTime,
    ),
)
agged
```

Supported backends

- BigQuery
- ClickHouse
- Dask
- DataFusion
- Druid
- DuckDB
- Exasol
- Flink
- Impala
- MSSQL
- MySQL
- Oracle
- pandas
- Polars
- PostgreSQL
- PySpark
- RisingWave
- Snowflake
- SQLite
- Trino

Scale from dev to prod with less rewriting

BUT: There are no golden tickets

- Floating point math exists
- Only SQL backends can execute SQL
- Some backend differences can't be abstracted away (regexes, data-dependent function behavior)

Maintain the same logic between streaming and batch

BUT: Streaming and batch are still different

- Some batch operators don't make sense in a streaming context

What's next?

- Time travel support
- Iceberg integration
- Pattern recognition
- Additional streaming backends

Try Ibis today!



<https://ibis-project.org>



ibis-project/ibis



ibisData



ibis-project



Phillip in the Cloud
cpcloud

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

```
conda install -c conda-forge ibis-framework  
conda install -c conda-forge ibis-[{backend}]
```

