

ABRiS: Avro Bridge for Apache Spark

Felipe Melo & Georgi Chochov
ABSA Group Limited

#SAISDev5 #ABRiS #ABSA

About Us



- ABSA is a Pan-African financial services provider
 - With Apache Spark at the core of its data engineering
- We try to fill gaps in the Hadoop eco-system
- Contributions to Apache Spark
- Spark-related open-source projects (github.com/AbsaOSS)
 - ABRiS – Avro SerDe for structured APIs (#SAISDev5)
 - Cobrix – A cobol data source
 - Atum – A completeness and accuracy library
 - Spline – A data lineage tracking and visualization tool (#SAISExp18)

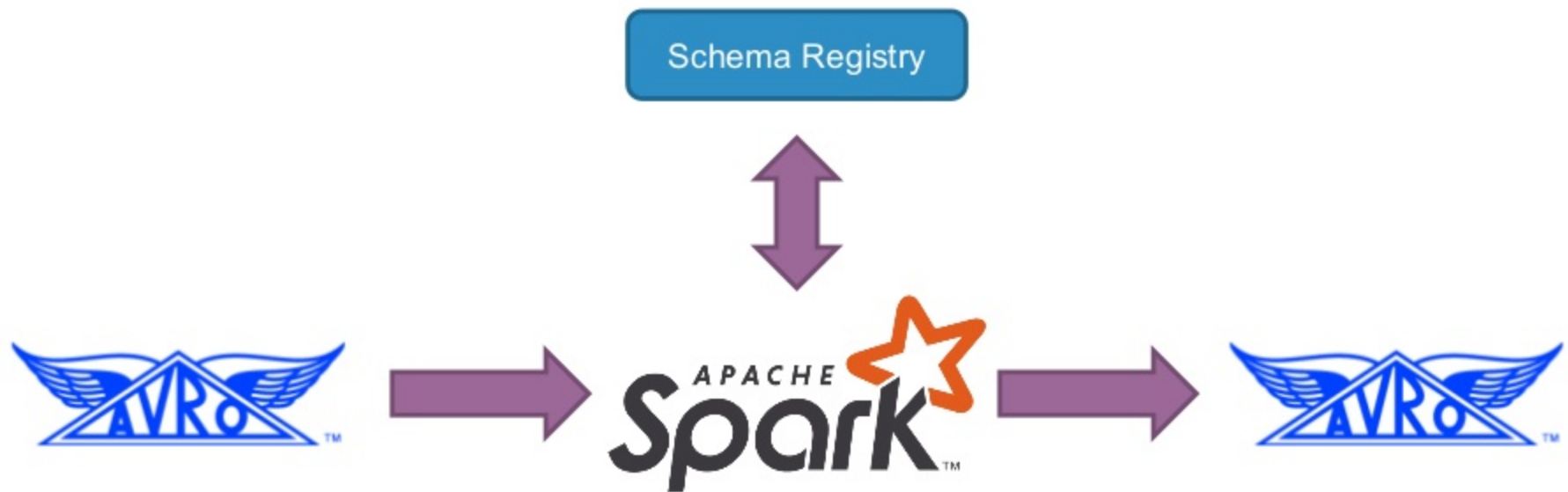
A Word on Spark 2.4

- ABRiS was initially developed for Spark 2.2
- Spark 2.4 is introducing built-in Avro conversion capabilities (Hurray!)
- But ABRiS still offers features that Spark 2.4 does not

