Building an E2E Analytics Pipeline with PyFlink

Marta Paes (@morsapaes)
Developer Advocate

ververica

About Ververica

Original Creators of
Apache Flink®

Enterprise Stream Processing
With Ververica Platform

Part of
Alibaba Group

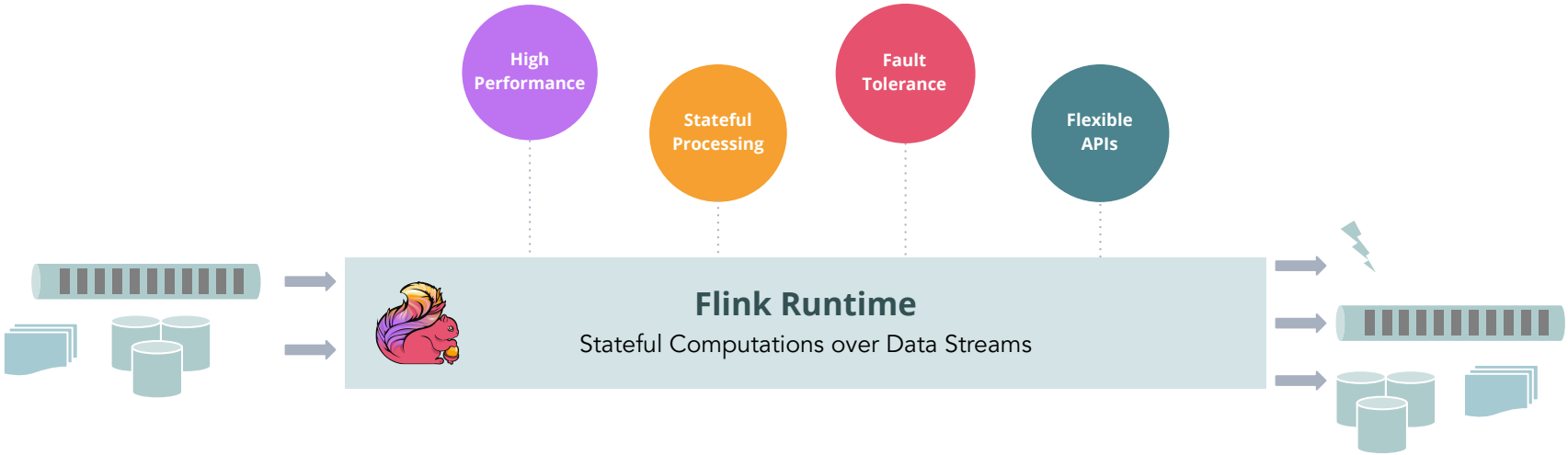
Apache Flink

Flink is an **open source** framework and **distributed** engine for **unified batch and stream processing**.

Learn more: flink.apache.org

Apache Flink

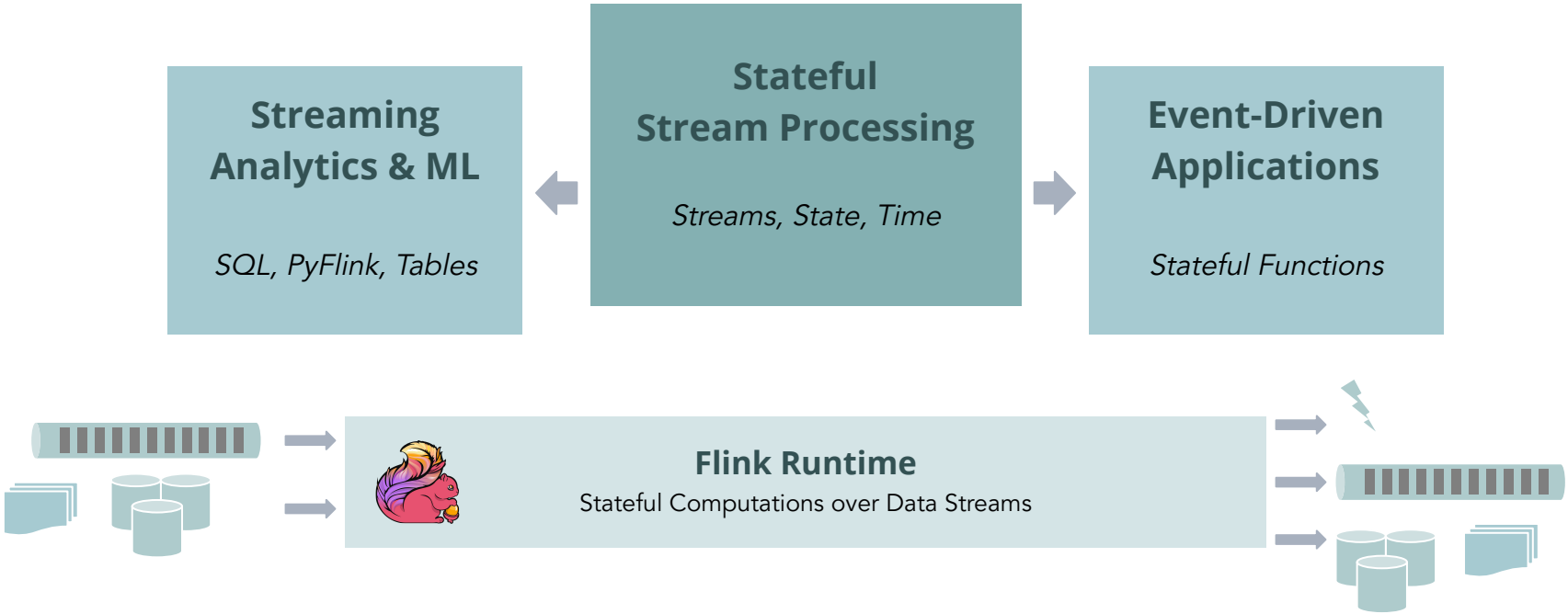
Flink is an **open source** framework and **distributed** engine for **unified batch and stream processing**.



