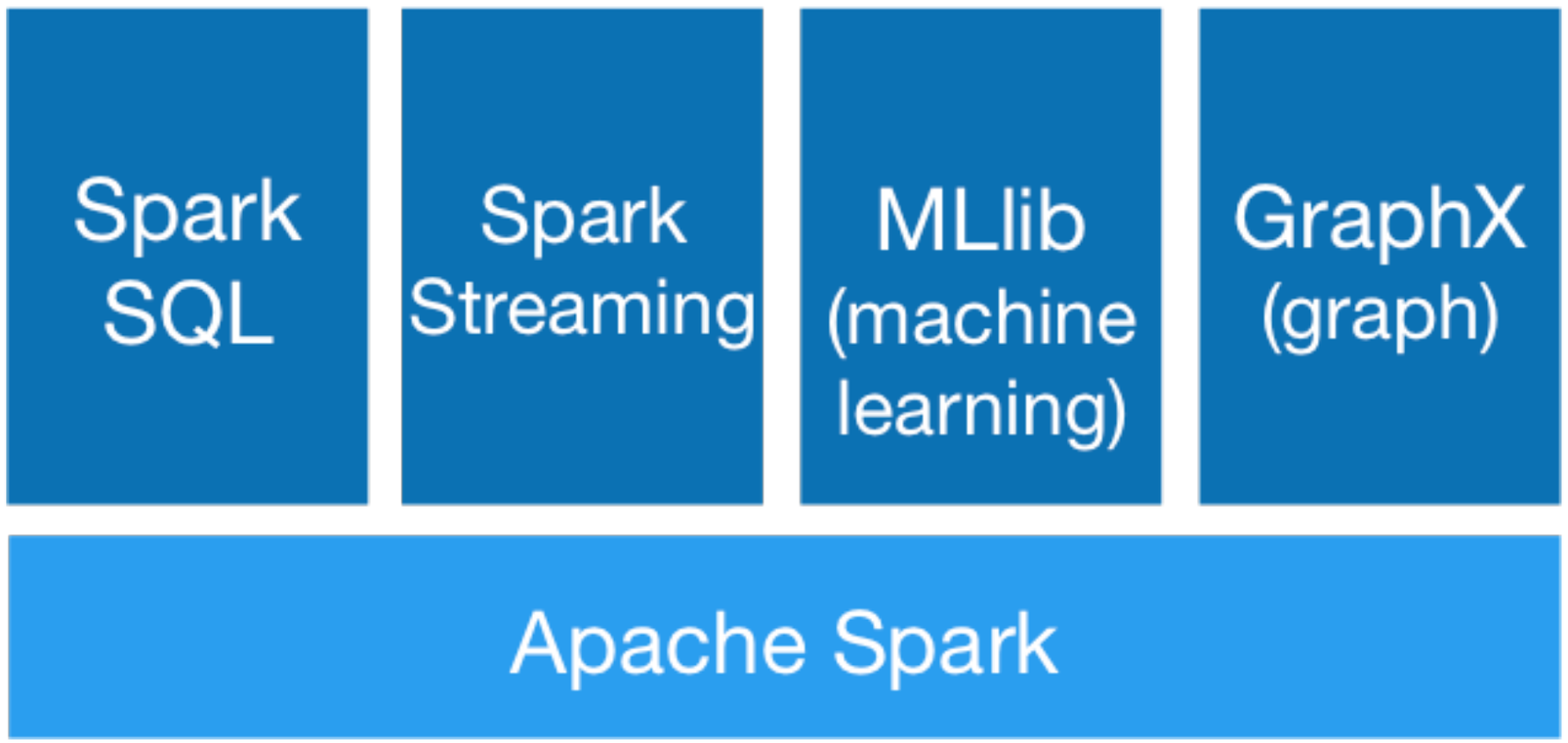
Structured Streams in Spark 2.0

Long Tran

Overview

- RDDs
  - RDD / Scala API
- D-Streams (0.7)
- SQL (1.3)
  - Dataframes API
  - Catalyst
  - Tungsten (1.4)
- Structured Streams (finally!) (2.0)
  - File API - future APIs
  - Exactly Once semantics