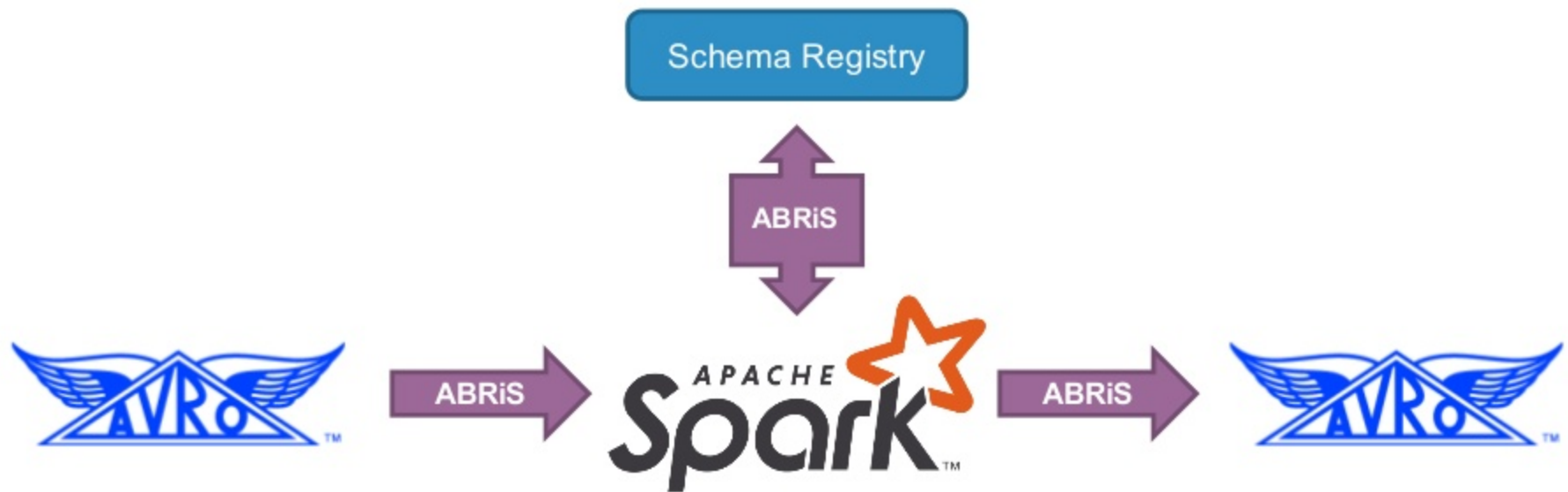
ABRiS Motivation

- Need for UDFs or map functions in Spark jobs to handle Avro data.
- Lack of support for Confluent distribution.
- Lack of support for schema management facilities such as Schema Registry.
- Lack of support for Spark structured streaming APIs.

