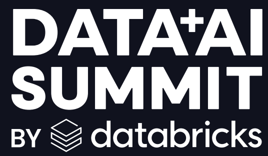
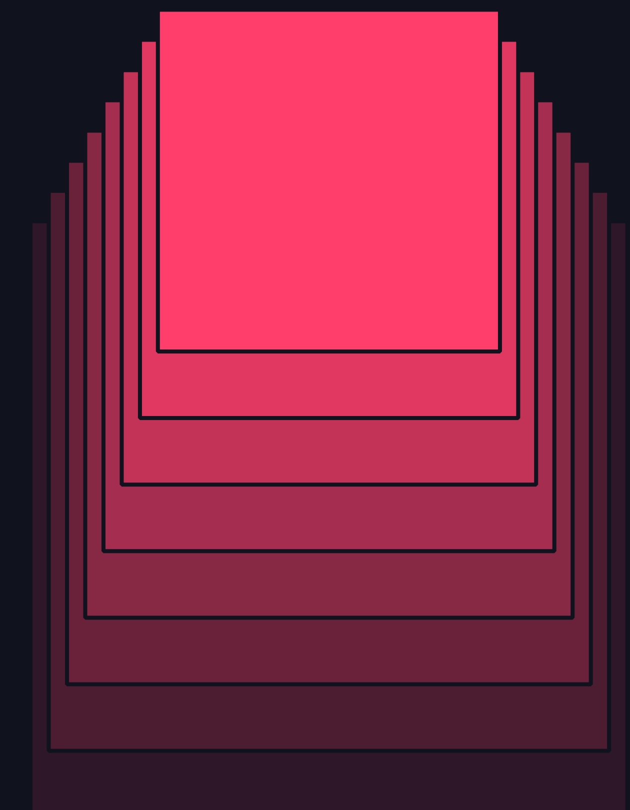
Product safe harbor statement

This information is provided to outline Databricks' general product direction and is for informational purposes only. Customers who purchase Databricks services should make their purchase decisions relying solely upon services, features, and functions that are currently available. Unreleased features or functionality described in forward-looking statements are subject to change at Databricks discretion and may not be delivered as planned or at all



Introducing the New Python Data Source API in Apache Spark



Allison Wang
Sr. Software Engineer, Databricks

Ryan Nienhuis,
Sr. Staff Product Manager, Databricks

©2024 Databricks Inc. — All rights reserved

Agenda

Exploring the new Python Data Source API in Apache Spark

- Introduction
- Why Python Data Source API?
- Deep Dive into the Python Data Source API
  - Demo
  - Data Source Reader
  - Data Source Writer
- Streaming APIs
- Q & A

©2024 Databricks Inc. — All rights reserved

Custom Integrations in Spark

How do I simply read and write data?

You have a couple options…

1. Use ForEachBatch / ForEach for streaming workloads
2. Build a custom integration in Scala/Java using the DataSource V2 API
3. Don’t build one; get the data in Delta using a custom app
4. Import a library

…Which have some drawbacks

1. ForEachBatch code is powerful but very hard to write well
2. Flexible but no API for Python developers
3. Added cost and latency copying data
4. Not optimized for Spark

©2024 Databricks Inc. — All rights reserved

Why don't I just do it in Python?

Pull in library, make some API calls, etc.

Custom data sources have lots of advantages

- End users can use like a built-in integration
  - df = spark.read.format("my_data_source")
  - df.write.format(“my_data_source”)
  - CREATE TABLE t(c1 INT, c2 INT) USING PYTHON `my_data_source`
- Rely on API for implemented partitioning and other Spark capabilities
- Build once and use across programming languages
- Custom data sources can be packaged and pip installed

©2024 Databricks Inc. — All rights reserved

How do we improve here?

Let’s provide as simple experience for Python developers

- Aimed to focus on simplicity versus flexibility
- Supports distributed scan, append-only, atomic writes
- Does not support limit or aggregate push down, complete/update modes

©2024 Databricks Inc. — All rights reserved

Introducing the New Python Data Source API

©2024 Databricks Inc. — All rights reserved

Python Data Source API

- Available in Apache Spark 4.0 and Databricks Runtime 15.2+
- Fully open source:
- Support both **read and write operations,** for both **batch and streaming**

©2024 Databricks Inc. — All rights reserved

Data Source Overview

Three easy steps to create and use custom data sources

Step 1: Create a Data Source

```
class MySource(DataSource):
    …
```

Step 2: Register the Data Source

Register the data source in the current Spark session using the Python data source class:

```
spark
.dataSource
.register(MySource)
```

Step 3: Read from or write to the data source

```
spark.read
.format("my-source")
.load(...)
```

```
df.write
.format("my-source")
.mode("append")
.save(...)
```