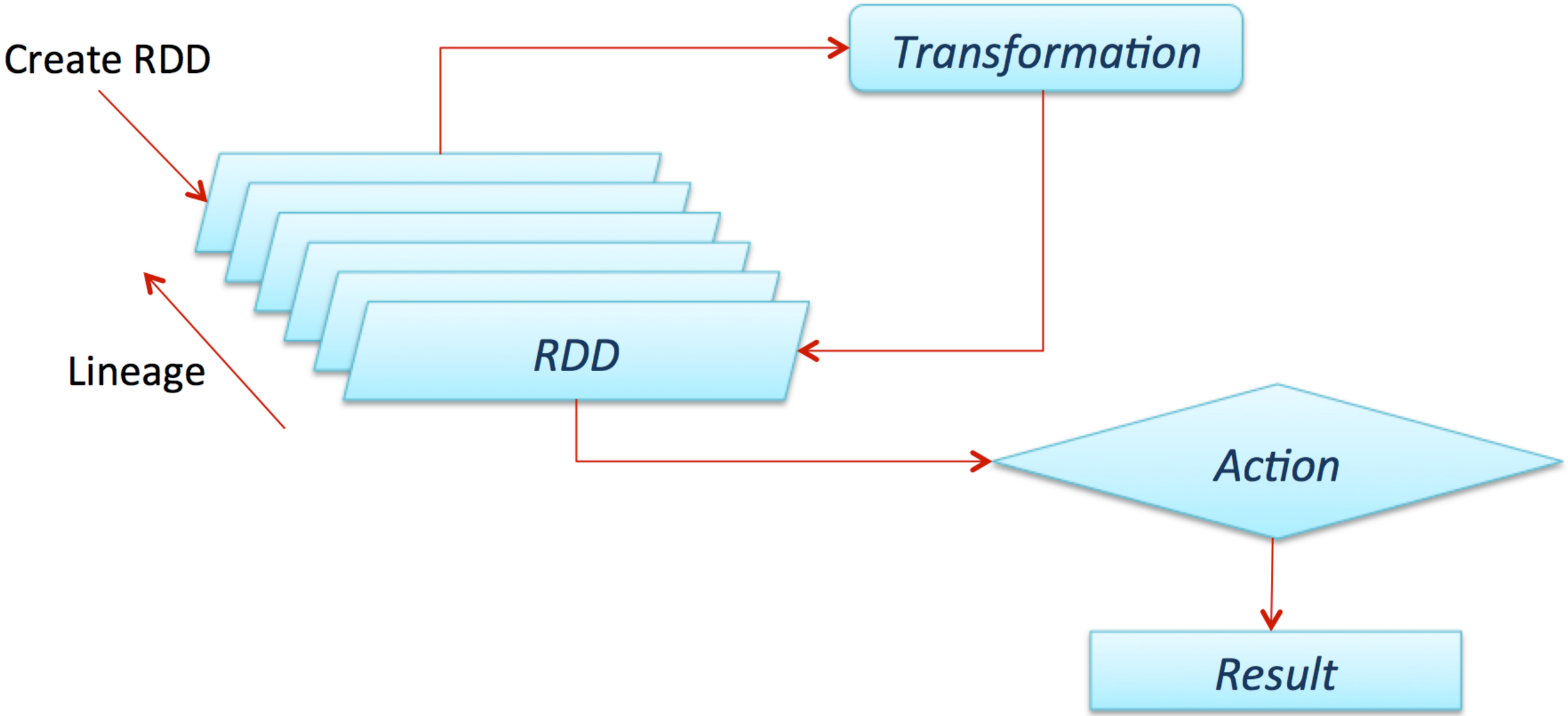


https://spark.apache.org/

RDD

Resilient Distributed Dataset

A parallelized, lazily evaluated, directed acyclic graph of computation.



https://dzone.com/refcardz/apache-spark

RDD + Scala Word Count

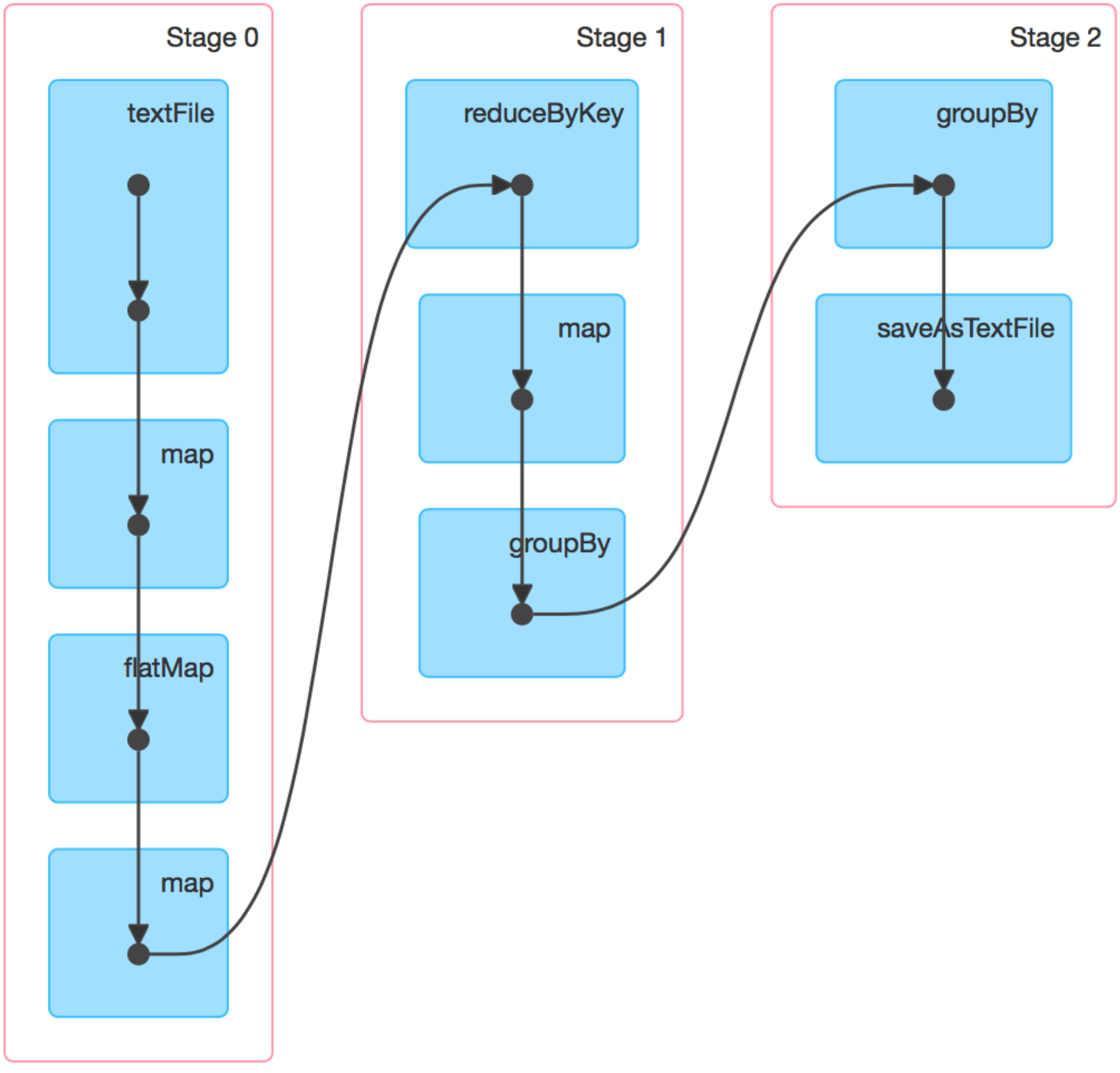
SCALA

```scala
val wordCount = words.toLowerCase
  .split(" ")
  .groupBy(word => word)
  .mapValues(value => value.length)
```

