Introduction to

by Xuehai Chen

Content

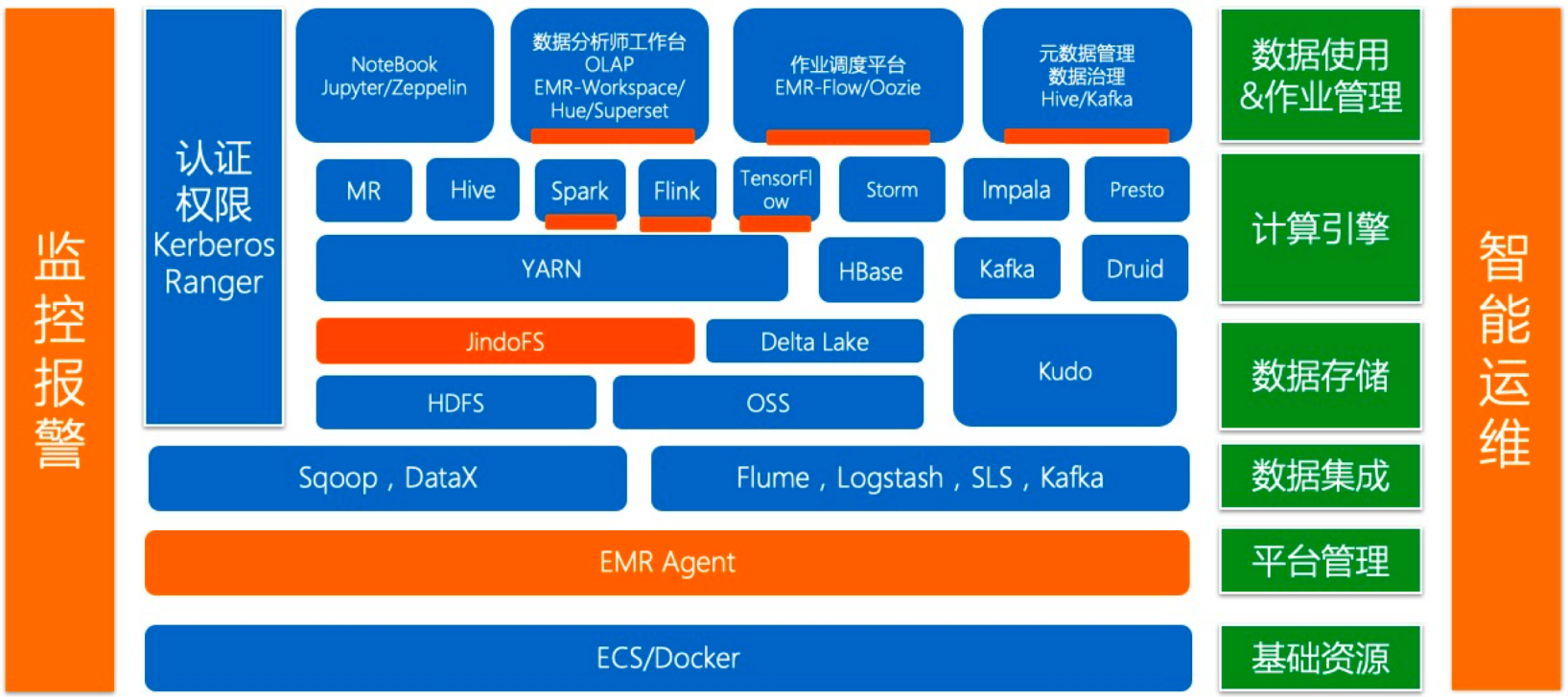
- Background
  - Hadoop ecosystem
  - Why we need Spark
  - What is Spark
  - Simple Example of Spark Application
- Spark Core
  - Terminology
  - General Architecture
  - Infrastructure
  - Data Structure
  - Scheduler Module
  - Storage Module
  - Execution Module
  - Tungsten
- Spark SQL
  - Data Structure
  - Catalyst & sql parse pipline
  - Code Generation
  - Join
- Deployment & Intergration
  - Storage Systems Intergration
  - Resource Manger Systems Intergration
  - SparkDriver Framework
- More
  - Beyond JVM
  - Beyond Batch Processing

