Data Source API V2

Wenchen Fan

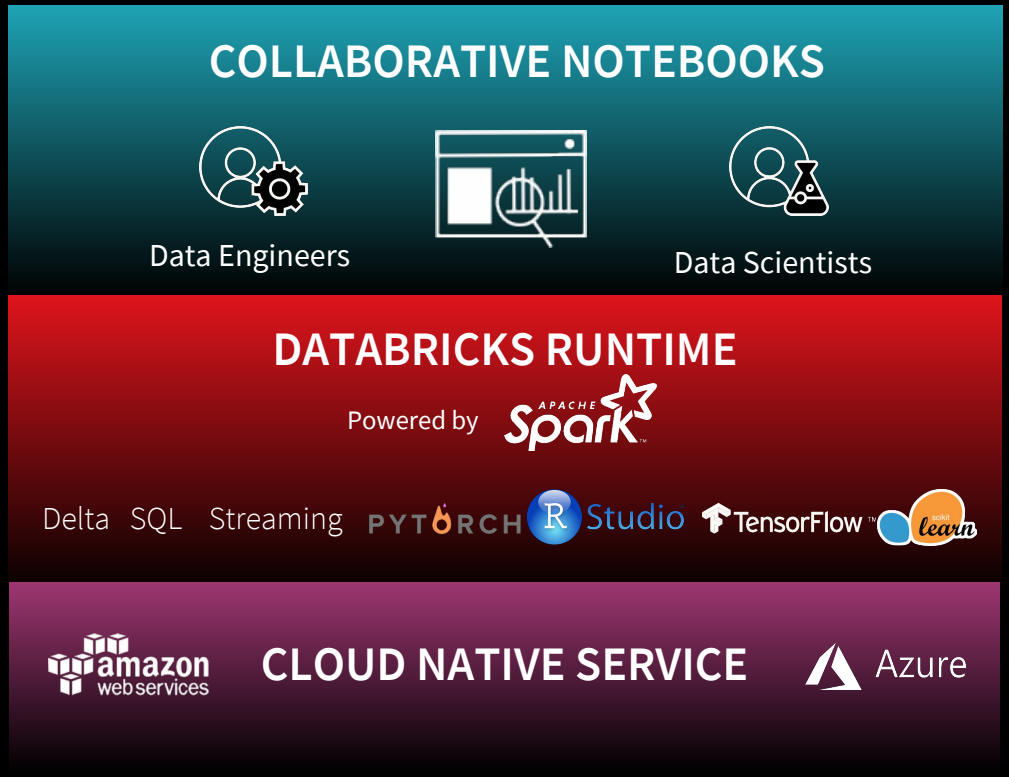
2018-6-6 | SF | Spark + AI Summit

Databricks' Unified Analytics Platform

Unifies Data Engineers and Data Scientists

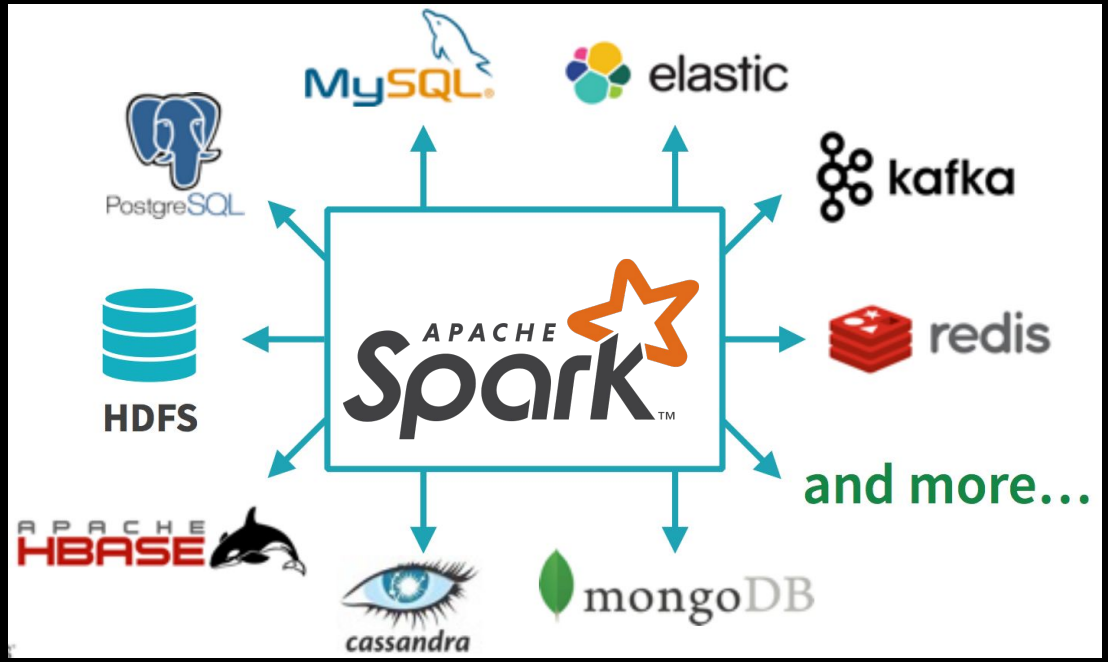
Unifies Data and AI Technologies

Eliminates infrastructure complexity



databricks

What is Data Source API?



What is Data Source API?

- Hadoop: InputFormat/OutputFormat
- Hive: Serde
- Presto: Connector

  …….

Defines how to read/write data from/to a storage system.

Ancient Age: Custom RDD

```
HadoopRDD/CassandraRDD/HBaseRDD/…
rdd.mapPartitions { it =>
  // custom logic to write to external storage
}
```

Good in the ancient ages when users writing Spark applications with RDD API.

New Requirements When Switching to Spark SQL

How to read data?

- How to read data concurrently and distributedly? (RDD only satisfy this)
- How to skip reading data by filters?
- How to speed up certain operations? (aggregate, limit, etc.)
- How to convert data using Spark's data encoding?
- How to report extra information to Spark? (data statistics, data partitioning, etc.)
- Structured Streaming Support

  …….

How to write data?

- How to write data concurrently and distributedly? (RDD only satisfy this)
- How to make the write operation atomic?
- How to clean up if write failed?
- How to convert data using Spark's data encoding?
- Structured streaming support

  …….

Data Source API V1 for Spark SQL

Data Source API V1

```scala
// Read
trait RelationProvider {
  def createRelation(context: SQLContext, options: Map[String, String]): BaseRelation
}

abstract class BaseRelation {
  def sizeInBytes: Long
}

trait TableScan {
  def buildScan(): RDD[Row]
}

trait PrunedFilteredScan {
  def buildScan(requiredColumns: Array[String], filters: Array[Filter]): RDD[Row]
}

// Write
trait InsertableRelation {
  def insert(data: DataFrame, overwrite: Boolean): Unit
}
```

Data Source API V1

Pros:

- Simple
- Works well for the most common cases

Data Source API V1

Cons:

- Coupled with other APIs. (SQLContext, RDD, DataFrame)

Data Source API V1

```scala
// Read
trait RelationProvider {
  def createRelation(context: SQLContext, options: Map[String, String]): BaseRelation
}

abstract class BaseRelation {
  def sizeInBytes: Long
}

trait TableScan {
  def buildScan(): RDD[Row]
}

trait PrunedFilteredScan {
  def buildScan(requiredColumns: Array[String], filters: Array[Filter]): RDD[Row]
}

// Write
trait InsertableRelation {
  def insert(data: DataFrame, overwrite: Boolean): Unit
}
```

Data Source API V1

Cons:

- Coupled with other APIs. (SQLContext, RDD, DataFrame)
- Hard to push down other operators.

Data Source API V1

```scala
// Read
trait RelationProvider {
  def createRelation(context: SQLContext, options: Map[String, String]): BaseRelation
}

abstract class BaseRelation {
  def sizeInBytes: Long
}

trait TableScan {
  def buildScan(): RDD[Row]
}

trait PrunedFilteredScan {
  def buildScan(requiredColumns: Array[String], filters: Array[Filter]): RDD[Row]
}

// Write
trait InsertableRelation {
  def insert(data: DataFrame, overwrite: Boolean): Unit
}
```

Data Source API V1

```scala
// Read
trait RelationProvider {
  def createRelation(context: SQLContext, options: Map[String, String]): BaseRelation
}

abstract class BaseRelation {
  def sizeInBytes: Long
}

trait TableScan {
  def buildScan(): RDD[Row]
}

trait PrunedFilteredScan {
  def buildScan(requiredColumns: Array[String], filters: Array[Filter]): RDD[Row]
}

// Write
trait InsertableRelation {
  def insert(data: DataFrame, overwrite: Boolean): Unit
}
```

buildScan(limit)
buildScan(limit, requiredCols)
buildScan(limit, filters)
buildScan(limit, requiredCols, filters)
…

Data Source API V1

Cons:

- Coupled with other APIs. (SQLContext, RDD, DataFrame)
- Hard to push down other operators.
- Hard to add different data encoding. (columnar scan)

Data Source API V1

```scala
// Read
trait RelationProvider {
  def createRelation(context: SQLContext, options: Map[String, String]): BaseRelation
}

abstract class BaseRelation {
  def sizeInBytes: Long
}

trait TableScan {
  def buildScan(): RDD[Row]
}

trait PrunedFilteredScan {
  def buildScan(requiredColumns: Array[String], filters: Array[Filter]): RDD[Row]
}

// Write
trait InsertableRelation {
  def insert(data: DataFrame, overwrite: Boolean): Unit
}
```

Data Source API V1

Cons:

- Coupled with other APIs. (SQLContext, RDD, DataFrame)
- Hard to push down other operators.
- Hard to add different data encoding. (columnar scan)
- Hard to implement writing.

Data Source API V1

```scala
// Read
trait RelationProvider {
  def createRelation(context: SQLContext, options: Map[String, String]): BaseRelation
}

abstract class BaseRelation {
  def sizeInBytes: Long
}

trait TableScan {
  def buildScan(): RDD[Row]
}

trait PrunedFilteredScan {
  def buildScan(requiredColumns: Array[String], filters: Array[Filter]): RDD[Row]
}

// Write
trait InsertableRelation {
  def insert(data: DataFrame, overwrite: Boolean): Unit
}
```

Data Source API V1

Cons:

- Coupled with other APIs. (SQLContext, RDD, DataFrame)
- Hard to push down other operators.
- Hard to add different data encoding. (columnar scan)
- Hard to implement writing.
- No streaming support

How to read data?

- How to read data concurrently and distributedly?
- How to skip reading data by filters?
- How to speed up certain operations?
- How to convert data using Spark's data encoding?
- How to report extra information to Spark?
- Structured streaming support

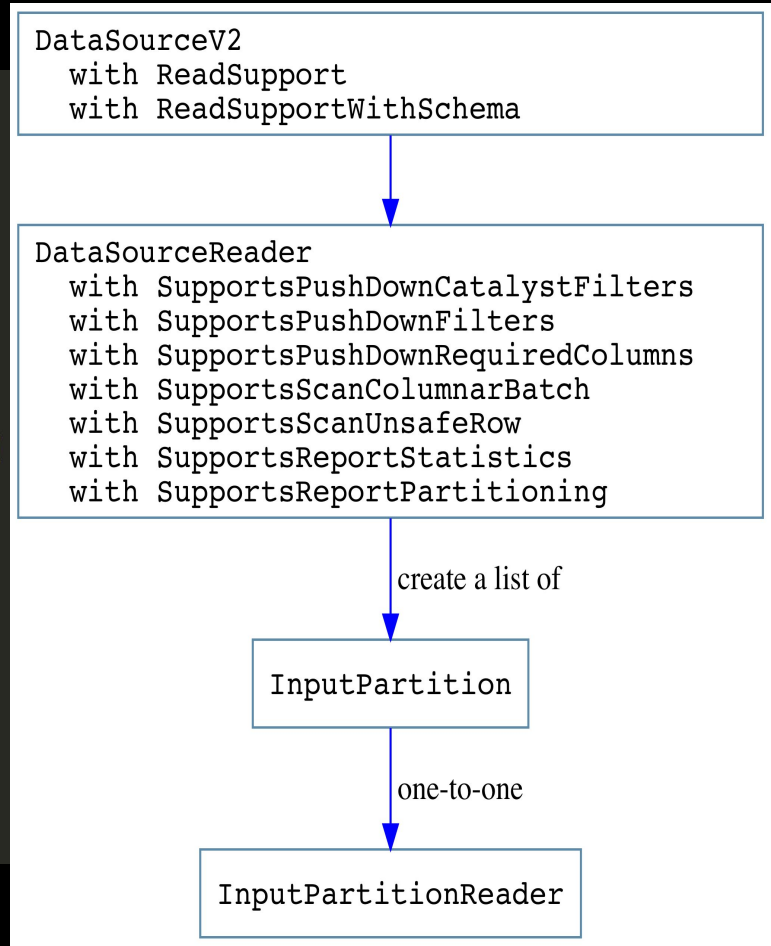
How to write data?

- How to write data concurrently and distributedly?
- How to make the write operation atomic?
- How to clean up if write failed?
- How to convert data using Spark's data encoding?
- Structured streaming support

What’s the design of Data Source API V2?

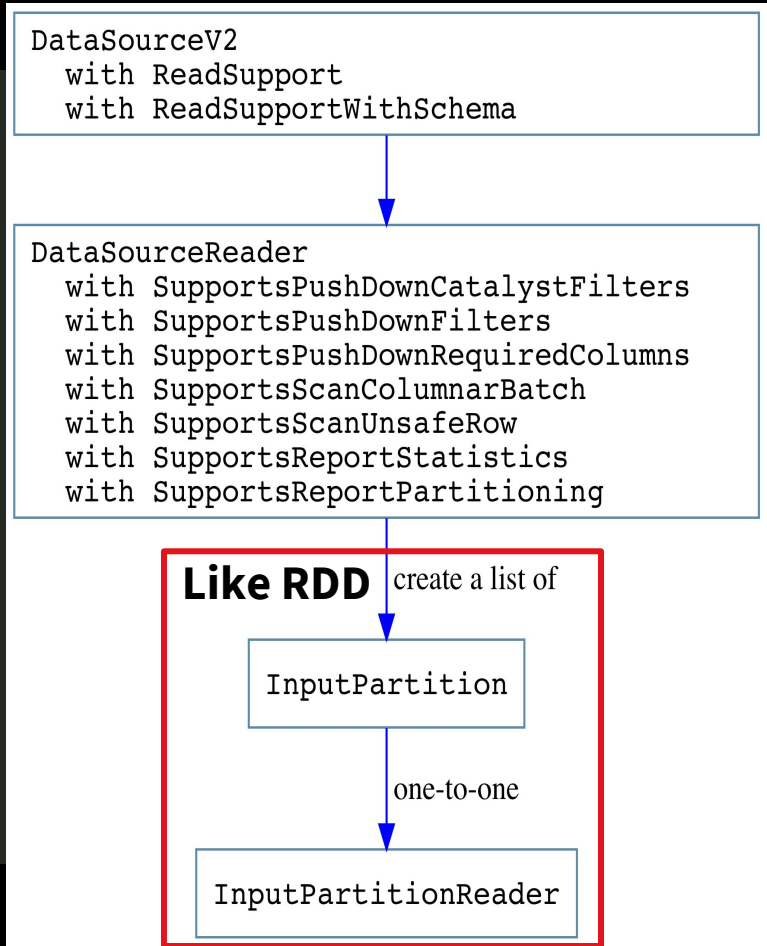
API Sketch (read)

```java
public interface DataSourceV2 {}

public interface ReadSupport extends DataSourceV2 {
  DataSourceReader createReader(DataSourceOptions options);
}

public interface DataSourceReader {
  List<InputPartition<Row>> planInputPartitions();
}

public interface SupportsPushDownFilters extends DataSourceReader {
  Filter[] pushFilters(Filter[] filters);
}

public interface InputPartition<T> extends Serializable {
  InputPartitionReader<T> createPartitionReader();
}

public interface InputPartitionReader<T> extends Closeable {
  boolean next() throws IOException;
  T get();
}
```



API Sketch (read)

```java
public interface DataSourceV2 {}

public interface ReadSupport extends DataSourceV2 {
  DataSourceReader createReader(DataSourceOptions options);
}

public interface DataSourceReader {
  List<InputPartition<Row>> planInputPartitions();
}

public interface SupportsPushDownFilters extends DataSourceReader {
  Filter[] pushFilters(Filter[] filters);
}

public interface InputPartition<T> extends Serializable {
  InputPartitionReader<T> createPartitionReader();
}

public interface InputPartitionReader<T> extends Closeable {
  boolean next() throws IOException;
  T get();
}
```



```java
public interface DataSourceV2 {}

public interface ReadSupport extends DataSourceV2 {
  DataSourceReader createReader(DataSourceOptions options);
}

public interface DataSourceReader {
  List<InputPartition<Row>> planInputPartitions();
}

public interface SupportsPushDownFilters extends DataSourceReader {
  Filter[] pushFilters(Filter[] filters);
}

public interface InputPartition<T> extends Serializable {
  InputPartitionReader<T> createPartitionReader();
}

public interface InputPartitionReader<T> extends Closeable {
  boolean next() throws IOException;
  T get();
}
```

Easy to extend

```java
public interface DataSourceV2 {}

public interface ReadSupport extends DataSourceV2 {
  DataSourceReader createReader(DataSourceOptions options);
}

public interface DataSourceReader {
  List<InputPartition<Row>> planInputPartitions();
}

public interface SupportsPushDownFilters extends DataSourceReader {
  Filter[] pushFilters(Filter[] filters);
}

public interface InputPartition<T> extends Serializable {
  InputPartitionReader<T> createPartitionReader();
}

public interface InputPartitionReader<T> extends Closeable {
  boolean next() throws IOException;
  T get();
}
```

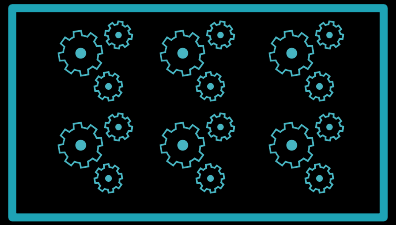
Easy to extend

```java
public interface SupportsPushDownLimit extends DataSourceReader {
  void pushLimit(int limit);
}

public interface SupportsScanColumnarBatch extends DataSourceReader {
  List<InputPartition<ColumnarBatch>> planBatchInputPartitions();
}
```