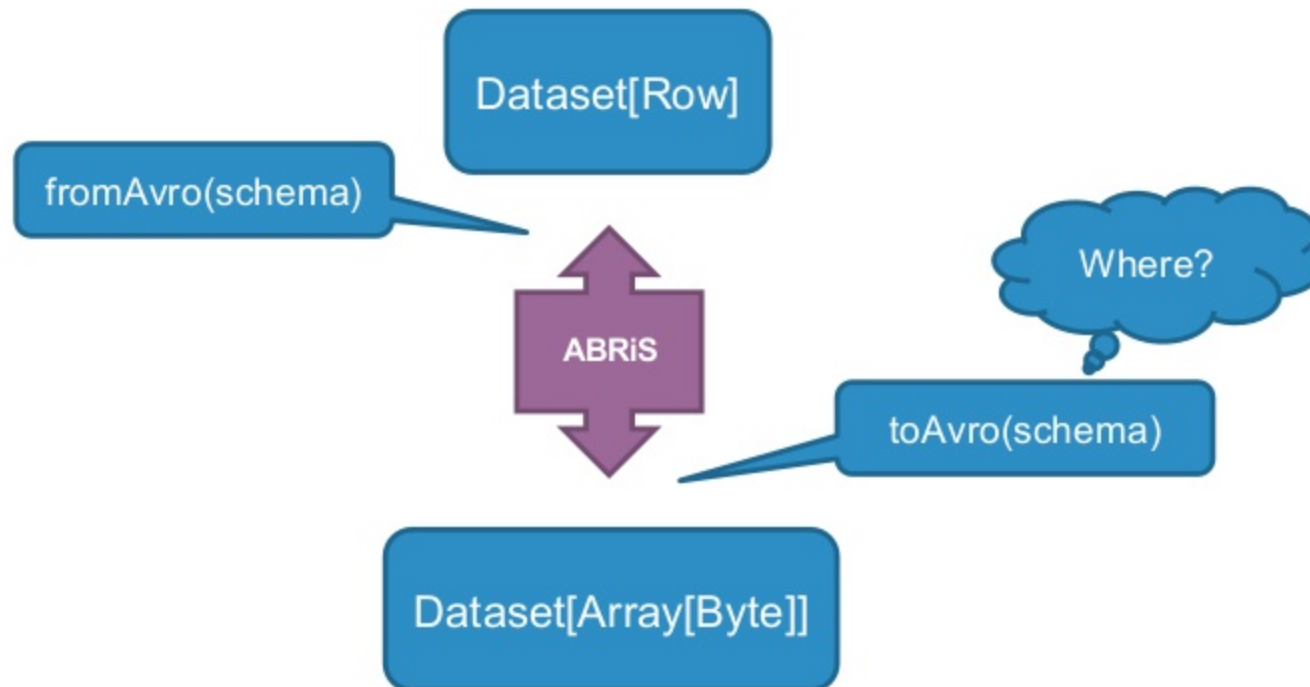
Bridging Avro and Spark



Bridging Avro and Spark



Bridging Avro and Spark



What we bring

- Ease of use
- Schema Retention Policy
- Schema Management
- Confluent Support
- Support for Kafka Keys

Use cases

Producer



Consumer



What we bring

- **Ease of use**
- Schema Retention Policy
- Schema Management
- Confluent Support
- Support for Kafka Keys

Writing to Kafka

```
val dataframe = ???
```

dataframe

```
.writeStream  
.format("kafka")  
.option("kafka.bootstrap.servers", "localhost:9092")  
.option("topic", "topic_name")  
.start()
```

Reading from Kafka

```
spark
  .readStream
  .format("kafka")
  .option("kafka.bootstrap.servers", "localhost:9092")
  .option("subscribe", "topic")

  .where("value.person.age > 30")
  .writeStream.format("console").start().awaitTermination()
```

Writing with an Avro schema

```
val avroSchema: Schema = ???
```

```
import za.co.absa.abris.avro.AvroSerDe._
```

```
dataframe
```

```
.toAvro(avroSchema)  
.writeStream  
.format("kafka")  
.option("kafka.bootstrap.servers", "localhost:9092")  
.option("topic", "topic_name")  
.start()
```

Reading with an Avro schema

```
val avroSchema: Schema = ???
```

```
import za.co.absa.abris.avro.AvroSerDe._
```

```
spark  
  .readStream  
  .format("kafka")  
  .option("kafka.bootstrap.servers", "localhost:9092")  
  .option("subscribe", "topic")  
  .fromAvro("value", avroSchema)(SchemaRetentionPolicies.RETAIN_ORIGINAL_SCHEMA)  
  .where("value.person.age > 30")  
  .writeStream.format("console").start().awaitTermination()
```

What we bring


- Ease of use
- **Schema Retention Policy**
- Schema Management
- Confluent Support
- Support for Kafka Keys

Schema Retention Policy

RETAIN_ORIGINAL_SCHEMA

key	value	topic	partition	offset	timestamp	timestampType
key 1	{'name': 'femel', 'age': 35}	test_topic	1	21	2018-...	...
key 2	{'name': 'user2', 'age': 28}	test_topic	2	22
...	...	test_topic

RETAIN_SELECTED_COLUMN_ONLY



username	age
femel	35
user2	28
...	...

What we bring

- Ease of use
- Schema Retention Policy
- **Schema Management**
- Confluent Support
- Support for Kafka Keys

Confluent Schema Registry

- Serving layer, RESTful interface for Avro schemas management
- Provides compatibility settings and evolution of schemas
- <https://github.com/confluentinc/schema-registry>

Reading with Schema Registry

```
val schemaRegistrySettings = Map(  
  SchemaManager.PARAM_SCHEMA_REGISTRY_URL    -> "http://somewhere:8081",  
  SchemaManager.PARAM_VALUE_SCHEMA_ID        -> "latest",  
  SchemaManager.PARAM_SCHEMA_REGISTRY_TOPIC  -> "topic"  
)  
  
import za.co.absa.abris.avro.AvroSerDe._  
  
spark  
  .readStream  
  .format("kafka")  
  .option("kafka.bootstrap.servers", "localhost:9092")  
  .option("subscribe", "topic")  
  .fromAvro("value", schemaRegistrySettings)(SchemaRetentionPolicies.RETAIN_ORIGINAL_SCHEMA)  
  .where("value.person.age > 30")  
  .writeStream.format("console").start().awaitTermination()
```

