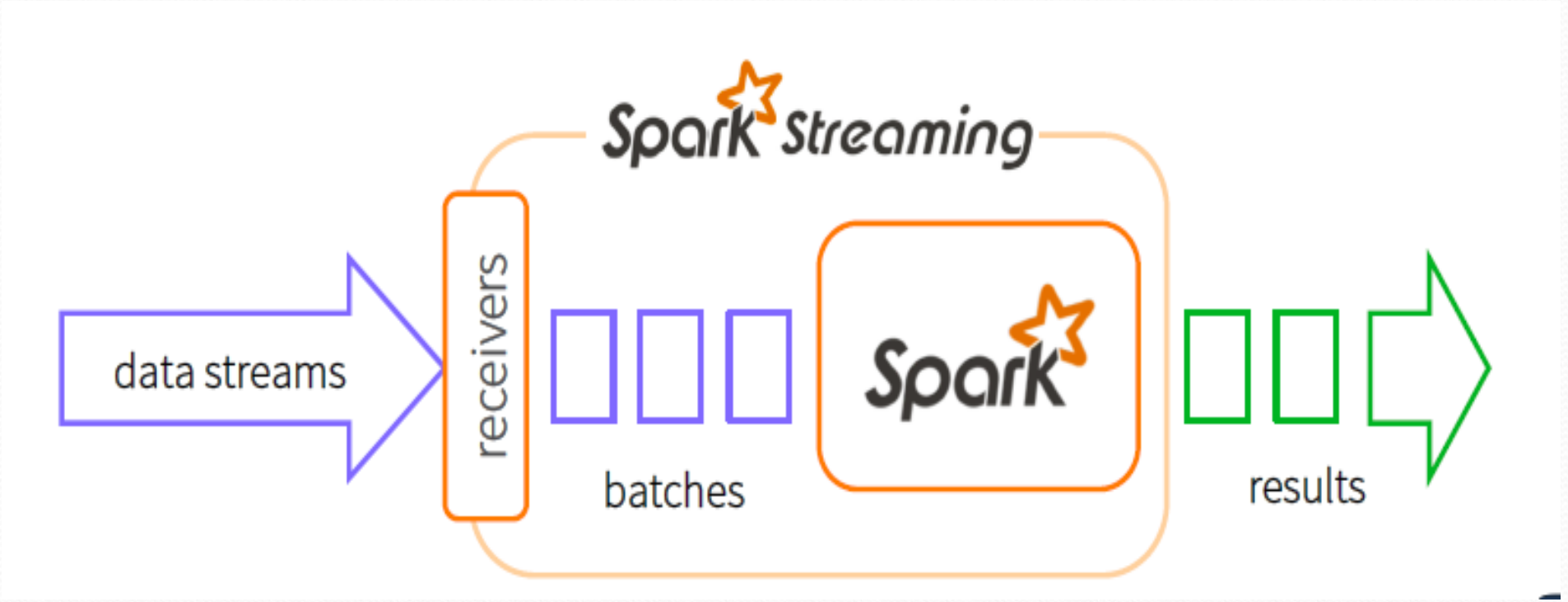
Apache Spark Streaming Library

Spark Streaming

- Features of Spark Streaming
  - High Level API (joins, windows etc.)
  - Fault – Tolerant (exactly once semantics achievable)
  - Deep Integration with Spark Ecosystem (MLlib, SQL, GraphX etc.)

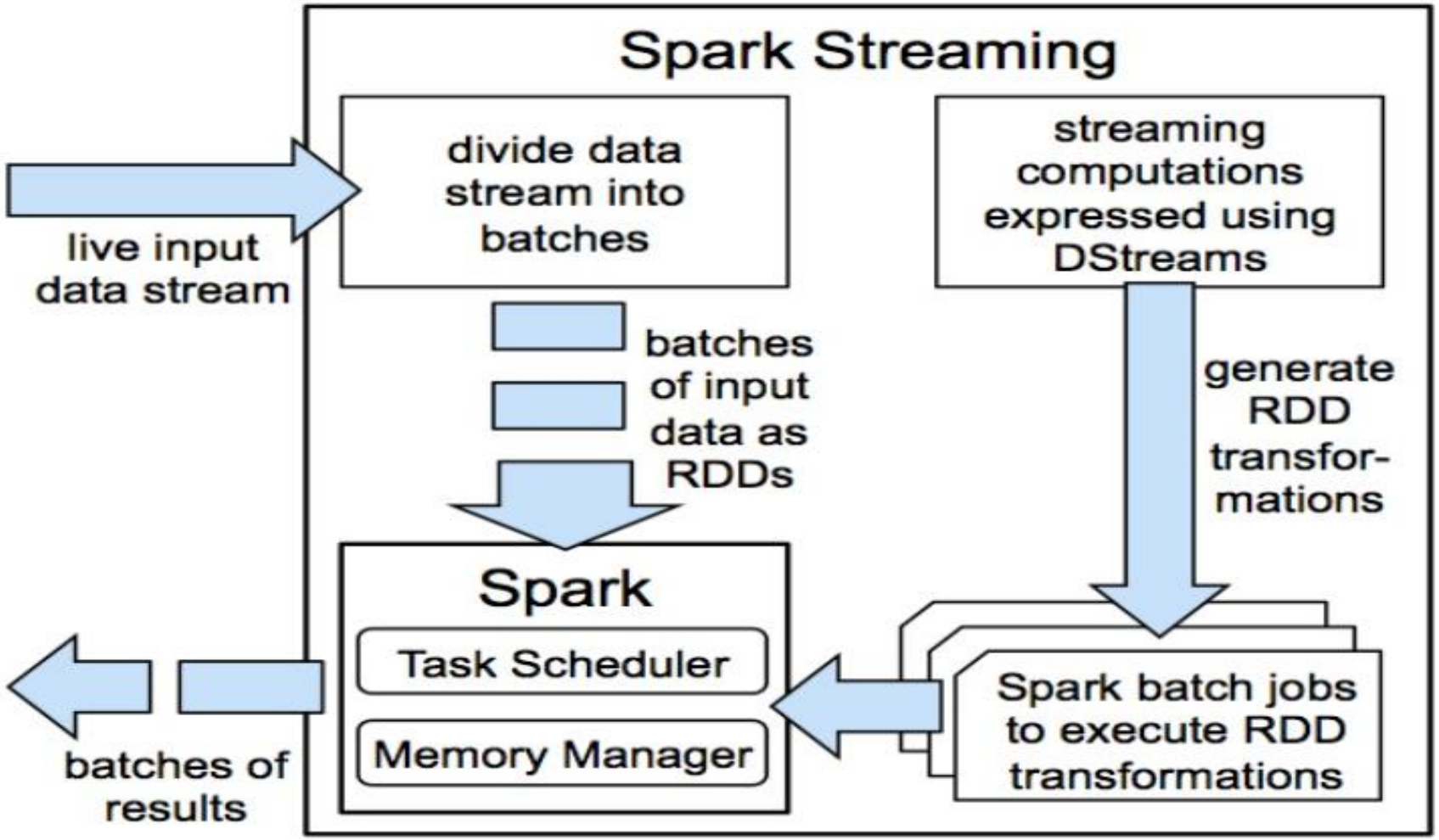
Spark Streaming

- Receivers receive data streams and chop them up into batches.
- Spark processes these batches and pushes out the results.