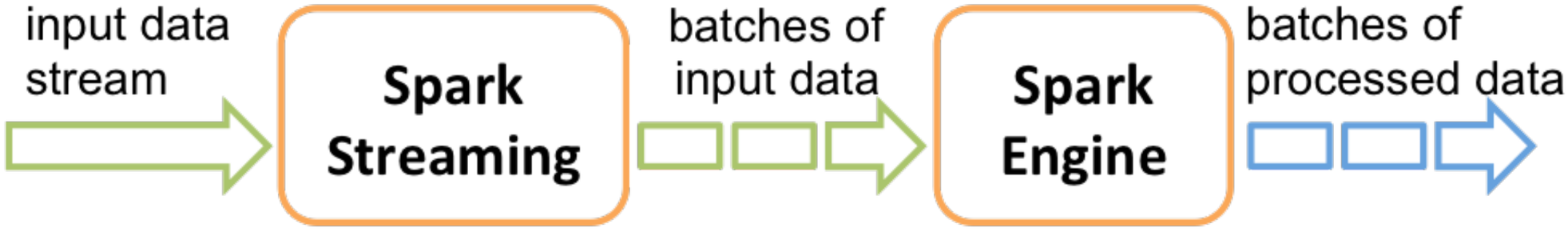
SPARK

```scala
val sc = new SparkContext()
val words = sc.textFile("shakespeare.txt")

val wordCount = words.map(line => line.toLowerCase)
    .flatMap(line => line.split(" "))
    .groupBy(word => word)
    .mapValues(value => value.size)
```

▼ DAG Visualization

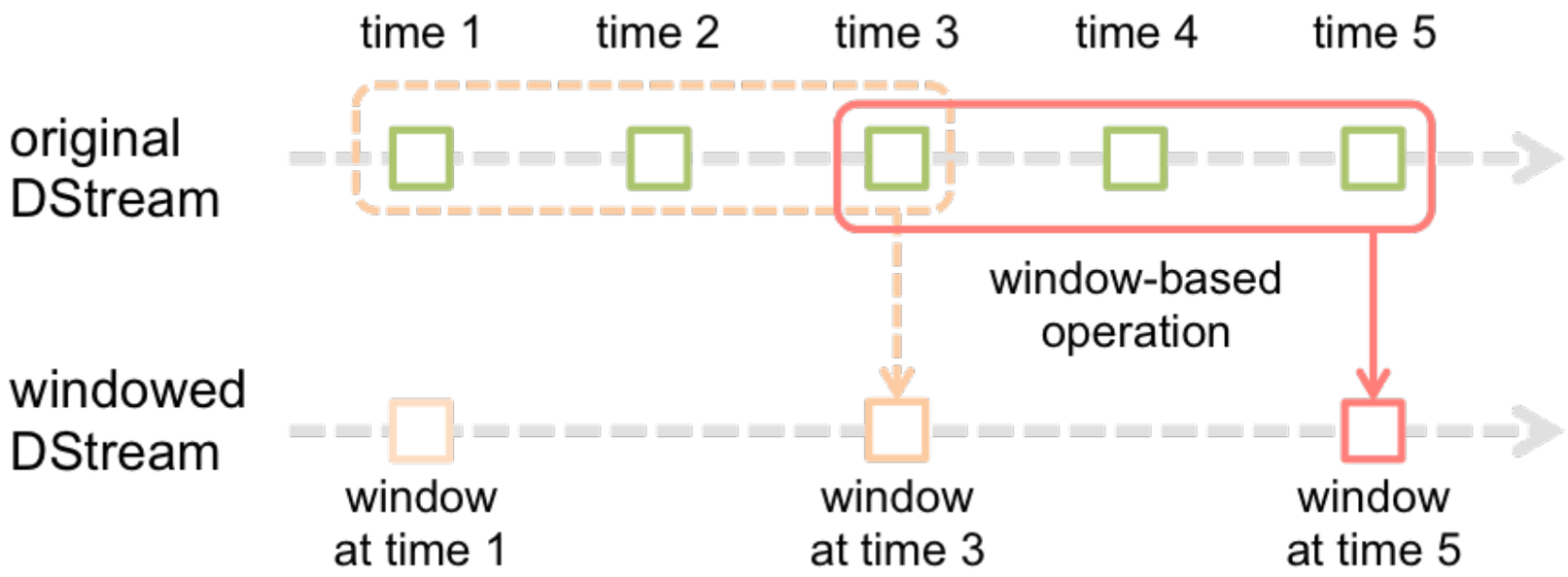


Spark Streaming



http://spark.apache.org/docs/latest/streaming-programming-guide.html

DStream API


time 1
time 2
time 3
time 4
time 5
original DStream
window-based operation
windowed DStream
window at time 1
window at time 3
window at time 5

DStream Word Count

```
val incomingFiles = ssc.textFileStream("inputDir/")

val wordCount  =  incomingFiles.flatMap(x => x.split(" "))
                               .map(word => (word, 1))
                               .reduceByKeyAndWindow(reduceFunc = _ + _,
                                 invReduceFunc = _ - _,
                                 slideDuration = Seconds(10),
                                 windowDuration = Seconds(30))
```