Read Process

Spark Driver

External Storage



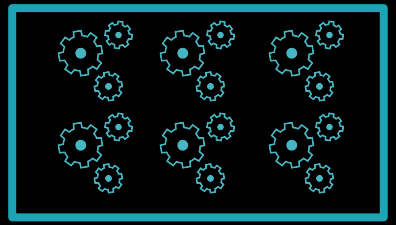
Spark Executors

Read Process

Spark Driver

1. a query plan generated by user
2. the leaf data scan node generates DataSourceReader

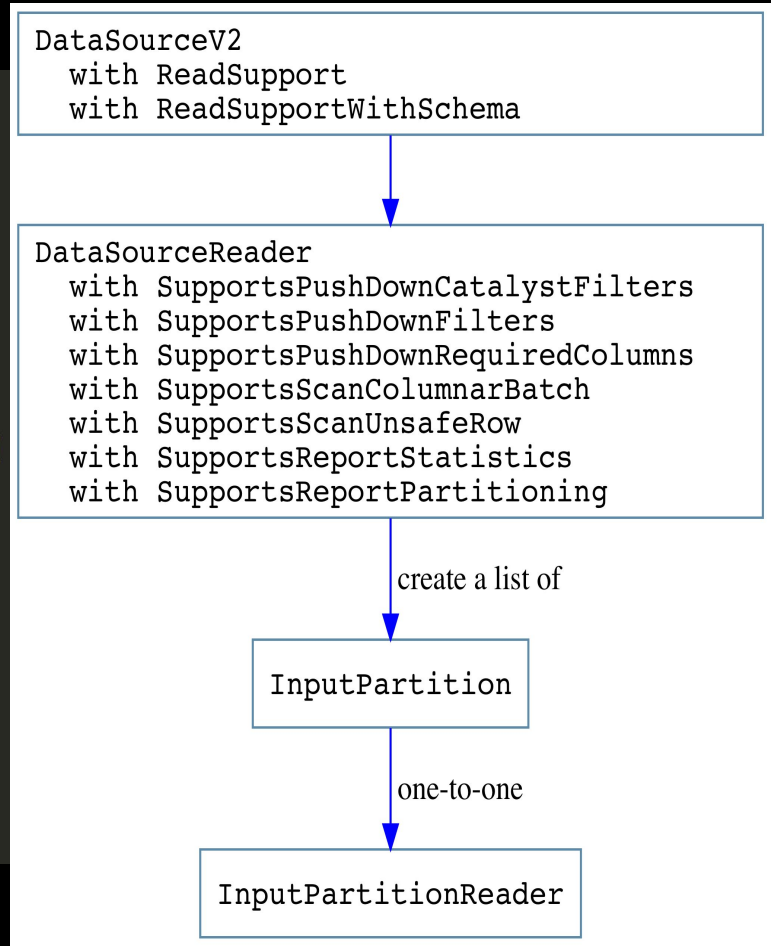
External Storage



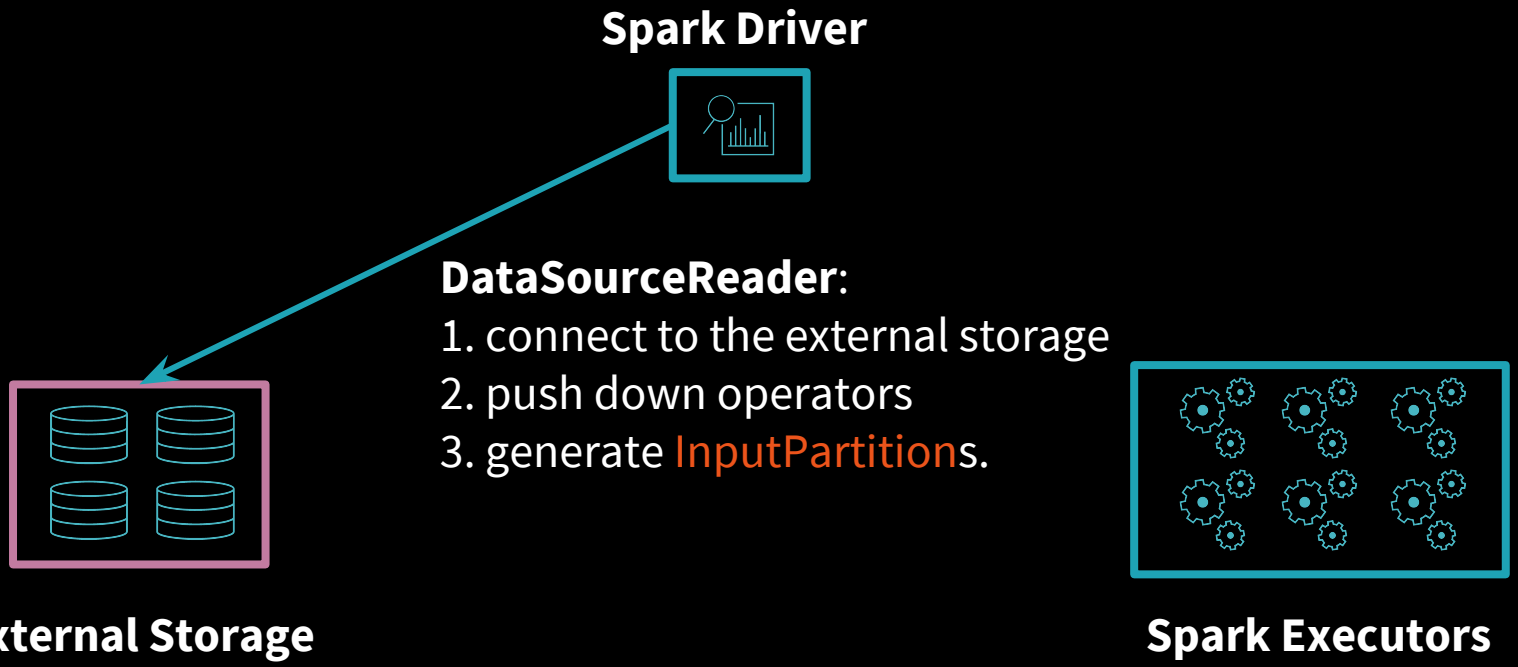
Spark Executors

API Sketch (read)

```java
public interface DataSourceV2 {}

public interface ReadSupport extends DataSourceV2 {
  DataSourceReader createReader(DataSourceOptions options);
}

public interface DataSourceReader {
  List<InputPartition<Row>> planInputPartitions();
}

public interface SupportsPushDownFilters extends DataSourceReader {
  Filter[] pushFilters(Filter[] filters);
}

public interface InputPartition<T> extends Serializable {
  InputPartitionReader<T> createPartitionReader();
}

public interface InputPartitionReader<T> extends Closeable {
  boolean next() throws IOException;
  T get();
}
```

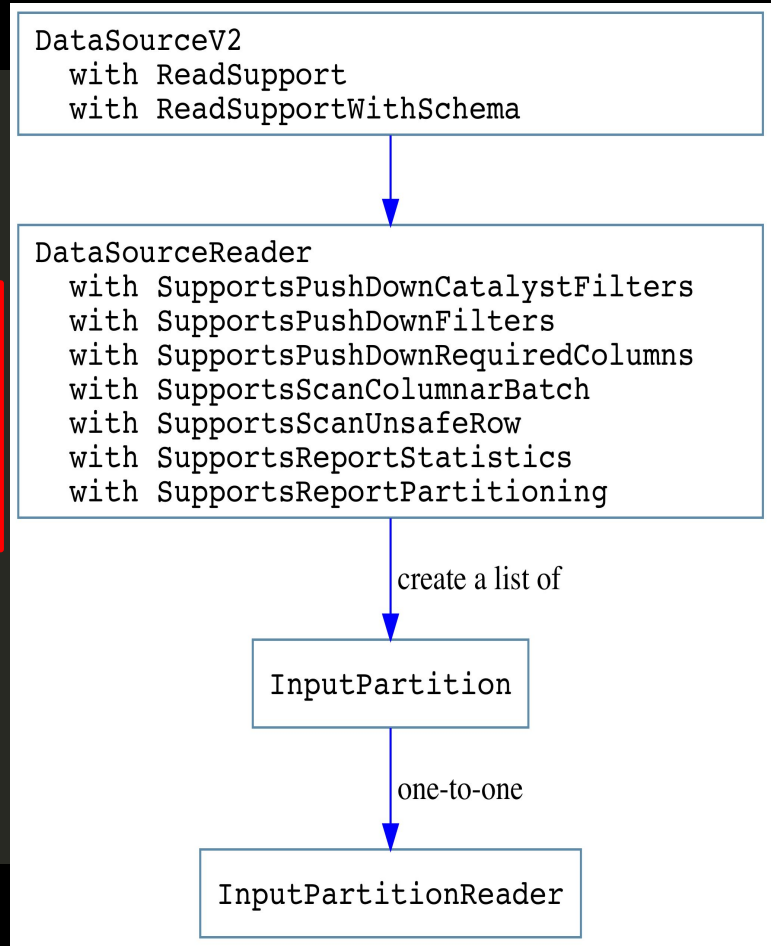


Read Process

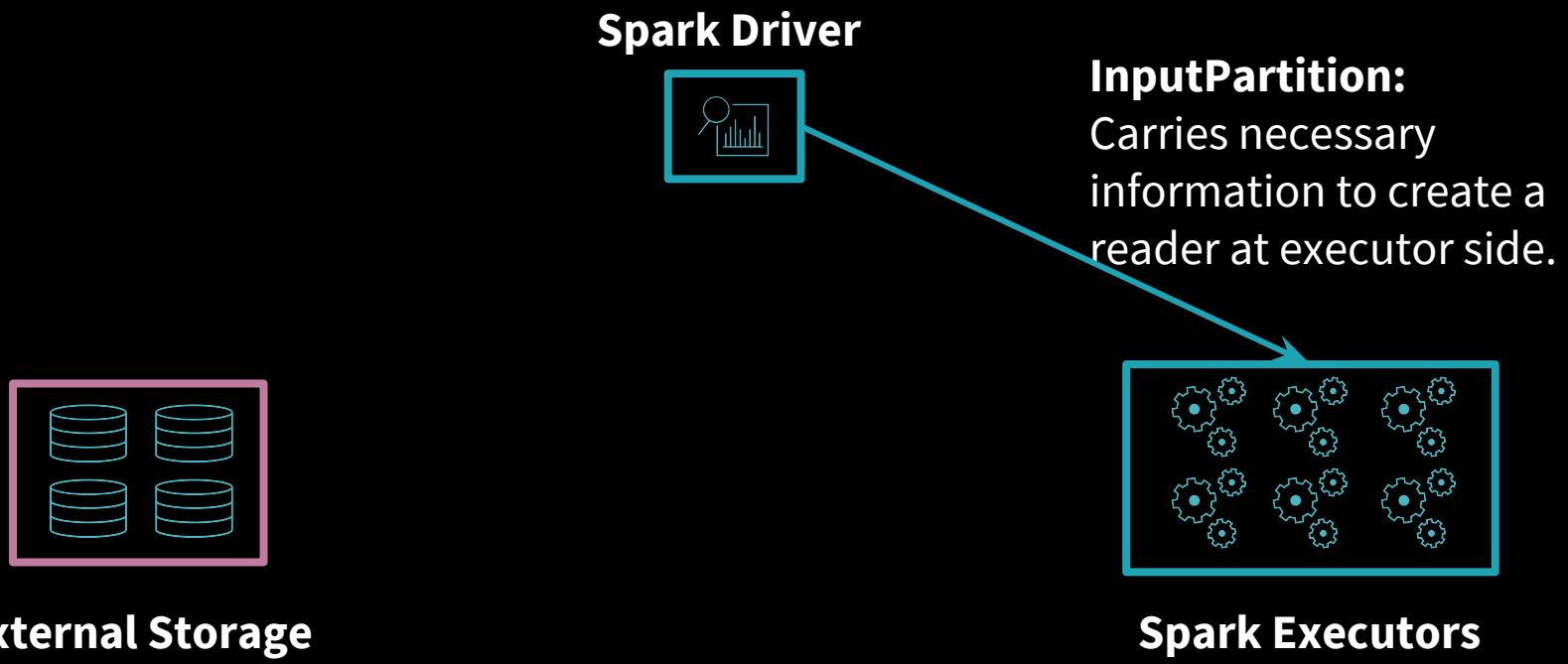


API Sketch (read)

```java
public interface DataSourceV2 {}

public interface ReadSupport extends DataSourceV2 {
  DataSourceReader createReader(DataSourceOptions options);
}

public interface DataSourceReader {
  List<InputPartition<Row>> planInputPartitions();
}

public interface SupportsPushDownFilters extends DataSourceReader {
  Filter[] pushFilters(Filter[] filters);
}

public interface InputPartition<T> extends Serializable {
  InputPartitionReader<T> createPartitionReader();
}

public interface InputPartitionReader<T> extends Closeable {
  boolean next() throws IOException;
  T get();
}
```

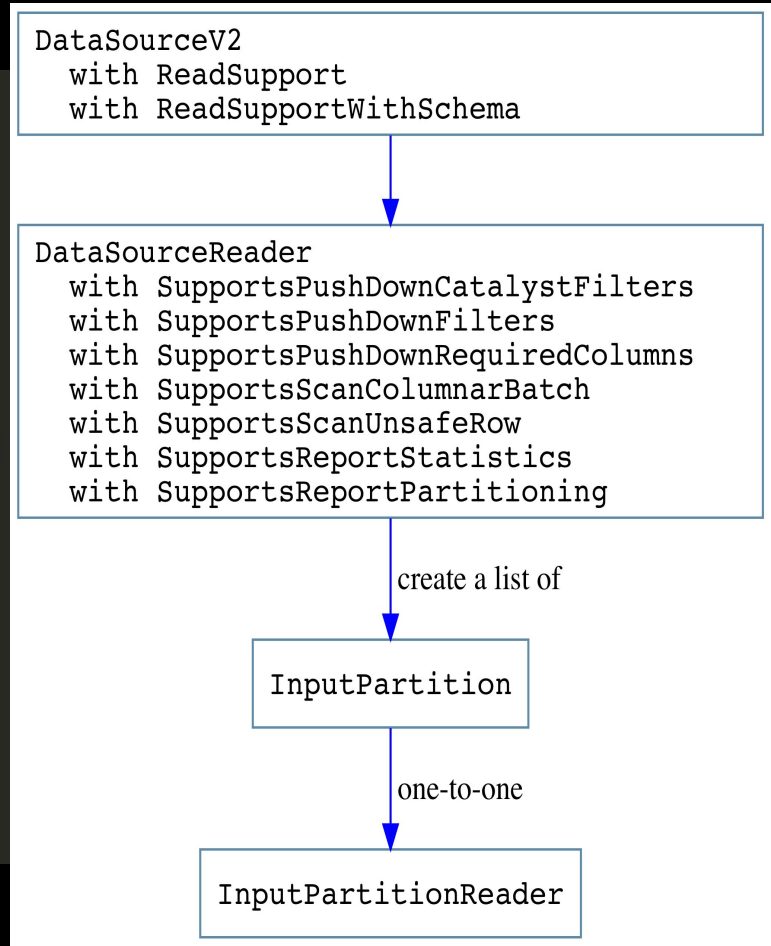


Read Process

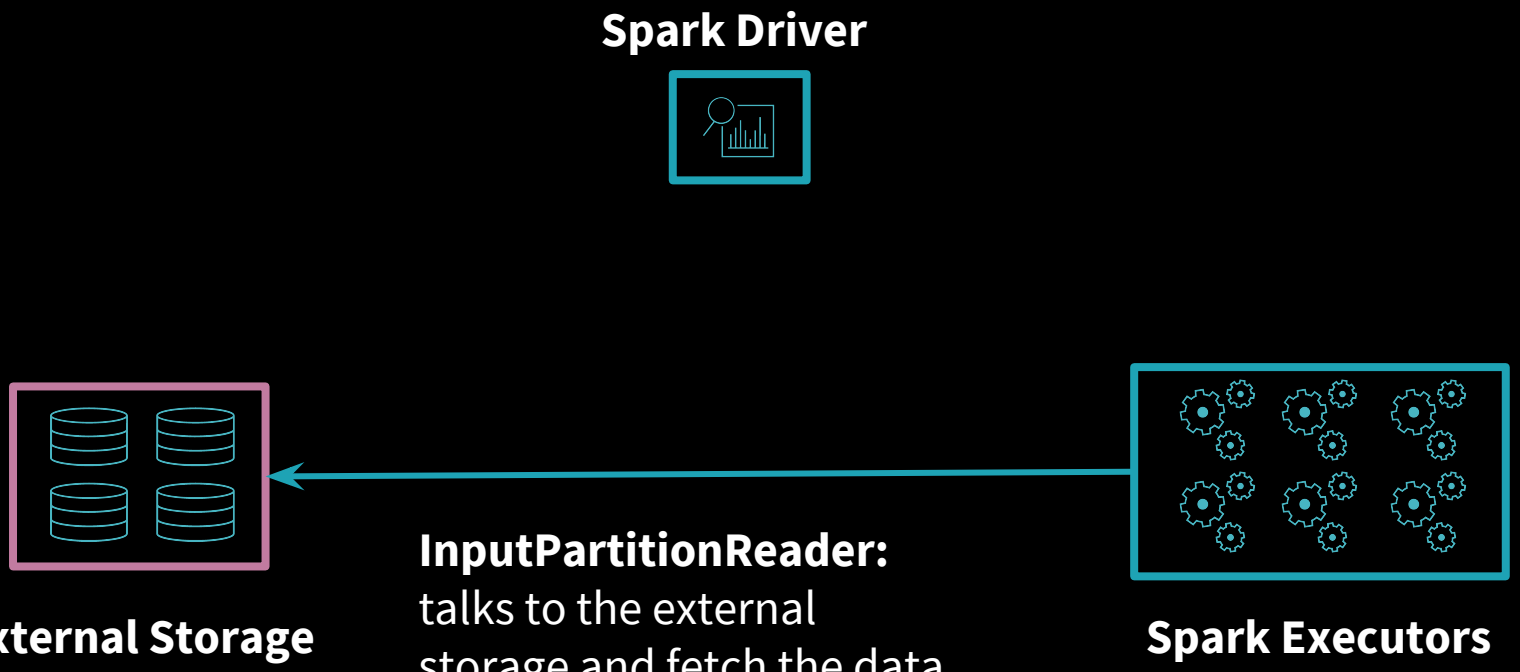


API Sketch (read)

```java
public interface DataSourceV2 {}

public interface ReadSupport extends DataSourceV2 {
  DataSourceReader createReader(DataSourceOptions options);
}

public interface DataSourceReader {
  List<InputPartition<Row>> planInputPartitions();
}

public interface SupportsPushDownFilters extends DataSourceReader {
  Filter[] pushFilters(Filter[] filters);
}

public interface InputPartition<T> extends Serializable {
  InputPartitionReader<T> createPartitionReader();
}

public interface InputPartitionReader<T> extends Closeable {
  boolean next() throws IOException;
  T get();
}
```

