Microsoft

Magpie

Python at Speed and Scale using Cloud Backends

Alekh Jindal, K. Venkatesh Emani, Maureen Daum, Olga Poppe, Brandon Haynes, Anna Pavlenko, Ayushi Gupta, Karthik Ramachandra, Carlo Curino, Andreas Mueller, Wentao Wu, Hiren Patel

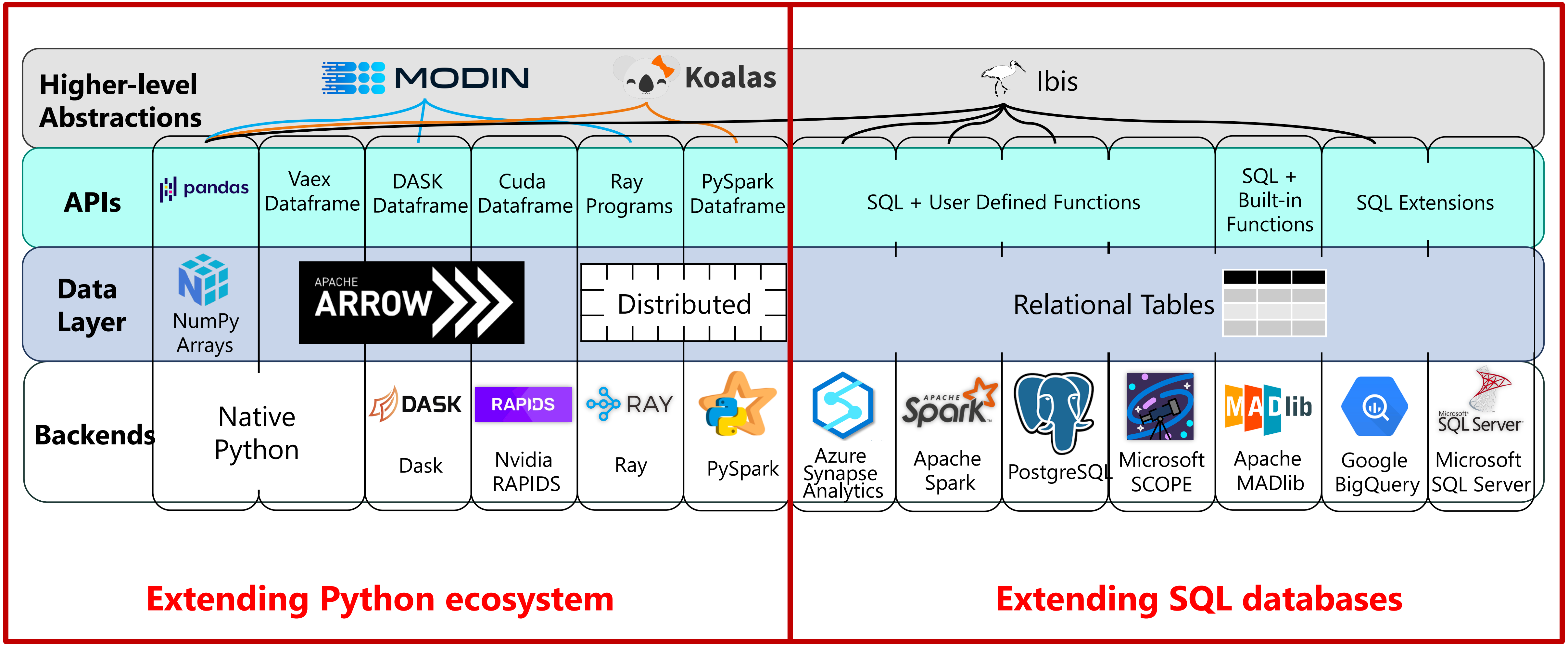
Gray Systems Lab, Microsoft
Azure Data, Microsoft
Microsoft Research
University of Washington

The Python and The Cloud

- De-facto for ad-hoc analysis
- Pandas dataframes highly popular
- Performance is a challenge

- Hyper-scale performance
- Several SQL processing backends
- Enterprise data already on cloud

The current landscape ... is a fragmented jungle!