SQL & DataFrames

API for computing structured data

DataFrames and SQL Word Count

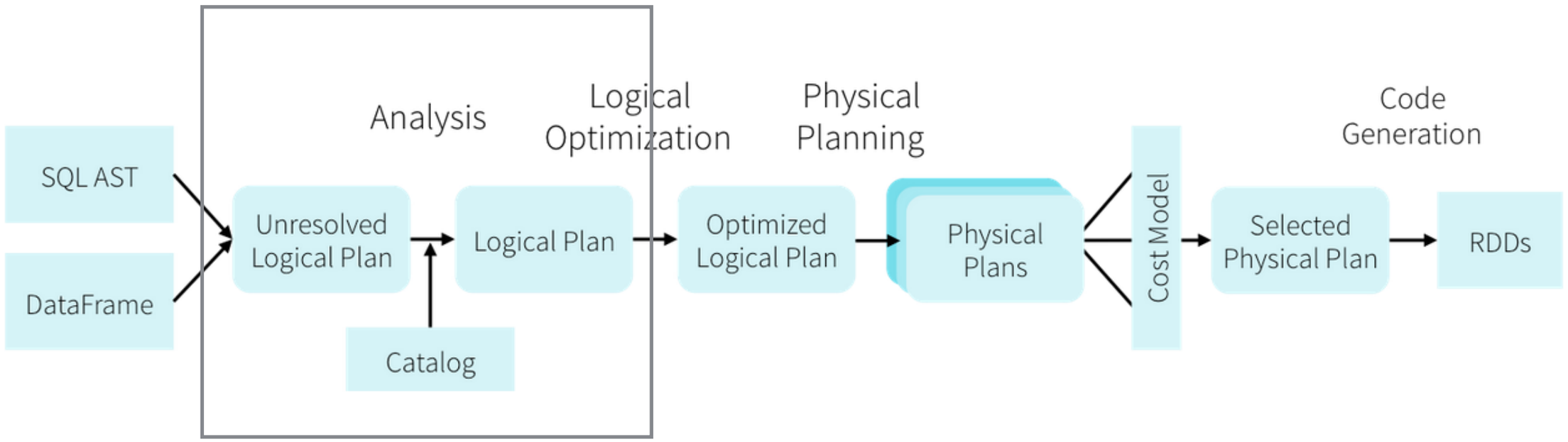
DataFrames

```
val linesDF = spark.sparkContext.textFile("data/all-shakespeare.txt").toDF("line")
val wordsDF = linesDF.select(explode(split('line, " ")).as("word"))
val wordCountDF = wordsDF.groupBy("word").count()
wordCountDF.show()
```

SQL

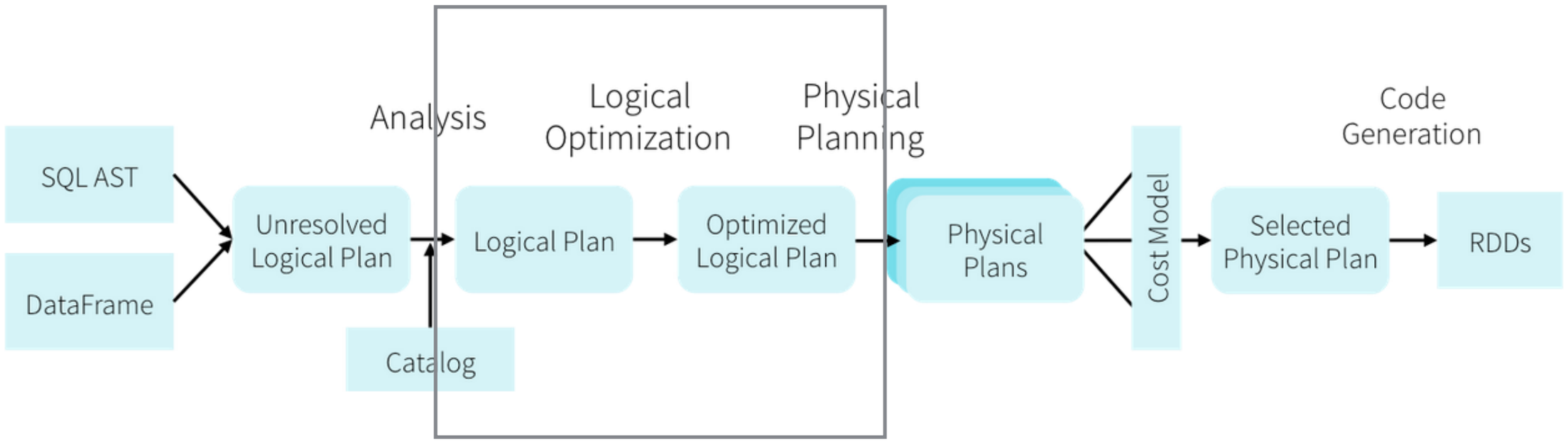
```
linesDF.createOrReplaceTempView("lineDF")
val wordCountSQL = spark.sql(
    """select substr(line,1,(instr(line," ")-1)) AS word,  count(*) AS count
      |   from lineDF
      |   group by  substr(line,1,(instr(line," ")-1))
      |   order by count desc""".stripMargin)
wordCountSQL.show
```

Catalyst

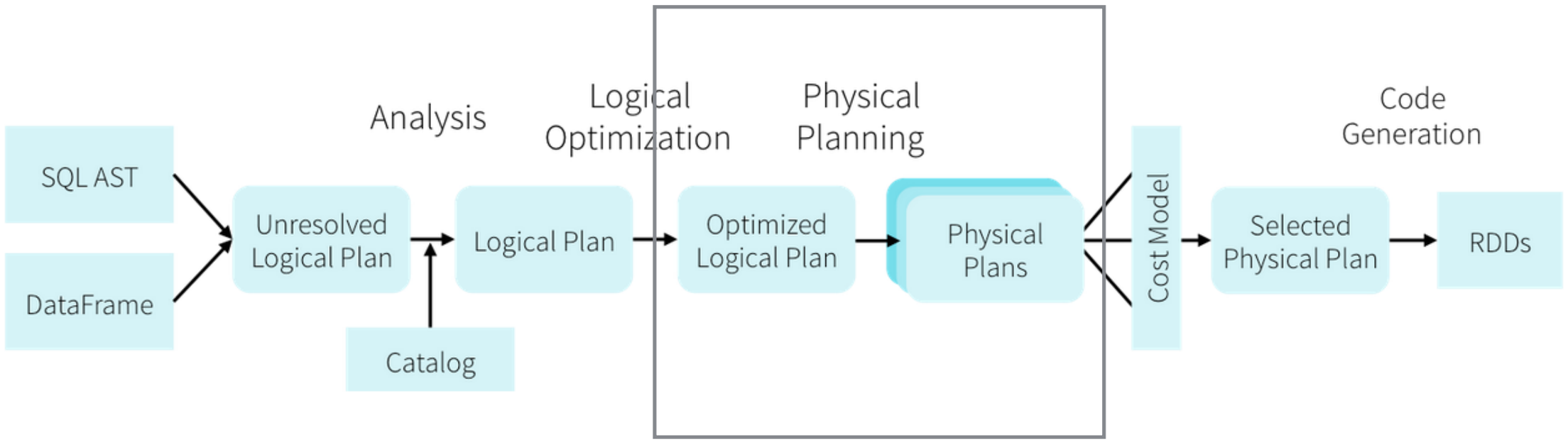


https://databricks.com/blog/2015/04/13/deep-dive-into-spark-sqls-catalyst-optimizer.html