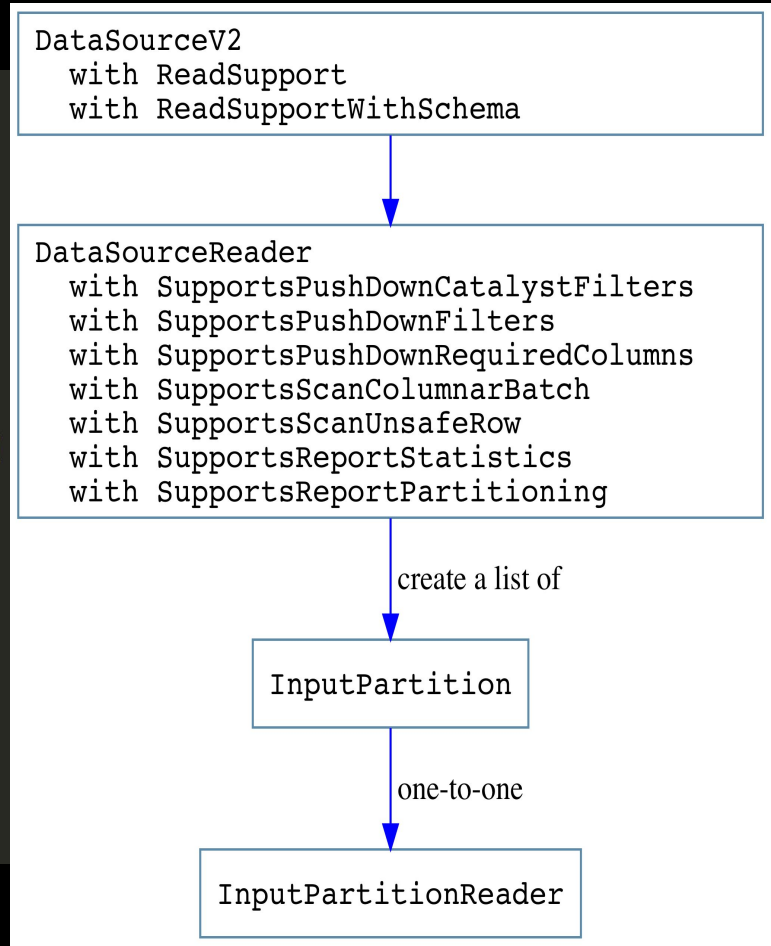


Read Process



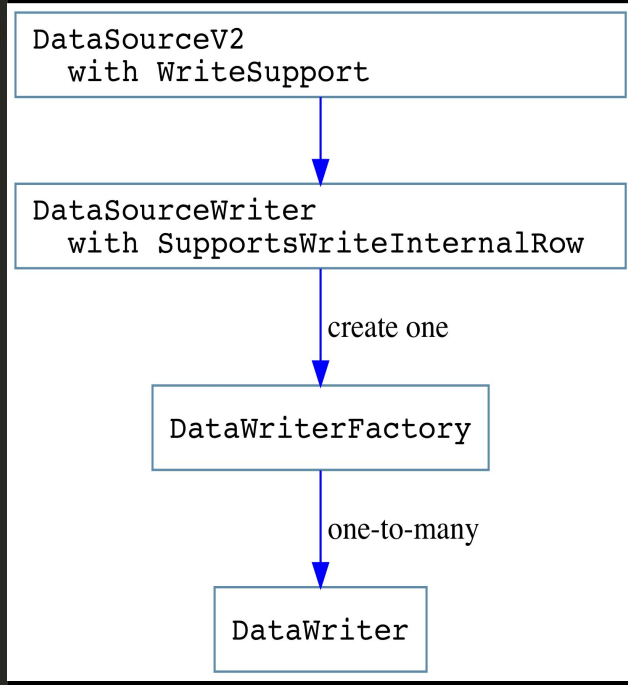
API Sketch (read)

```
public interface DataSourceV2 {}

public interface ReadSupport extends DataSourceV2 {
  DataSourceReader createReader(DataSourceOptions options);
}

public interface DataSourceReader {
  List<InputPartition<Row>> planInputPartitions();
}

public interface SupportsPushDownFilters extends DataSourceReader {
  Filter[] pushFilters(Filter[] filters);
}

public interface InputPartition<T> extends Serializable {
  InputPartitionReader<T> createPartitionReader();
}

public interface InputPartitionReader<T> extends Closeable {
  boolean next() throws IOException;
  T get();
}
```



API Sketch (write)

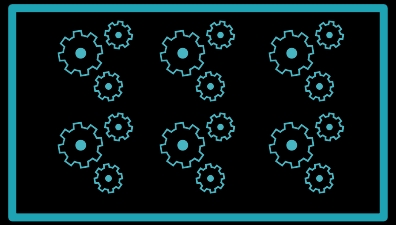
```scala
public interface WriteSupport extends DataSourceV2 {
  Optional<DataSourceWriter> createWriter(
      String jobId, StructType schema, SaveMode mode, DataSourceOptions options);
}

public interface DataSourceWriter {
  DataWriterFactory<Row> createWriterFactory();

  void commit(WriterCommitMessage[] messages);

  void abort(WriterCommitMessage[] messages);
}

public interface DataWriterFactory<T> extends Serializable {
  DataWriter<T> createDataWriter(int partitionId, int attemptNumber);
}

public interface DataWriter<T> {
  void write(T record) throws IOException;

  WriterCommitMessage commit() throws IOException;

  void abort() throws IOException;
}
```



Write Process

Spark Driver

External Storage



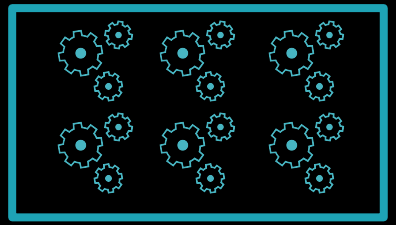
Spark Executors

Write Process

Spark Driver

1. a query plan generated by user
2. root data write node generates DataSourceWriter

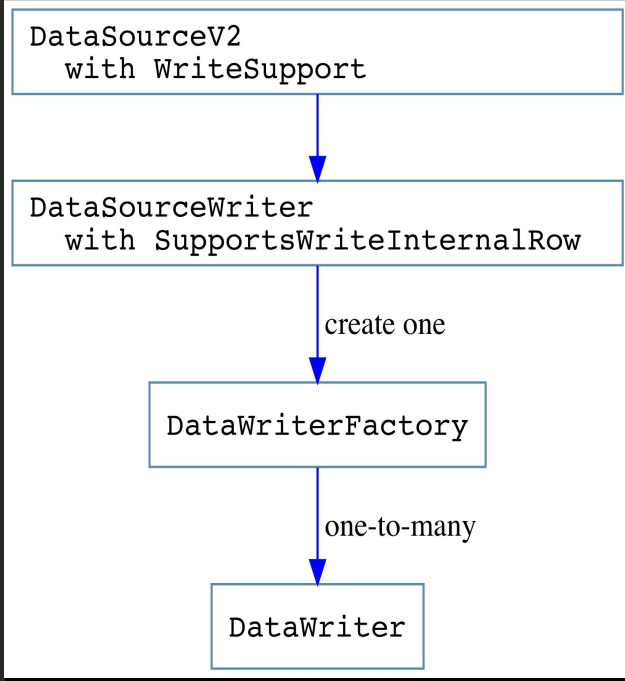
External Storage



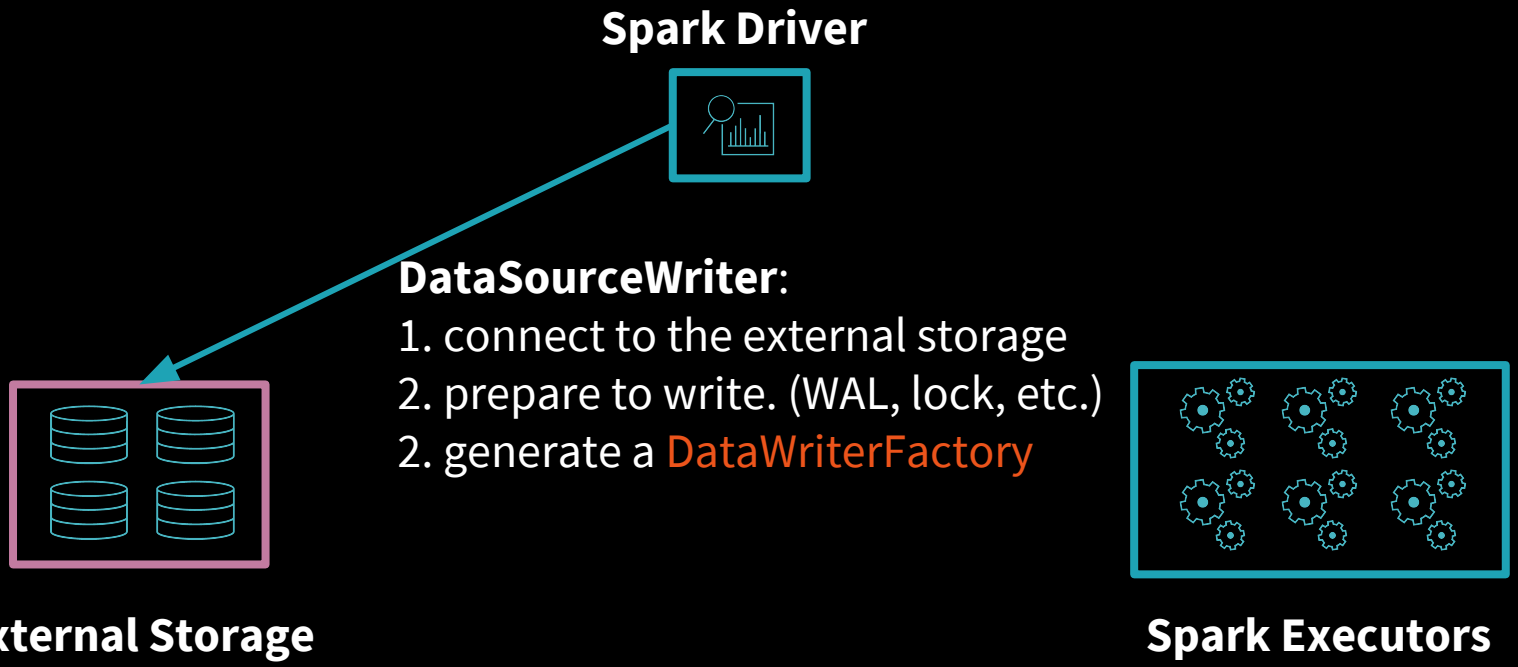
Spark Executors

API Sketch (write)

```scala
public interface WriteSupport extends DataSourceV2 {
  Optional<DataSourceWriter> createWriter(
      String jobId, StructType schema, SaveMode mode, DataSourceOptions options);
}

public interface DataSourceWriter {
  DataWriterFactory<Row> createWriterFactory();

  void commit(WriterCommitMessage[] messages);

  void abort(WriterCommitMessage[] messages);
}

public interface DataWriterFactory<T> extends Serializable {
  DataWriter<T> createDataWriter(int partitionId, int attemptNumber);
}

public interface DataWriter<T> {
  void write(T record) throws IOException;

  WriterCommitMessage commit() throws IOException;

  void abort() throws IOException;
}
```

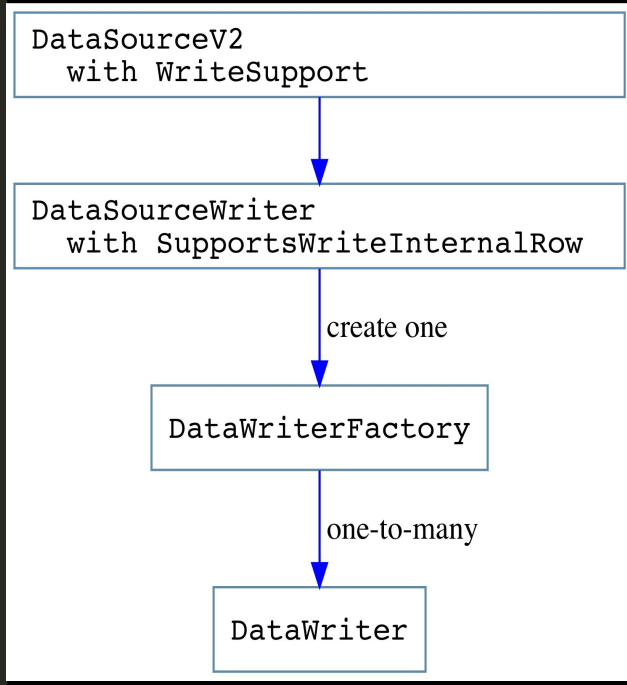


Write Process

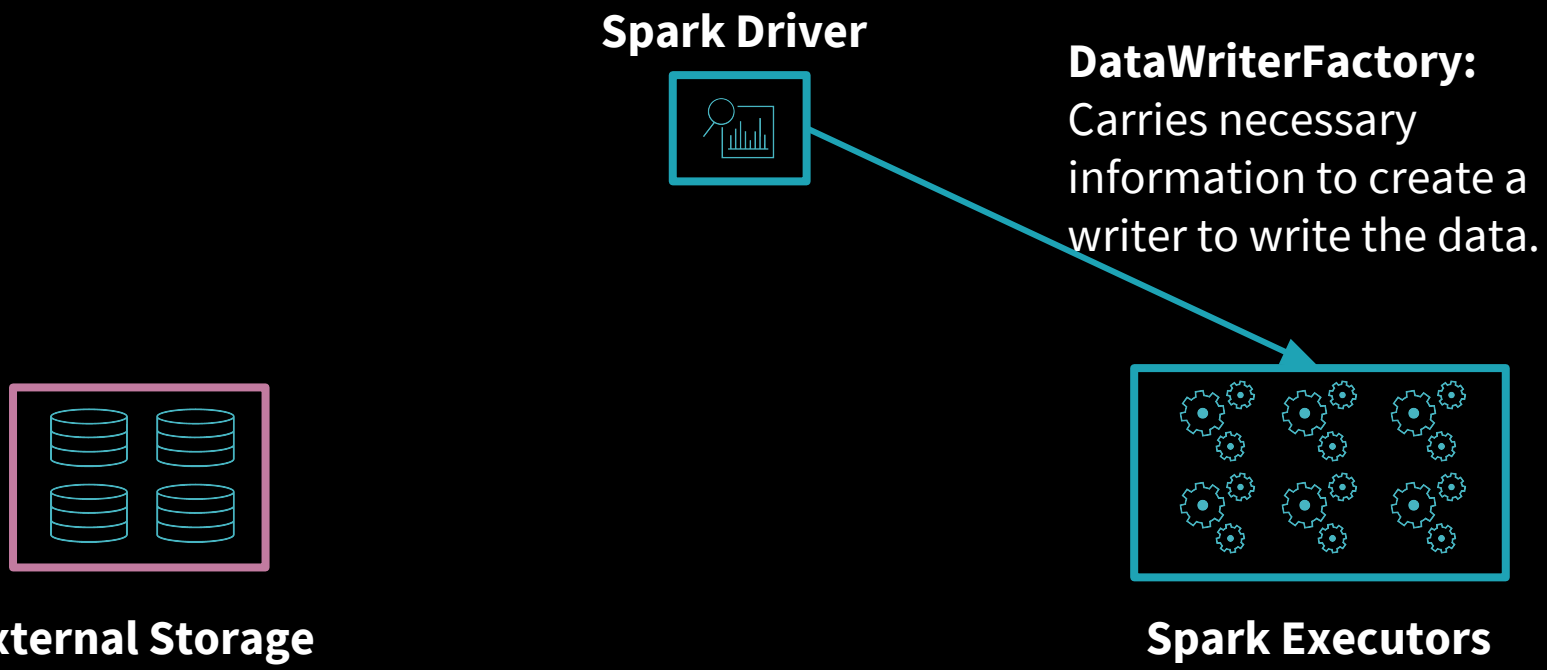


API Sketch (write)

```
public interface WriteSupport extends DataSourceV2 {
  Optional<DataSourceWriter> createWriter(
      String jobId, StructType schema, SaveMode mode, DataSourceOptions options);
}

public interface DataSourceWriter {
  DataWriterFactory<Row> createWriterFactory();

  void commit(WriterCommitMessage[] messages);

  void abort(WriterCommitMessage[] messages);
}

public interface DataWriterFactory<T> extends Serializable {
  DataWriter<T> createDataWriter(int partitionId, int attemptNumber);
}

public interface DataWriter<T> {
  void write(T record) throws IOException;

  WriterCommitMessage commit() throws IOException;

  void abort() throws IOException;
}
```