Learn more: flink.apache.org

Use Cases

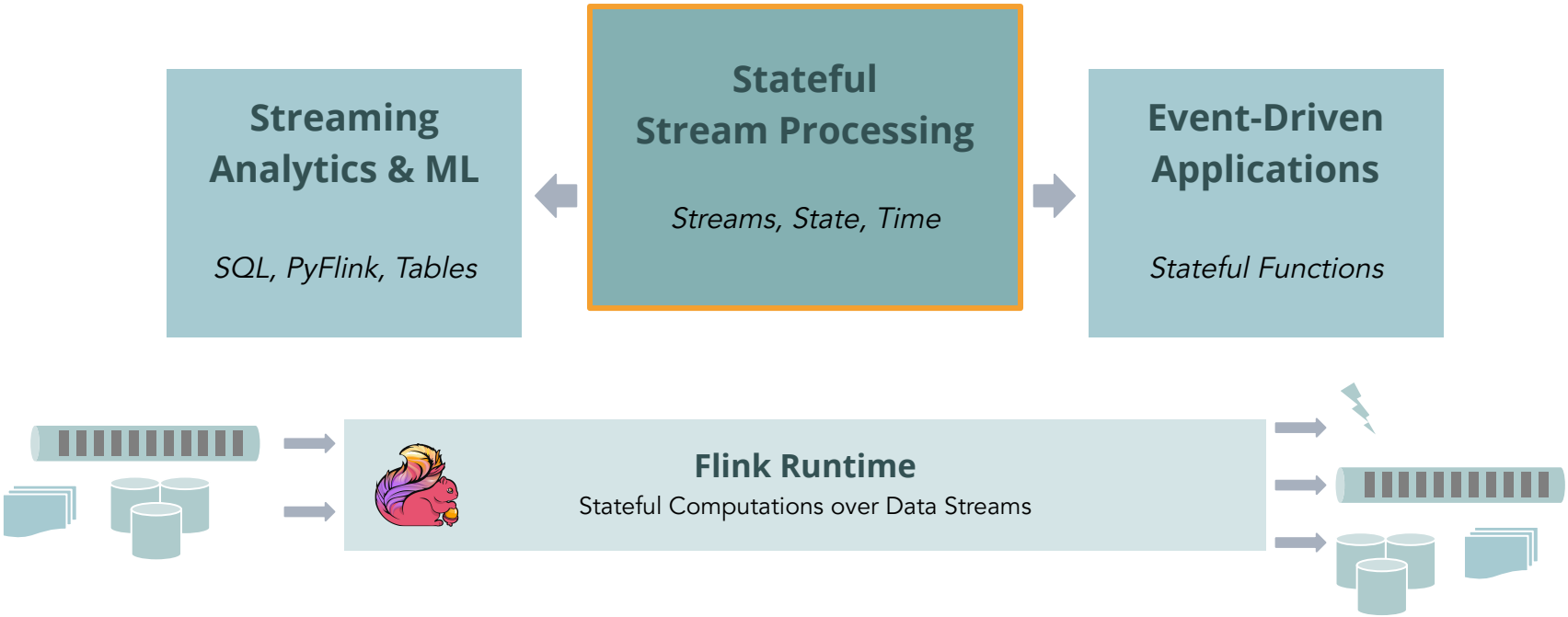
This gives you a robust foundation for a wide range of use cases:



Learn more: flink.apache.org

Use Cases

Classical, core stream processing use cases that build on the primitives of **streams**, **state** and **time**.



Learn more: flink.apache.org

Stateful Stream Processing

Classical, core stream processing use cases that build on the primitives of **streams**, **state** and **time**.

- Explicit control over these primitives
- Complex computations and customization
- Maximize **performance** and **reliability**