Catalyst


Analysis
Logical Optimization
Physical Planning
Code Generation
SQL AST
DataFrame
Unresolved Logical Plan
Logical Plan
Optimized Logical Plan
Physical Plans
Cost Model
Selected Physical Plan
RDDs
Catalog

Catalyst


Analysis
Logical Optimization
Physical Planning
Code Generation
SQL AST
DataFrame
Unresolved Logical Plan
Catalog
Logical Plan
Optimized Logical Plan
Physical Plans
Cost Model
Selected Physical Plan
RDDs

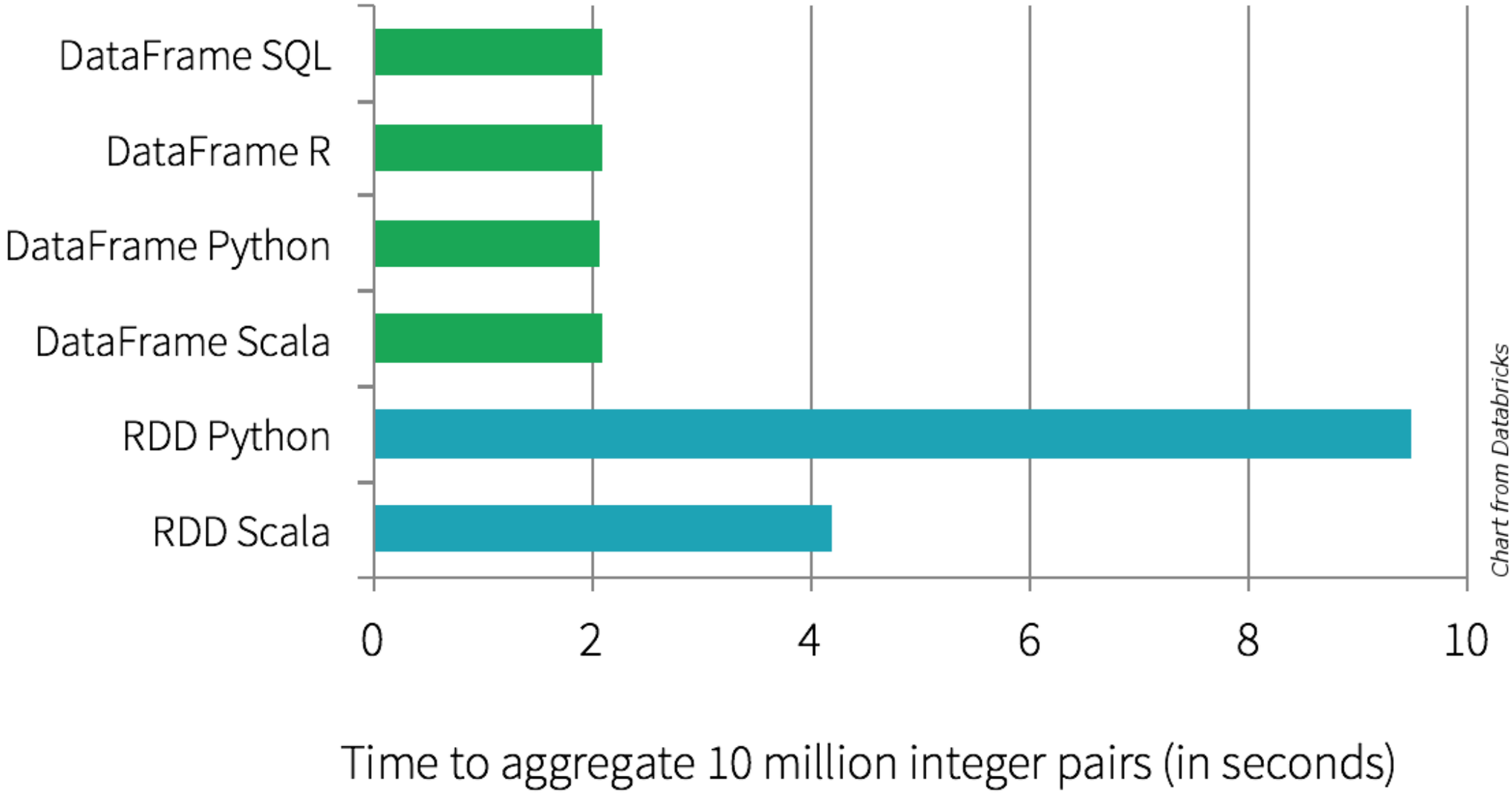
Tungsten

Memory Management and Binary Processing

www.slideshare.net/databricks/2015-0616-spark-summit

Tungsten

Cache Aware Computation


DataFrame SQL
DataFrame R
DataFrame Python
DataFrame Scala
RDD Python
RDD Scala
0
2
4
6
8
10
Chart from Databricks
Time to aggregate 10 million integer pairs (in seconds)