Writing with Schema Registry

```
val schemaRegistrySettings = Map(  
  SchemaManager.PARAM_SCHEMA_REGISTRY_URL    -> "http://somewhere:8081",  
  SchemaManager.PARAM_VALUE_SCHEMA_ID        -> "latest"  
)  
  
import za.co.absa.abris.avro.AvroSerDe._  
  
dataframe  
  .toAvro("topic", "schemaName", "schemaNamespace")(Some(schemaRegistrySettings))  
  .writeStream  
  .format("kafka")  
  .option("kafka.bootstrap.servers", "localhost:9092")  
  .option("topic", "topic_name")  
  .start()
```

Notes on Schema Registry

- Schema is registered if Schema Registry settings are provided **AND** schemas are compatible
 - Leverages Confluent Schema Registry client
- If no schema is provided, it is inferred from the Dataframe
 - `.toAvro("topic_name", "schemaName", "schemaNamespace")`

What we bring

- Ease of use
- Schema Retention Policy
- Schema Management
- **Confluent Support**
- Support for Kafka Keys

Notes on Confluent Kafka

- Confluent Kafka writers append the schema's ID to the top of the payload
- *io.confluent.kafka.serializers.AbstractKafkaAvroSerializer*

```
84      ByteArrayOutputStream out = new ByteArrayOutputStream();  
85      out.write(MAGIC_BYTE);  
86      out.write(ByteBuffer.allocate(idSize).putInt(id).array());
```


Writing to Confluent Kafka

```
val schemaRegistrySettings = Map(  
  SchemaManager.PARAM_SCHEMA_REGISTRY_URL -> "http://somewhere:8081",  
  SchemaManager.PARAM_VALUE_SCHEMA_ID    -> "latest"  
)  
  
import za.co.absa.abris.avro.AvroSerDe._  
  
dataframe  
  .toConfluentAvro("topic_name", "schemaName", "schemaNamespace")(schemaRegistrySettings)  
    (SchemaRetentionPolicies.RETAIN_ORIGINAL_SCHEMA)  
  .writeStream  
  .format("kafka")  
  .option("kafka.bootstrap.servers", "localhost:9092")  
  .option("topic", "topic_name")  
  .start()
```


Reading from Confluent Kafka

```
val schemaRegistrySettings = Map(  
  SchemaManager.PARAM_SCHEMA_REGISTRY_URL    -> "http://somewhere:8081",  
  SchemaManager.PARAM_VALUE_SCHEMA_ID        -> "latest",  
  SchemaManager.PARAM_SCHEMA_REGISTRY_TOPIC  -> "topic"  
)  
  
import za.co.absa.abris.avro.AvroSerDe._  
  
spark  
  .readStream  
  .format("kafka")  
  .option("kafka.bootstrap.servers", "localhost:9092")  
  .option("subscribe", "topic")  
  .fromConfluentAvro("value", None, schemaRegistrySettings)  
    (SchemaRetentionPolicies.RETAIN_ORIGINAL_SCHEMA)  
  .where("value.person.age > 30")  
  .writeStream.format("console").start().awaitTermination()
```

What we bring

- Ease of use
- Schema Retention Policy
- Schema Management
- Confluent Support
- **Support for Kafka Keys**

Writing to Kafka – Avro keys

```
val keySchema: Schema = ???  
val valueSchema: Schema = ???  
  
import za.co.absa.abris.avro.AvroSerDeWithKeyColumn._  
  
dataframe  
  .toAvro(keySchema, valueSchema)  
  .writeStream  
  .format("kafka")  
  .option("kafka.bootstrap.servers", "localhost:9092")  
  .option("topic", "topic_name")  
  .start()
```

Reading from Kafka – Avro keys

```
val keyAvroSchema: Schema = ???  
val valueAvroSchema: Schema = ???
```

```
import za.co.absa.abris.avro.AvroSerDeWithKeyColumn._
```

```
spark  
  .readStream  
  .format("kafka")  
  .option("kafka.bootstrap.servers", "localhost:9092")  
  .option("subscribe", "topic")  
  .fromAvro(keyAvroSchema, valueAvroSchema)  
    (SchemaRetentionPolicies.RETAIN_ORIGINAL_SCHEMA)  
  .where("value.person.age > 30")  
  .writeStream.format("console").start().awaitTermination()
```

Writing to Kafka – Plain keys

```
val valueSchema: Schema = ???
```

```
import za.co.absa.abris.avro.AvroSerDeWithKeyColumn._
```