©2024 Databricks Inc. — All rights reserved

DEMO

©2024 Databricks Inc. — All rights reserved

pyspark-data-sources

An open source repo with demo Python data sources

- All examples in the following demo are open source
- You can install them using
    - `pip install pyspark-data-sources[all]`

©2024 Databricks Inc. — All rights reserved

Demo: Fake Data Source

A data source that generates synthetic data

©2024 Databricks Inc. — All rights reserved

Demo: REST API Data Source

A data source that fetches data from a REST API

©2024 Databricks Inc. — All rights reserved

Demo: HuggingFace Datasets

A data source that fetches datasets from HuggingFace

©2024 Databricks Inc. — All rights reserved

DATA SOURCE READ

```
spark.read
    .format("my-source")
    .option("key", "value")
    .load()
```

Warning: Quiz at the end

©2024 Databricks Inc. — All rights reserved

Simple Example

A simple data source that generates one row

```python
from pyspark.sql.datasource import DataSource, DataSourceReader

class SimpleDataSource(DataSource):
   @classmethod
   def name(self):
       return "simple"

   def schema(self):
       return "id int, name string"

   def reader(self, schema):
       return SimpleReader()

class SimpleReader(DataSourceReader):
   def read(self, partition):
       yield (1, "Alice")
```


©2024 Databricks Inc. — All rights reserved

Simple Example

A simple data source that generates one row

- Register the data source:

```
spark.dataSource.register(SimpleDataSource)
```

- Load the data using its name:

```
spark.read.format("simple").load().show()

+---+-----+
| id| name|
+---+-----+
|  1|Alice|
+---+-----+
```


©2024 Databricks Inc. — All rights reserved

DataSource

The base class for Python data source

```
class DataSource(ABC):

    def __init__(self, options: Dict[str, str]) -> None:
        self.options = options

    @classmethod
    def name(cls) -> str:
        ...

    def schema(self) -> Union[StructType, str]:
        ...

    def reader(self, schema: StructType):
        ...

  ...
```


©2024 Databricks Inc. — All rights reserved

DataSource

The base class for Python data source

```python
class DataSource(ABC):

    def __init__(self, options: Dict[str, str]) -> None:
        self.options = options

    @classmethod
    def name(cls) -> str:
        ...

    def schema(self) -> Union[StructType, str]:
        ...

    def reader(self, schema: StructType):
        ...

  ...
```

User-defined values passed in via `option()` or `options()`:

```python
spark.read
.option("key", "value")

df.write
.option("key", "value")
```


©2024 Databricks Inc. — All rights reserved

DataSource

The base class for Python data source

```
class DataSource(ABC):

    def __init__(self, options: Dict[str, str]) -> None:
        self.options = options

    @classmethod
    def name(cls) -> str:
        ...

    def schema(self) -> Union[StructType, str]:
        ...

    def reader(self, schema: StructType):
        ...

  ...
```

`path` is special key that stores the path value from `load()` and `save()`:

```
spark.read
.load("path/to/file")

df.write
.save("path/to/file")

self.options["path"]
> "path/to/file"
```


©2024 Databricks Inc. — All rights reserved

Simple Example

A simple data source that generates one row

```
from pyspark.sql.datasource import DataSource, DataSourceReader

class SimpleDataSource(DataSource):
   @classmethod
   def name(self):
       return "simple"

   def schema(self):
       return "id int, name string"

   def reader(self, schema):
       return SimpleReader()

class SimpleReader(DataSourceReader):
   def read(self, partition):
       yield (1, "Alice")
```

Short name of the data source that is used in `format()`.

©2024 Databricks Inc. — All rights reserved

Simple Example

A simple data source that generates one row

```
from pyspark.sql.datasource import DataSource, DataSourceReader

class SimpleDataSource(DataSource):
    @classmethod
    def name(self):
        return "simple"

    def schema(self):
        return "id int, name string"

    def reader(self, schema):
        return SimpleReader()

class SimpleReader(DataSourceReader):
    def read(self, partition):
        yield (1, "Alice")
```

Default schema when reading from a data source.

It can be static:

- DDL Sting
- StructType

Or dynamically determined

©2024 Databricks Inc. — All rights reserved

Simple Example

A simple data source that generates one row

```
from pyspark.sql.datasource import DataSource, DataSourceReader

class SimpleDataSource(DataSource):
    @classmethod
    def name(self):
        return "simple"

    def schema(self):
        return "id int, name string"

    def reader(self, schema):
        return SimpleReader()

class SimpleReader(DataSourceReader):
    def read(self, partition):
        yield (1, "Alice")
```

Instantiate a data source reader.