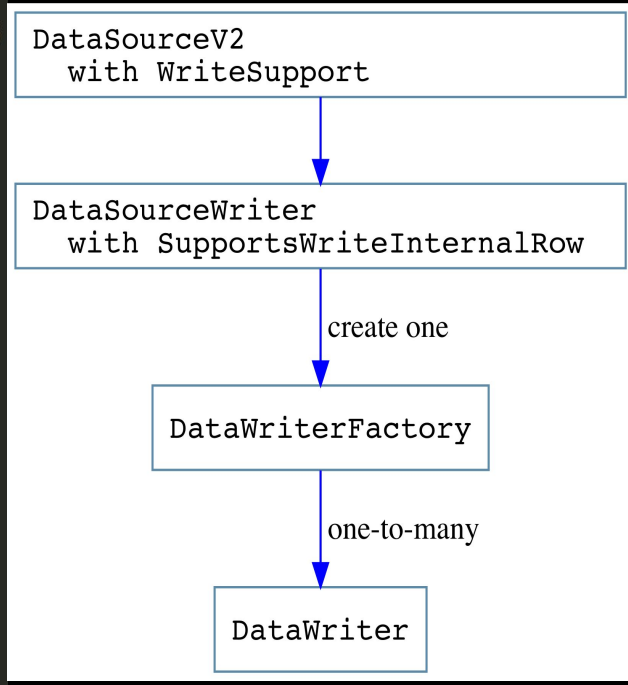


Write Process

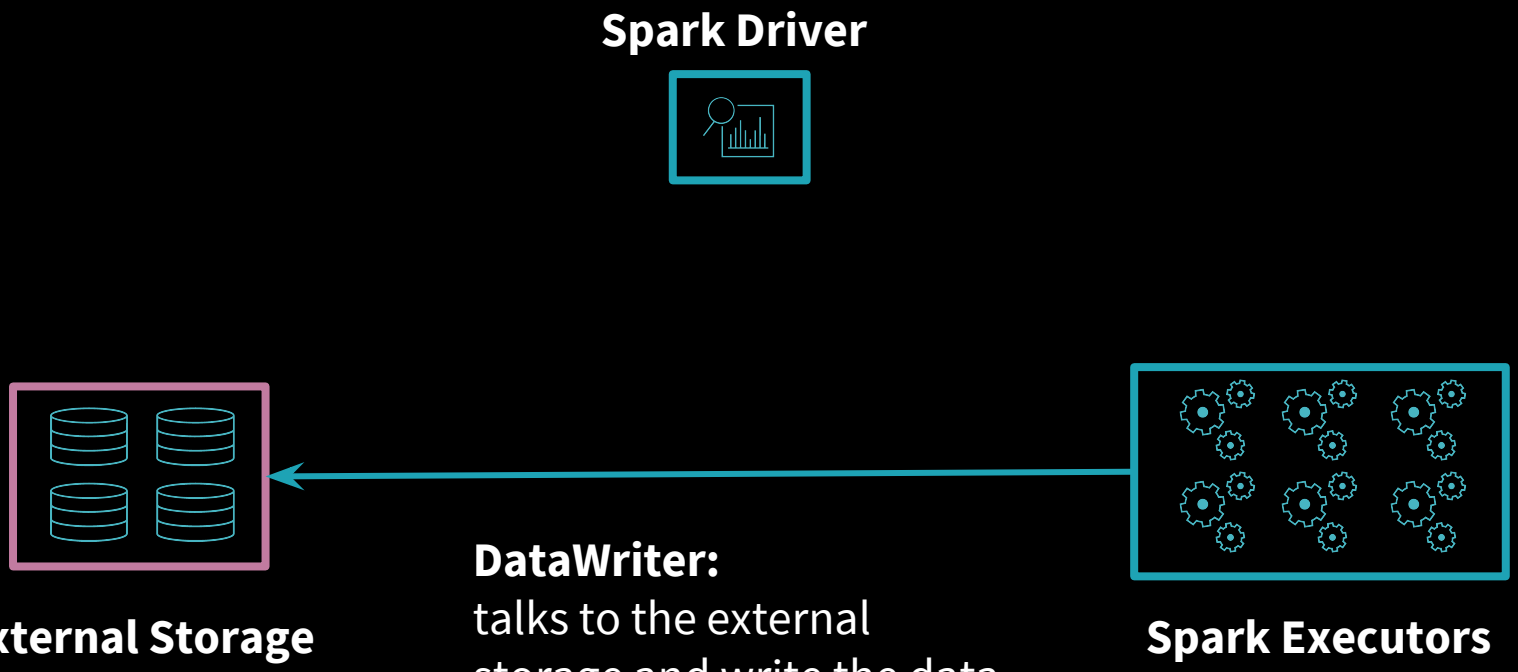


API Sketch (write)

```
public interface WriteSupport extends DataSourceV2 {
  Optional<DataSourceWriter> createWriter(
      String jobId, StructType schema, SaveMode mode, DataSourceOptions options);
}

public interface DataSourceWriter {
  DataWriterFactory<Row> createWriterFactory();

  void commit(WriterCommitMessage[] messages);

  void abort(WriterCommitMessage[] messages);
}

public interface DataWriterFactory<T> extends Serializable {
  DataWriter<T> createDataWriter(int partitionId, int attemptNumber);
}

public interface DataWriter<T> {
  void write(T record) throws IOException;

  WriterCommitMessage commit() throws IOException;

  void abort() throws IOException;
}
```

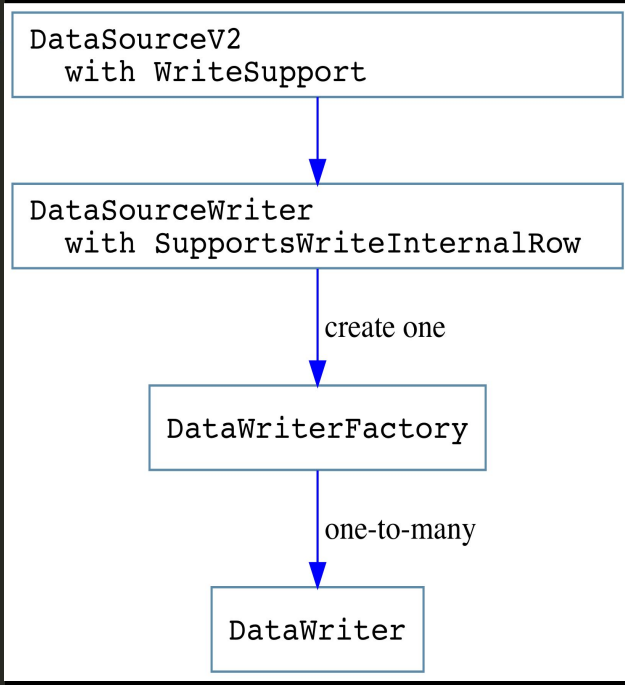


Write Process

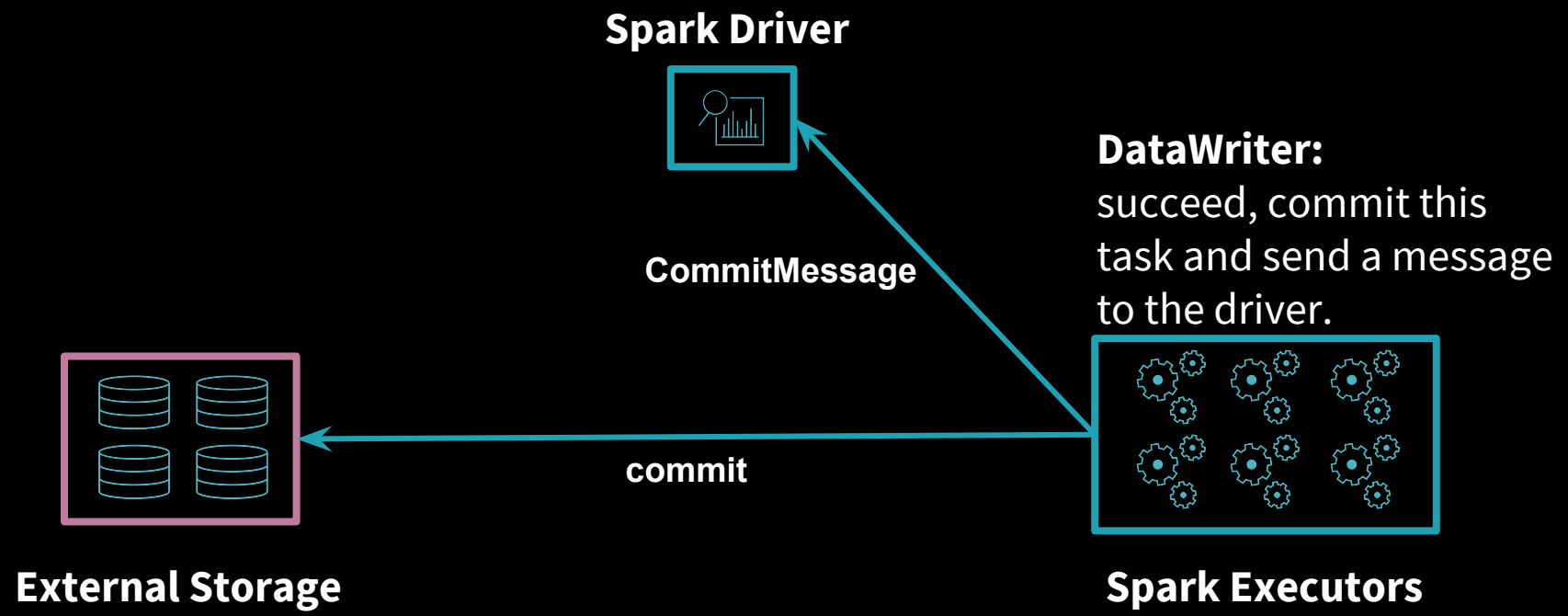


API Sketch (write)

```scala
public interface WriteSupport extends DataSourceV2 {
  Optional<DataSourceWriter> createWriter(
      String jobId, StructType schema, SaveMode mode, DataSourceOptions options);
}

public interface DataSourceWriter {
  DataWriterFactory<Row> createWriterFactory();

  void commit(WriterCommitMessage[] messages);

  void abort(WriterCommitMessage[] messages);
}

public interface DataWriterFactory<T> extends Serializable {
  DataWriter<T> createDataWriter(int partitionId, int attemptNumber);
}

public interface DataWriter<T> {
  void write(T record) throws IOException;

  WriterCommitMessage commit() throws IOException;

  void abort() throws IOException;
}
```

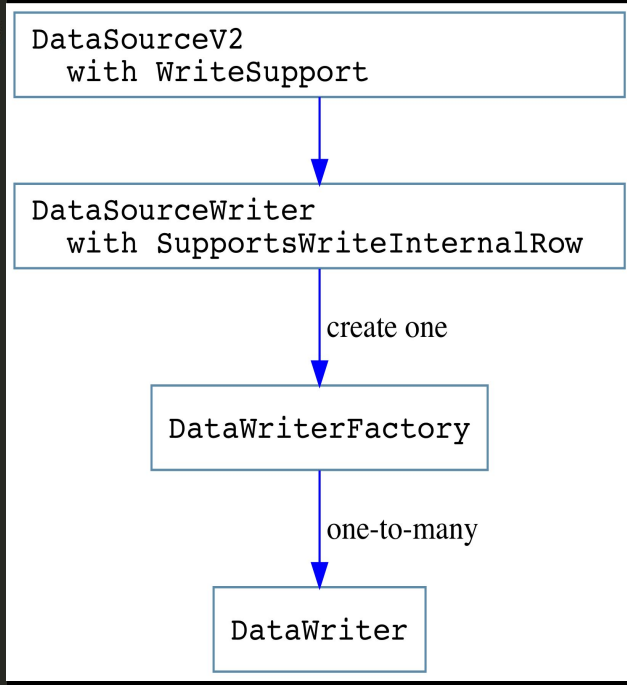


Write Process

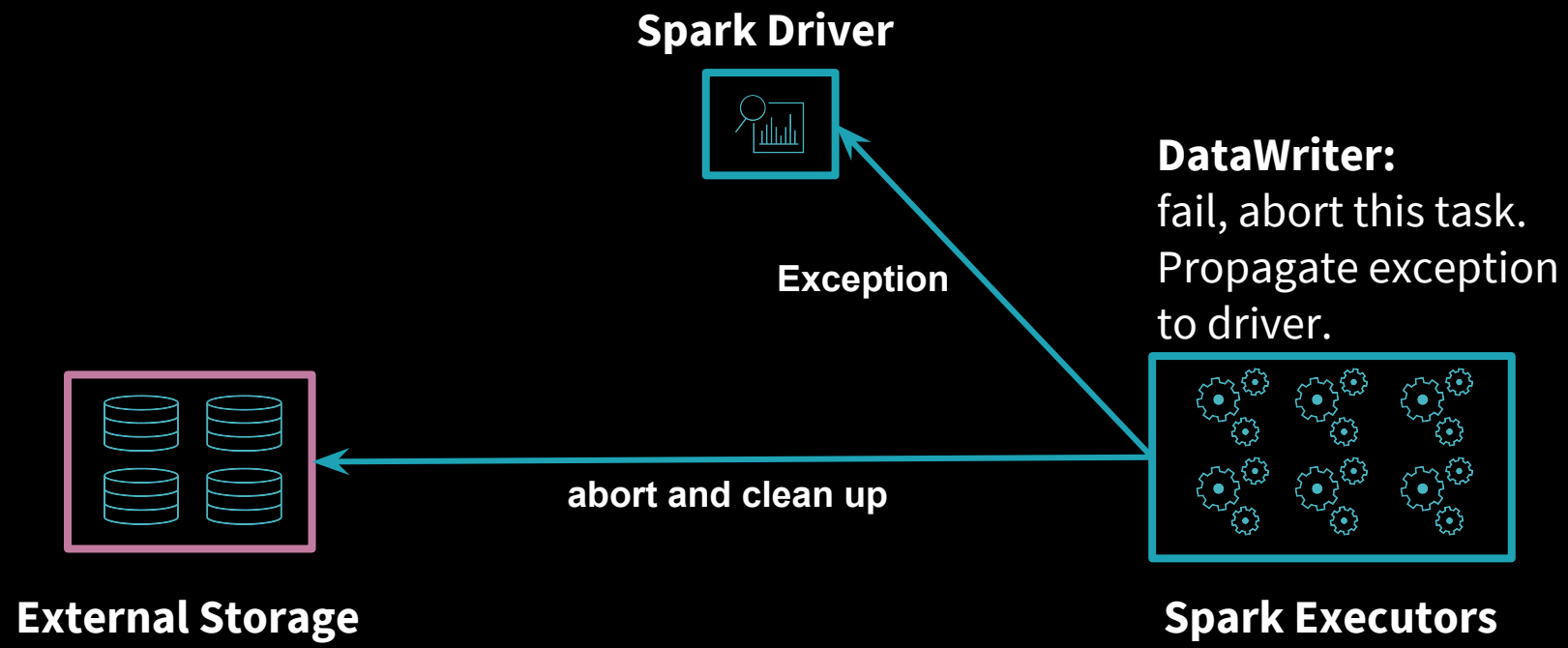


API Sketch (write)

```java
public interface WriteSupport extends DataSourceV2 {
  Optional<DataSourceWriter> createWriter(
      String jobId, StructType schema, SaveMode mode, DataSourceOptions options);
}

public interface DataSourceWriter {
  DataWriterFactory<Row> createWriterFactory();

  void commit(WriterCommitMessage[] messages);

  void abort(WriterCommitMessage[] messages);
}

public interface DataWriterFactory<T> extends Serializable {
  DataWriter<T> createDataWriter(int partitionId, int attemptNumber);
}

public interface DataWriter<T> {
  void write(T record) throws IOException;

  WriterCommitMessage commit() throws IOException;

  void abort() throws IOException;
}
```

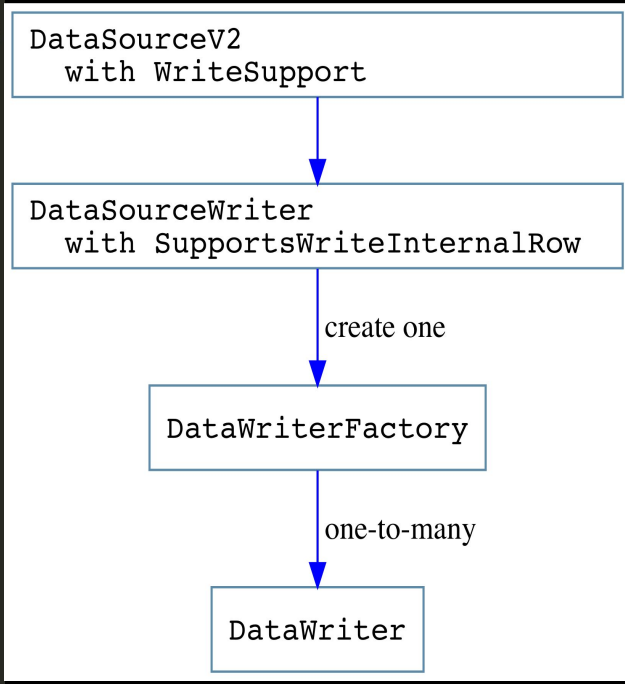


Write Process



API Sketch (write)

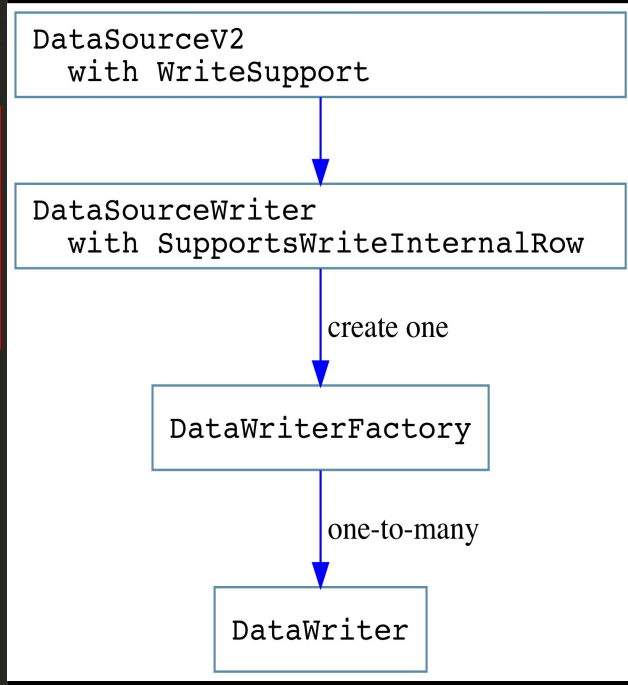
```scala
public interface WriteSupport extends DataSourceV2 {
  Optional<DataSourceWriter> createWriter(
      String jobId, StructType schema, SaveMode mode, DataSourceOptions options);
}

public interface DataSourceWriter {
  DataWriterFactory<Row> createWriterFactory();

  void commit(WriterCommitMessage[] messages);

  void abort(WriterCommitMessage[] messages);
}

public interface DataWriterFactory<T> extends Serializable {
  DataWriter<T> createDataWriter(int partitionId, int attemptNumber);
}

public interface DataWriter<T> {
  void write(T record) throws IOException;

  WriterCommitMessage commit() throws IOException;

  void abort() throws IOException;
}
```