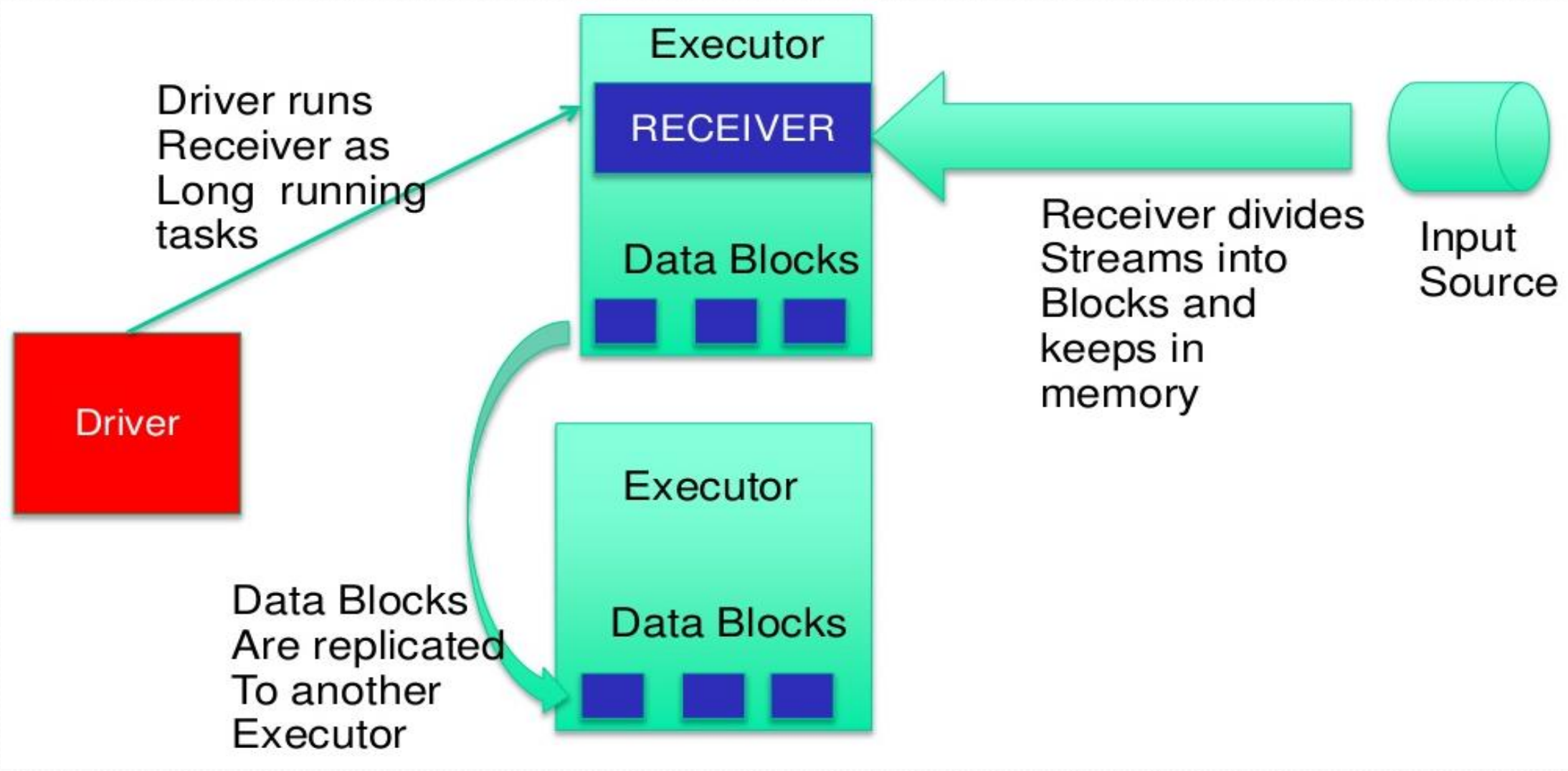


Architecture

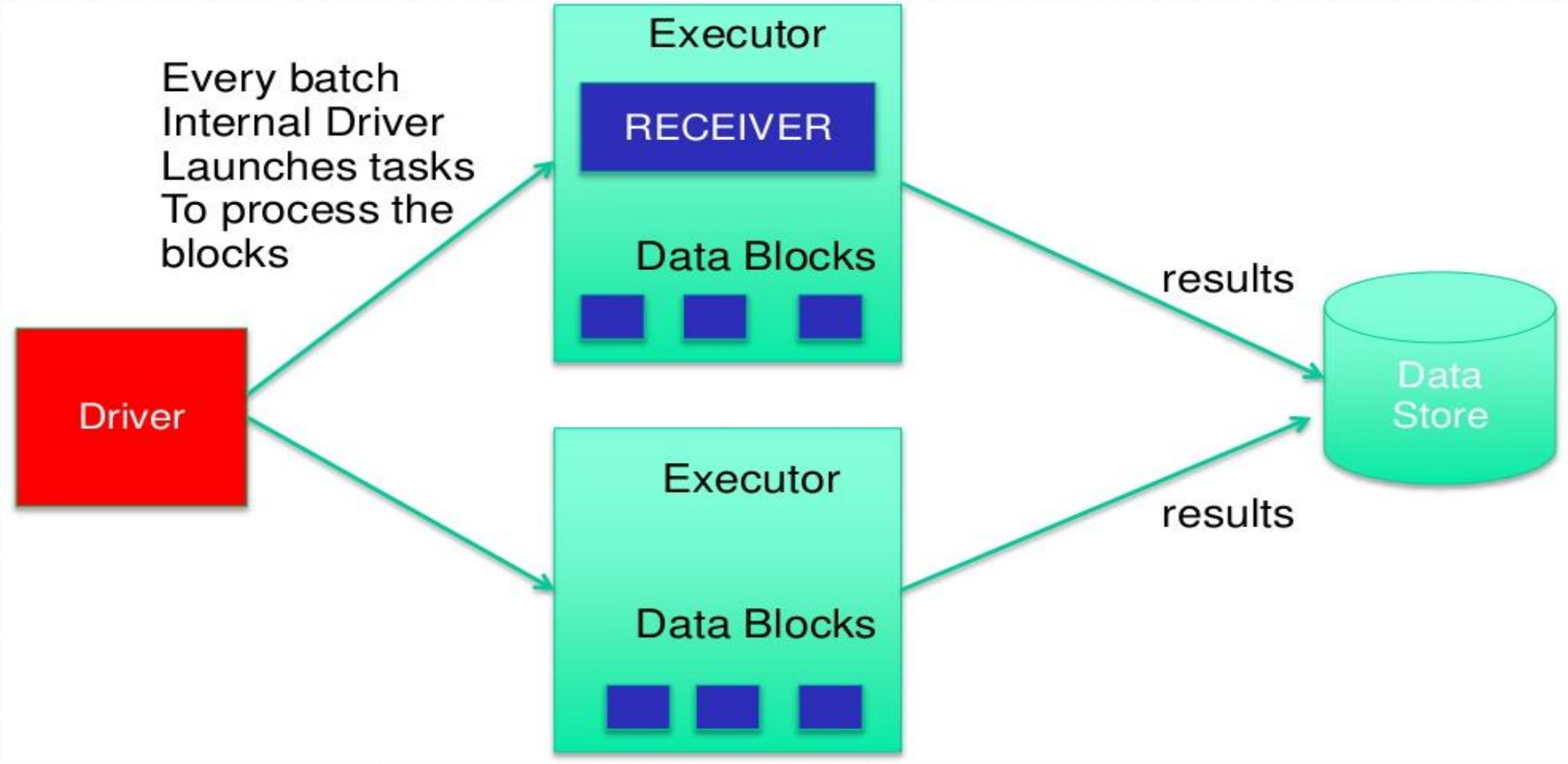
High Level Overview



Receiving Data



Processing Data



Word Count with Kafka

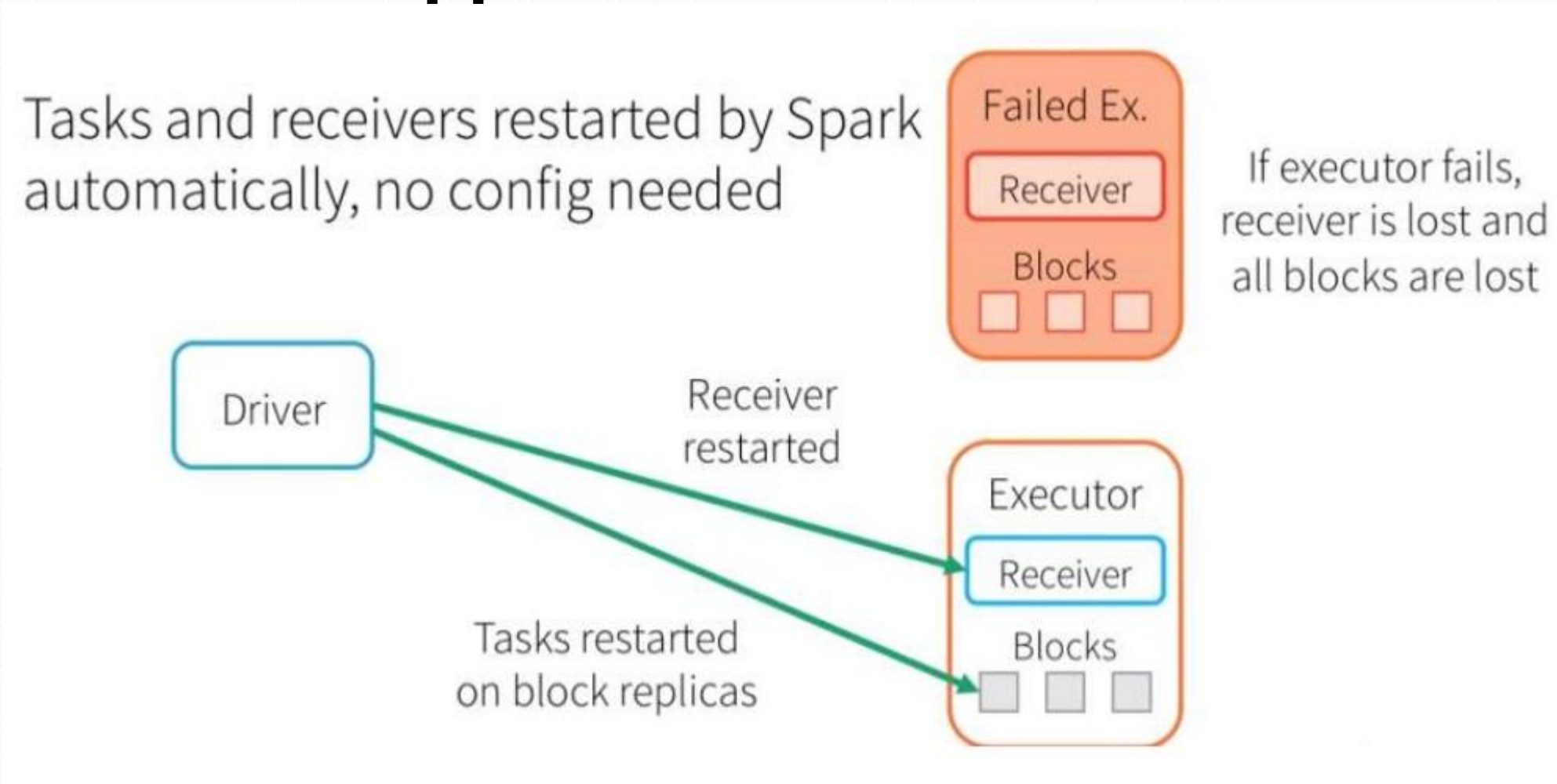
```
object WordCount {
  def main(args: Array[String]) {
    val context = new StreamingContext(new SparkConf(), Seconds(1))
    val lines = KafkaUtils.createStream(context, ...)
    val words = lines.flatMap(_.split(" "))
    val wordCounts = words.map(x => (x,1)).reduceByKey(_ + _)
    wordCounts.print()
    context.start()
    context.awaitTermination()
  }
}
```