BACK

Background - Hadoop ecosystem

- The Google File System(2003)
- MapReduce: Simplified Data Processing on Large Clusters(2004)
- Bigtable: A Distributed Storage System for Structured Data(2006)

- HDFS
- MapReduce
- HBase



Aliyun EMR Architecture

BACK

Background - Why we need Spark

Speed

- Run workloads 100x faster.

Ease of Use

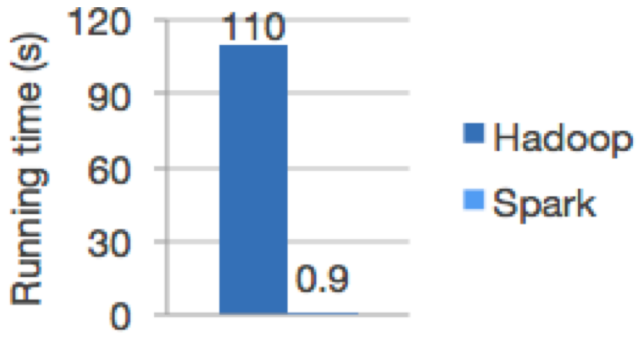
- Write applications quickly in Java, Scala, Python, R, and SQL.

Generality

- Combine SQL, streaming, and complex analytics.

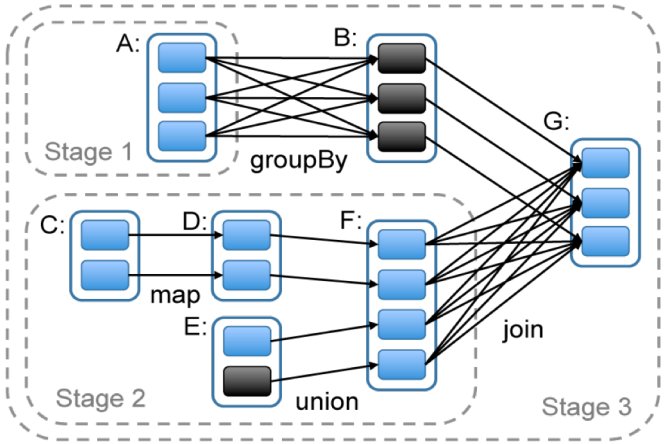
Runs Everywhere

- Spark runs on Hadoop, Apache Mesos, Kubernetes, standalone, or in the cloud. It can access diverse data sources.