Write Process

API Sketch (write)

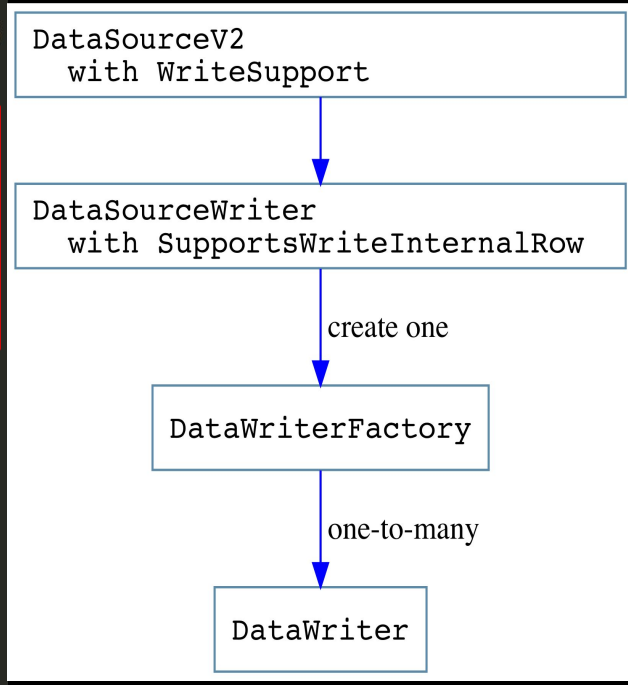
```java
public interface WriteSupport extends DataSourceV2 {
  Optional<DataSourceWriter> createWriter(
      String jobId, StructType schema, SaveMode mode, DataSourceOptions options);
}

public interface DataSourceWriter {
  DataWriterFactory<Row> createWriterFactory();

  void commit(WriterCommitMessage[] messages);

  void abort(WriterCommitMessage[] messages);
}

public interface DataWriterFactory<T> extends Serializable {
  DataWriter<T> createDataWriter(int partitionId, int attemptNumber);
}

public interface DataWriter<T> {
  void write(T record) throws IOException;

  WriterCommitMessage commit() throws IOException;

  void abort() throws IOException;
}
```



Write Process

API Sketch (write)

```
public interface WriteSupport extends DataSourceV2 {
  Optional<DataSourceWriter> createWriter(
      String jobId, StructType schema, SaveMode mode, DataSourceOptions options);
}

public interface DataSourceWriter {
  DataWriterFactory<Row> createWriterFactory();

  void commit(WriterCommitMessage[] messages);

  void abort(WriterCommitMessage[] messages);
}

public interface DataWriterFactory<T> extends Serializable {
  DataWriter<T> createDataWriter(int partitionId, int attemptNumber);
}

public interface DataWriter<T> {
  void write(T record) throws IOException;

  WriterCommitMessage commit() throws IOException;

  void abort() throws IOException;
}
```



Streaming Data Source API V2

Structured Streaming Deep Dive:
https://tinyurl.com/y9bze7ae

Continuous Processing in Structured Streaming:
https://tinyurl.com/ydbdhxbz

Ongoing Improvements

- Catalog Supports: standardize the DDL logical plans, proxy DDL commands to data source, integrate data source catalog. (SPARK-24252)
- More operator pushdown: limit pushdown, aggregate pushdown, join pushdown, etc. (SPARK-22388, SPARK-22390, SPARK-24130, ...)

Thank you

Wenchen Fan (wenchen@databricks.com)

databricks

Apache Spark Data Source V2 : Example

Gengliang Wang

Spark Summit 2018, SF

About me

- Gengliang Wang (Github: gengliangwang)

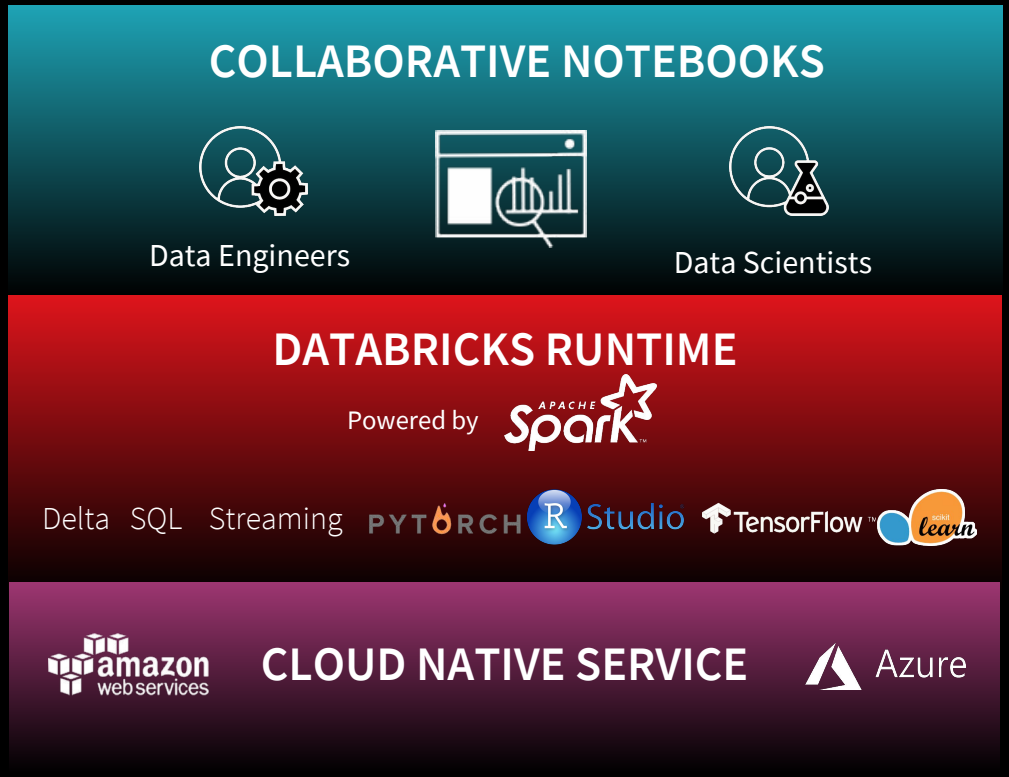
- Software Engineer at databricks

Databricks' Unified Analytics Platform

Unifies Data Engineers and Data Scientists

Unifies Data and AI Technologies

Eliminates infrastructure complexity



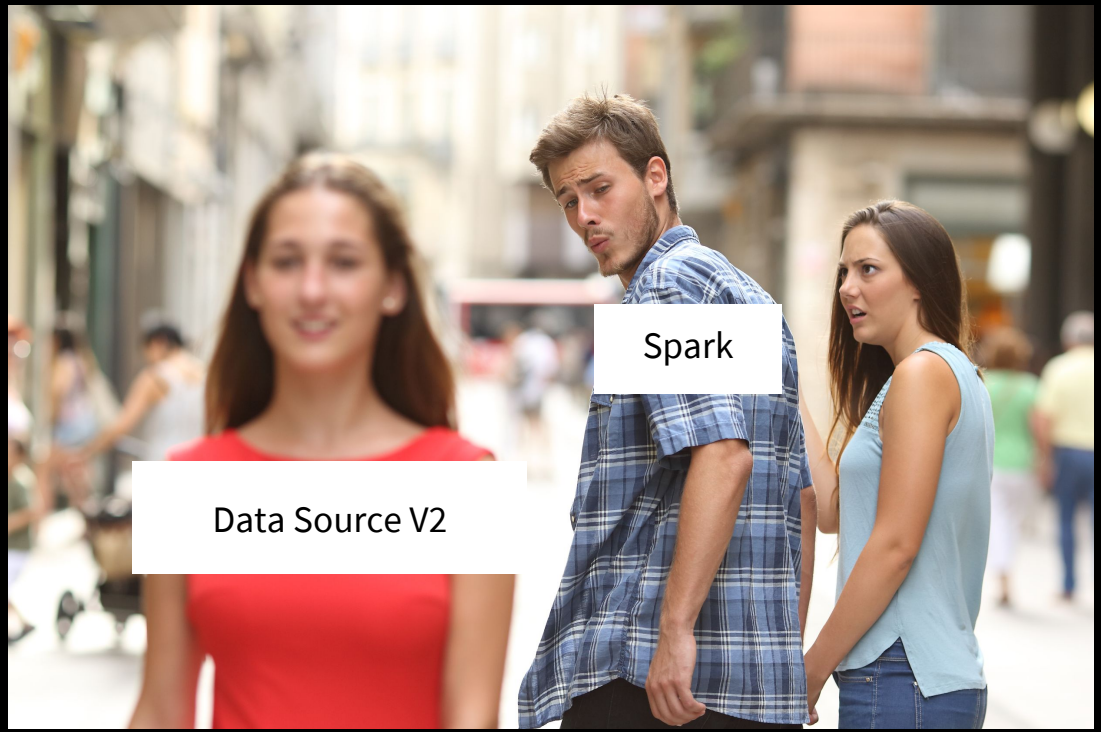
databricks

About this talk

- Part II of Apache Data Source V2 session.
  - See Wenchen's talk for background and design details.

- How to implement Parquet data source with the V2 API

We are migrating...



Read Parquet files

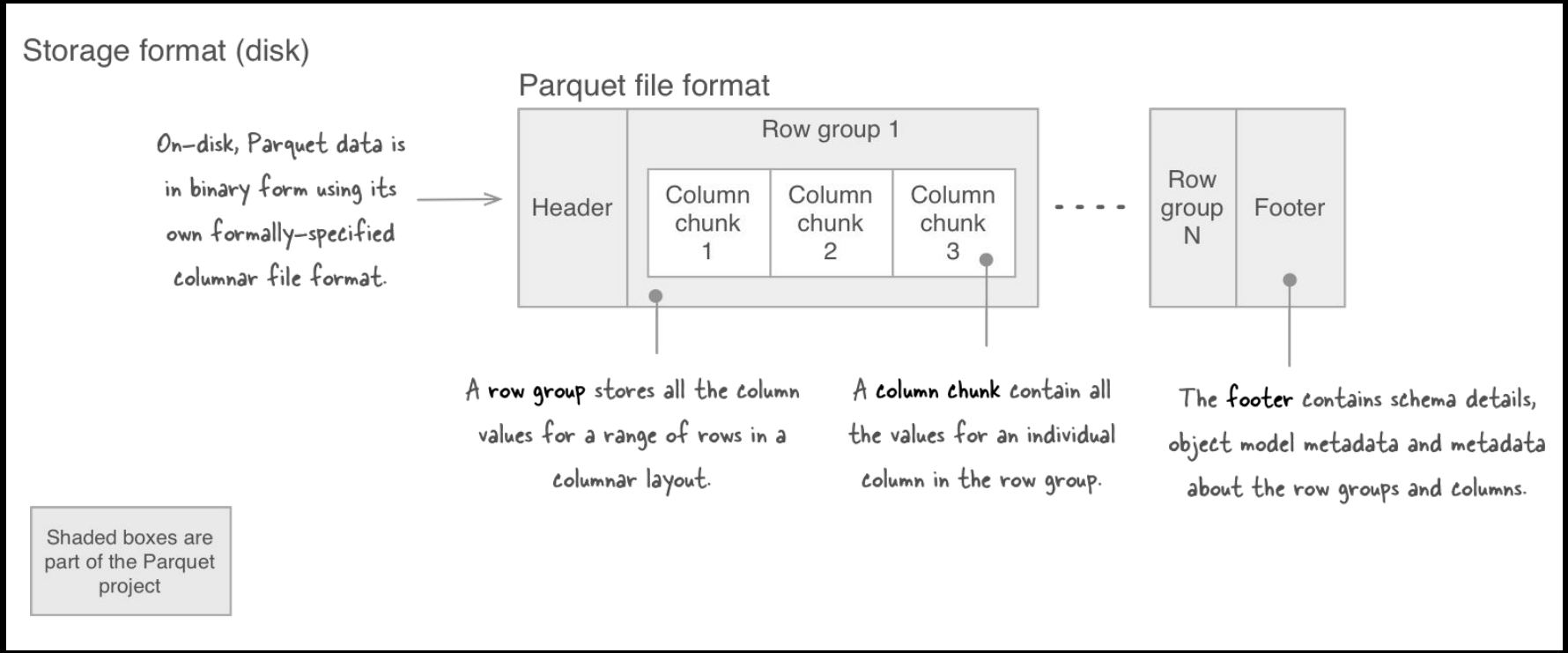
Query example

```
trainingData = spark.read.parquet("/data/events")
                    .where("city = 'San Francisco' and year = 2018")
                    .select("timestamp").collect()
```

Goal

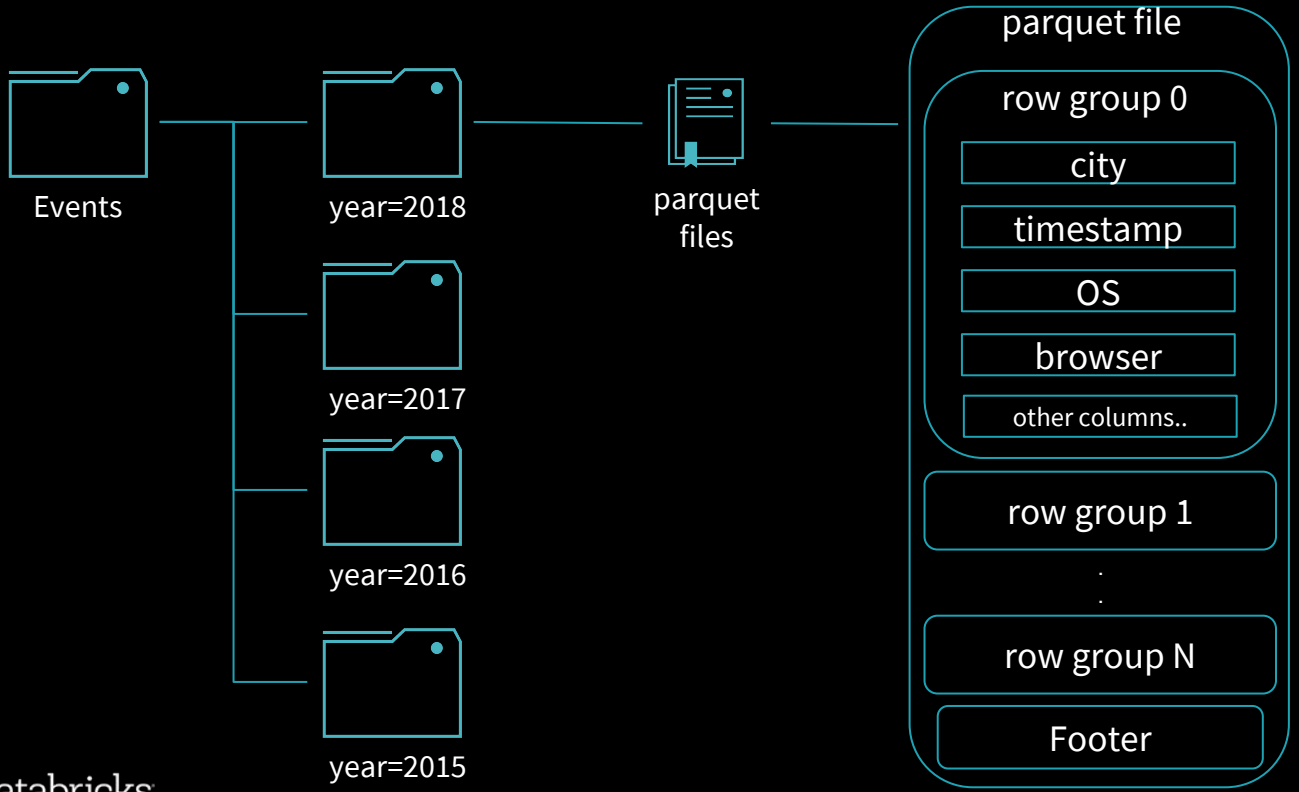
- Understand data and skip unneeded data
- Split file into partitions for parallel read