Fault Tolerance and Reliability

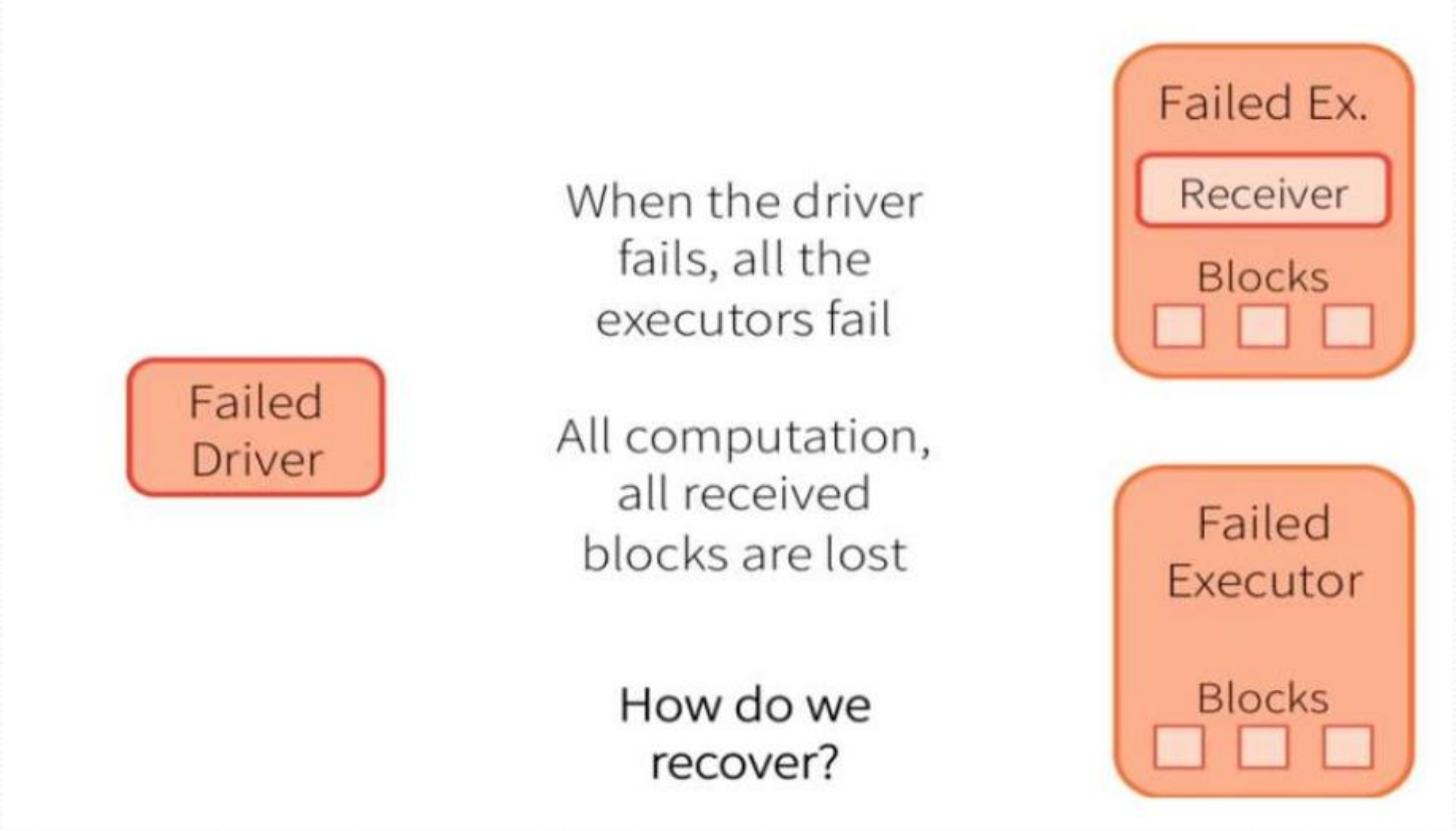
Fault Tolerance

- Why Care ?
- Different guarantees for Data Loss
  - At least Once
  - Exactly Once
- What all can fail ?
  - Driver
  - Executor

What happens when Executor fails ?



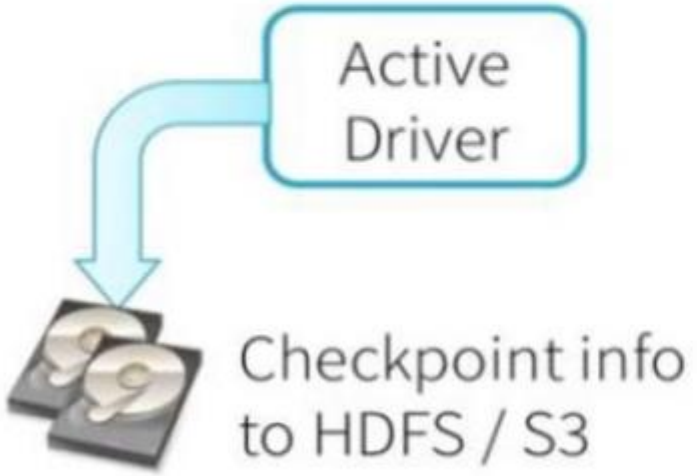
What happens when Driver fails ?