Higher-level Abstractions

MODIN

Koalas

Ibis

APIs

pandas

Vaex Dataframe

DASK Dataframe

Cuda Dataframe

Ray Programs

PySpark Dataframe

SQL + User Defined Functions

SQL + Built-in Functions

SQL Extensions

Data Layer

NumPy Arrays

APACHE ARROW

Distributed

Relational Tables

Backends

Native Python

Dask

Nvidia RAPIDS

Ray

PySpark

Azure Synapse Analytics

Apache Spark

PostgreSQL

Microsoft SCOPE

Apache MADlib

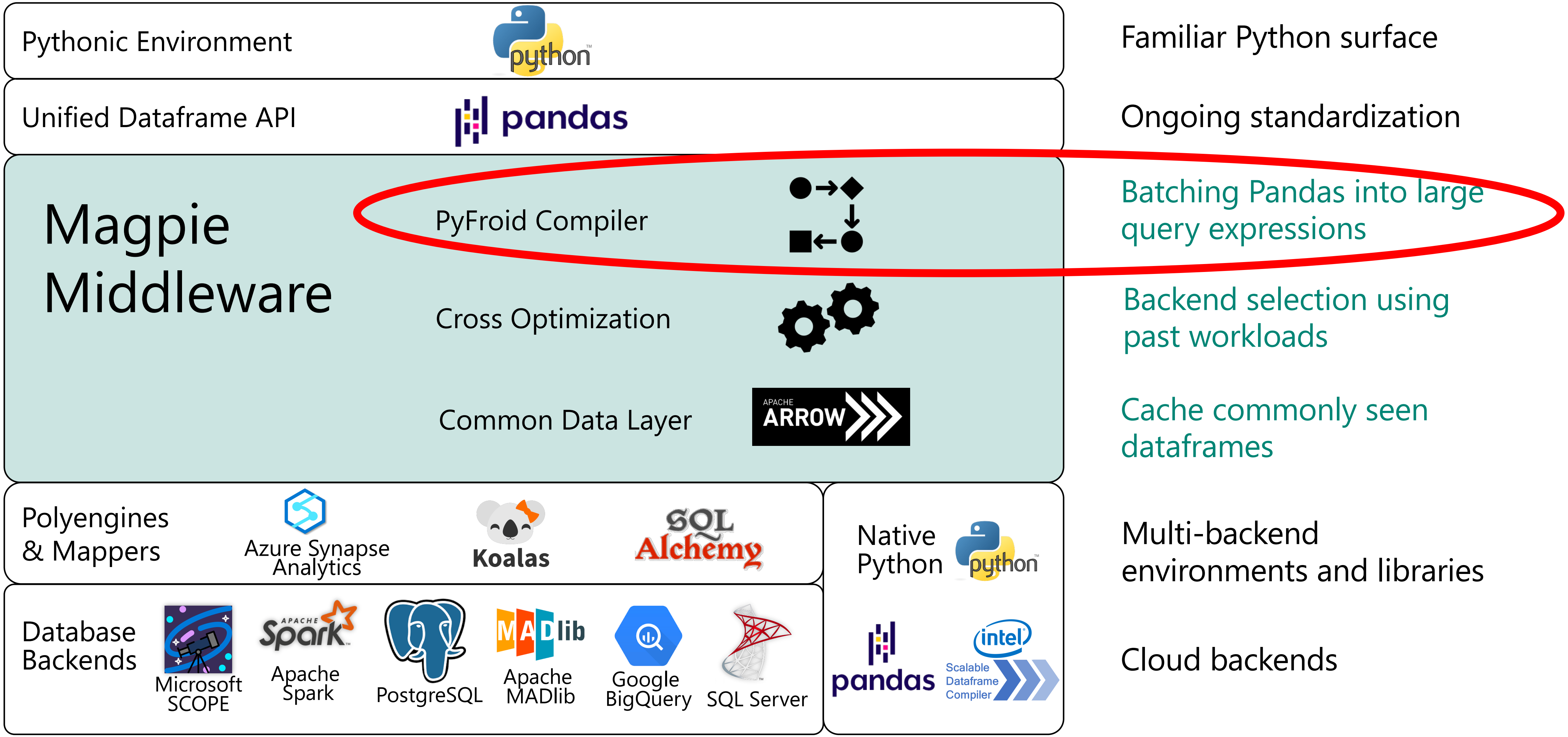
Google BigQuery

Microsoft SQL Server

Extending Python ecosystem

Extending SQL databases

Magpie



Pythonic Environment

Familiar Python surface

Unified Dataframe API

Ongoing standardization

Magpie Middleware

PyFroid Compiler

Batching Pandas into large query expressions

Cross Optimization

Backend selection using past workloads

Common Data Layer

Cache commonly seen dataframes

Polyengines & Mappers

Azure Synapse Analytics

Koalas

Native Python

Multi-backend environments and libraries

Database Backends

Microsoft SCOPE

Apache Spark

PostgreSQL

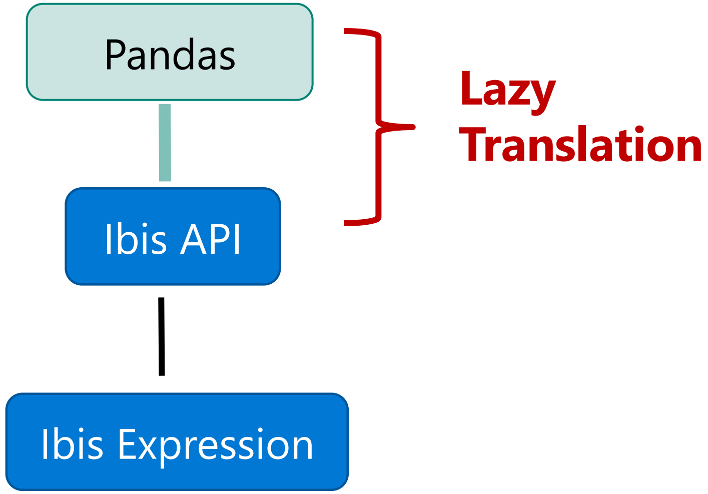
Apache MADlib

Google BigQuery

SQL Server

Cloud backends

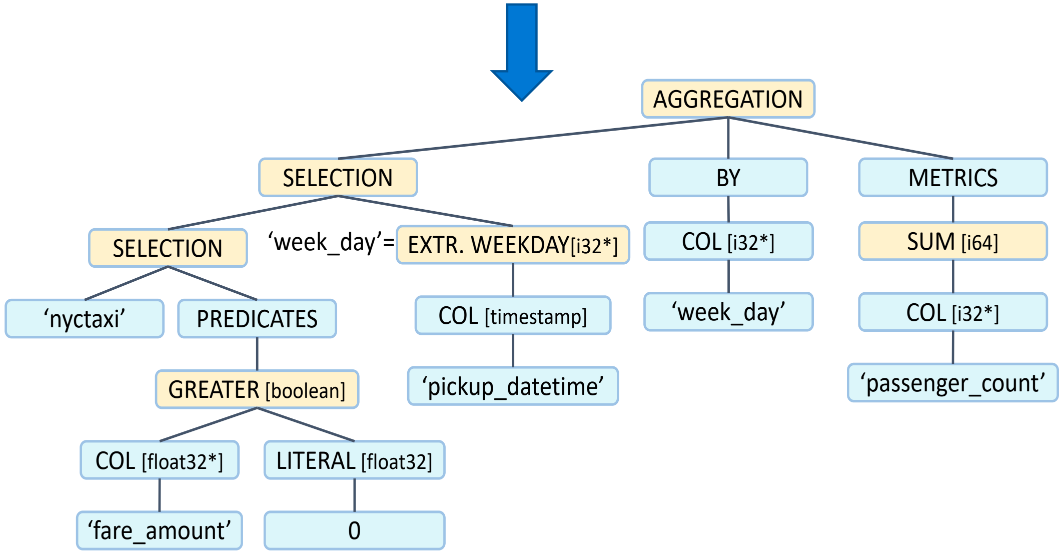
Batching Pandas



The number of taxi trips per weekday over the NYC Taxi dataset

```
import pyfroid.pandas as pd                              # vs import pandas as pd
df = pd.read_sql(‘nyctaxi’, con)                         # fetch data
df = df[df.fare_amount > 0]                              # filter bad rows
df[‘day’] = df.pickup_datetime.dt.dayofweek              # add features
df = df.groupby([‘day’])[‘passenger_count’].sum()        # aggregation
print(df)                                                # use dataframe
```