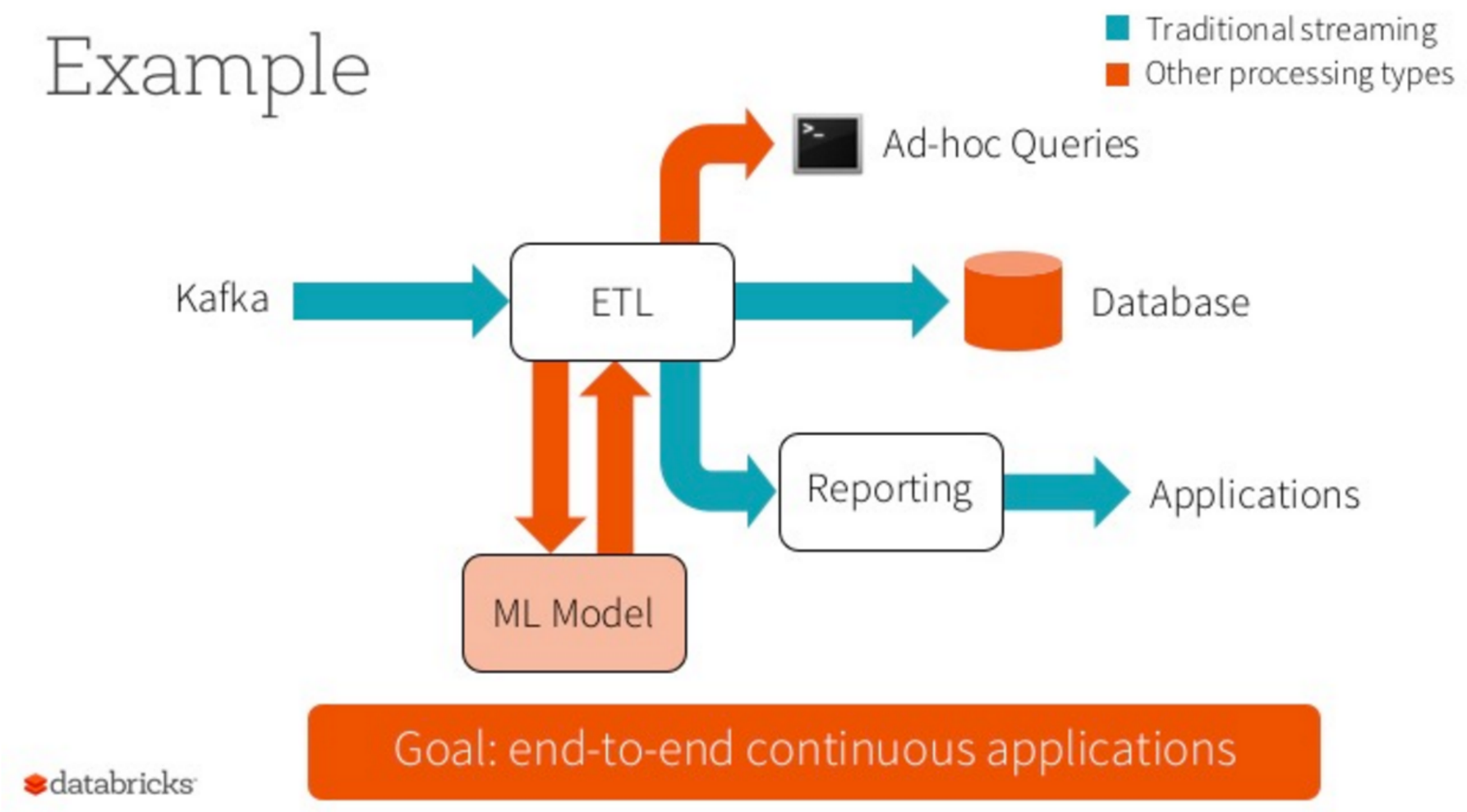
Spark 2.0 is the ALPHA RELEASE of Structured Streaming

Structured Streaming provides fast, scalable, fault-tolerant, end-to-end exactly-once stream processing without the user having to reason about streaming.

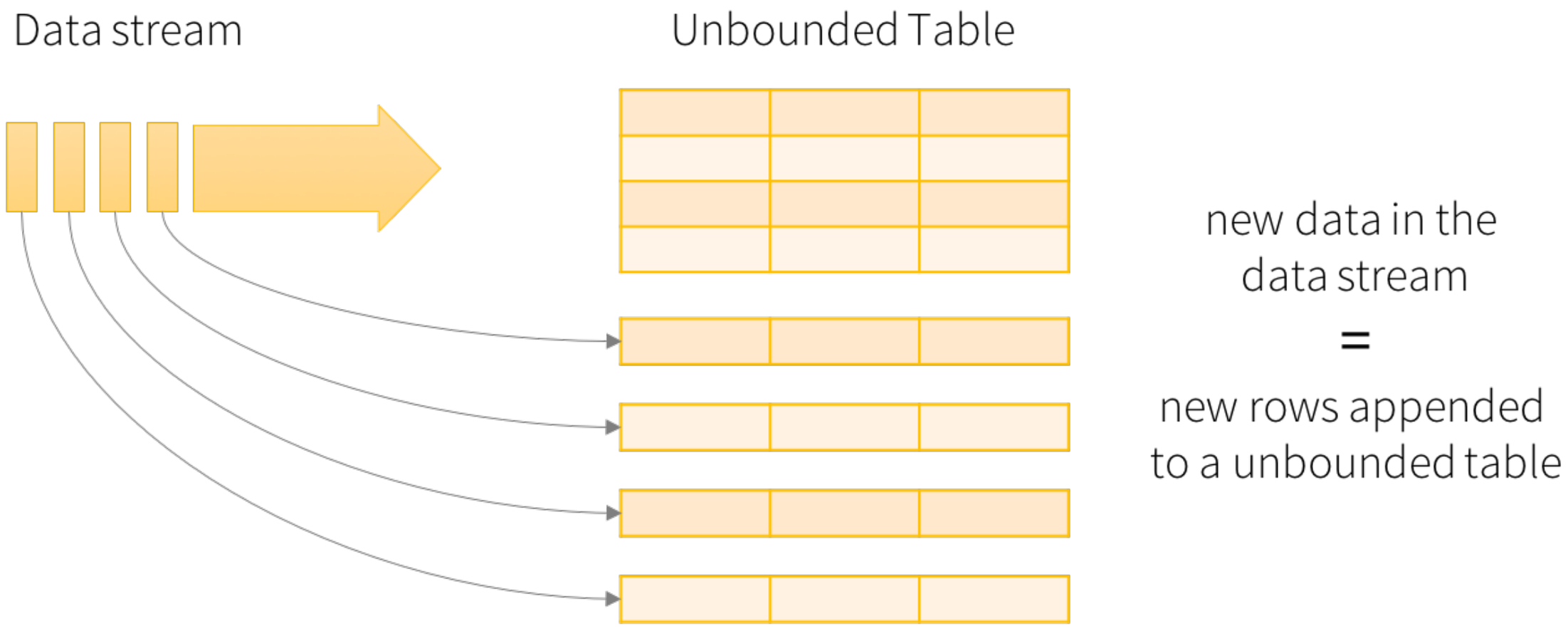
Classic Streaming

(interactions with other systems
left to the user)