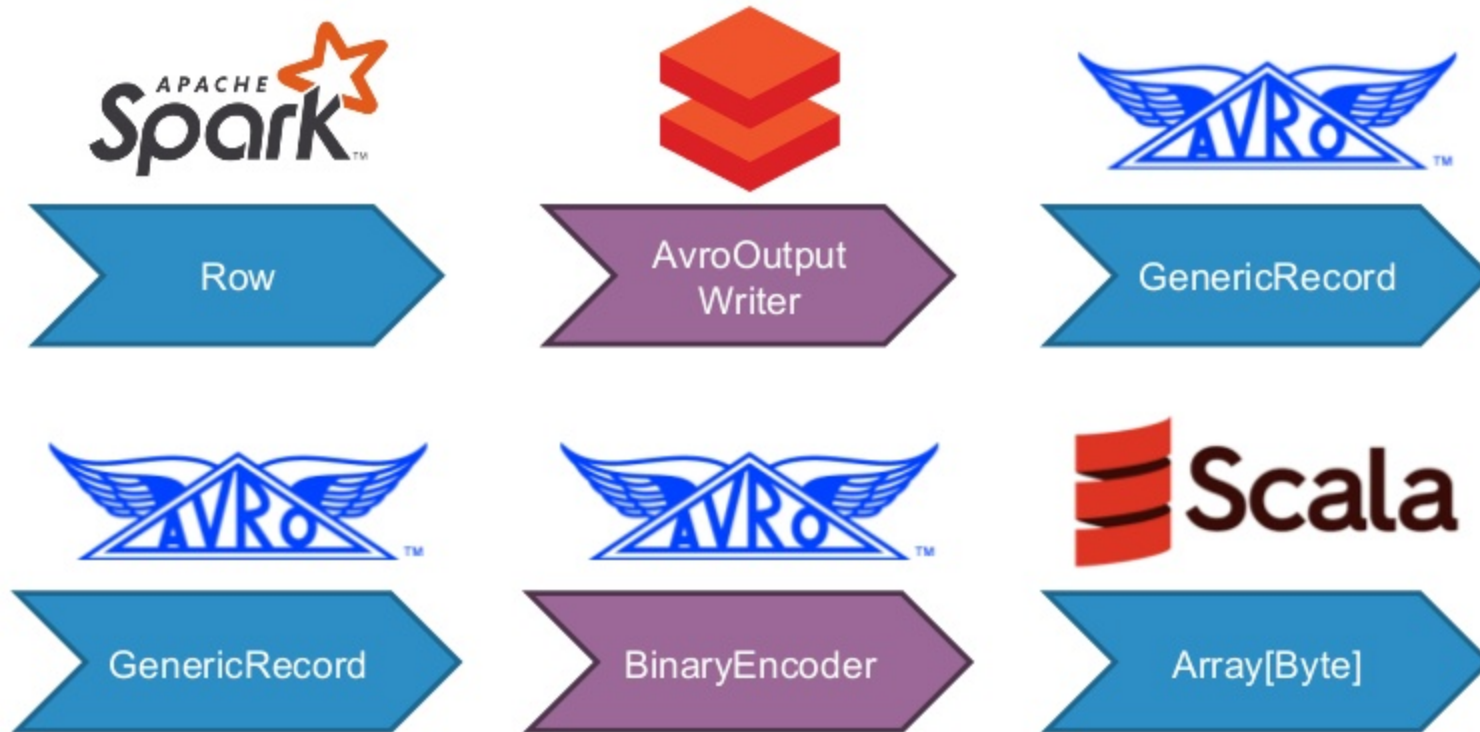
```
dataframe
```

```
.toAvroWithPlainKey(valueSchema)  
.writeStream  
.format("kafka")  
.option("kafka.bootstrap.servers", "localhost:9092")  
.option("topic", "topic_name")  
.start()
```