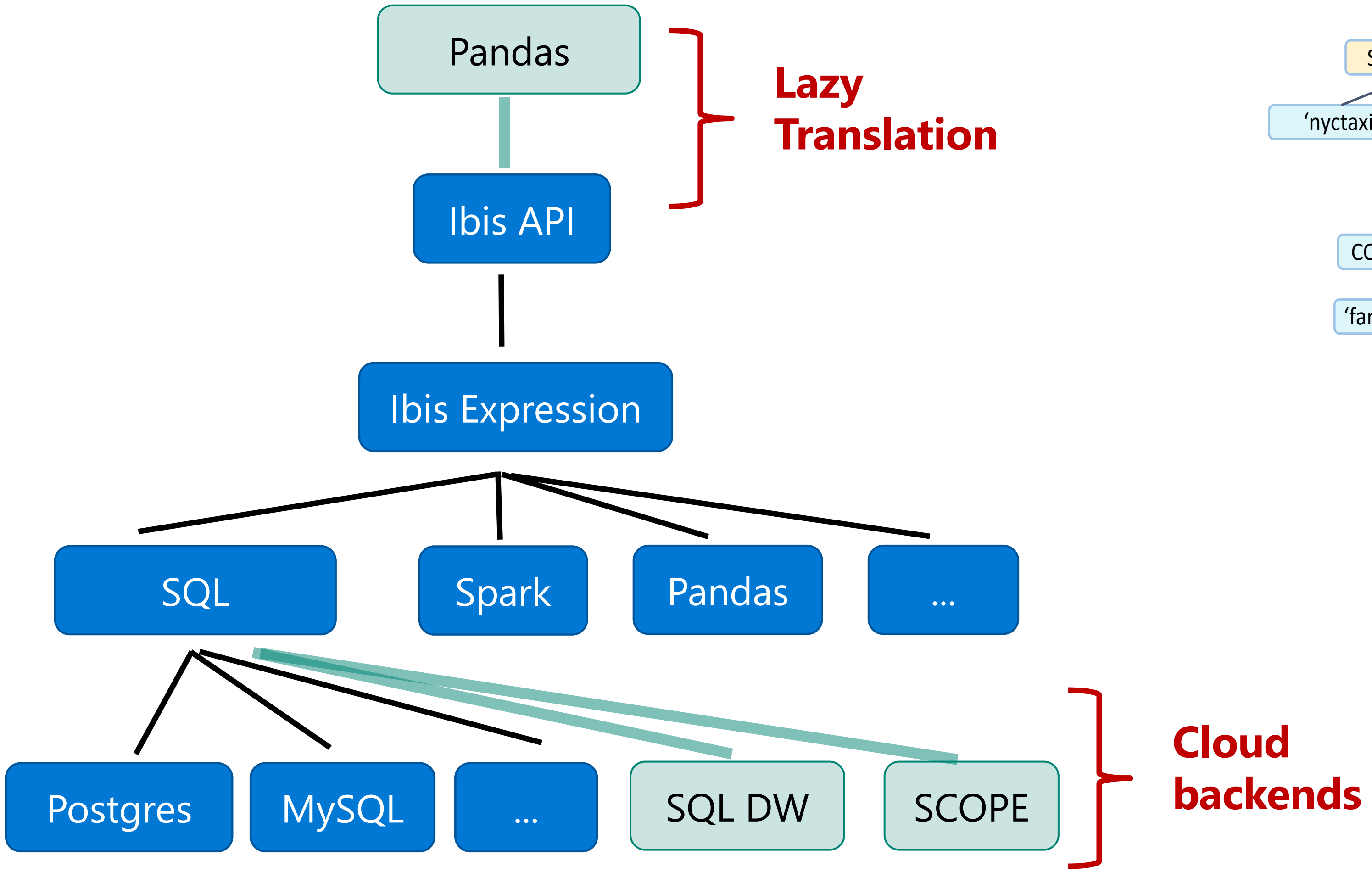
Pandas Dataframe Program



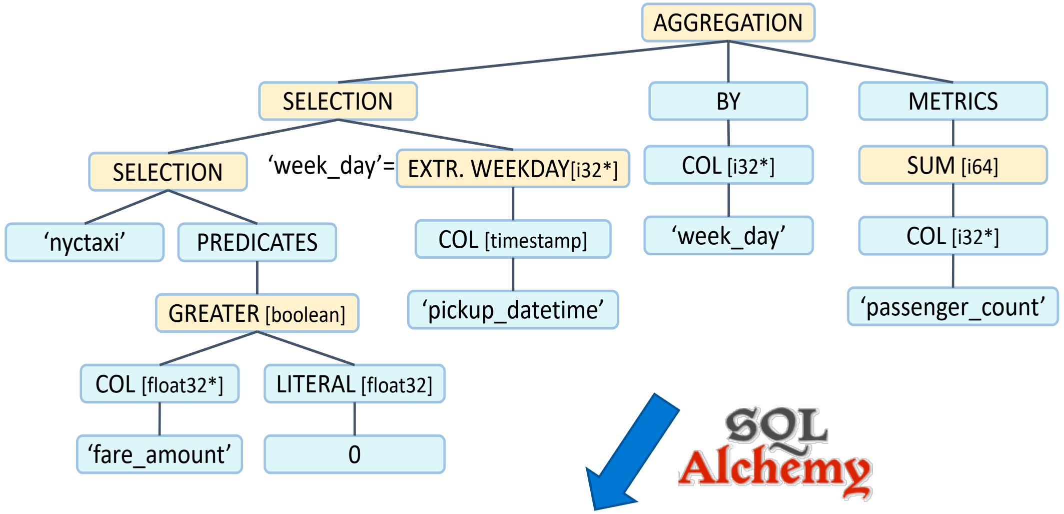
Intermediate Representation

Blue parts: already in IBIS, **Green** parts: our contributions

Pushing Data Science down

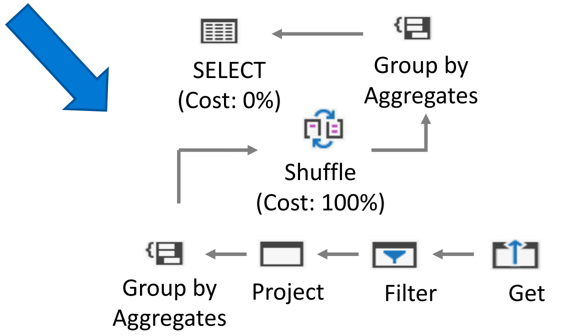


Blue parts: already in IBIS, **Green** parts: our contributions



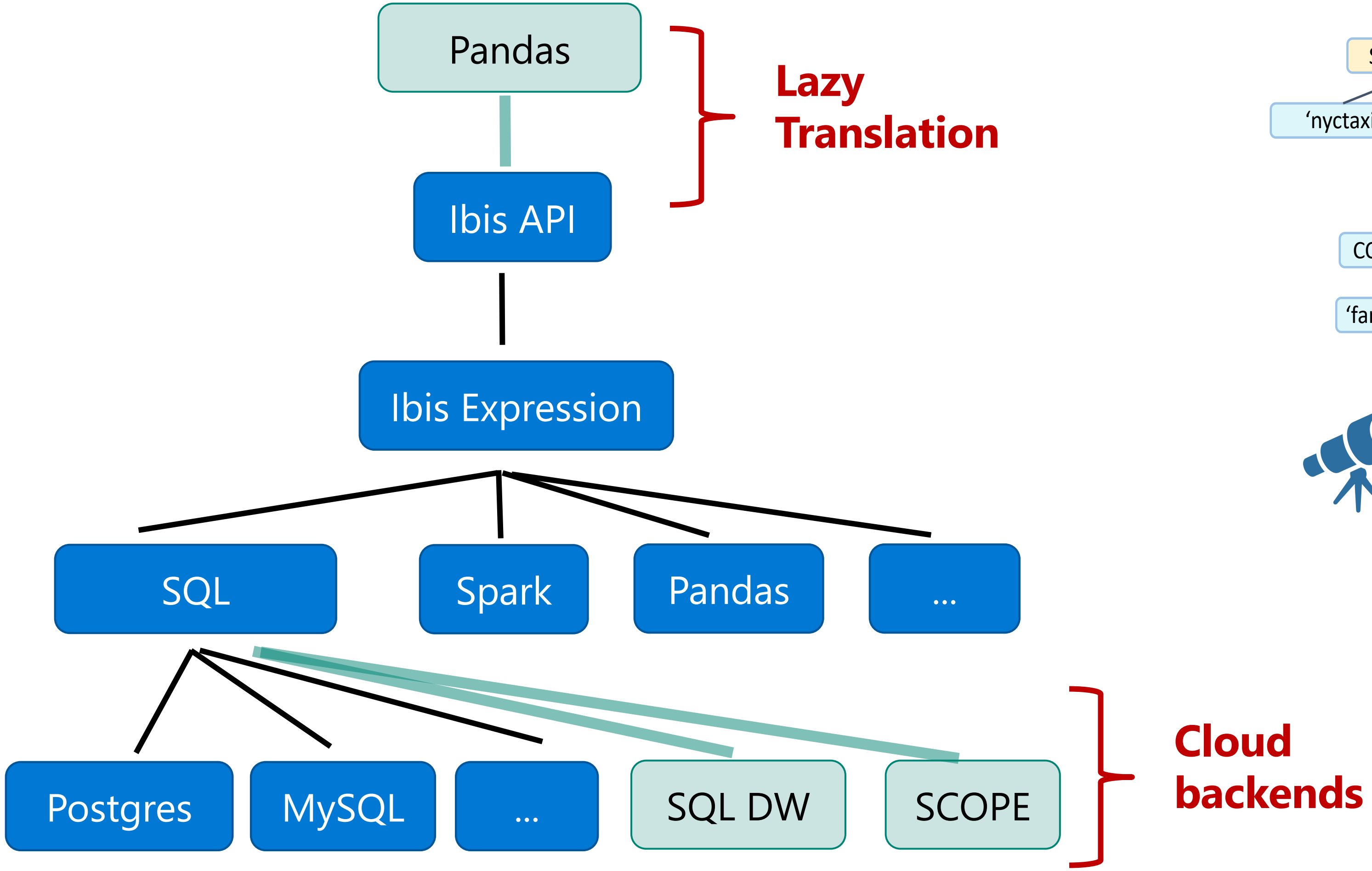
```
SELECT DATEPART(WEEKDAY, pickup_datetime) AS day,
             SUM(passenger_count)
FROM nyctaxi WHERE fare_amount > 0
GROUP BY DATEPART(WEEKDAY, pickup_datetime)
```