Continuous Applications

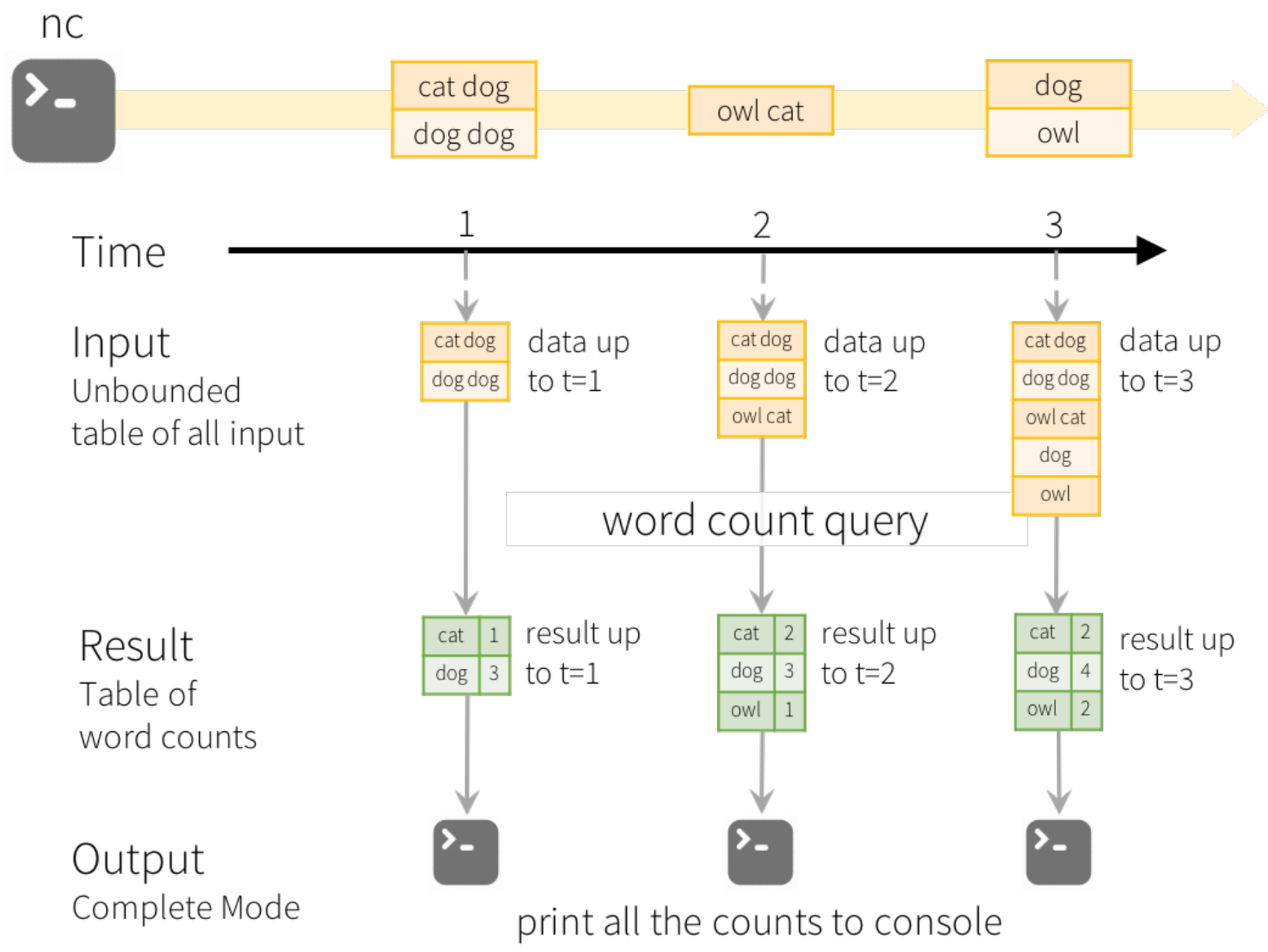


Programming Model



Data stream as an unbounded table

https://spark.apache.org/docs/latest/img/structured-streaming-stream-as-a-table.png