Example Use Cases

NETFLIX

ING

aws

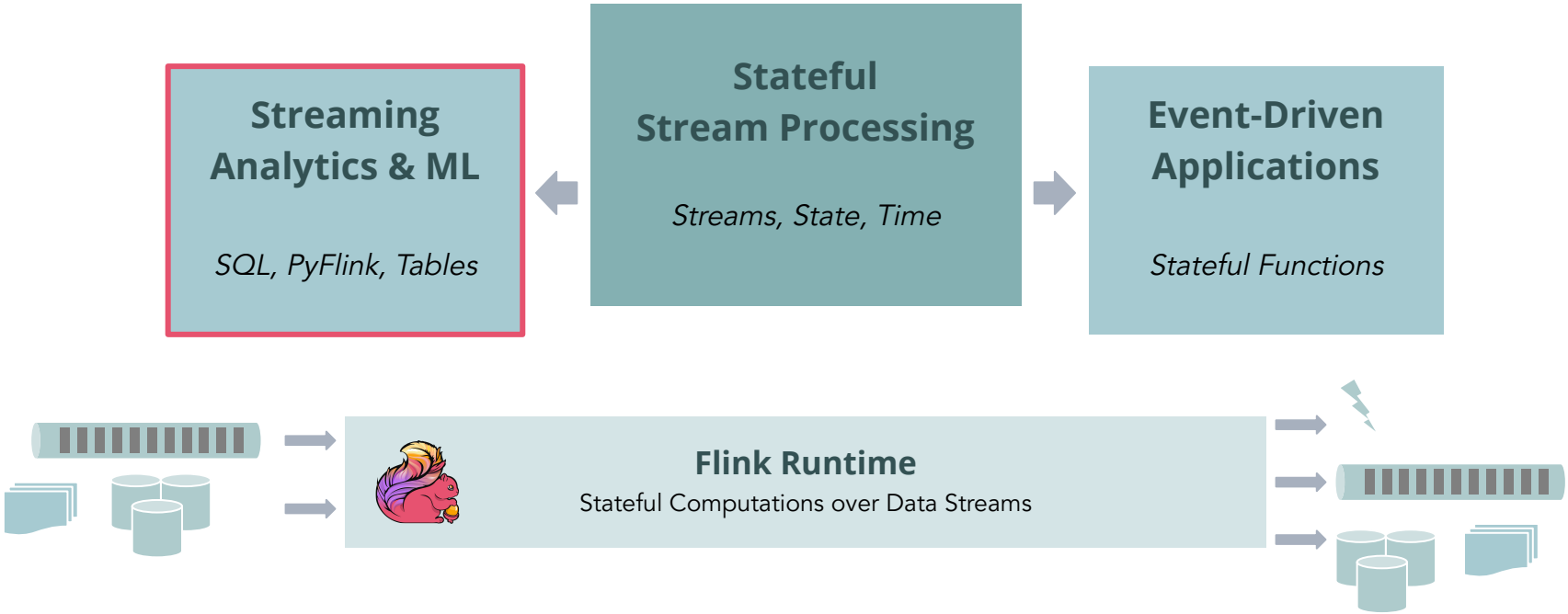
Large-scale Data Pipelines

ML-Based Fraud Detection

Service Monitoring & Anomaly Detection

Use Cases

More high-level or domain-specific use cases that can be modeled with SQL or Python and dynamic tables.



Learn more: flink.apache.org

Streaming Analytics & ML

More high-level or domain-specific use cases that can be modeled with SQL or Python and dynamic tables.

- Focus on logic, not implementation
- Mixed workloads (batch and streaming)
- Maximize **developer speed** and **autonomy**

Example Use Cases

Uber

criteo

Unified Online/Offline Model Training	E2E Streaming Analytics Pipelines	ML Feature Generation

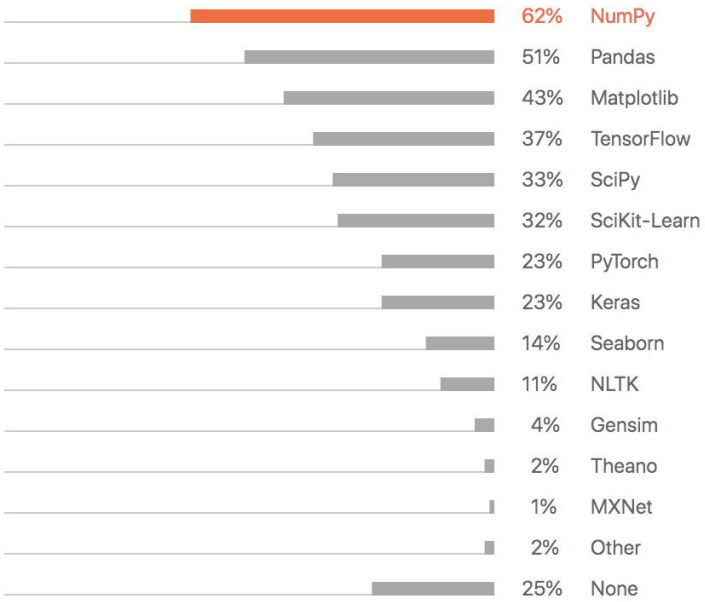
More Flink Users

Learn More: Powered by Flink, Speakers – Flink Forward San Francisco 2019, Speakers – Flink Forward Europe 2019

Why PyFlink?

Python is...pretty stacked?

What data science frameworks do you use in addition to Python?