T-SQL Statement

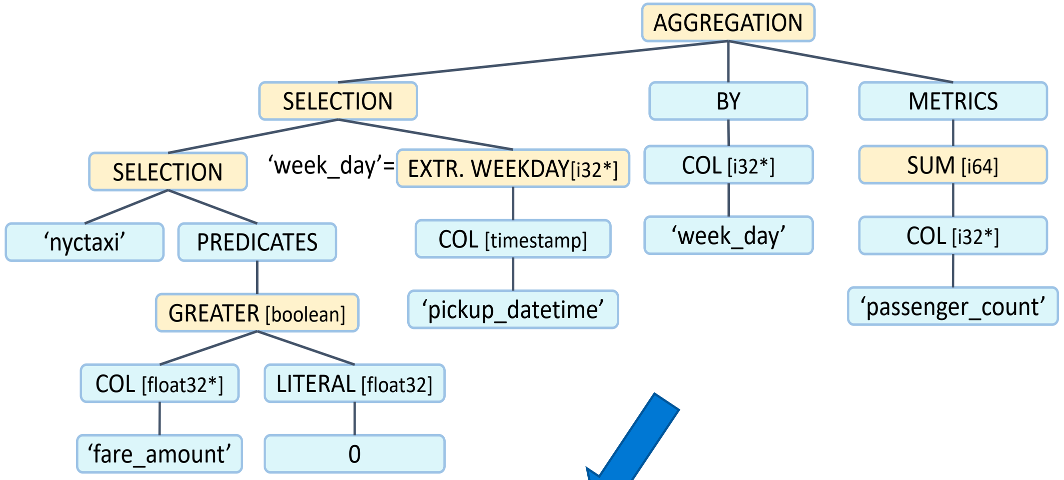


SQL DW Execution Plan

Pushing Data Science down



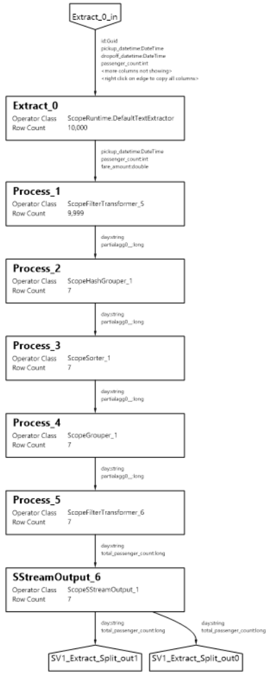
Blue parts: already in IBIS, **Green** parts: our contributions



pyscope

```
df = script.extract(path, schema
        .select("fare_amount > 0")
        .groupby("day")
        .project("pickup_datetime
.DayOfWeek.ToString() AS day",
"passenger_count")
```