©2024 Databricks Inc. — All rights reserved

Intro to DataSourceReader

A base class for defining how to read from a data source

- The `DataSourceReader` is responsible for defining how data is read from a source and split for parallel processing.
- It includes two methods: `partitions` and `read`

```
class DataSourceReader(ABC):

    def partitions(self) -> Sequence[InputPartition]:
        ...

    @abstractmethod
    def read(self, partition: InputPartition) -> Iterator[Row]:
        ...
```


©2024 Databricks Inc. — All rights reserved

Intro to DataSourceReader

A base class for defining how to read from a data source

- The `DataSourceReader` is responsible for defining how data is read from a source and split for parallel processing.
- It includes two methods: `partitions` and `read`

```
class DataSourceReader(ABC):

    def partitions(self) -> Sequence[InputPartition]:
        ...

    @abstractmethod
    def read(self, partition: InputPartition) -> Iterator[Row]:
        ...
```

How to split the data for parallel processing

©2024 Databricks Inc. — All rights reserved

Intro to DataSourceReader

A base class for defining how to read from a data source

- The `DataSourceReader` is responsible for defining how data is read from a source and split for parallel processing.
- It includes two methods: `partitions` and `read`

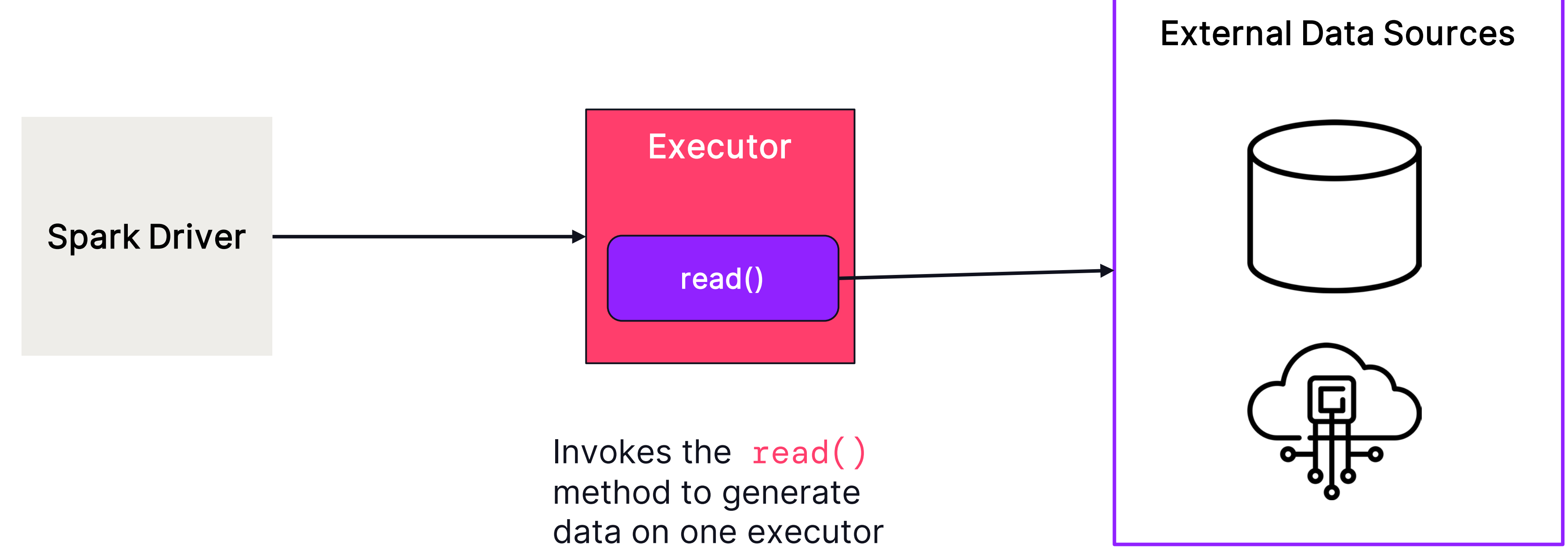
```
class DataSourceReader(ABC):

    def partitions(self) -> Sequence[InputPartition]:
        ...

    @abstractmethod
    def read(self, partition: InputPartition) -> Iterator[Row]:
        ...
```