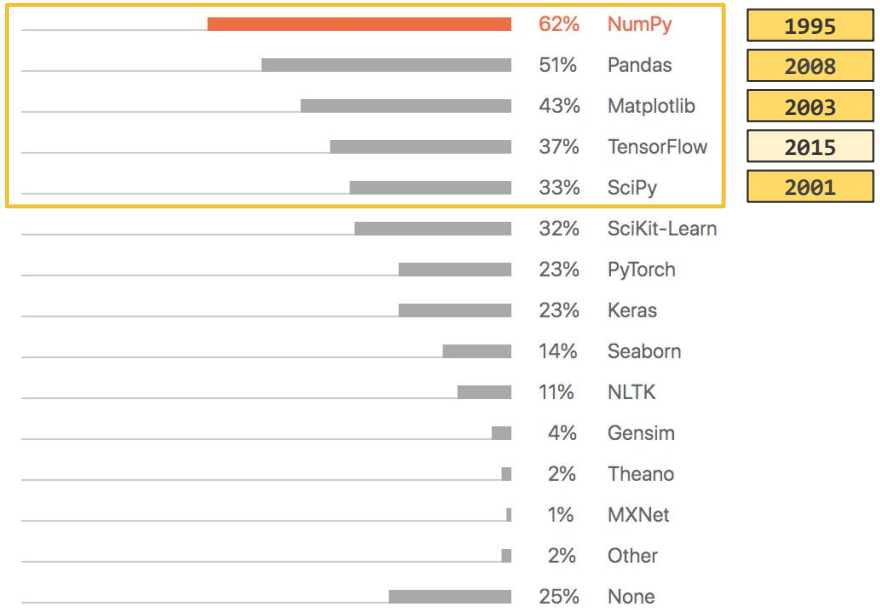


Mature analytics stack, with libraries that are **fast** and **intuitive**.

Source: [JetBrains' Developer Ecosystem Report 2020]

...and also timeless!

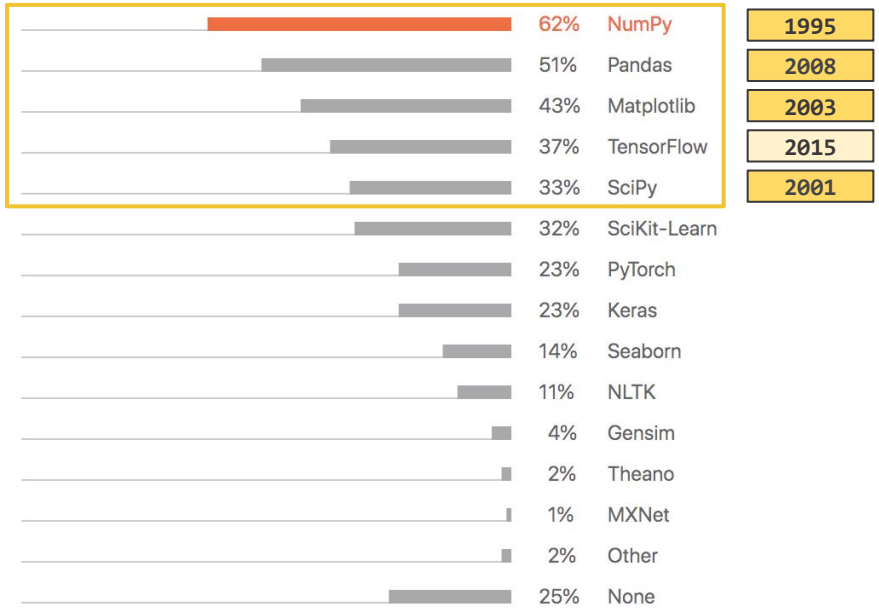
What data science frameworks do you use in addition to Python?



Mature analytics stack, with libraries that are **fast** and **intuitive**.

...and also timeless!

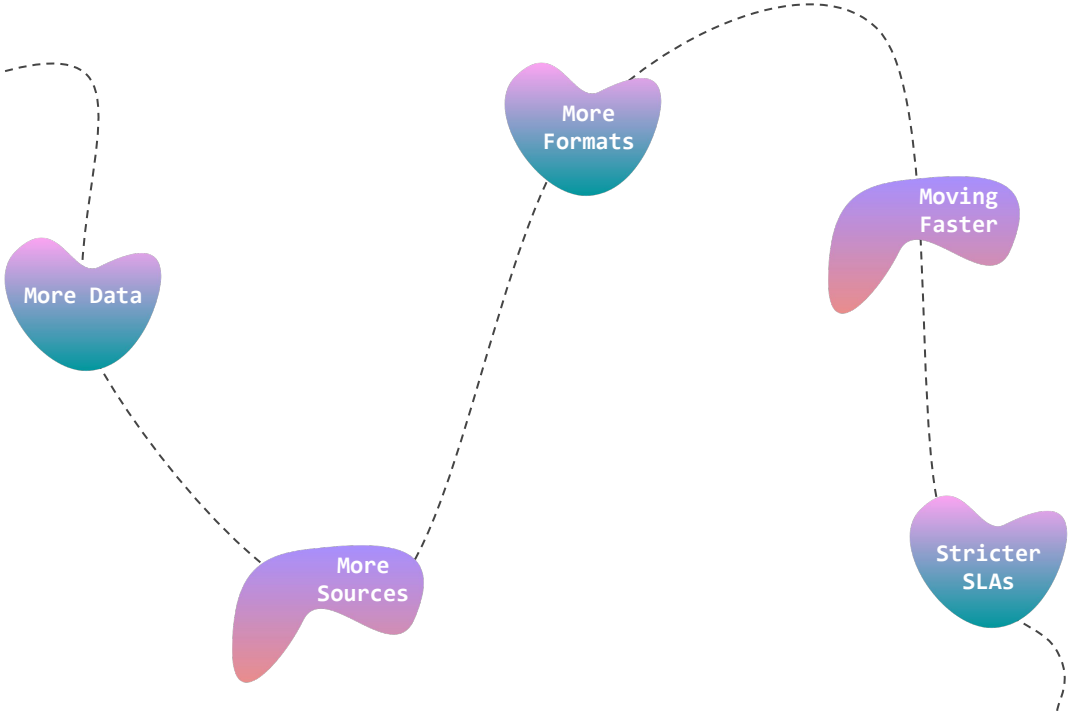
What data science frameworks do you use in addition to Python?