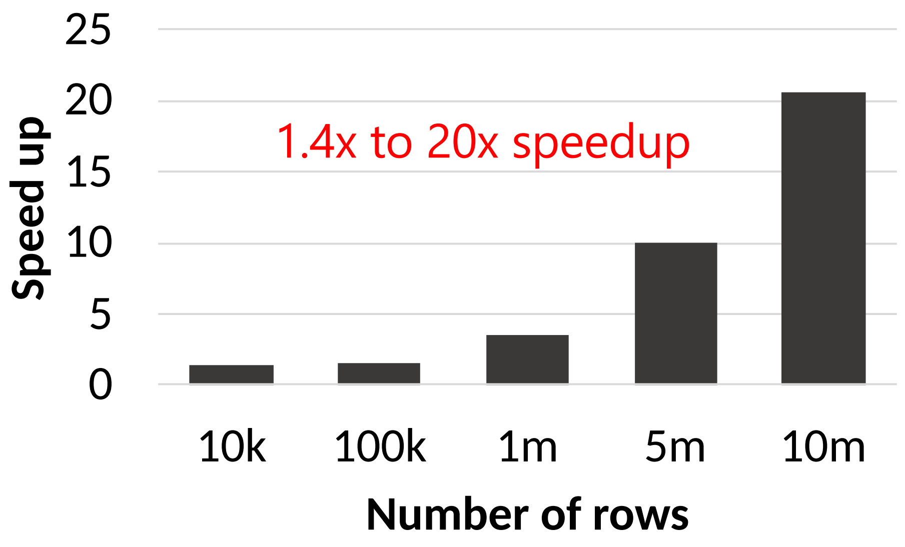
SCOPE Script



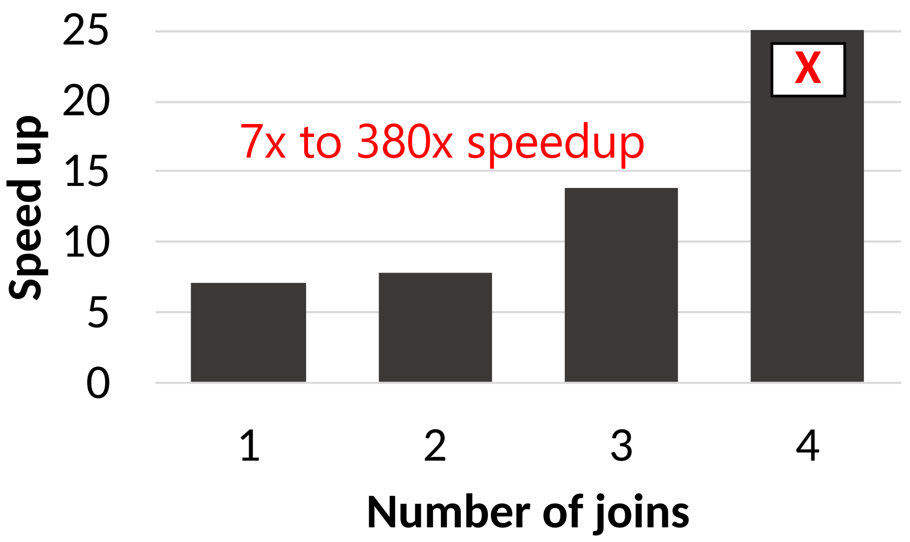
SCOPE Execution Plan

Impact: speed-up using SQL DW

Growing input size

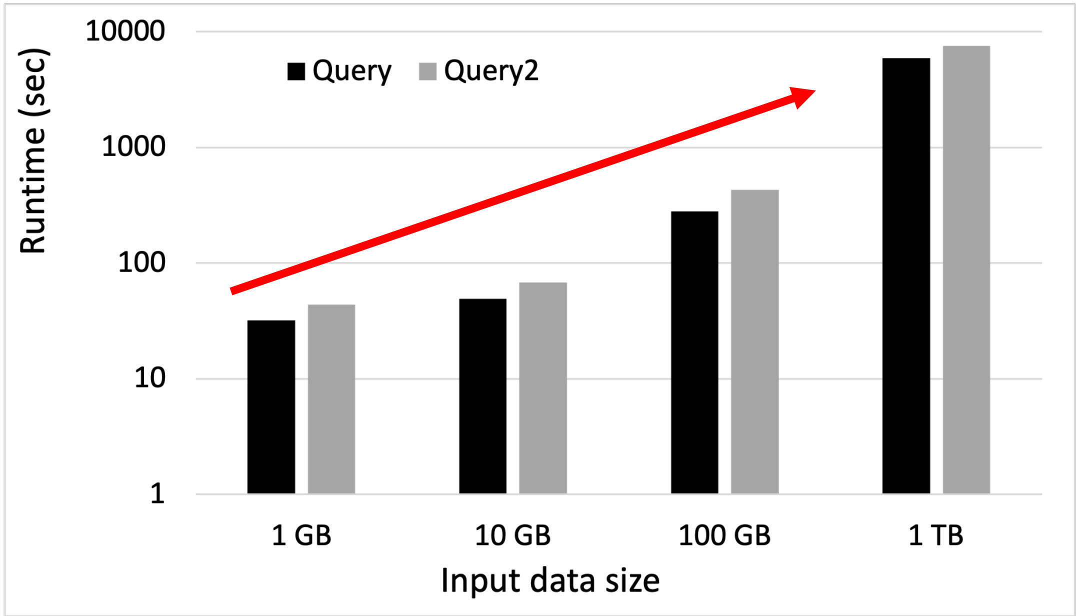


Growing query complexity

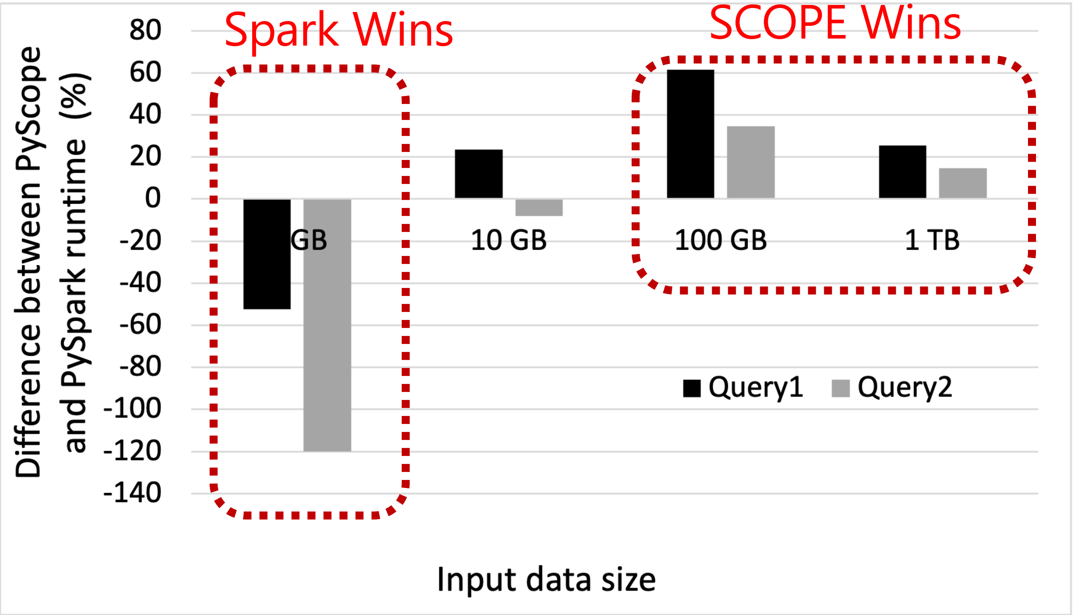


Impact: scale-out using SCOPE

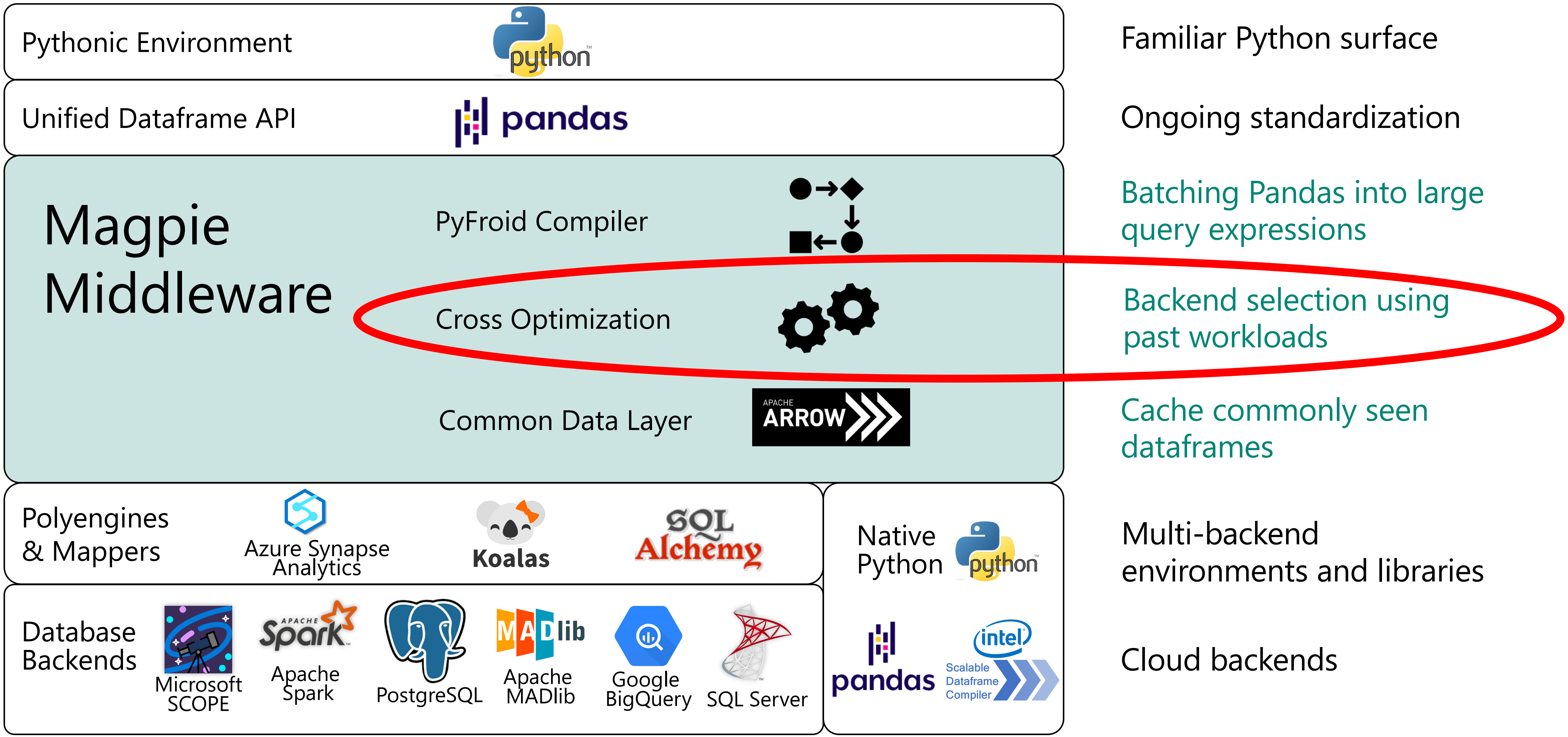
Scale data science to big data!