Parquet 101



ref: Understanding how Parquet integrates with Avro, Thrift and Protocol Buffers

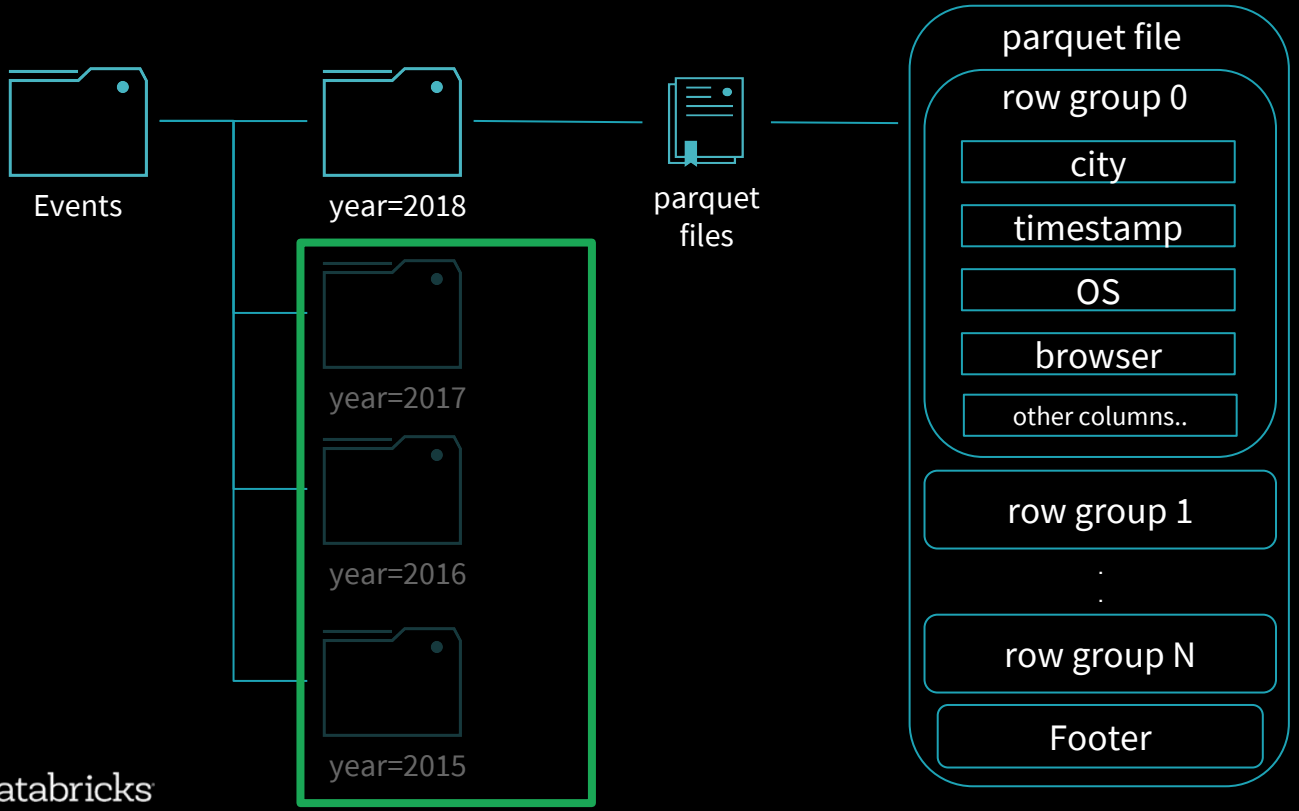
Data layout


Events
year=2018
year=2017
year=2016
year=2015
parquet files
parquet file
row group 0
city
timestamp
OS
browser
other columns..
row group 1
row group N
Footer

pseudo-code

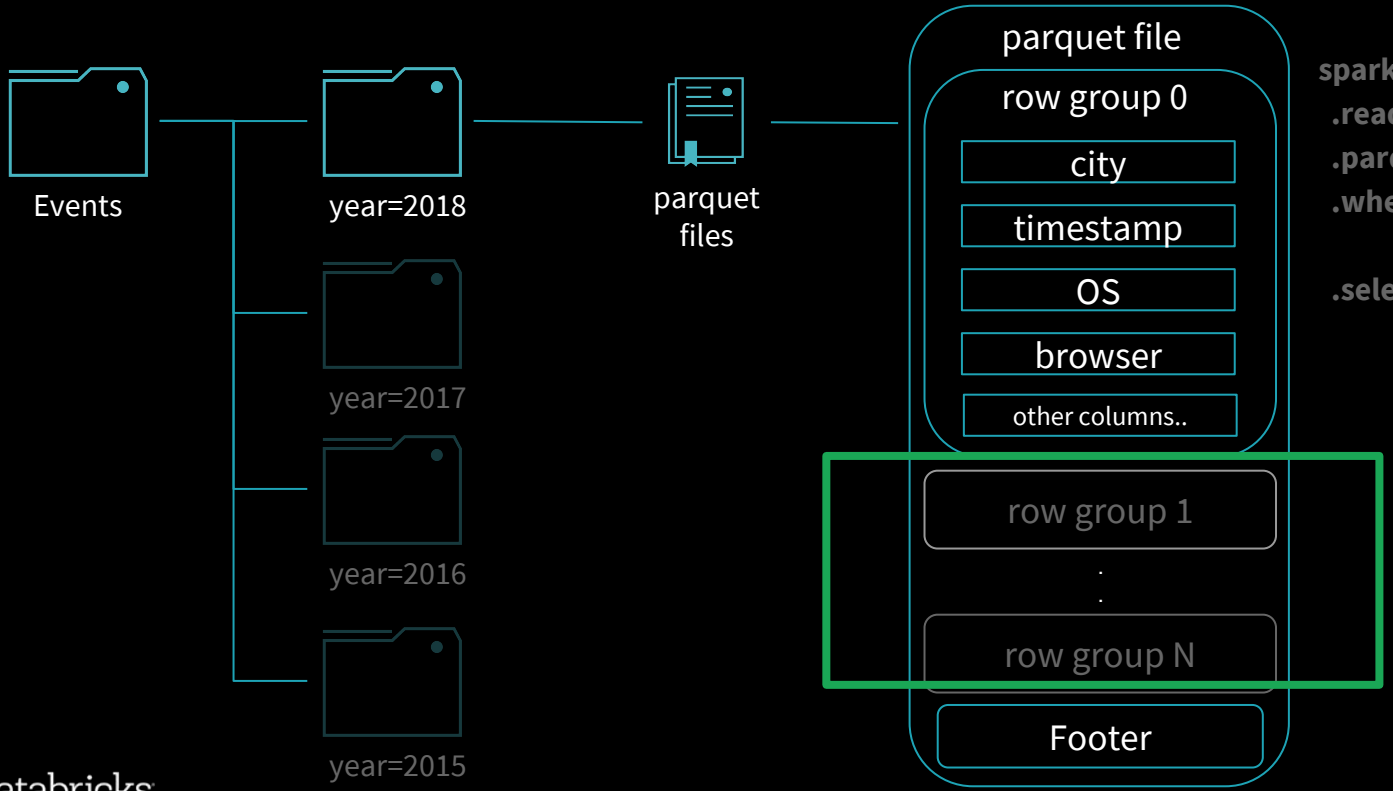
```
class ParquetDataSource extends DataSourceReader {
    override def readSchema(): StructType = {
        fileIndex
          .listFiles()
          .map(readSchemaInFooter)
          .reduce(mergeSchema)
    }
}
```

Prune partition columns




```
spark
  .read
  .parquet("/data/events")
  .where("city = 'San Francisco' and
                year = 2018")
  .select("timestamp").collect()
```

Skip row groups

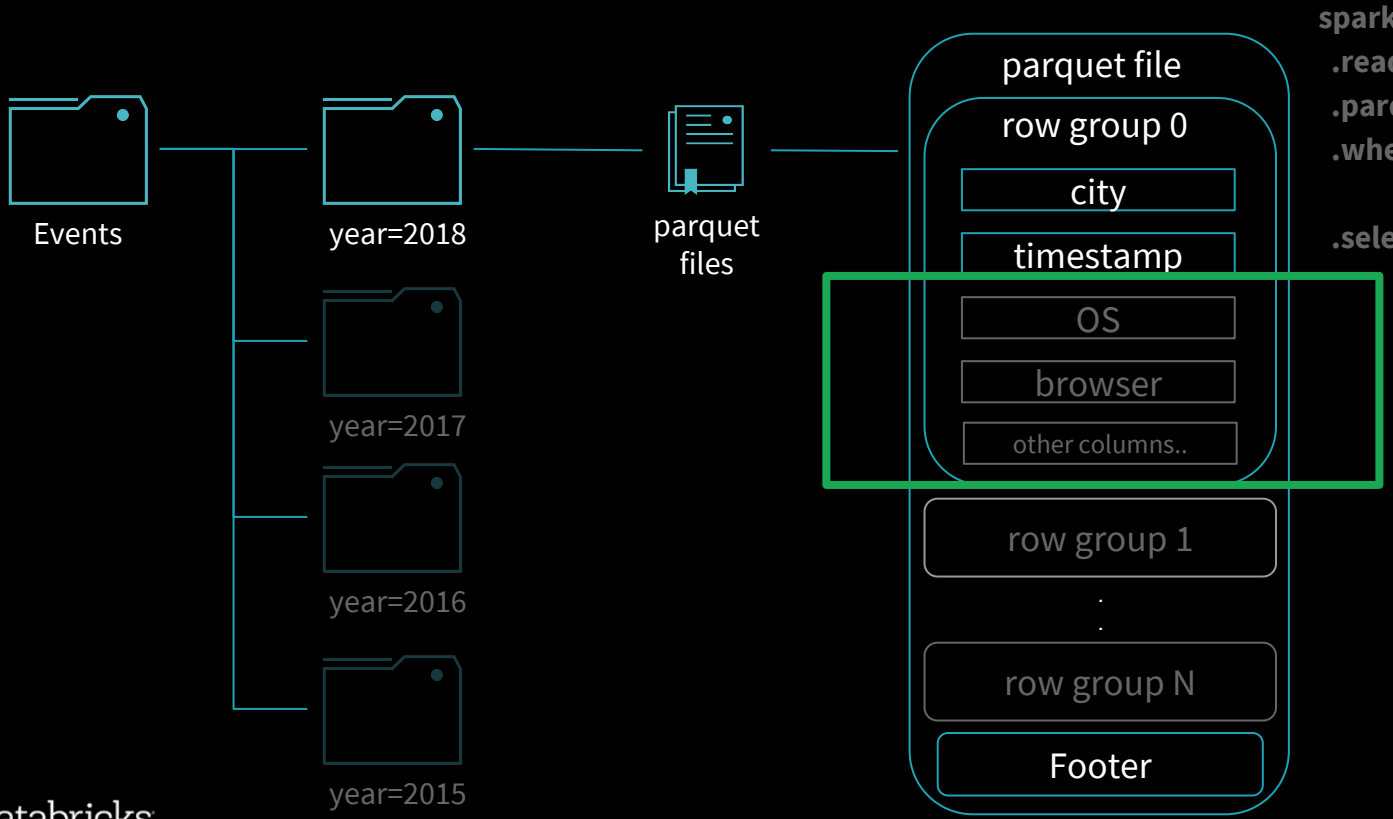



```
spark
  .read
  .parquet("/data/events")
  .where("city = 'San Francisco' and
          year = 2018")
  .select("timestamp").collect()
```

pseudo-code

```
class ParquetDataSource extends DataSourceReader with SupportsPushDownFilters {
    override def pushFilters(filters: Array[Filter]): Array[Filter] = {
        (partitionFilters, dataFilters) =
            filters.span(_.outputSet.subsetOf(partitionColumns))
        dataFilters
    }
}
// For the selected row groups, we still need to evaluate data filters in Spark
// To be continued in #planInputPartitions
```

Prune columns




```
spark
  .read
  .parquet("/data/events")
  .where("city = 'San Francisco' and
          year = 2018")
  .select("timestamp").collect()
```

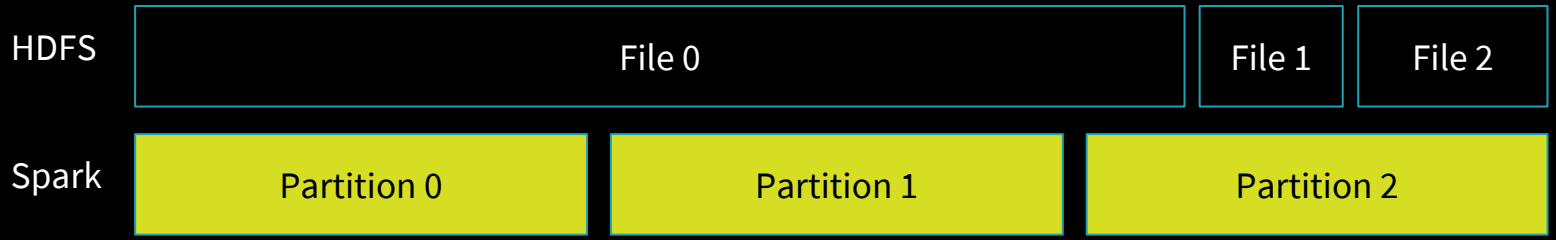
pseudo-code

```
class ParquetDataSource extends DataSourceReader with SupportsPushDownFilters
        with SupportsPushDownRequiredColumns {
        var requiredSchema: StructType = _
        override def pruneColumns(requiredSchema: StructType): Unit = {
                this.requiredSchema = requiredSchema
        }
}

// To be continued in #planInputPartitions
```

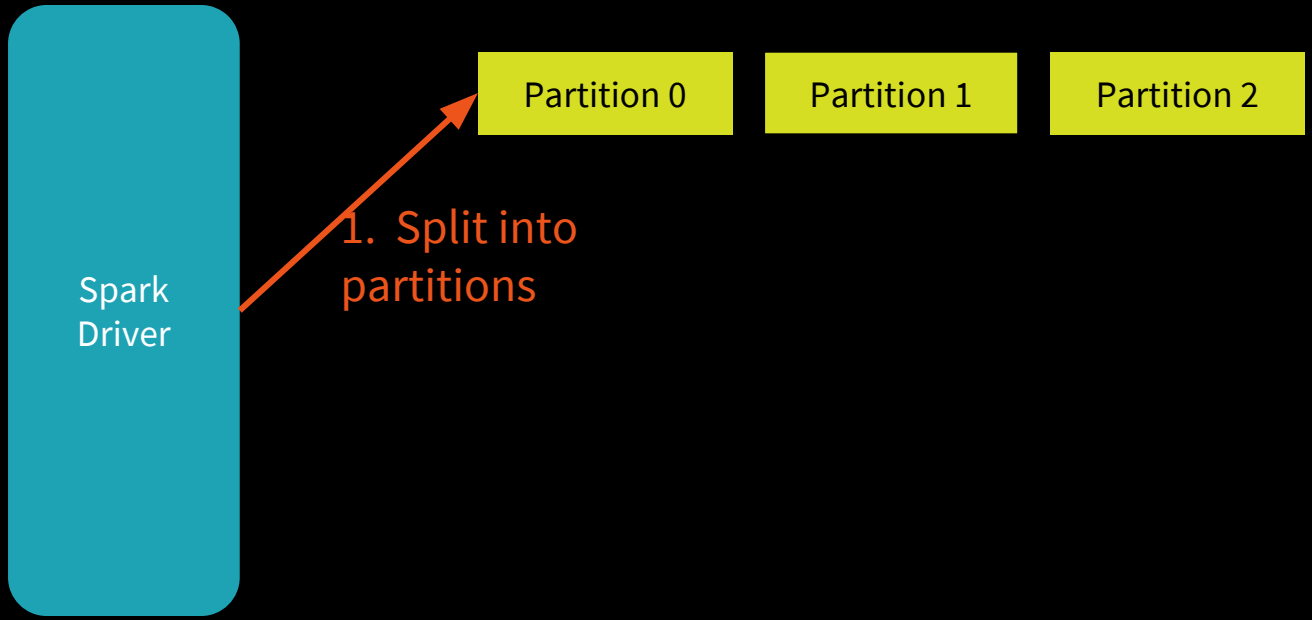
Goal

- Understand data and skip unneeded data
- Split files into partitions for parallel read