How to read the data for each partition

©2024 Databricks Inc. — All rights reserved

Reading from a Data Source

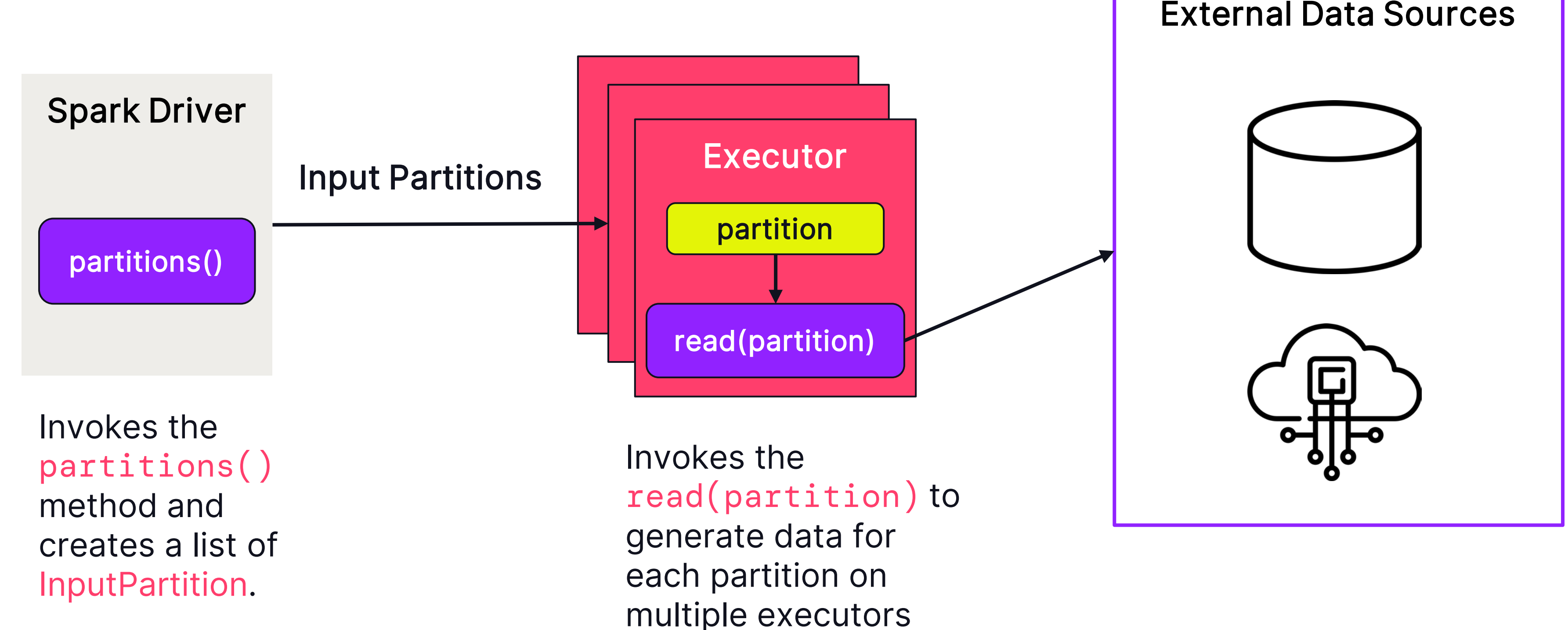
Without partitions



©2024 Databricks Inc. — All rights reserved

Reading from a Data Source

With partitions



©2024 Databricks Inc. — All rights reserved

Simple Example with Partitions

Reading data in parallel

```
@dataclass
class RangePartition(InputPartition):
    start: int
    end: int

class SimpleReader(DataSourceReader):

    def partitions(self):
        return [RangePartition(1, 3), RangePartition(3, 5)]

    def read(self, partition):
        for i in range(partition.start, partition.end):
            yield (i, f"name_{i}")
```

Subclass the `InputPartition`

Implement the `partitions()` method to return two range partitions

©2024 Databricks Inc. — All rights reserved

Simple Example with Partitions

Reading data in parallel

```
from pyspark.sql.functions import spark_partition_id

df = spark.read.format("simple").load()
df.withColumn("partition_id", spark_partition_id()).show()

+---+------+------------+
| id|  name|partition_id|
+---+------+------------+
|  1|name_1|           0|
|  2|name_2|           0|
|  3|name_3|           1|
|  4|name_4|           1|
+---+------+------------+
```

RangePartition(1, 3)

RangePartition(3, 5)

©2024 Databricks Inc. — All rights reserved

Partitioning Strategy

Key considerations for implementing readable data sources

- When to implement partitions?
  - Handling large datasets
  - Parallelization to boost performance
- Why not partitioning?
  - Simpler to manage
  - Data volume might not require it
  - Data source constraints
- Common strategies for partitioning
  - Range partitions (e.g, start_date, end_date)