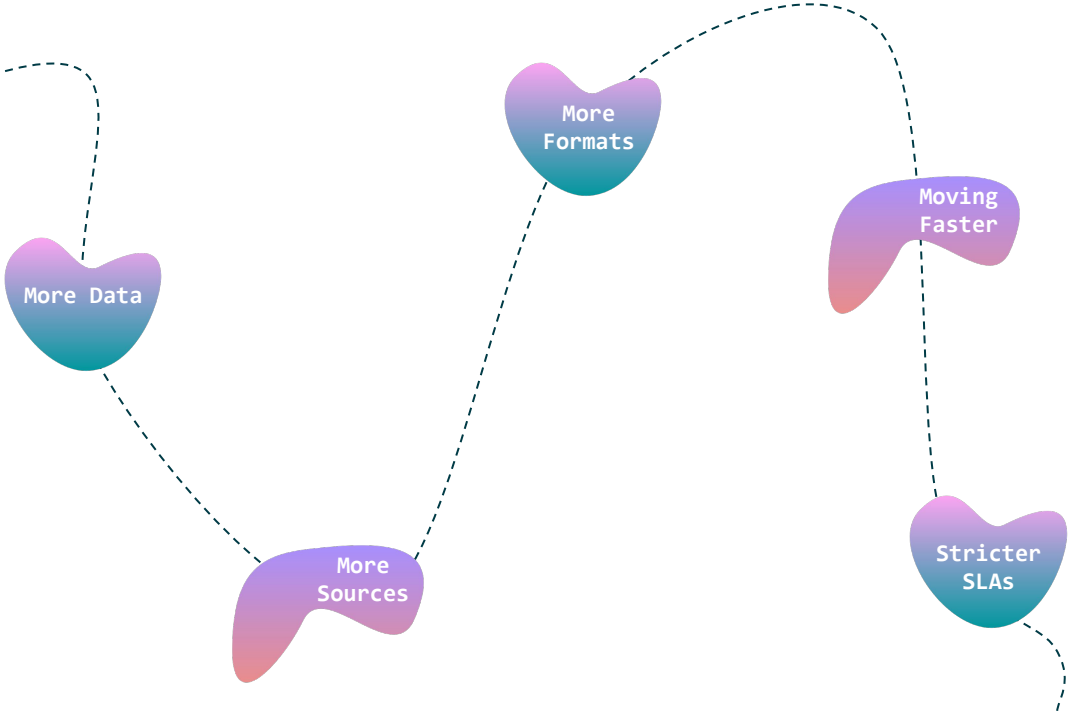
Mature analytics stack, with libraries that are **fast** and **intuitive**.

Older libraries are mostly **restricted** to a data size that **fits in memory** (RAM), and designed to run on a **single core** (CPU).

This is a problem, because



This is a problem, because



But you still want to use these powerful libraries, right?

Why PyFlink?

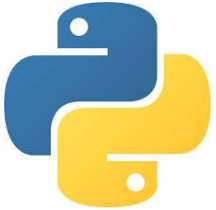
Java

Scala

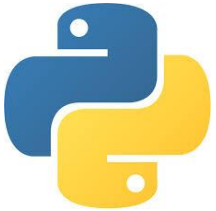
Why PyFlink?

Expose the functionality of Flink beyond the JVM

Why PyFlink?



Why PyFlink?



Distribute and **scale** the functionality of Python through Flink

Flink at Alibaba scale

Double 11 / Singles Day

Alibaba Group

incl. sub-second updates to the GMV dashboard

Real-time Data Applications

Search | Recomm. | Ads | BI | Security

Data Size

1.7EB

Throughput (Peak)

4B events/sec

State Size (Biggest)