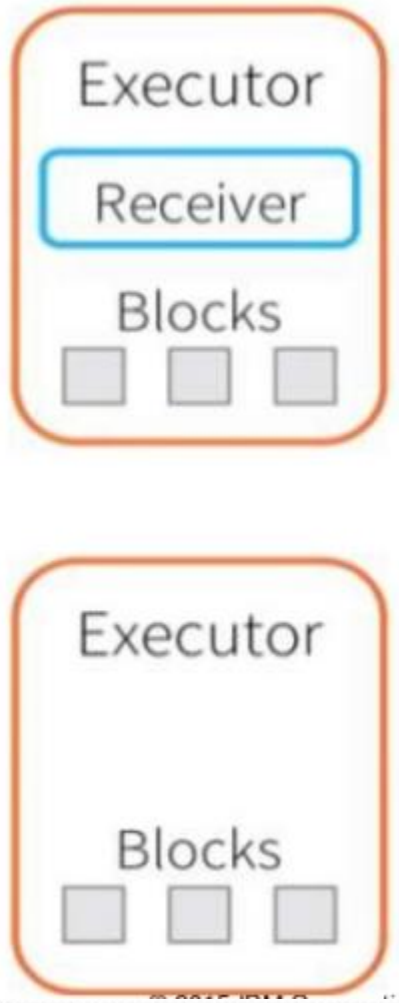


Recovering Driver – DStream Checkpointing

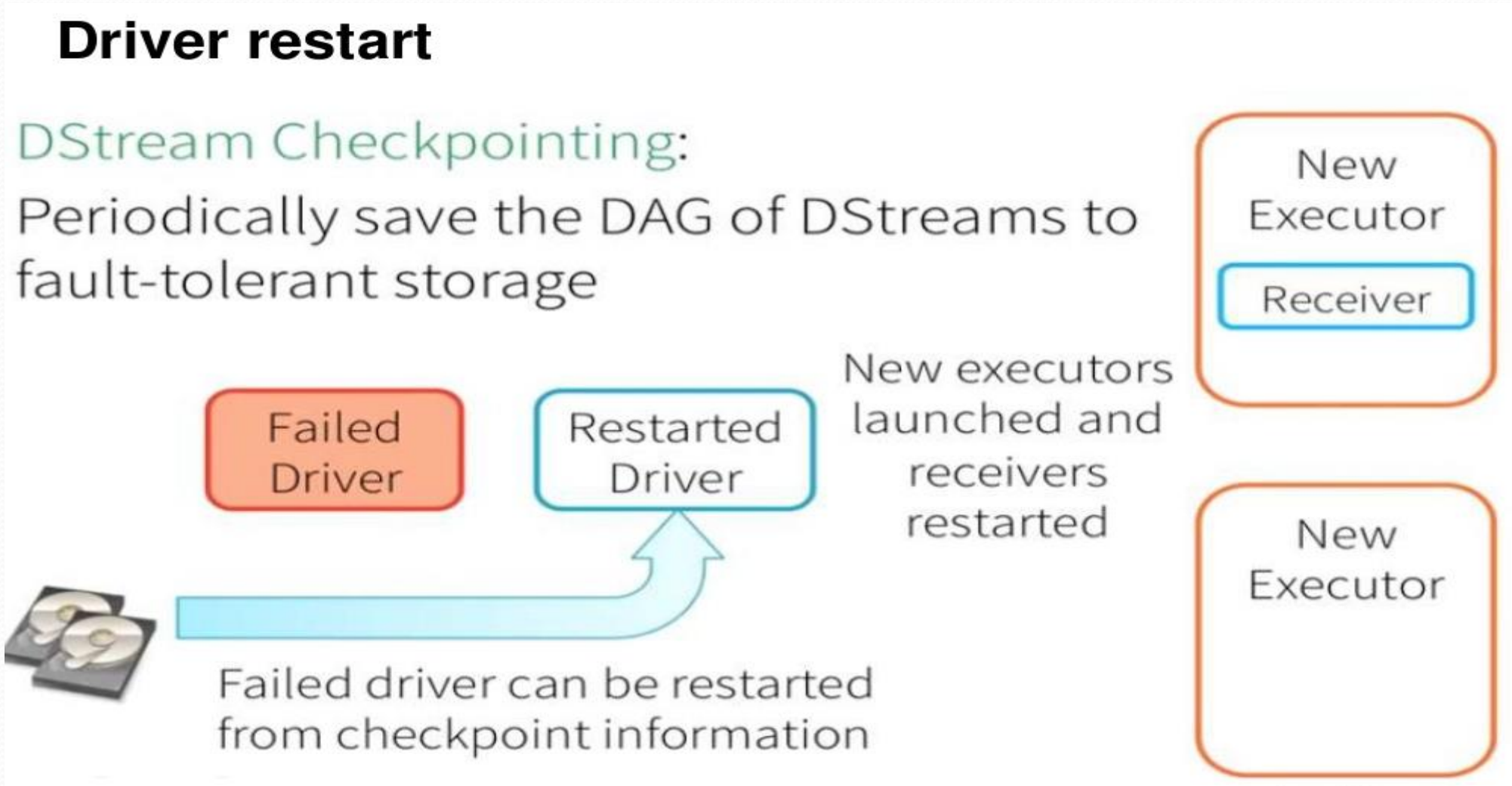
DStream Checkpointing:

Periodically save the DAG of DStreams to fault-tolerant storage





Recovering Driver – DStream Checkpointing



Driver restart – ToDO List

- Configure automatic driver restart
  - Spark Standalone
  - YARN

- Set Checkpoint in HDFS compatible file system

```
streamingContext.checkpiont(hdfsDirectory)
```

- Ensure the Code uses checkpoints for recovery

```
Def setupStreamingContext() : StreamingContext = {
    Val context = new StreamingContext(…)
    Val lines = KafkaUtils.createStream(…)
    …
    Context.checkpoint(hdfsDir)

Val context = StreamingContext.getOrCreate(hdfsDir, setupStreamingContext)
Context.start()
```

Configuring Automatic Driver Restart

Spark Standalone – Use spark-submit with “cluster” mode and “--supervise”