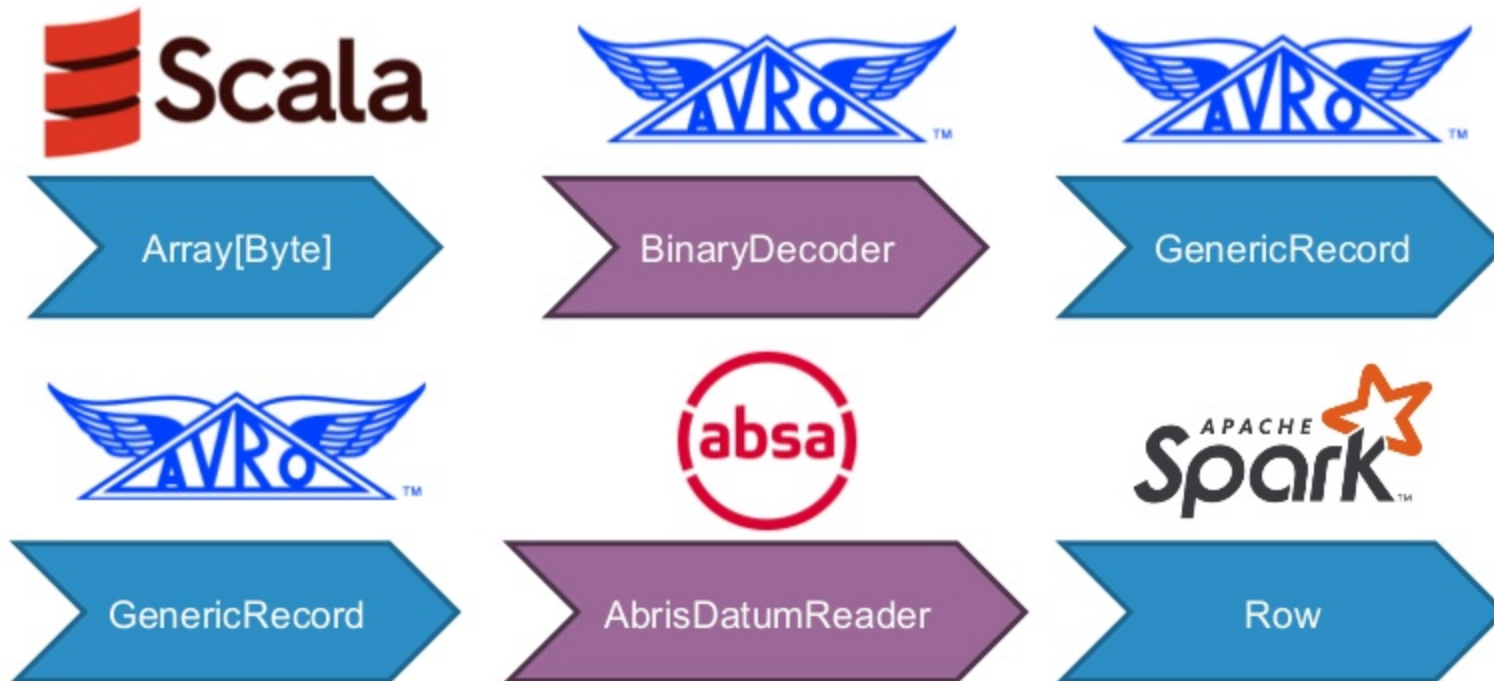
ABRiS ecosystem



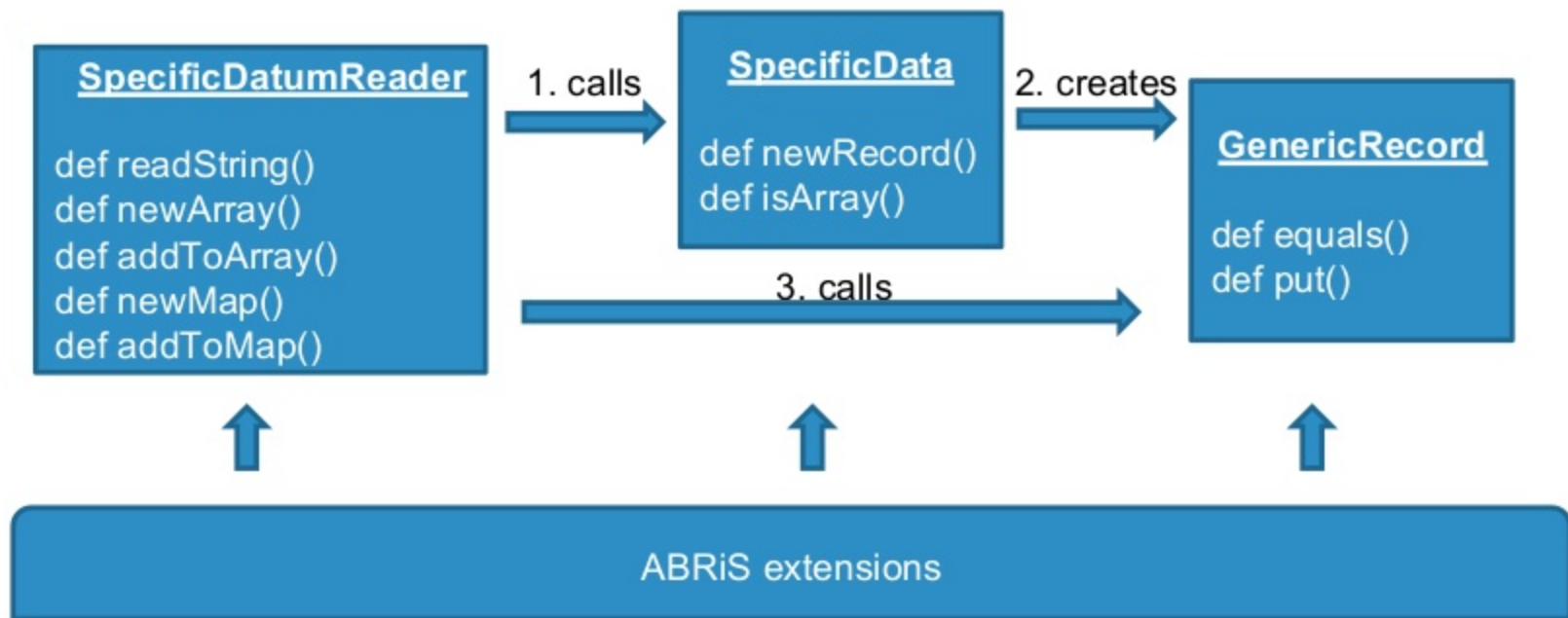
Under the hood – Spark to Avro



Under the hood – Avro To Spark



Under the hood – Avro Decoding



Avro encoding inside Spark

```
def toAvro(rows: Dataset[Row], schemas: SchemasProcessor)(schemald: Option[Int]): Dataset[Array[Byte]] = {
```

Avro encoding inside Spark

// trait for objects that can produce Avro and Spark schemas from each other

```
trait SchemasProcessor extends Serializable {  
  def getAvroSchema(): Schema  
  def getSparkSchema(): StructType  
}
```

Avro encoding inside Spark

```
def toAvro(rows: Dataset[Row], schemas: SchemasProcessor)(schemald: Option[Int]):Dataset[Array[Byte]] = {  
  implicit val recEncoder: Encoder[Array[Byte]] = Encoders.BINARY  
  
  rows.mapPartitions { partition =>  
  
    val avroSchema = schemas.getAvroSchema()  
    val sparkSchema = schemas.getSparkSchema()  
  
    partition.map { row =>  
      SparkAvroConversions.rowToBinaryAvro(row, sparkSchema, avroSchema, schemald)  
    }  
  }  
}
```

Avro decoding inside Spark

```
def fromAvroToRow(dataframe: Dataset[Row], schema: Schema): Dataset[Row] = {  