SCOPE vs Spark



Magpie



Pythonic Environment — Familiar Python surface

Unified Dataframe API — Ongoing standardization

Magpie Middleware

PyFroid Compiler — Batching Pandas into large query expressions

Cross Optimization — Backend selection using past workloads

Common Data Layer — Cache commonly seen dataframes

Polyengines & Mappers: Azure Synapse Analytics, Koalas, SQL Alchemy

Native Python

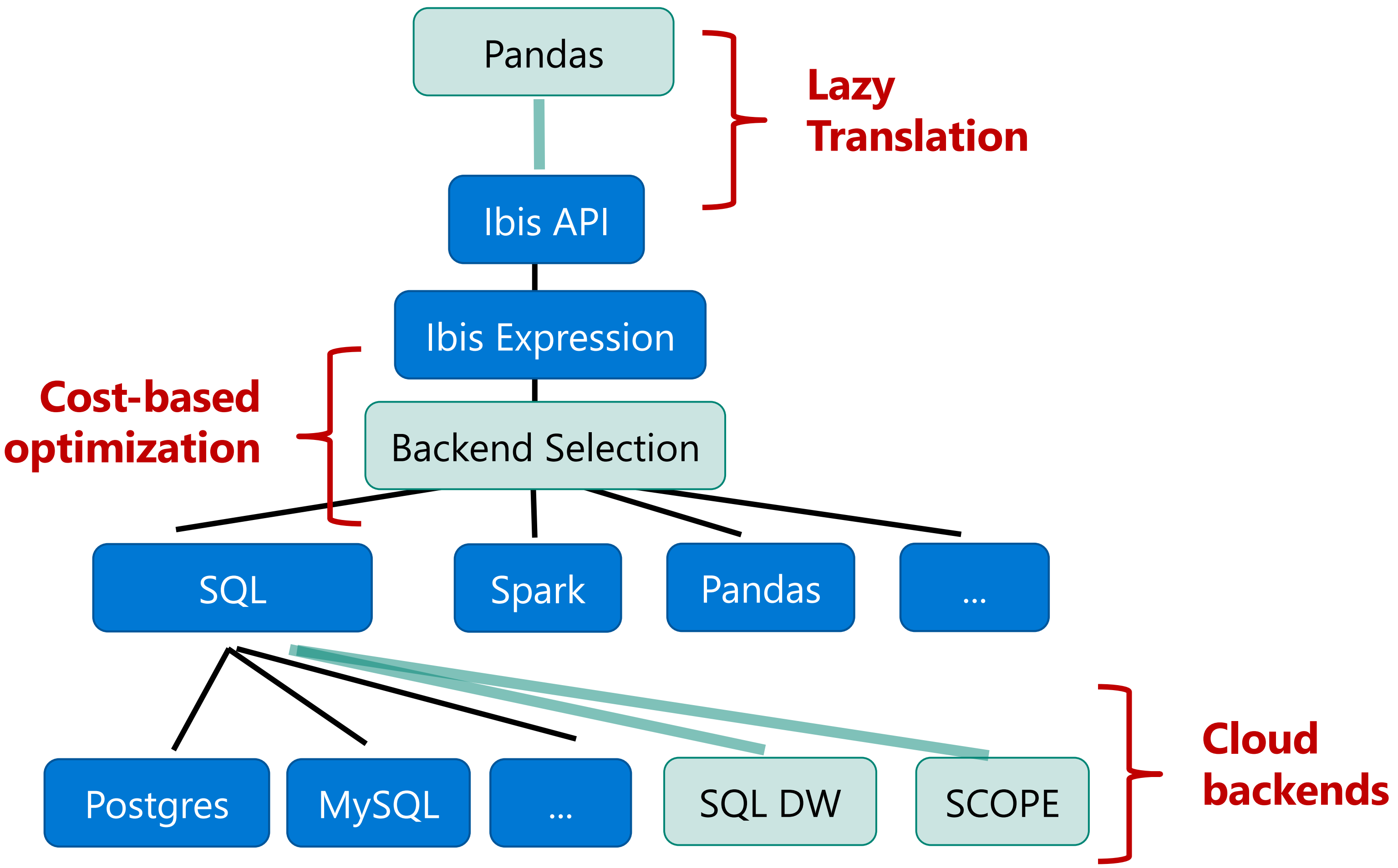
Multi-backend environments and libraries

Database Backends: Microsoft SCOPE, Apache Spark, PostgreSQL, Apache MADlib, Google BigQuery, SQL Server

pandas, intel Scalable Dataframe Compiler

Cloud backends

Backend Selection

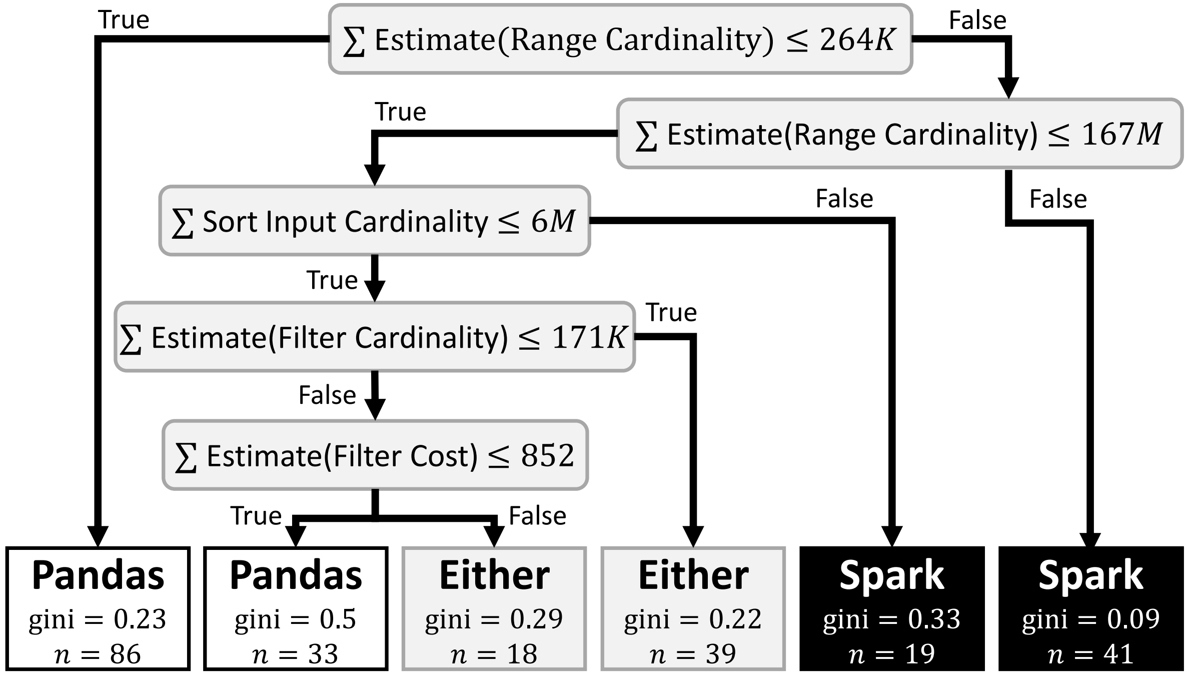


- Leverage past workloads from cloud backends to learn a decision tree
- At compile time:
  - User provides the list of available backends
  - Compile the plan into a common representation
  - Infer best backend using the decision tree

Blue parts: already in IBIS, **Green** parts: our contributions

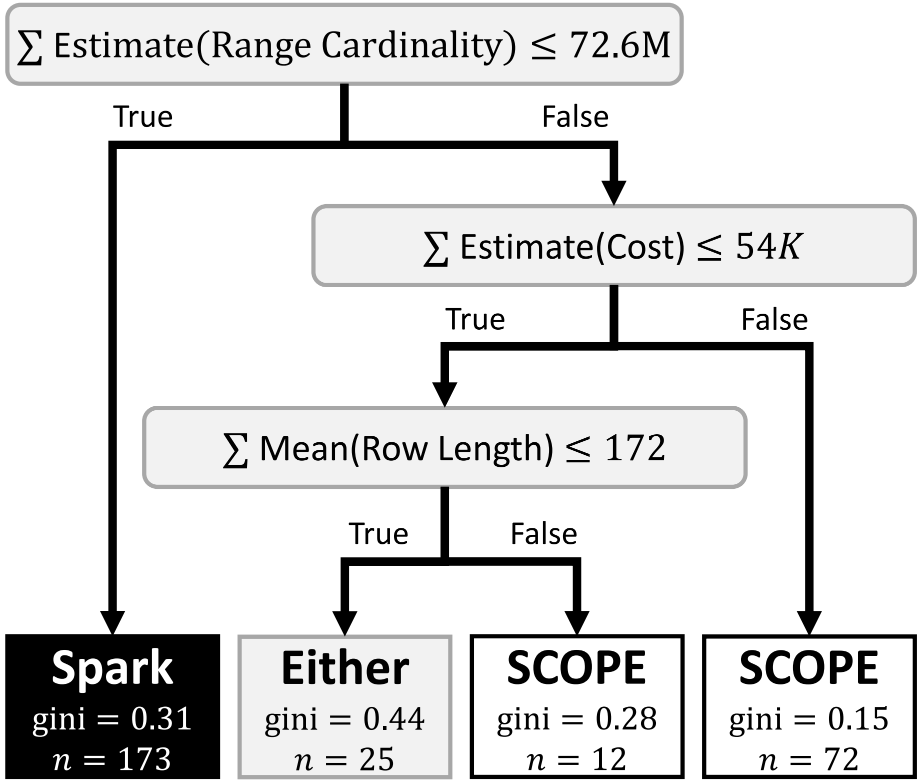
Scenario 1: Pandas vs PySpark

- Question:
  - When to switch to a cluster?
  - Or to local execution?
- Decision tree:
  - 84% accuracy on test set
  - On Pandas:
    - 84% median improvement
    - Up to 99% improvement