100TB

Latency

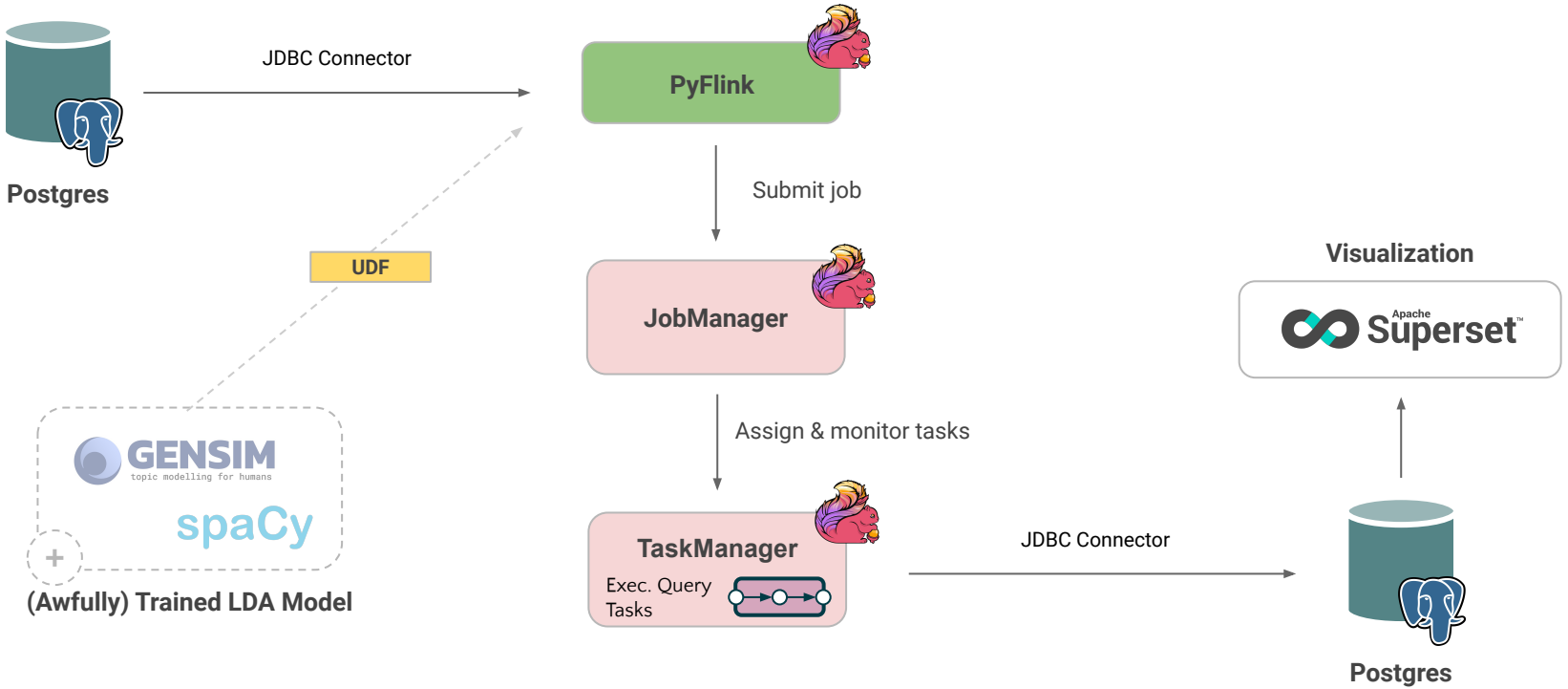
Sub-sec

Learn more: Alibaba Cloud Unveils 'Magic' Behind the World's Largest Online Shopping Festival

Demo

Can we use PyFlink to identify the most frequent topics in the Flink User Mailing List?

The Demo Environment



DEMO

Step 1. Create the source and sink tables.

```
ddl_jdbc_source = """CREATE TABLE user_ml_messages (
                            message_date TIMESTAMP(3),
                            message_id STRING,
                            message_in_reply_to STRING,
                            message_from_name STRING,
                            message_from_email STRING,
                            message_subject STRING,
                            message_body_html STRING,
                            message_body_plain STRING,
                            PRIMARY KEY(message_id) NOT ENFORCED
                ) WITH (
                    'connector'= 'jdbc',
                    'url' = 'jdbc:postgresql://postgres:5432/postgres',
                    'table-name' = 'perceval.stg_flink_user_ml',
                    'username' = 'postgres',
                    'password' = 'postgres'
                )"""

create_jdbc_source = t_env.execute_sql(ddl_jdbc_source)
```

Connector Properties

```
ddl_jdbc_sink = """CREATE TABLE flink_user_ml_topics (
                            message_id STRING,
                            message_date TIMESTAMP(3),
                            message_from_name STRING,
                            topic VARCHAR(2)
                        ) WITH (
                            'connector'= 'jdbc',
                            'url' = 'jdbc:postgresql://postgres:5432/postgres',
                            'table-name' = 'perceval.flink_user_ml_topics',
                            'username' = 'postgres',
                            'password' = 'postgres'
                        )"""


create_jdbc_sink = t_env.execute_sql(ddl_jdbc_sink)
```

DDL Statement Execution

DEMO

Step 2. Write and register a UDF to clean and classify the messages.

```
@udf(input_types=DataTypes.STRING(), result_type=DataTypes.STRING())
def class_model(m):

    lda = LdaModel.load("model/lda_model/lda_model_user_ml")

    v = tokenizer.pre_process(m)

    vector = lda[v]

    topics = sorted(vector, key=lambda x: x[1], reverse=True)

    return str(topics[0][0])


t_env.register_function("CLASS_TOPIC", class_model)
```

Text pre-processing and classification logic

UDF Registration

How would this perform if it were defined as a Pandas UDF?

Learn more: NLP with LDA - Analyzing Topics in the Enron Email dataset

DEMO

Step 3. Build your query, that will insert your results into the sink table.

```
t_env.from_path("user_ml_messages") \
     .filter("message_subject.isNotNull") \
     .select("message_id, message_date, message_from_name, CLASS_TOPIC(message_subject)")

t_env.execute_insert("flink_user_ml_topics")
```

OR

```
t_env.execute_sql("INSERT INTO flink_user_ml_topics \
                   SELECT message_id, message_date, message_from_name, CLASS_TOPIC(message_subject) \
                   FROM user_ml_messages WHERE message_subject IS NOT NULL")
```