  implicit val rowEncoder: ExpressionEncoder[Row] = AvroToRowEncoderFactory.createRowEncoder(schema)  
  
  // has to convert into String and re-parse it inside the 'map' operation since Avro Schema is not serializable  
  val plainSchema = schema.toString()  
  
  dataframe.mapPartitions { partition =>  
    val avroDecoder = new AvroToRowConverter(  
      Some(AvroReaderFactory.createAvroReader(AvroSchemaUtils.parse(plainSchema))))  
  
    partition.map(avroRecord => avroDecoder.convert(avroRecord))  
  }  
}
```

Conclusion

- Seamless integration between Spark Structured APIs and Avro
- Schema Registry support
 - the standard for Avro schema management
- Provides Avro support for Spark pre-2.4
- Will interoperate seamlessly with Spark 2.4

Coming soon

- Unified and Spark 2.4 compliant API
 - to_avro
 - from_avro
- Other Schema Registry naming strategies
 - TopicNamingStrategy (✓)
 - Since Schema Registry 5.0, July 2018
 - RecordNamingStrategy (X)
 - TopicRecordNamingStrategy (X)

Thank you!

- Questions?
- Comments?
- <https://github.com/AbsaOSS/ABRiS>
- Contacts
 - Felipe Melo
 - felipesmmelo@gmail.com
 - Georgi Chochov
 - g.chochov@gmail.com