©2024 Databricks Inc. — All rights reserved

How It Works

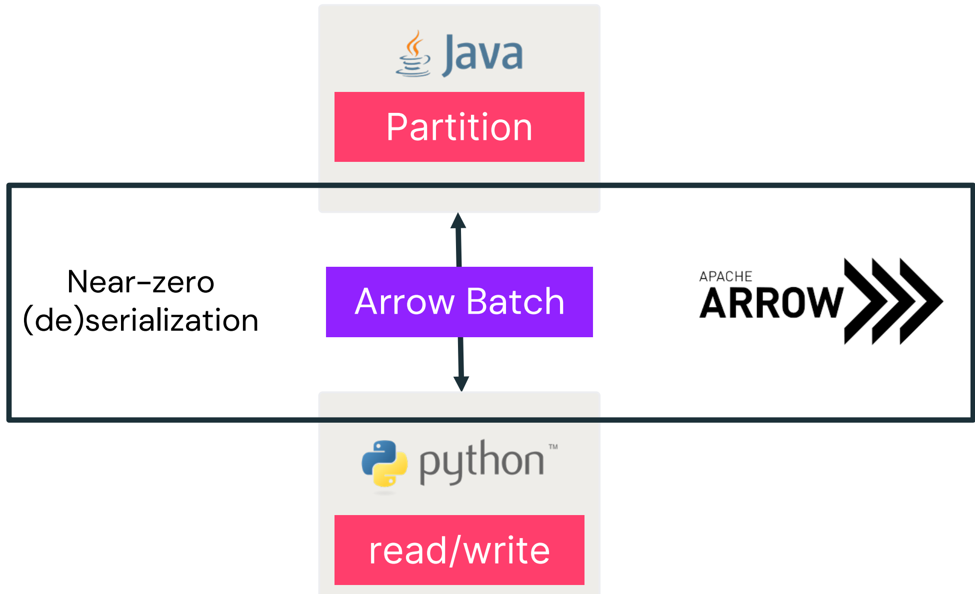
DataSource V2 API

Python data source is implemented on top of DataSource V2.

- PythonBatchWriterFactory.scala
- PythonCustomMetric.scala
- PythonDataSourceV2.scala
- PythonMicroBatchStream.scala
- PythonPartitionReaderFactory.scala
- PythonScan.scala
- PythonScanBuilder.scala

Apache Arrow

Use Apache Arrow for (de)serialization to improve performance.



©2024 Databricks Inc. — All rights reserved

Quiz Time!

©2024 Databricks Inc. — All rights reserved

Quiz 1

Understanding Data Source API basics

What method in the DataSourceReader must be overridden to specify how to read data from each partition?

A) `schema`

B) `partitions`

C) `read`

D) `name`

©2024 Databricks Inc. — All rights reserved

Quiz 1

Understanding Data Source API basics

What method in the DataSourceReader must be overridden to specify how to read data from each partition?

A) schema

B) partitions

C) read

D) name

©2024 Databricks Inc. — All rights reserved

Quiz 2

Data source registration and usage

How do you register a custom data source class SimpleDataSource in Spark?

A) spark.add_datasource(SimpleDataSource)

B) spark.dataSource.register(SimpleDataSource)

C) spark.dataSource.register("SimpleDataSource")

D) You don't need to 🙂

©2024 Databricks Inc. — All rights reserved

Quiz 2

Data source registration and usage

How do you register a custom data source class SimpleDataSource in Spark?

A) `spark.add_datasource(SimpleDataSource)`

B) `spark.dataSource.register(SimpleDataSource)`

C) `spark.dataSource.register(“SimpleDataSource”)`

D) You don’t need to 🙂

©2024 Databricks Inc. — All rights reserved

DATA SOURCE WRITE

```
df.write
    .format("my-source")
    .mode("append")
    .option("key", "value")
    .save("path/to/file")
```


©2024 Databricks Inc. — All rights reserved

Simple Example with Writer

A simple writable data source

```python
from pyspark.sql.datasource import DataSource, DataSourceWriter,
WriterCommitMessage

class SimpleDataSource(DataSource):
  @classmethod
  def name(self):
      return "simple"

  def writer(self, schema, overwrite):
      return SimpleWriter()

class SimpleWriter(DataSourceWriter):
   def write(self, iterator):
       for row in iterator:
           pass
       return WriterCommitMessage()
```

Implement the writer() method to make a data source writable.