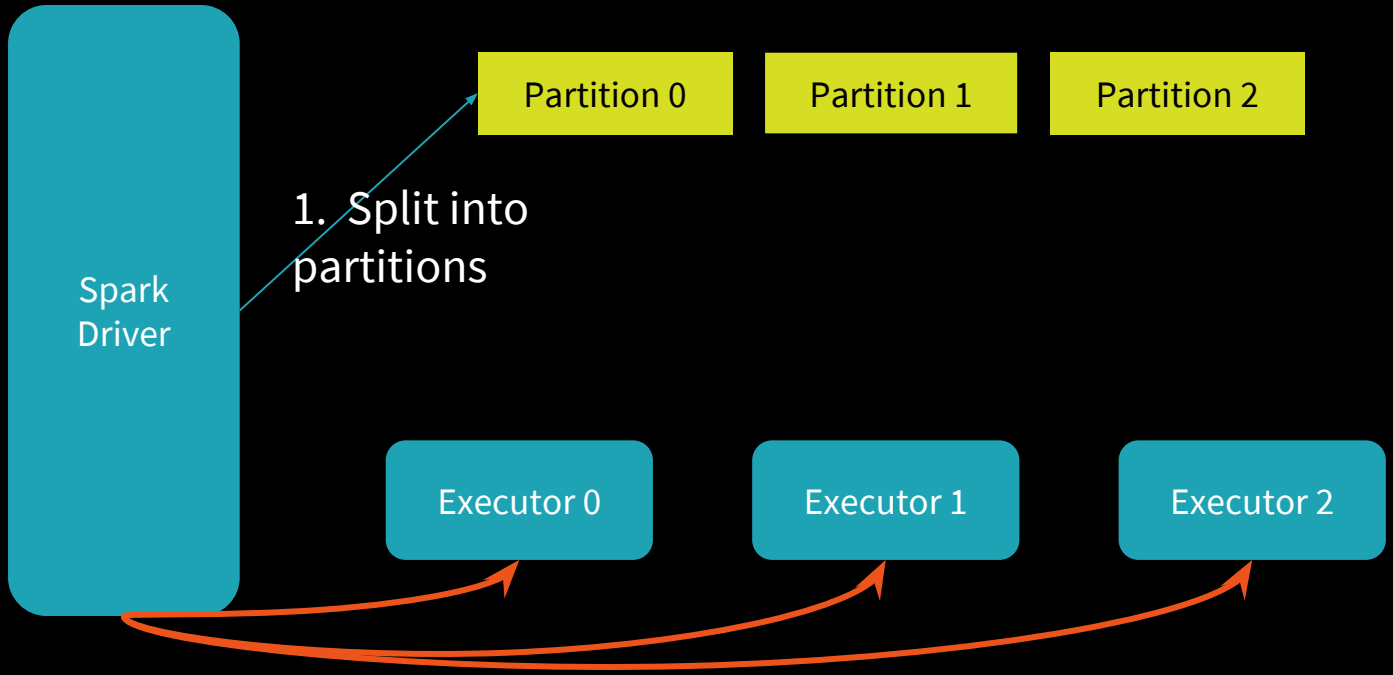
Partitions of same size



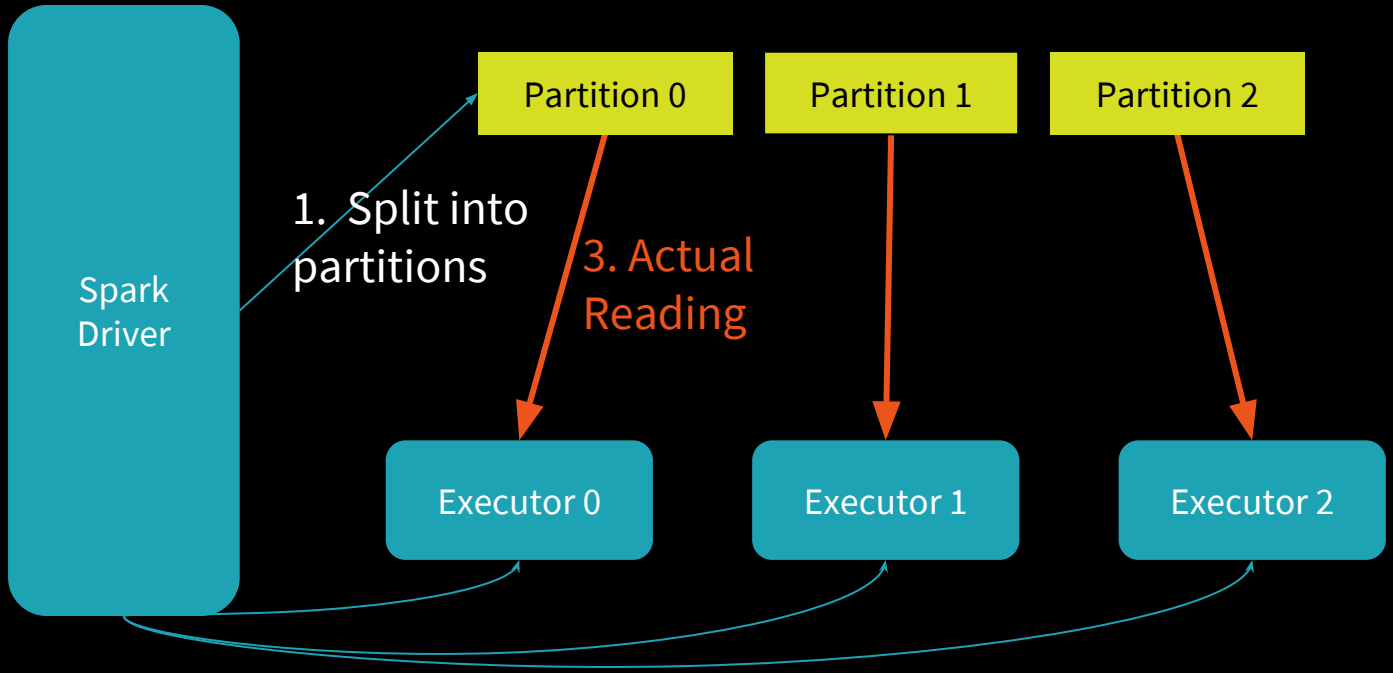
Driver: plan input partitions



Driver: plan input partitions



Executor: Read distributedly



pseudo-code

```
class ParquetDataSource extends DataSourceReader with SupportsPushDownFilters
      with SupportsPushDownRequiredColumns  {
      override def planInputPartitions(): List[InputPartition[Row]] = {
            val filePartitions = makeFilePartitions(fileIndex.listFiles(partitionFilters))
            filePartitions.map { filePartition =>
                  // Row group skipping
                  ParquetInputFormat.setFilterPredicate(hadoopConf, dataFilters)
                  // Read requested columns from parquet file to Spark rows
                  ParquetReader(filePartition, requiredSchema)
       }
}
```

Summary

- Basic
  - determine schema
  - plan input partitions
- Mixins for optimization
  - push down filters
  - push down required columns
  - scan columnar data
  - ...

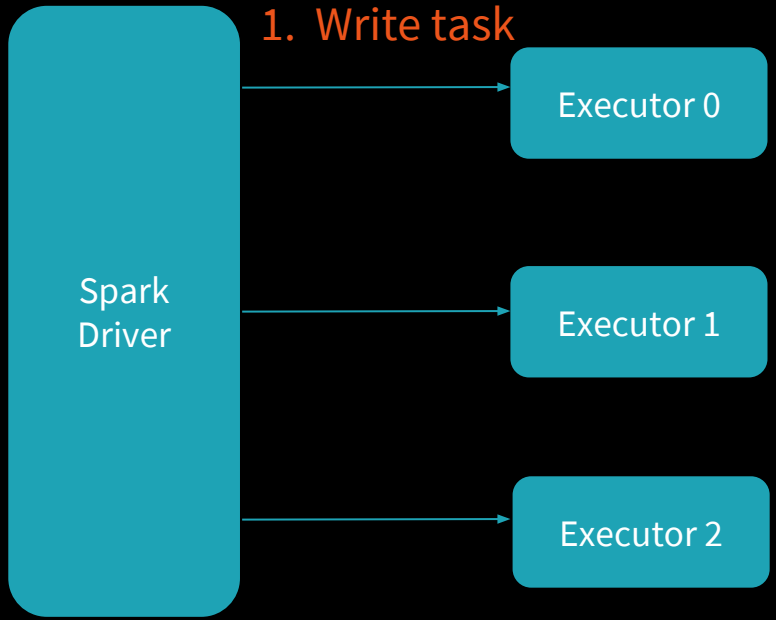
Parquet Writer on HDFS

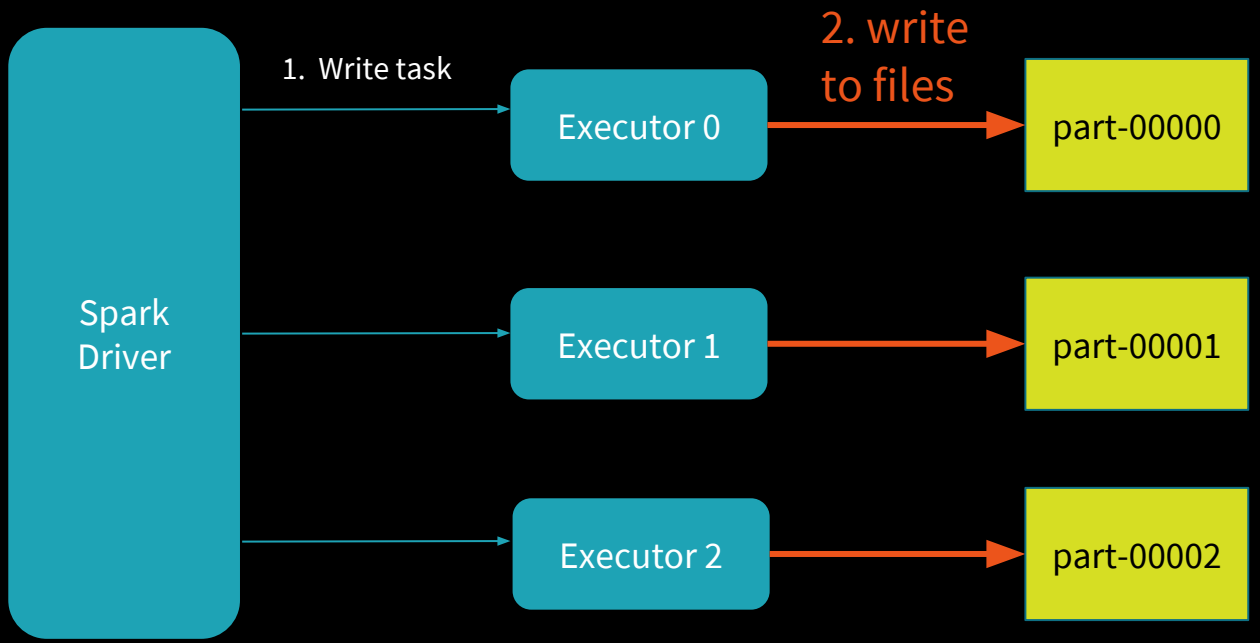
Query example

```
data = spark.read.parquet("/data/events")
                .where("city = 'San Francisco' and year = 2018")
                .select("timestamp")
data.write.parquet("/data/results")
```