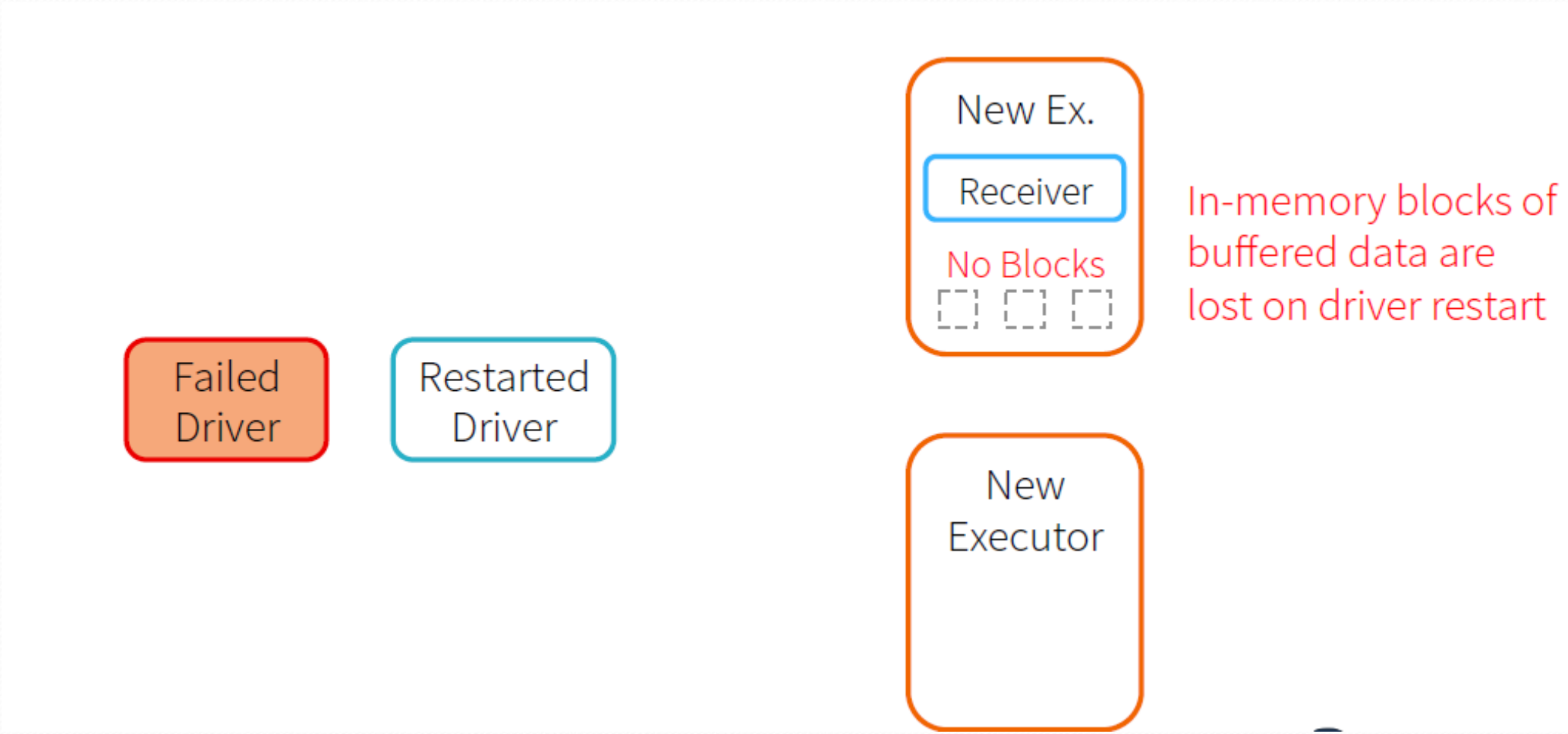
See http://spark.apache.org/docs/latest/spark-standalone.html

YARN – Use spark-submit in “cluster” mode

See YARN config “yarn.resourcemanager.am.max-attempts”

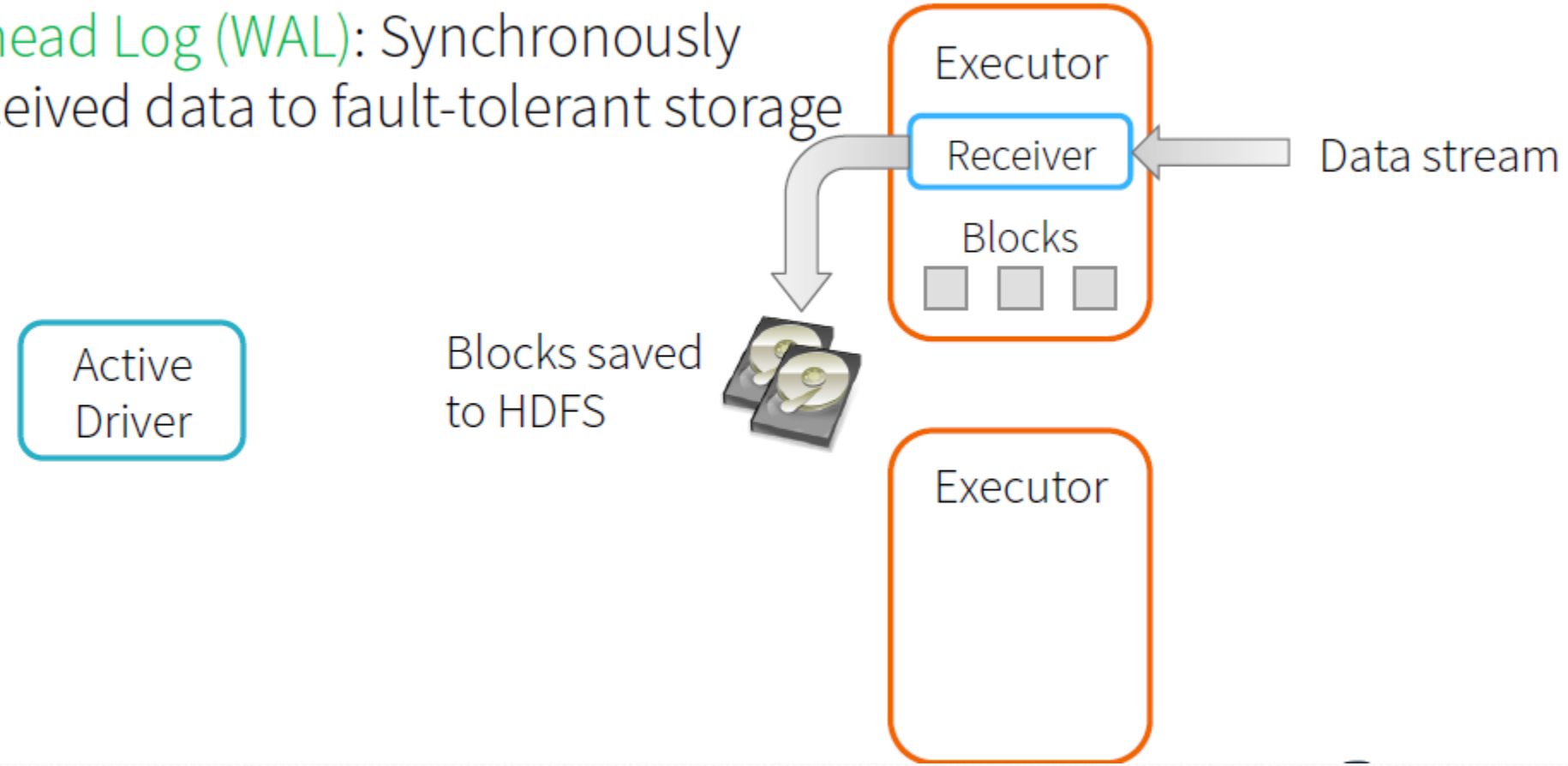
Mesos – Marathon can restart applications or use the “--supervise” flag.