©2024 Databricks Inc. — All rights reserved

Simple Example with Writer

A simple writable data source

- Register the data source (again):

```
spark.dataSource.register(SimpleDataSource)
```

- Write a dataframe into the sink:

```
df = spark.range(10).repartition(2)

df.write.format("simple").mode("append").save()

> (2) spark jobs
```


©2024 Databricks Inc. — All rights reserved

Intro to DataSourceWriter

A base class for defining how to write data

- The DataSourceWriter is responsible for defining how data is written in Spark.
- It has three methods designed to handle different aspect of the write process: `write`, `commit`, and `abort`

```
class DataSourceWriter(ABC):

    @abstractmethod
    def write(self, iterator: Iterator[Row]) -> "WriterCommitMessage":
        ...

    def commit(self, messages: List["WriterCommitMessage"]) -> None:
        ...

    def abort(self, messages: List["WriterCommitMessage"]) -> None:
        ...
```


©2024 Databricks Inc. — All rights reserved

Intro to DataSourceWriter

A base class for defining how to write data

- The DataSourceWriter is responsible for defining how data is written in Spark.
- It has three methods designed to handle different aspect of the write process: `write`, `commit`, and `abort`

```
class DataSourceWriter(ABC):

    @abstractmethod
    def write(self, iterator: Iterator[Row]) -> "WriterCommitMessage":
        ...

    def commit(self, messages: List["WriterCommitMessage"]) -> None:
        ...

    def abort(self, messages: List["WriterCommitMessage"]) -> None:
        ...
```

Process all rows in a given partition passed as an iterator.

Returns a `WriterCommitMessage` that will be used by the commit and abort methods.

©2024 Databricks Inc. — All rights reserved

Intro to DataSourceWriter

A base class for defining how to write data

- The DataSourceWriter is responsible for defining how data is written in Spark.
- It has three methods designed to handle different aspect of the write process: `write`, `commit`, and `abort`

```
class DataSourceWriter(ABC):

    @abstractmethod
    def write(self, iterator: Iterator[Row]) -> "WriterCommitMessage":
        ...

    def commit(self, messages: List["WriterCommitMessage"]) -> None:
        ...

    def abort(self, messages: List["WriterCommitMessage"]) -> None:
        ...
```

Handles transactional aspect of the write process

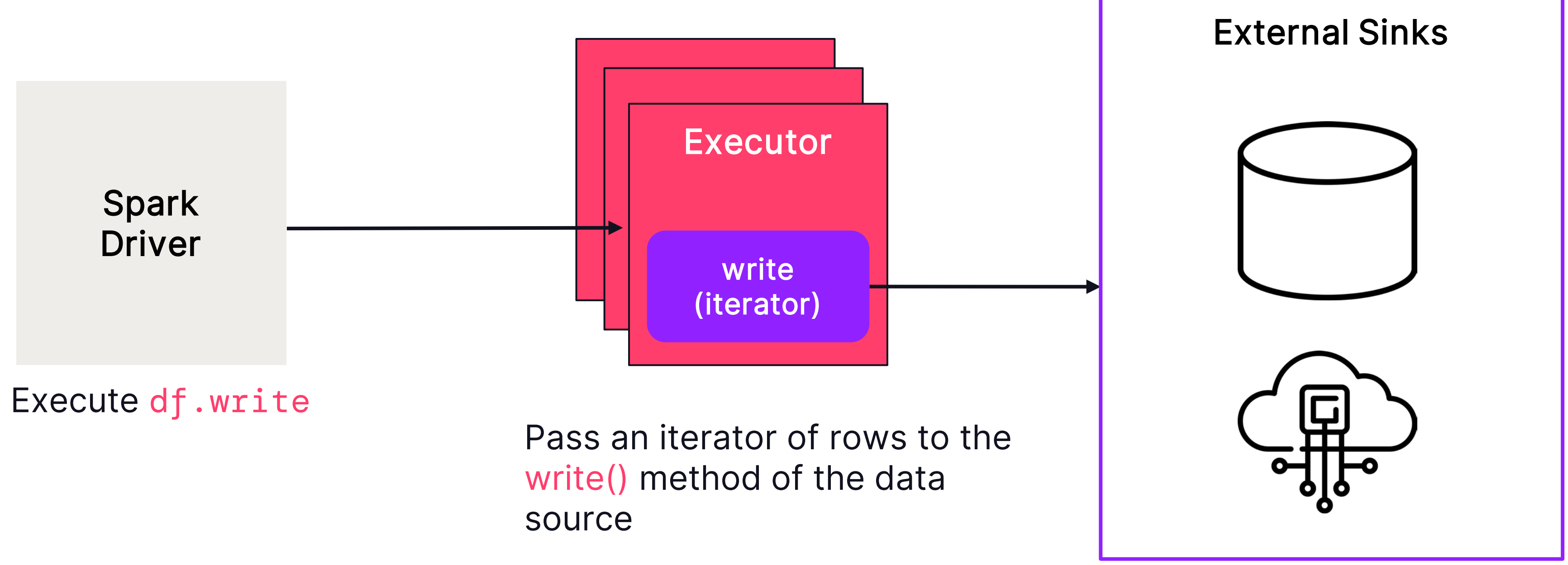
Commit: Finalizes successful write operations