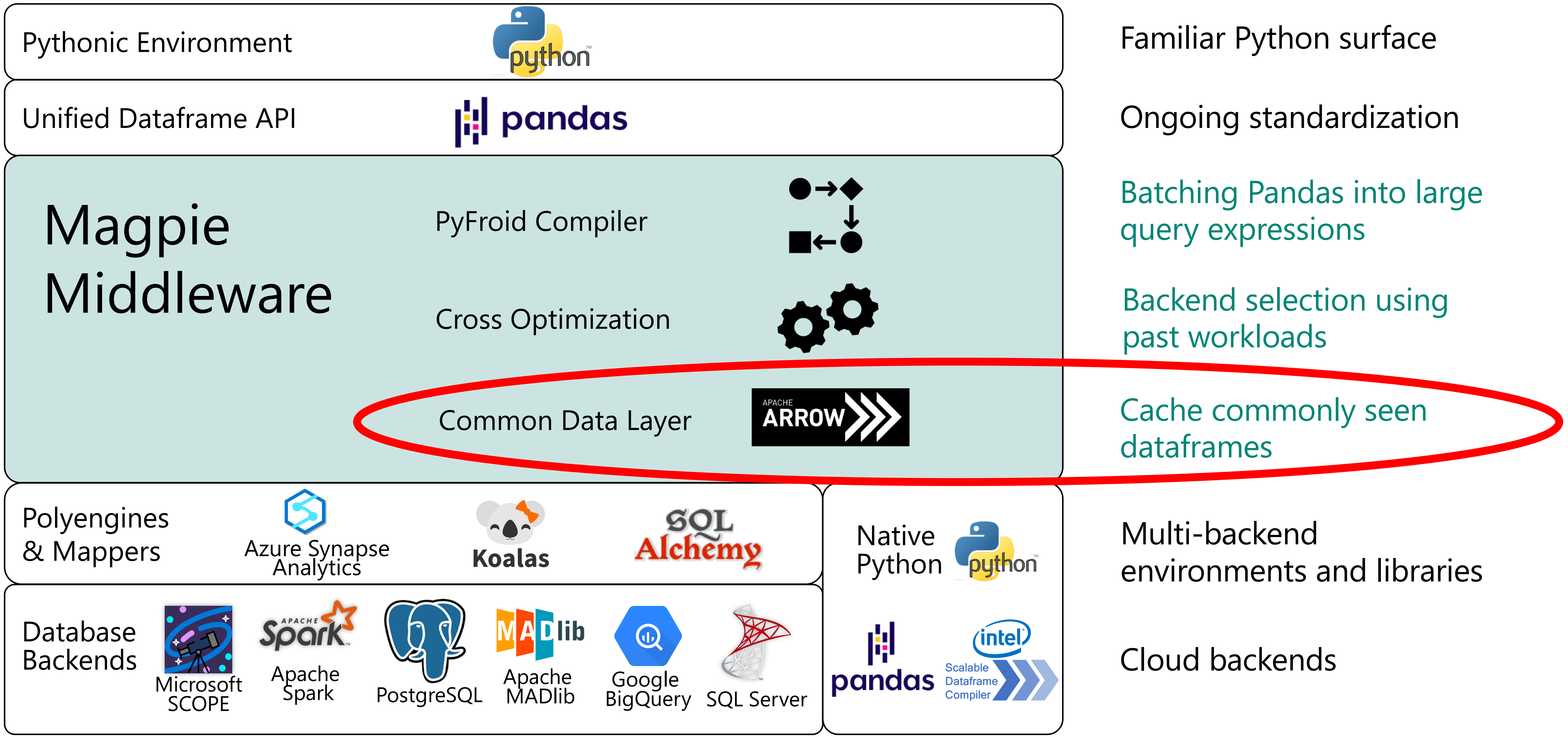

$\sum$ Estimate(Range Cardinality) $\leq 264K$
True
False
$\sum$ Estimate(Range Cardinality) $\leq 167M$
True
False
$\sum$ Sort Input Cardinality $\leq 6M$
False
True
$\sum$ Estimate(Filter Cardinality) $\leq 171K$
True
False
$\sum$ Estimate(Filter Cost) ≤ 852
True
False
Pandas
gini = 0.23
n = 86
Pandas
gini = 0.5
n = 33
Either
gini = 0.29
n = 18
Either
gini = 0.22
n = 39
Spark
gini = 0.33
n = 19
Spark
gini = 0.09
n = 41

Scenario 2: PyScope vs PySpark

- Cosmos has both SCOPE and Spark engines now
- Question: which one to use for data science?
- Decision tree
    - 87% accuracy on test set
    - On Spark:
        - Median improvement 85%
        - Up to 98% improvement