Received blocks lost on Restart!



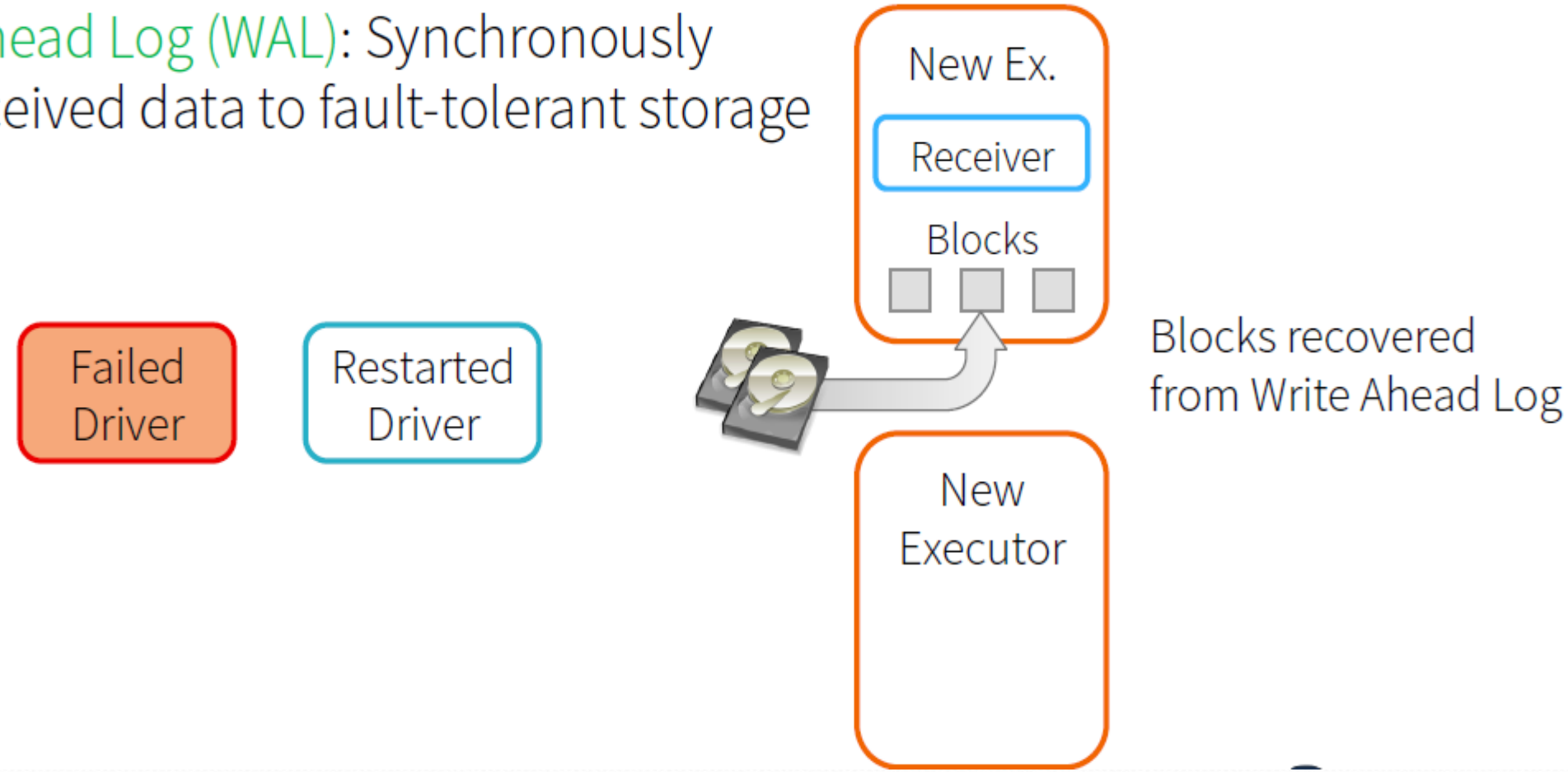
Recovering data with Write Ahead Logs

Write Ahead Log (WAL): Synchronously save received data to fault-tolerant storage



Recovering data with Write Ahead Logs

Write Ahead Log (WAL): Synchronously save received data to fault-tolerant storage



Configuration – Enabling WAL

1. Enable checkpointing, logs written in checkpoint directory

3. Enabled WAL in SparkConf configuration

```
sparkConf.set("spark.streaming.receiver.writeAheadLog.enable", "true")
```