Abort: Ensures proper cleanup if a write task fails

©2024 Databricks Inc. — All rights reserved

Initiating the Write Process

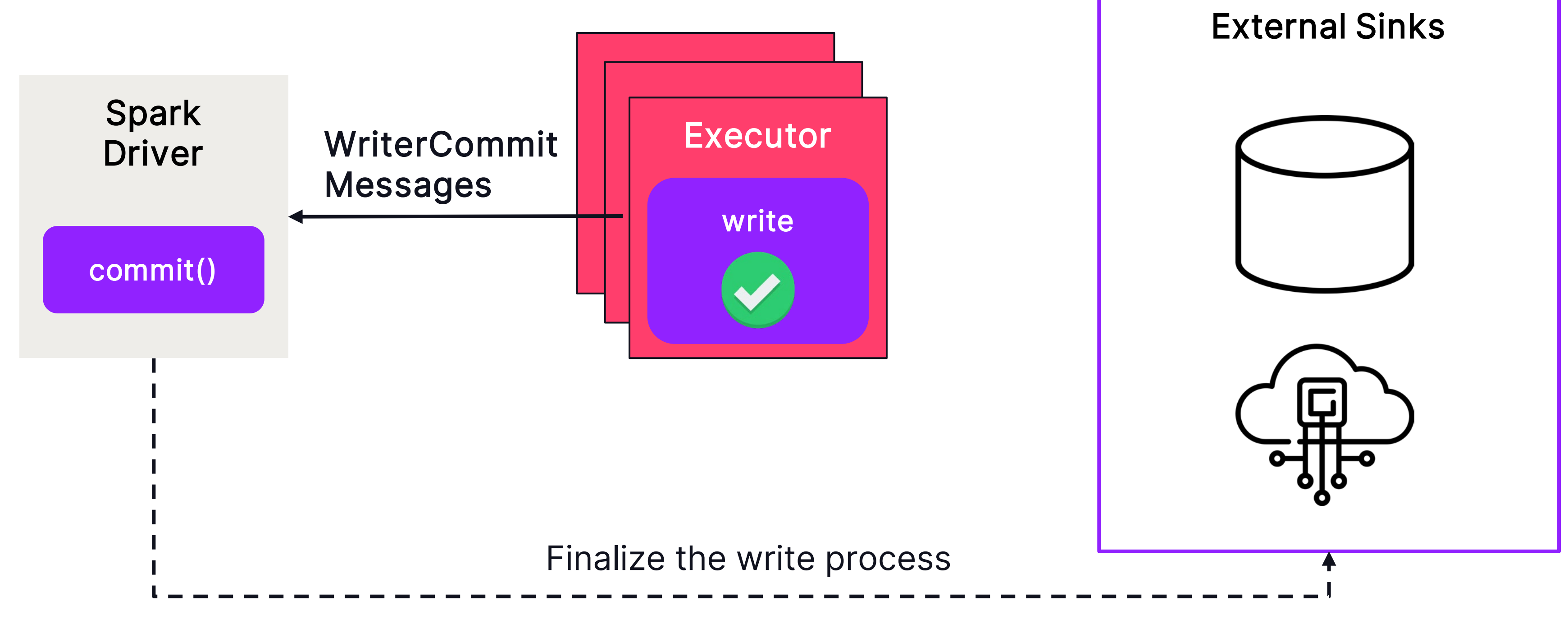
How the commit protocol works



©2024 Databricks Inc. — All rights reserved

Handling Task Success - Commit

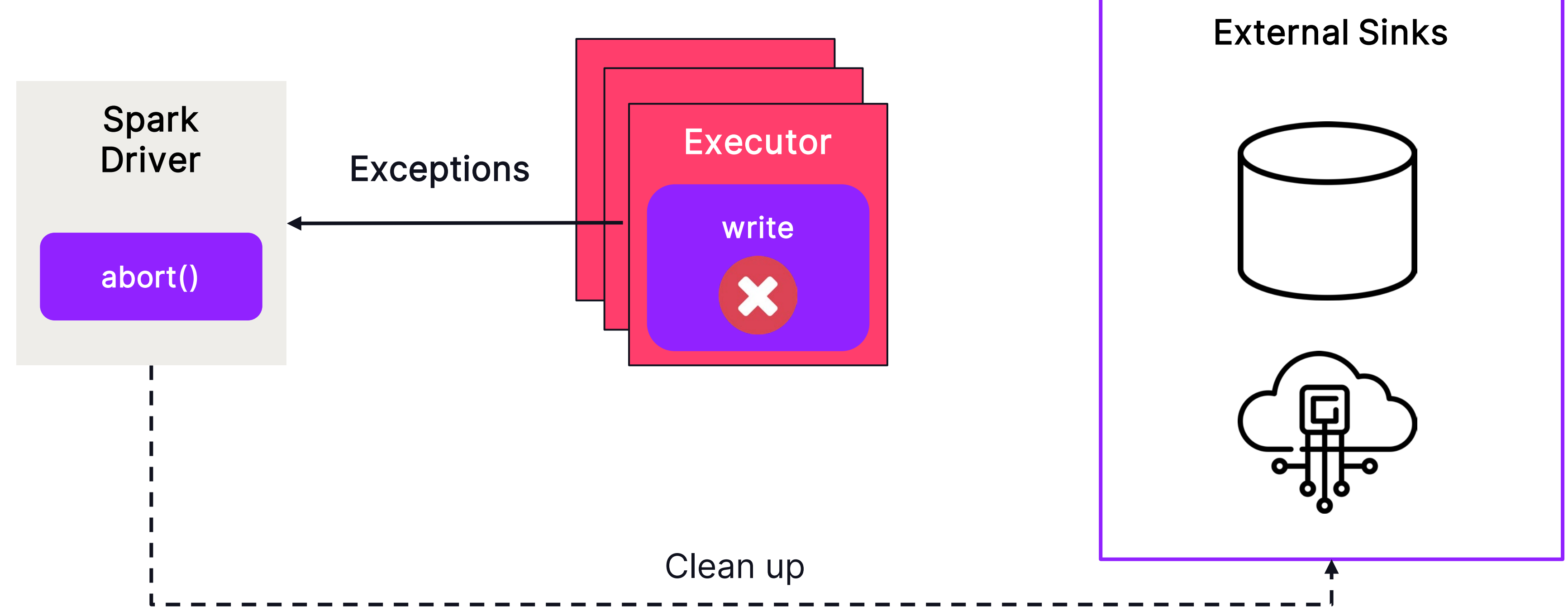
How the commit protocol works



©2024 Databricks Inc. — All rights reserved

Handling Task Failure - Abort

How the commit protocol works



©2024 Databricks Inc. — All rights reserved

STREAMING DATA SOURCE

```
spark.readStream
    .format("my-source")
    .option("key", "value")
    .load()
    .writeStream
    .format("my-sink")
    .start()
```

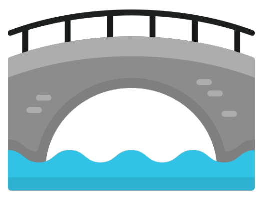

©2024 Databricks Inc. — All rights reserved

DEMO

©2024 Databricks Inc. — All rights reserved

Conclusion

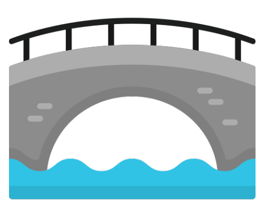
Before Python data source API



©2024 Databricks Inc. — All rights reserved

Conclusion

After Python data source API



©2024 Databricks Inc. — All rights reserved

Other Recommended Sessions

View after the conference to learn about Spark OSS, Python, and Streaming

Session	Area
Your Guide to Data Engineering on the Data Intelligence Platform	Data Engineering
Supercharge Your Data Warehouse: Introducing Python Stored Procedures	PySpark
Databricks Streaming: Project Lightspeed Goes Hyperspeed	Streaming
What's Next for the Upcoming Apache Spark 4.0	Spark OSS