DEMO

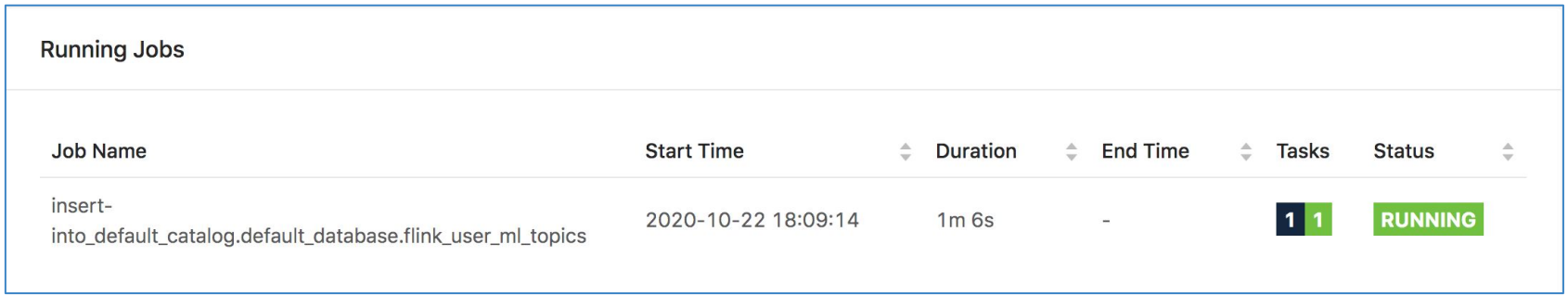
Step 4. Submit the job (and dependencies) to the cluster.

```
docker-compose exec jobmanager ./bin/flink run -py /opt/pyflink-ff2020/pipeline.py \
--pyArchives /opt/pyflink-ff2020/lda_model.zip#model \
--pyFiles /opt/pyflink-ff2020/tokenizer.py -d
```

LDA Model + Dictionary

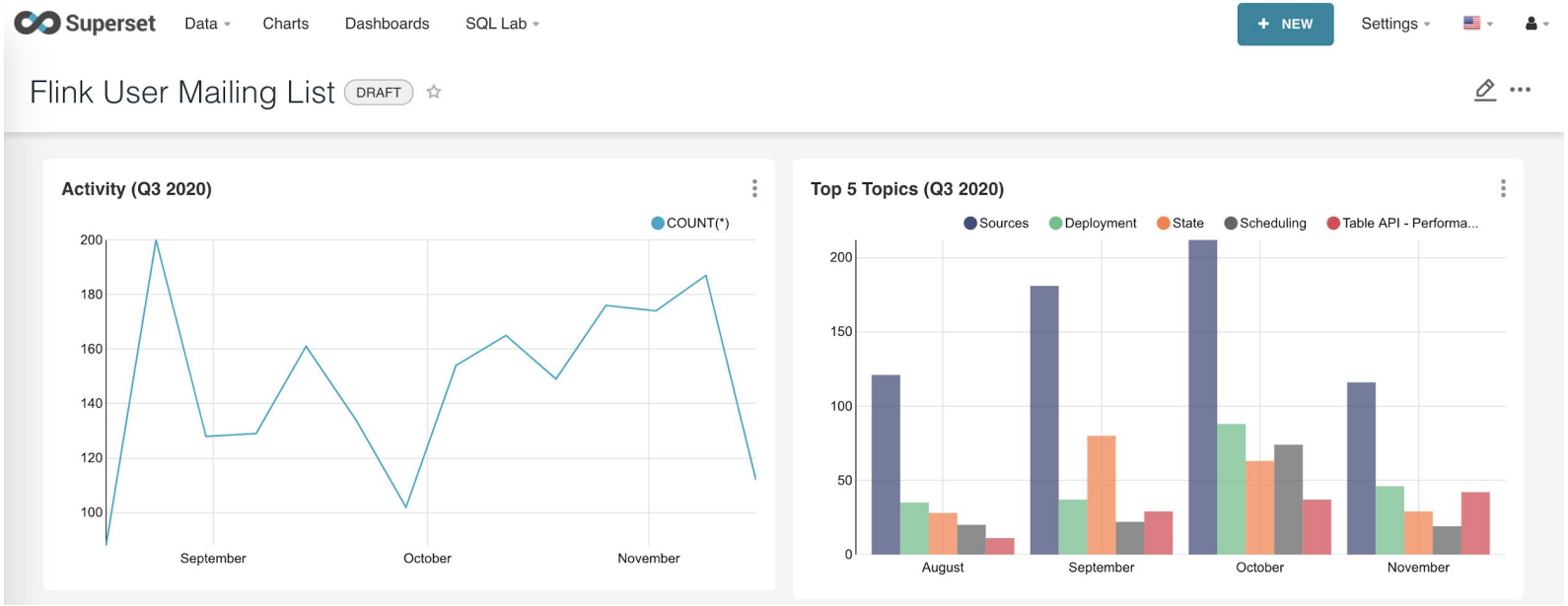
Pre-processing Class

Flink Web UI



DEMO

Step 5. Visualize in Superset!



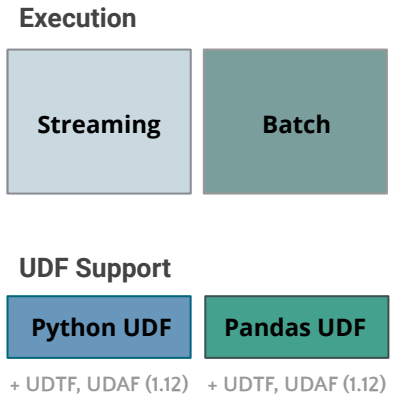
PyFlink in a Nutshell*

- Native SQL integration
- Unified APIs for batch and streaming
- Support for a large set of operations (incl. complex joins, windowing, pattern matching/CEP)

* As of Flink 1.11, only the Table API is exposed through PyFlink. The low-level DataStream API is going to be supported in 1.12.