Magpie



Pythonic Environment

Familiar Python surface

Unified Dataframe API

Ongoing standardization

Magpie Middleware

PyFroid Compiler

Batching Pandas into large query expressions

Cross Optimization

Backend selection using past workloads

Common Data Layer

Cache commonly seen dataframes

Polyengines & Mappers

Azure Synapse Analytics

Koalas

SQL Alchemy

Native Python

Multi-backend environments and libraries

Database Backends

Microsoft SCOPE

Apache Spark

PostgreSQL

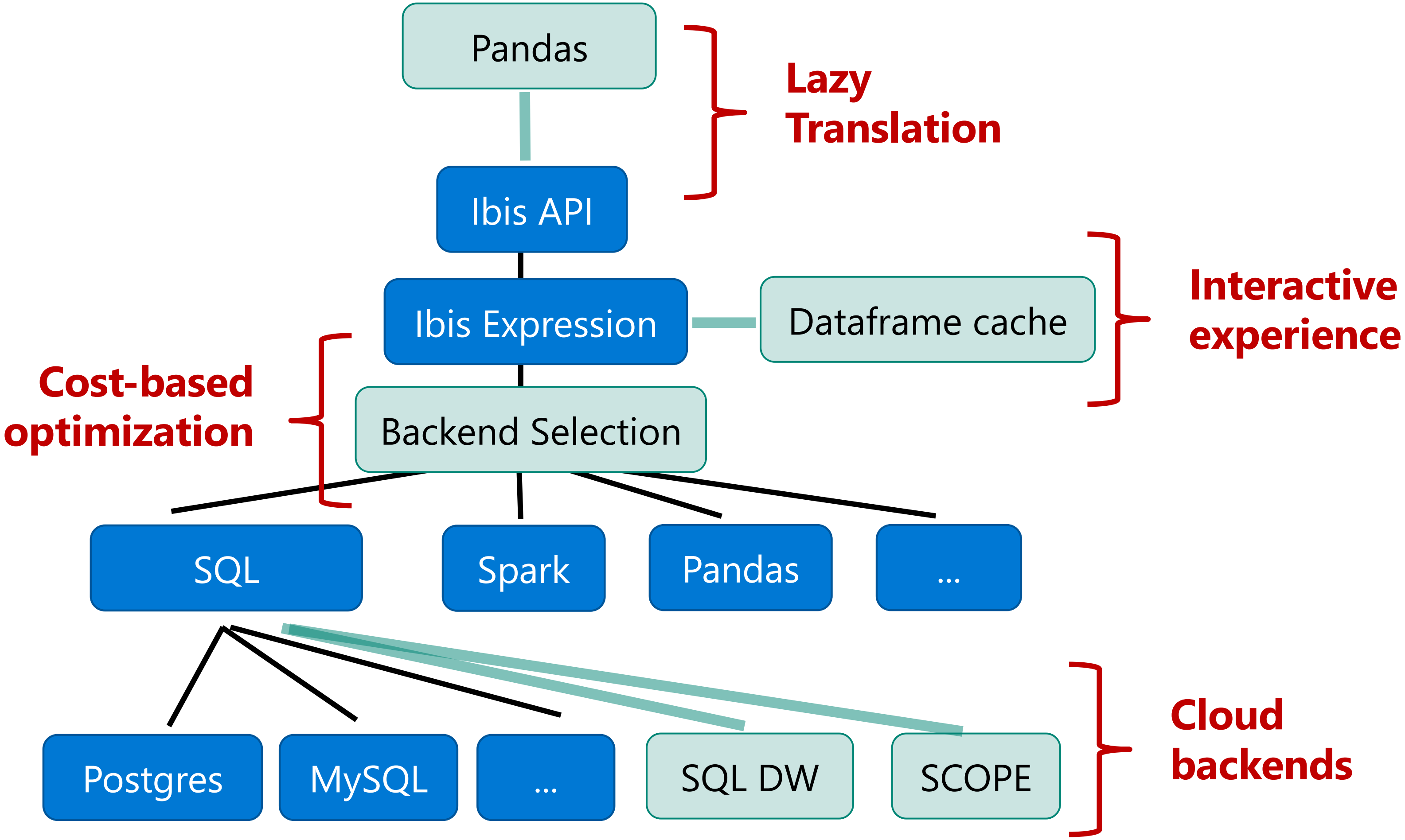
Apache MADlib

Google BigQuery

SQL Server

Cloud backends

Common Data Layer



Pandas

Ibis API

Ibis Expression

Dataframe cache

Backend Selection

SQL

Spark

Pandas

...

Postgres

MySQL

...

SQL DW

SCOPE

Lazy Translation

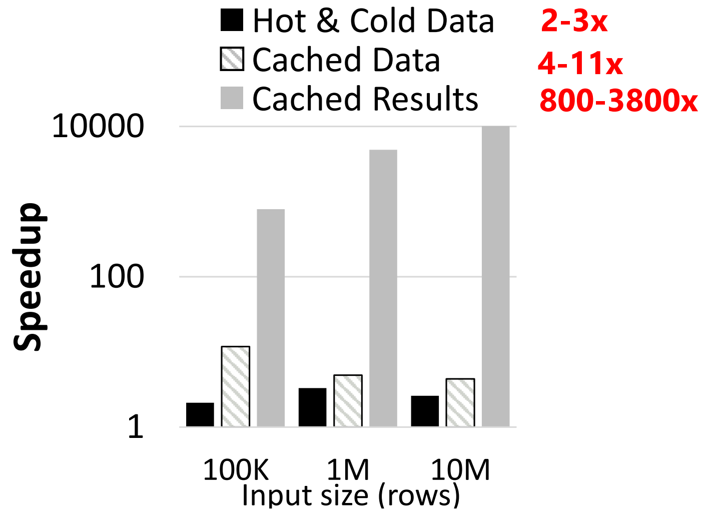
Interactive experience

Cost-based optimization

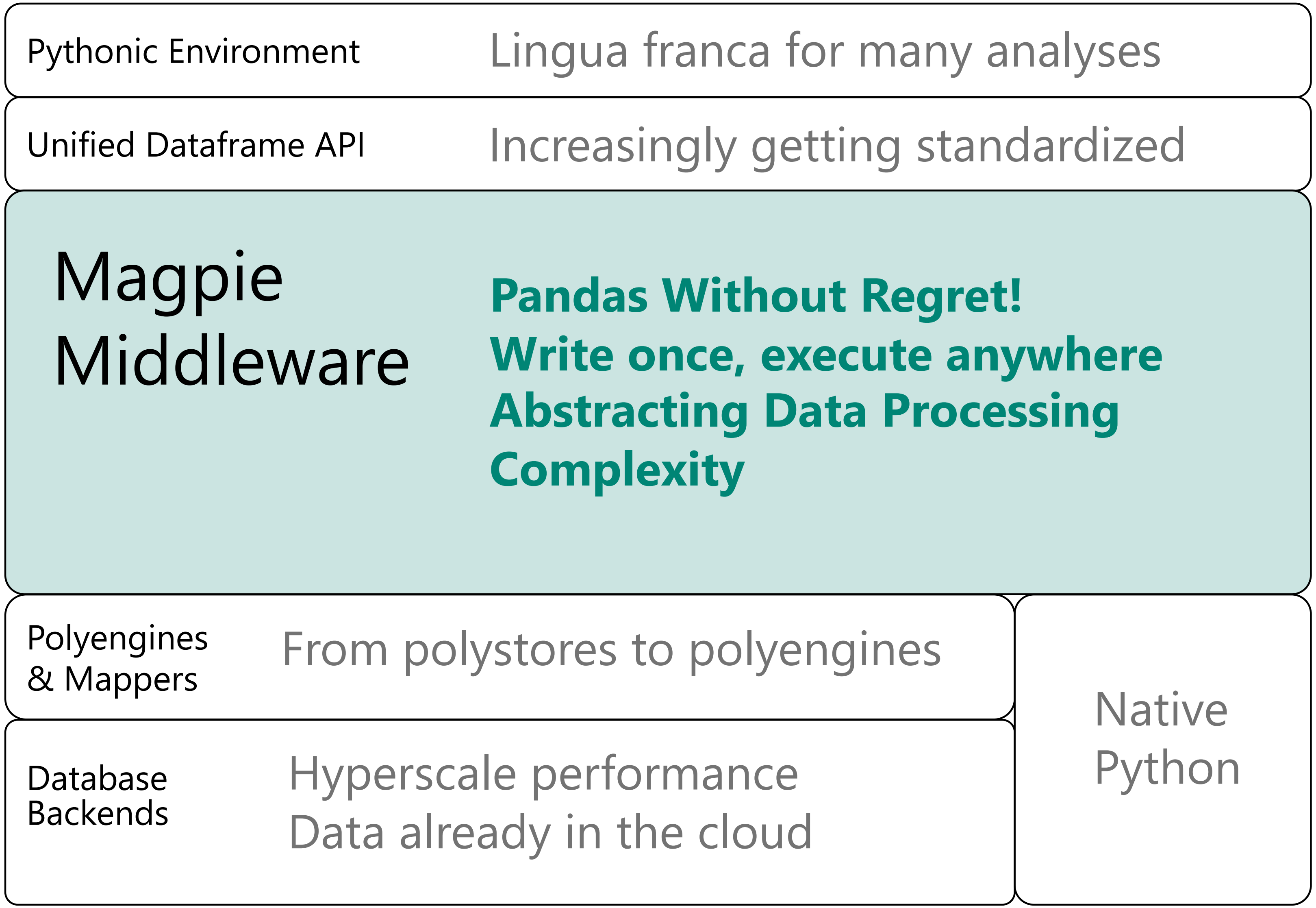
Cloud backends

Blue parts: already in IBIS, **Green** parts: our contributions

- Dataframe cache
  - Generate unique signatures
  - Store repeated dataframes in ArrowFlight server
  - Skip accessing the backend in case of cache hit



Summary



Pythonic Environment — Lingua franca for many analyses

Unified Dataframe API — Increasingly getting standardized

Magpie Middleware

Pandas Without Regret!
Write once, execute anywhere
Abstracting Data Processing Complexity

Polyengines & Mappers — From polystores to polyengines

Database Backends — Hyperscale performance
Data already in the cloud

Native Python

Gray
Systems
Lab

https://azuredata.microsoft.com/

Hiring Summer Interns!

Microsoft