Introducing Databricks' New Native Ingestion Connectors	Streaming
Streaming Data Pipelines: From Supernovas to LLMs	Streaming, LLM
Exploring UDTFs (User-Defined Table Functions) in PySpark	PySpark

©2024 Databricks Inc. — All rights reserved

Take Home Exercises

Build your own data sources!

- [**Easy**] Implement a REST API data source that reads data from a REST API (e.g financial data, weather data).
- [**Intermediate**] Implement a **streaming** REST API data source
- [**Intermediate**] Implement a data source that fetches Kaggle datasets.
- [**Hard**] Implement a data source and sink that reads data from and writes data to Excel files.

©2024 Databricks Inc. — All rights reserved

Learn more at the summit!

Tells us what you think

- We kindly request your valuable feedback on this session.
- Please take a moment to rate and share your thoughts about it.
- You can conveniently provide your feedback and rating through the **Mobile App**.

What to do next?

- Discover more related sessions in the mobile app!
- Visit the Demo Booth: Experience innovation firsthand!
- More Activities: Engage and connect further at the Databricks Zone!

Get trained and certified

- Visit the Learning Hub Experience at Moscone West, 2nd Floor!
- Take complimentary certification at the event; come by the Certified Lounge
- Visit our Databricks Learning website for more training, courses and workshops! databricks.com/learn

DATA+AI SUMMIT

©2024 Databricks Inc. — All rights reserved