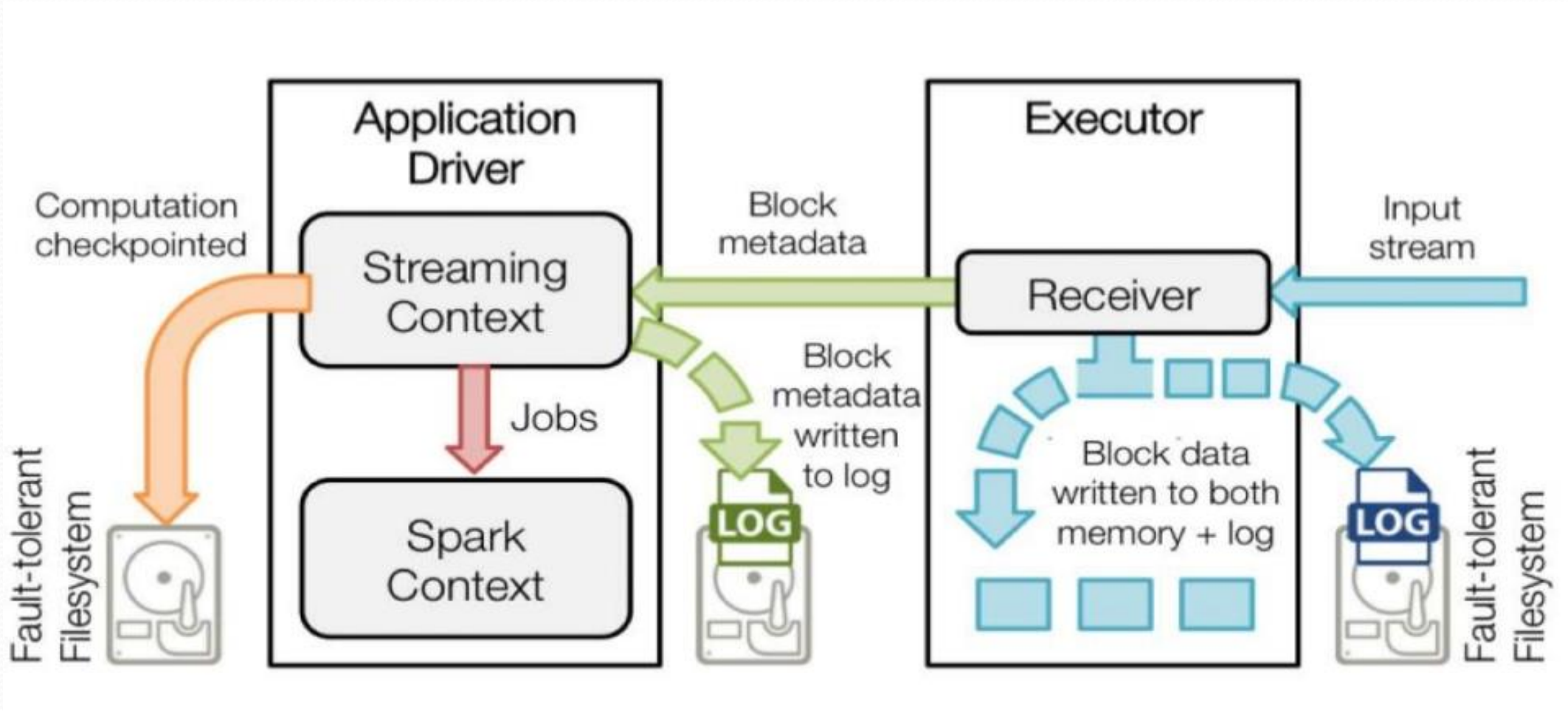
3. Receiver should also be *reliable*

   Acknowledge source only after data saved to WAL

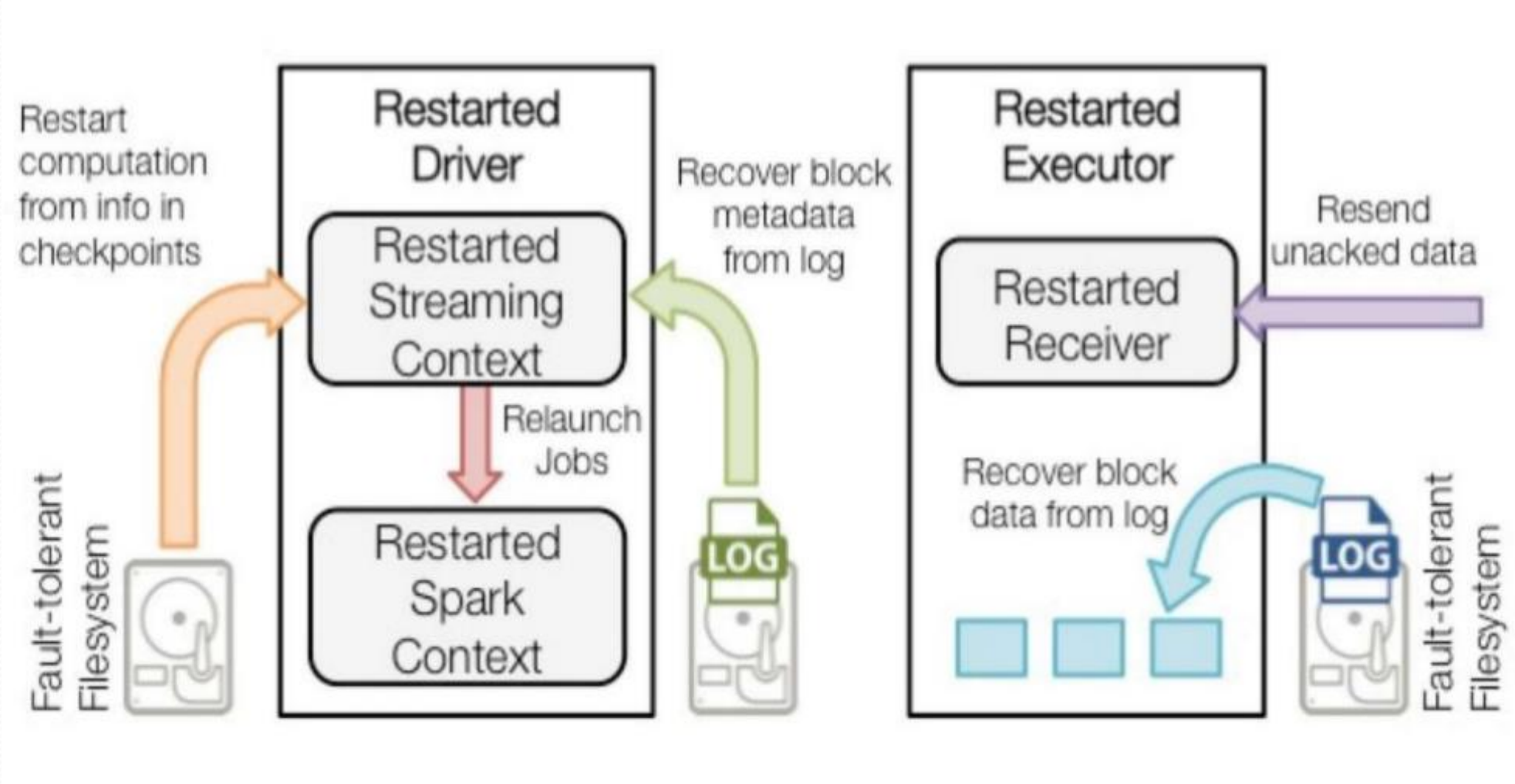
   Unacked data will be replayed from source by restarted receiver

5. Disable in-memory replication (already replicated by HDFS)

Normal Processing



Restarting Failed Driver



RDD Checkpointing

- Stateful stream processing can lead to long RDD lineages
- Long lineage = bad for fault-tolerance, too much recomputation
- RDD checkpointing saves RDD data to the fault-tolerant storage to limit lineage and recomputation