Goal

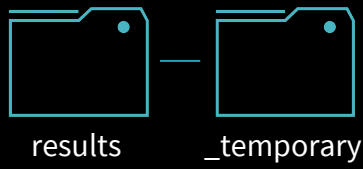
- Parallel
- Transactional


1. Write task
Spark Driver
Executor 0
Executor 1
Executor 2

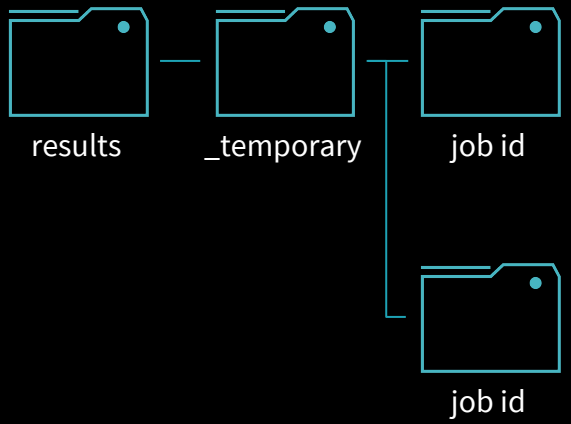


Each task writes to different temporary paths

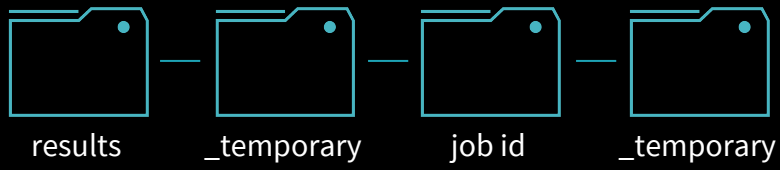
Everything should be temporary



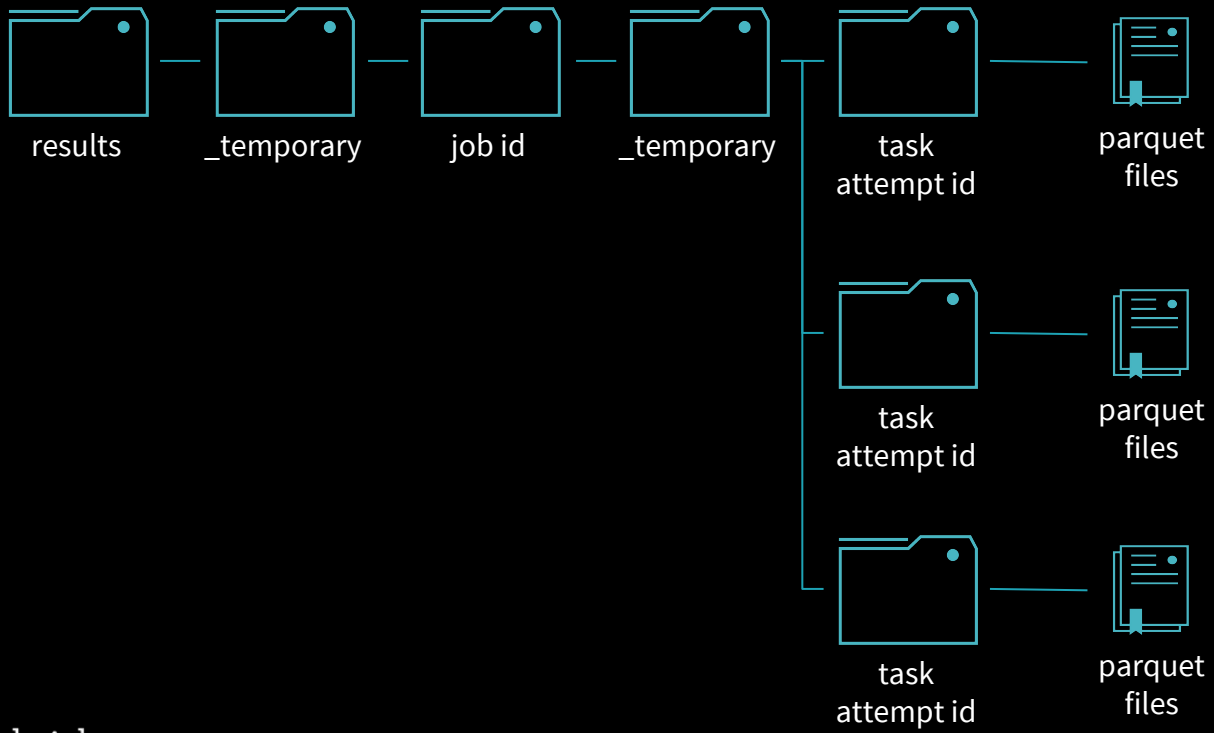
Files should be isolated between jobs



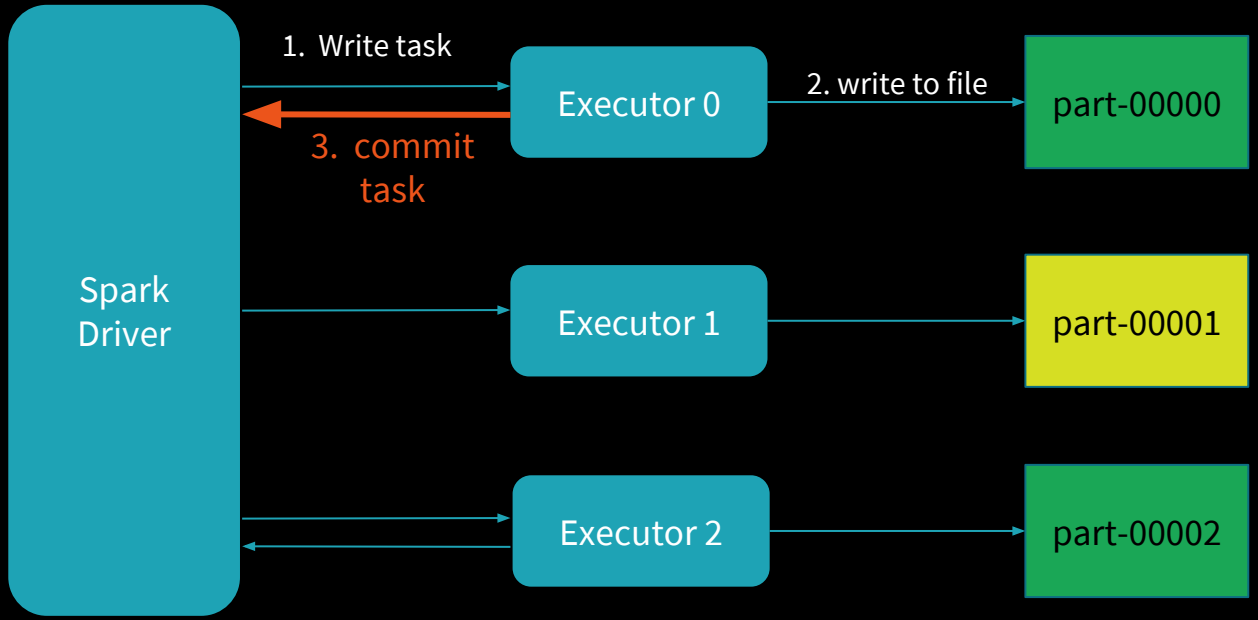
Task output is also temporary



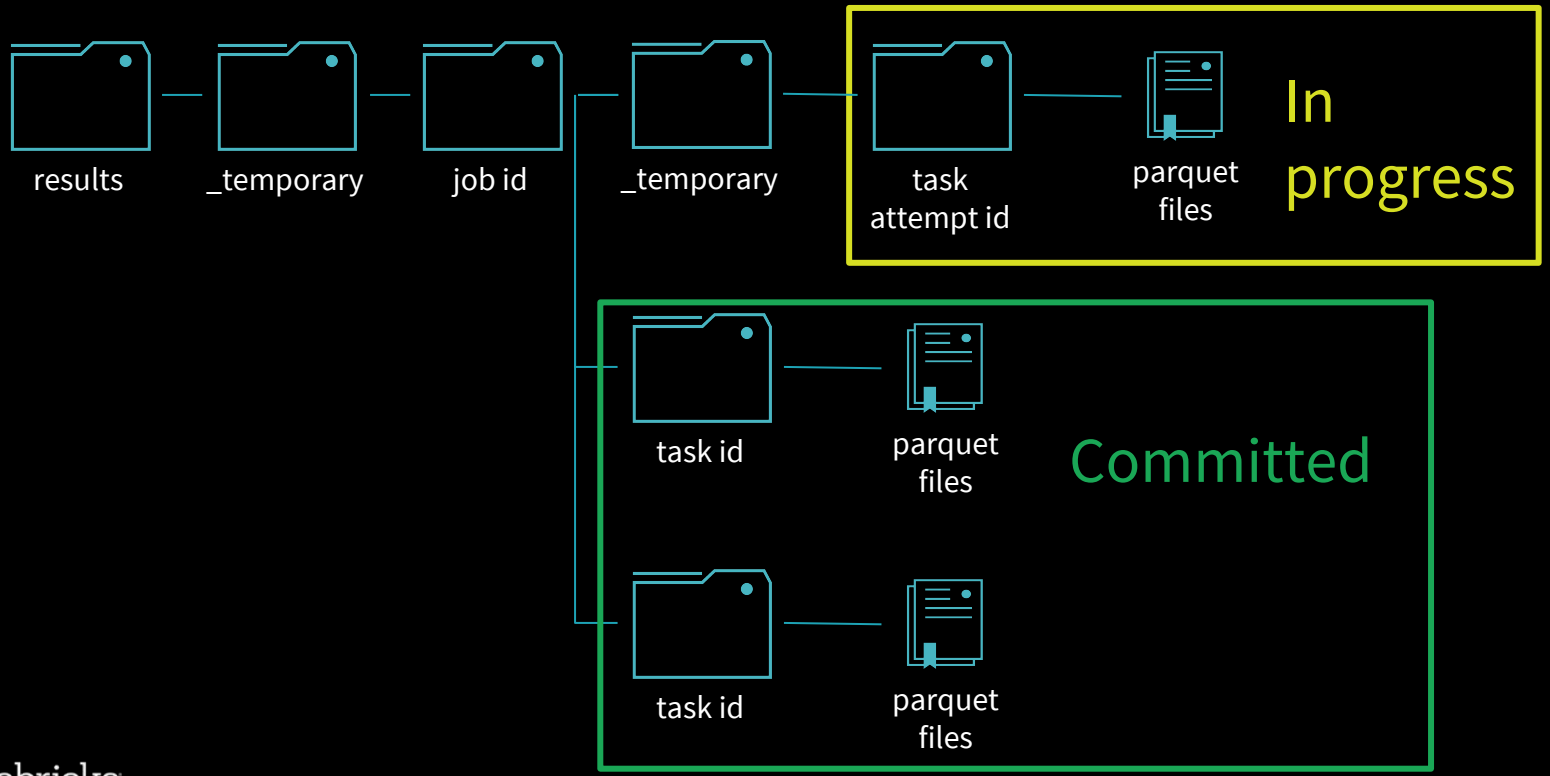
Files should be isolated between tasks



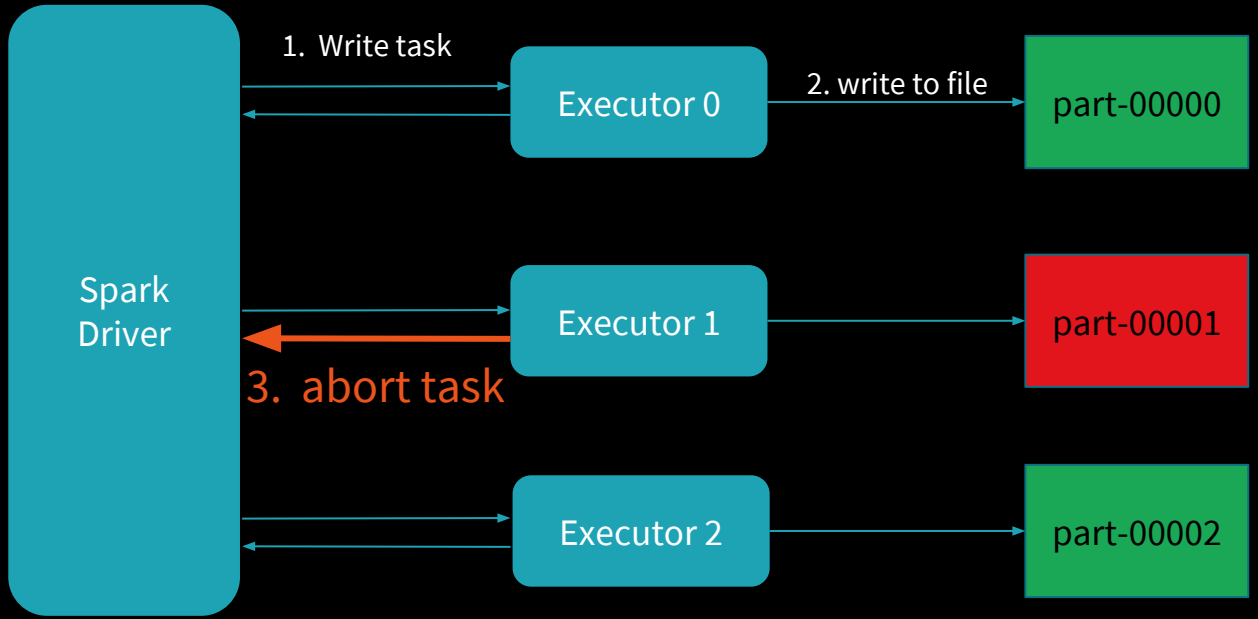
Commit task


Spark Driver
1. Write task
3. commit task
Executor 0
Executor 1
Executor 2
2. write to file
part-00000
part-00001
part-00002

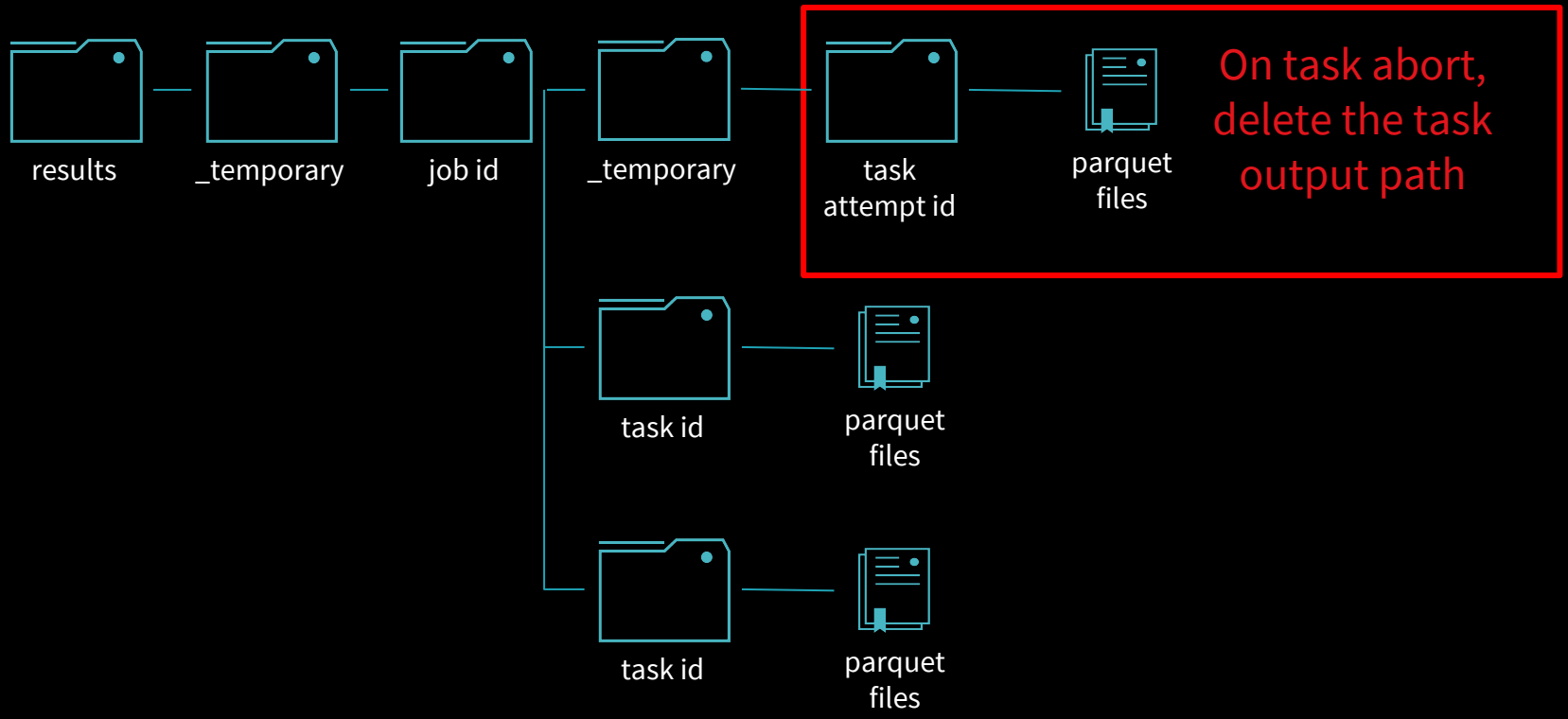
File layout


results
_temporary
job id
_temporary
task attempt id
parquet files
In progress
task id
parquet files
Committed
task id
parquet files

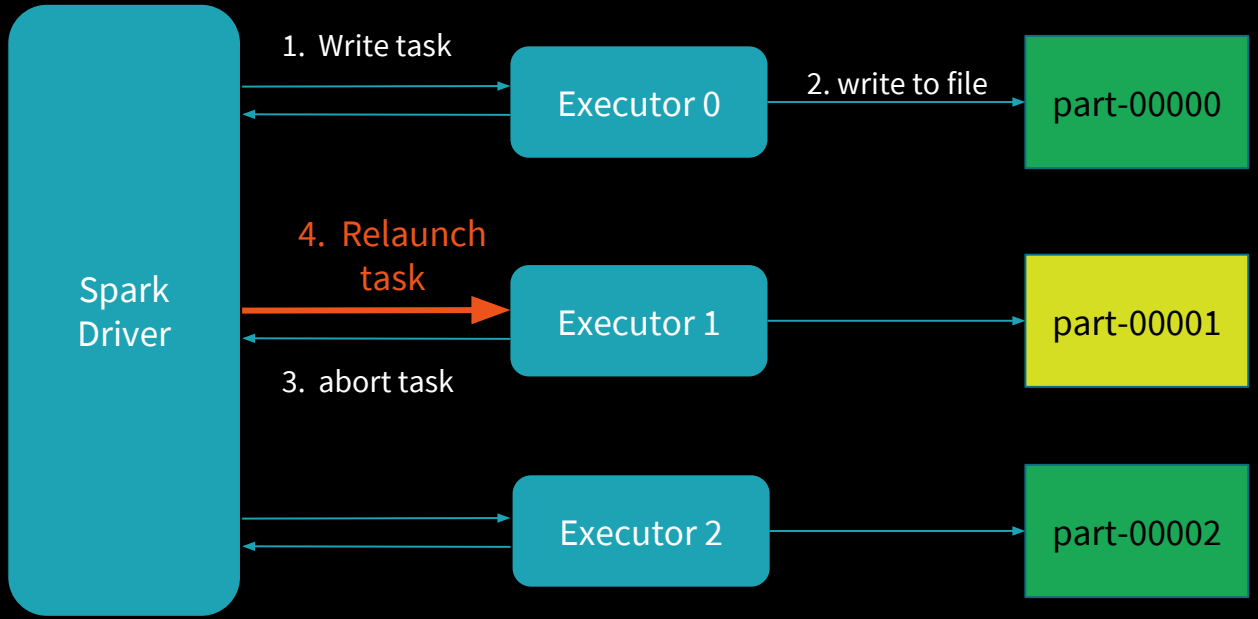
If task aborts..



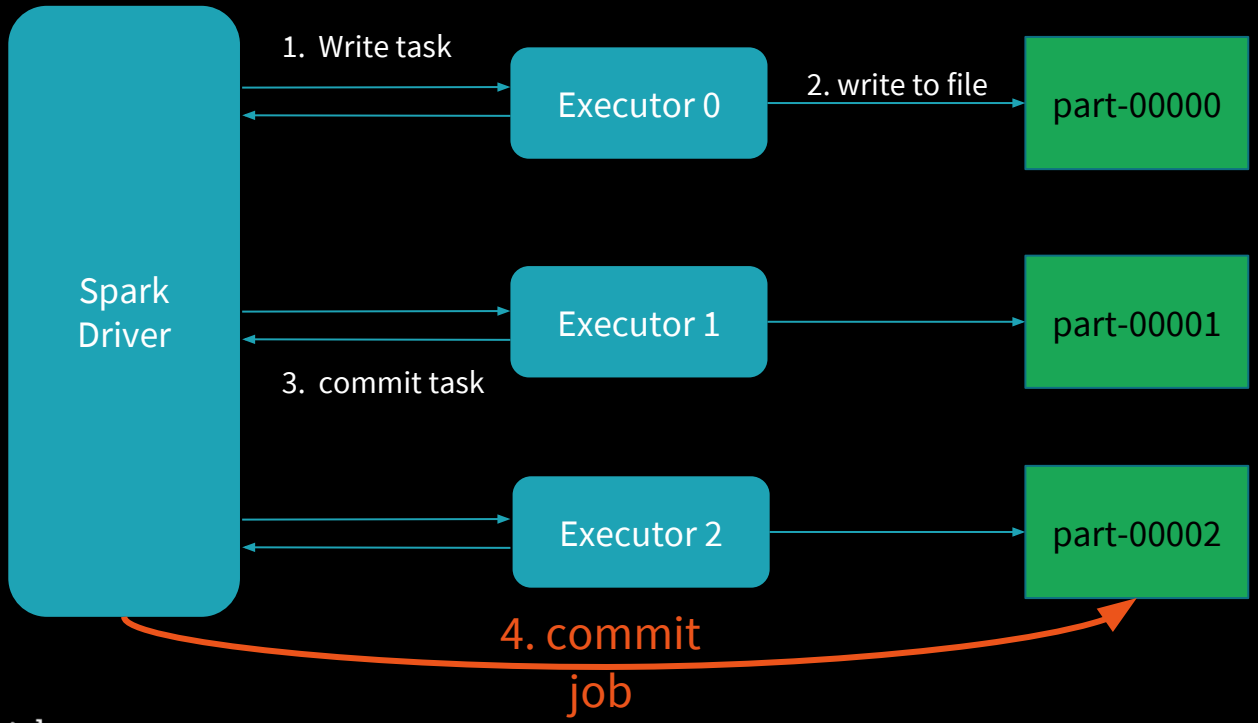
File layout


results
_temporary
job id
_temporary
task attempt id
parquet files
On task abort, delete the task output path
task id
parquet files
task id
parquet files

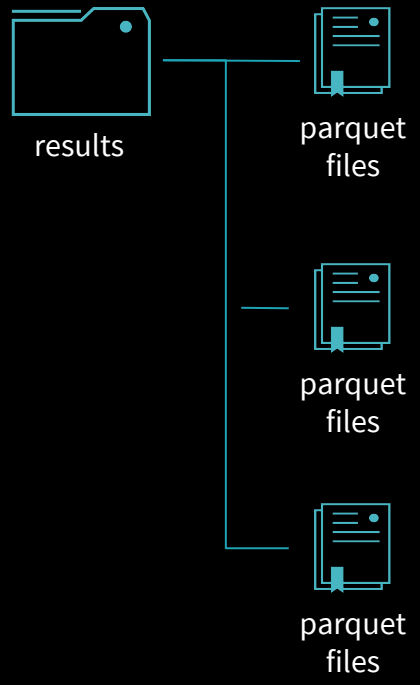
Relaunch task



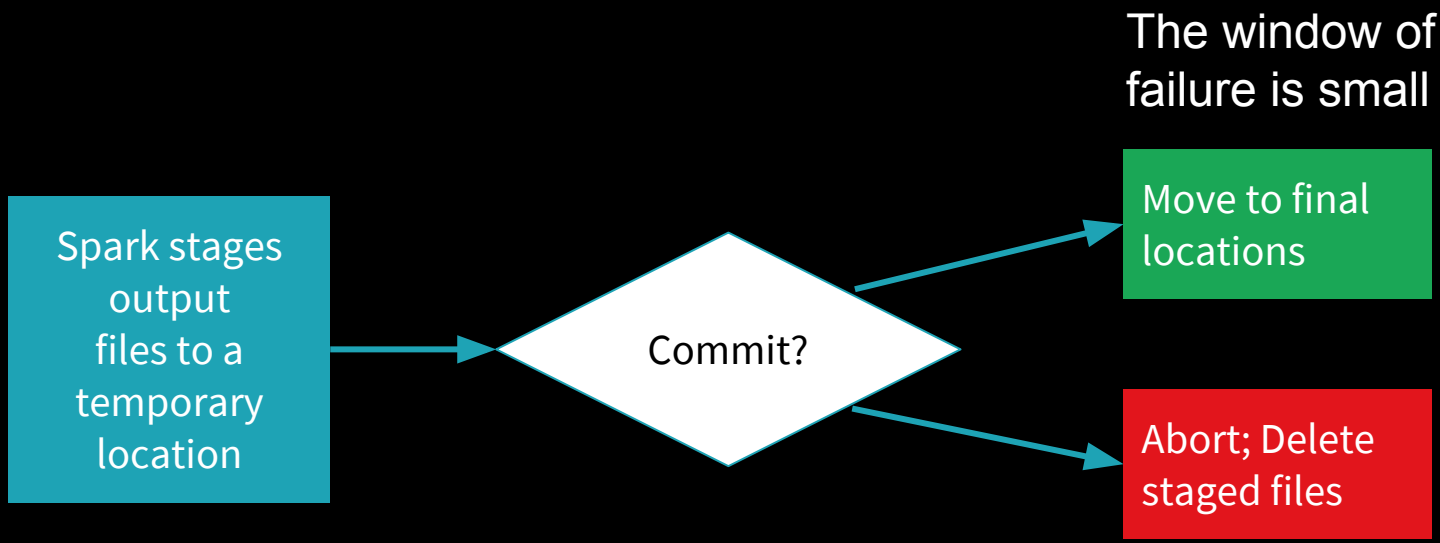
Distributed and Transactional Write



File layout



Almost transactional



See Eric Liang's talk in Spark summit 2017

pseudo-code

```
class ParquetDataSource extends DataSourceWriter with SupportsWriteInternalRow {
    override def createInternalRowWriterFactory(): DataWriterFactory[InternalRow] = {
        val parquetOutputFactory = ParquetOutputFactory(dataSchema, partitionSchema)
        ParquetWriterFactory(this.outputPath, parquetOutputFactory)
    }

    override def commit(messages: Array[WriterCommitMessage]): Unit = {
      committedTaskPaths.foreach { taskPath =>
            mergePath(taskPath, this.outputPath)
      }
    }

    override def abort(messages: Array[WriterCommitMessage]): Unit = {
      fs.delete(pendingJobAttemptsPath)
    }
}
```

```
class ParquetWriterFactory(
      path: Path,
      outputFactory: ParquetOutputFactory)
      extends DataWriterFactory[InternalRow] {
        override def createDataWriter(
              partitionId: Int,
              attemptNumber: Int,
              epochId: Long): DataWriter[InternalRow] = {
           val writer = outputFactory.newInstance()
            ParquetWriter(writer, partitionId, attemptNumber)
         }
   }
```

```
class ParquetWriter(writer: ParquetOutputWriter, partitionId: Int, attemptNumber: Int)
extends DataWriter[InternalRow] {
    val pendingPath = new pendingTaskAttemptPath(partitionId, attemptNumber)
    override def write(record: InternalRow): Unit = {
      writer.write(pendingPath)
    }

    override def commit(): WriterCommitMessage = {
      mergePath(pendingPath, pendingJobAttemptsPath)
    }

    override def abort(): Unit = {
      fs.delete(pendingPath)
    }
}
```

Thank you

Gengliang Wang (gengliang.wang@databricks.com)

databricks