PyFlink in a Nutshell*

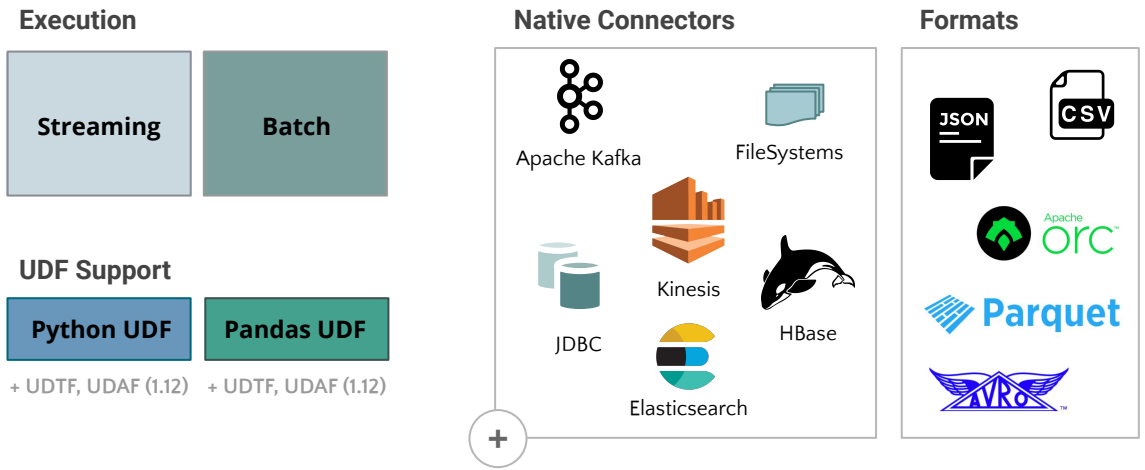
- Native SQL integration
- Unified APIs for batch and streaming
- Support for a large set of operations (incl. complex joins, windowing, pattern matching/CEP)



* As of Flink 1.11, only the Table API is exposed through PyFlink. The low-level DataStream API is going to be supported in 1.12.

PyFlink in a Nutshell*

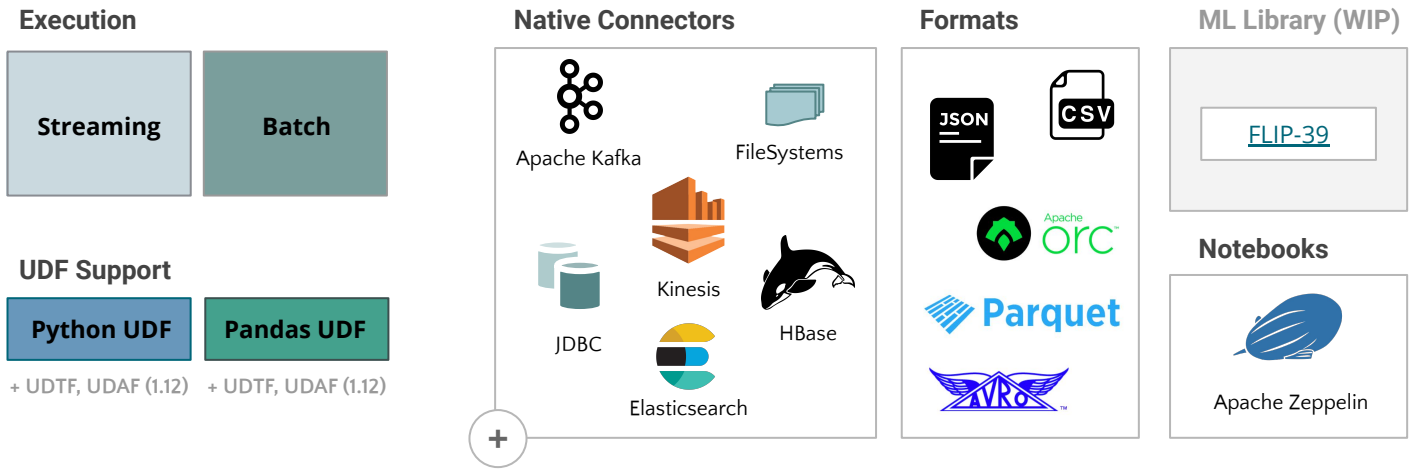
- Native SQL integration
- Unified APIs for batch and streaming
- Support for a large set of operations (incl. complex joins, windowing, pattern matching/CEP)



* As of Flink 1.11, only the Table API is exposed through PyFlink. The low-level DataStream API is going to be supported in 1.12.

PyFlink in a Nutshell*

- Native SQL integration
- Unified APIs for batch and streaming
- Support for a large set of operations (incl. complex joins, windowing, pattern matching/CEP)



* As of Flink 1.11, only the Table API is exposed through PyFlink. The low-level DataStream API is going to be supported in 1.12.

Thank you, Data Science UA!

Follow me on Twitter: @morsapaes

Learn more about Flink: https://flink.apache.org/

ververica