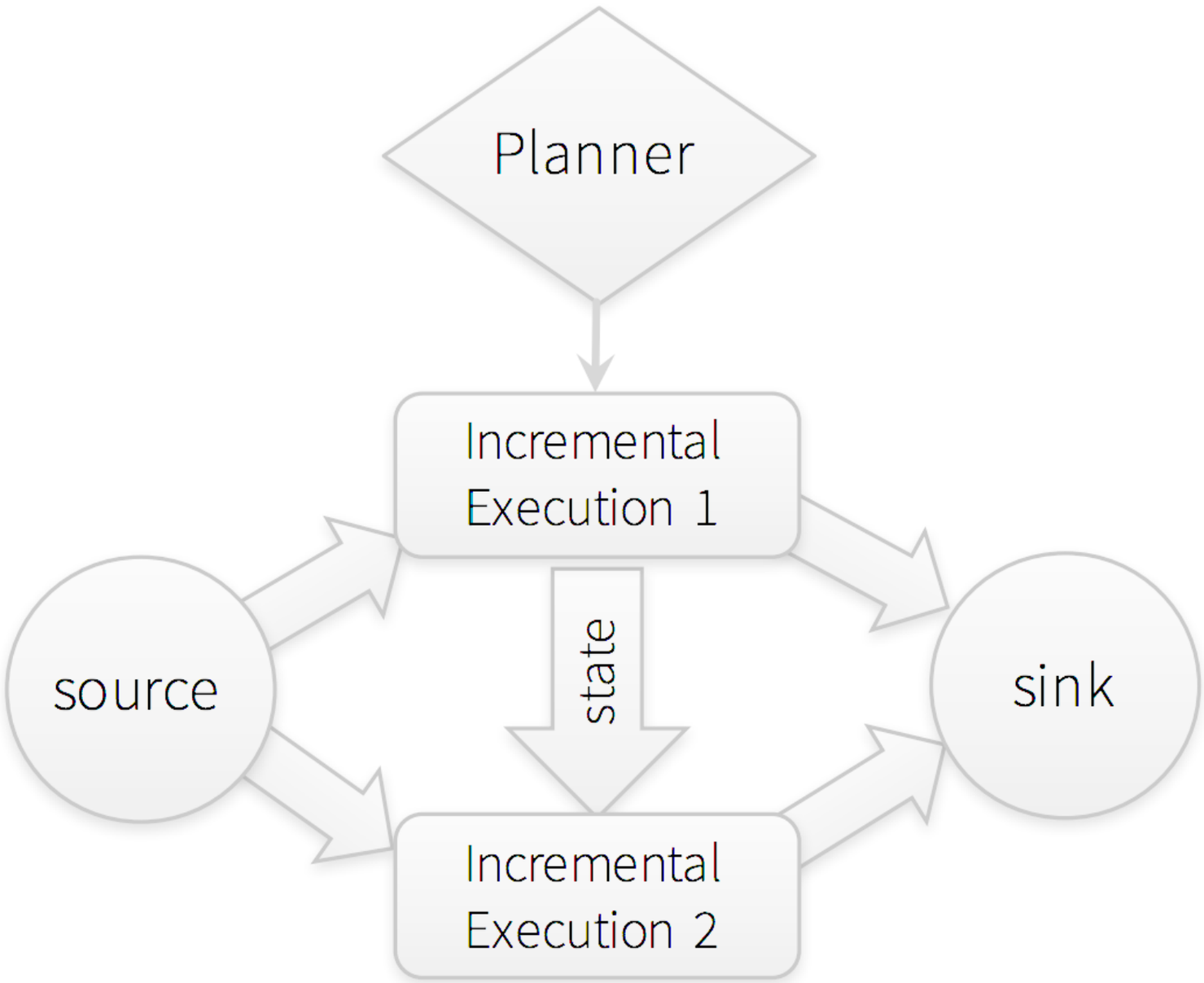


Model of the Quick Example

Structured Stream Word Count

```scala
val spark = SparkSession
  .builder
  .appName("StructuredStreamWordCount")
  .master("local")
  .getOrCreate()

import spark.implicits._

val lines = spark.readStream.text("inputDir/")

val words = lines.as[String].flatMap(_.split(" "))

val wordCounts = words.groupBy("value").count()

val query = wordCounts.writeStream
  .outputMode("complete")
  .format("console")
  .start()

query.awaitTermination()
```

end-to-end exactly once guarantees



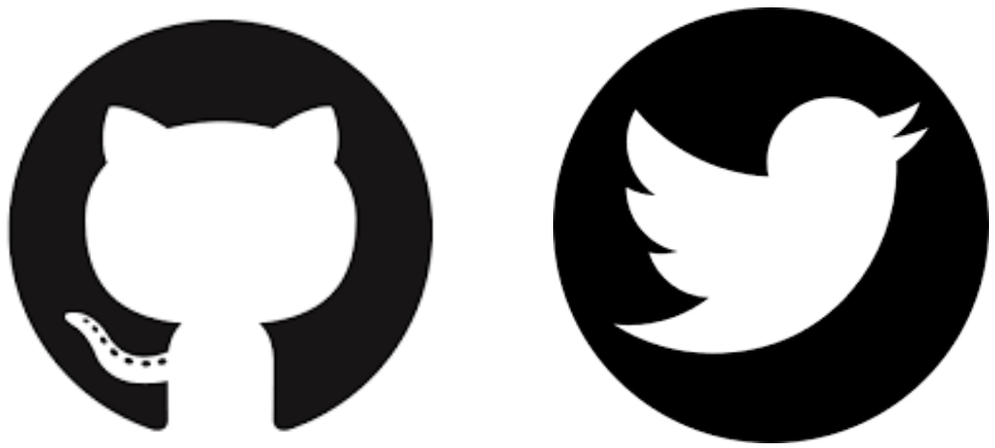
Conclusions

- Dataframe and SQL for streaming
- Catalyst!
- Tungsten!
- Unified API for batch and streaming (+ ML + GraphFrames)
- BIs, DBAs, Data Scientists can now do streaming!
- Exactly once guarantees
- No need to reason about intervals
- Event Time primitives

Future

- Current support for reading file streams only
- Kafka Integration (2.1)
- Public API for sources and sinks
- Watermarks
- ML Integration - continuously updated models

@LooooongTran



Sources

- https://databricks.com/blog/2016/07/28/structured-streaming-in-apache-spark.html
- https://databricks.com/blog/2016/07/28/continuous-applications-evolving-streaming-in-apache-spark-2-0.html
- https://www.youtube.com/watch?v=rl8dIzTpxrI
- https://www.youtube.com/watch?v=fn3WeMZZcCk
- https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html
- https://www.oreilly.com/learning/apache-spark-2-0--introduction-to-structured-streaming
- https://databricks.com/blog/2015/04/28/project-tungsten-bringing-spark-closer-to-bare-metal.html
- https://databricks.com/blog/2015/04/13/deep-dive-into-spark-sqls-catalyst-optimizer.html
- https://www.youtube.com/watch?v=1a4pgYzeFwE
- https://www.youtube.com/watch?v=5ajs8EIPWGI
- http://www.kdnuggets.com/2016/05/spark-tungsten-burns-brighter.html
- https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-sql-whole-stage-codegen.html