Logistic regression in Hadoop and Spark

Speed



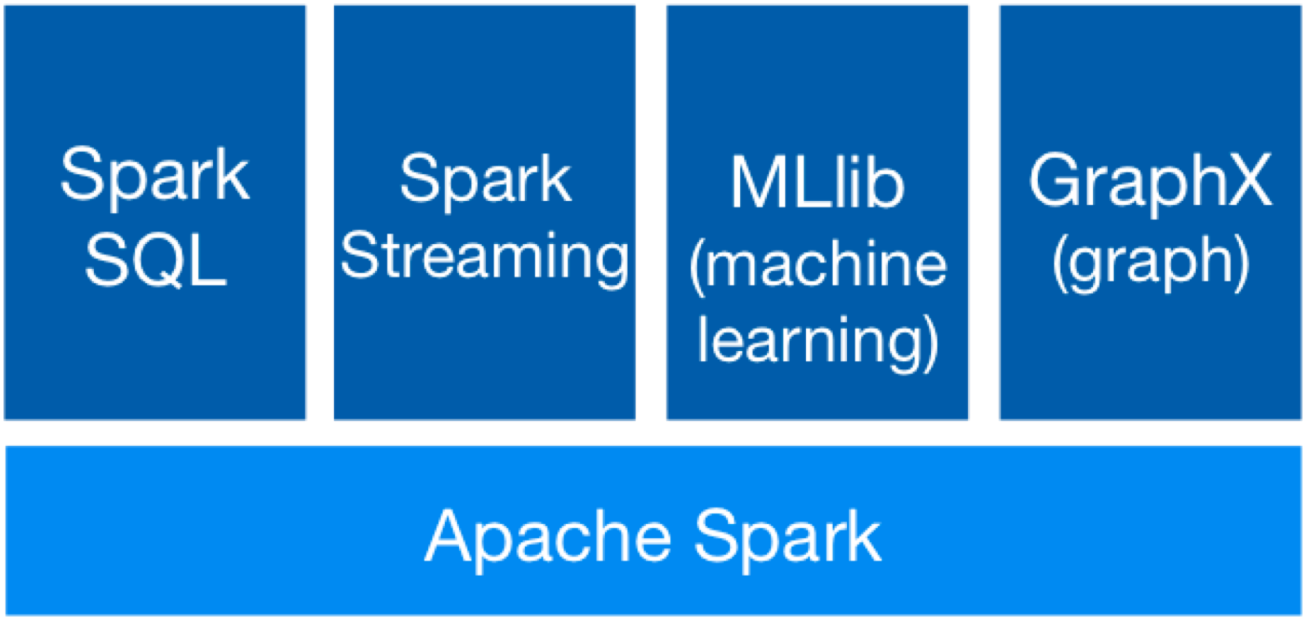
RDD transform

Fault Tolerance

BACK

Background - What is Spark

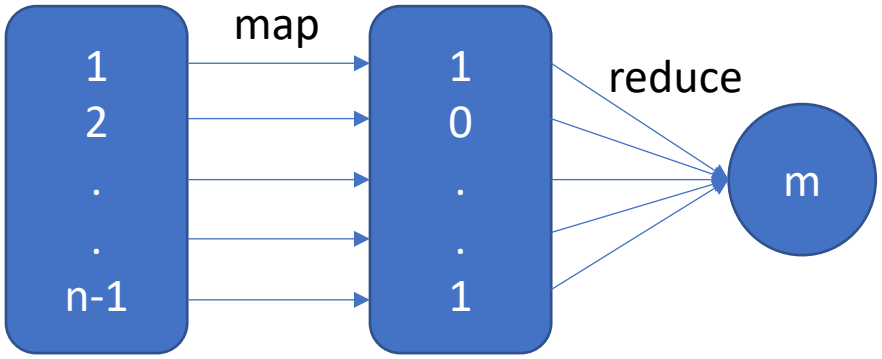
- Apache Spark is a fast and general-purpose cluster computing system.
- It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs.
- It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming.



BACK

Background – Simple Example of Spark Application

```scala
/** Computes an approximation to pi */
object SparkPi {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession
      .builder
      .appName( name = "Spark Pi")
      .getOrCreate()
    val slices = if (args.length > 0) args(0).toInt else 2
    val n = math.min(100000L * slices, Int.MaxValue).toInt // avoid overflow
    val count = spark.sparkContext.parallelize( seq = 1 until n, slices).map { i =>
      val x = random * 2 - 1
      val y = random * 2 - 1
      if (x*x + y*y <= 1) 1 else 0
    }.reduce(_ + _)
    println(s"Pi is roughly ${4.0 * count / (n - 1)}")
    spark.stop()
  }
}
```



BACK

Spark Core - Terminology

- ***SparkContext:*** Main entry point for Spark functionality. A SparkContext represents the connection to a Spark cluster, and can be used to create RDDs, accumulators and broadcast variables on that cluster.

- ***Driver:*** The process running the main() function of the application and creating the SparkContext.

- ***Executor:*** A process launched for an application on a worker node, that runs tasks and keeps data in memory or disk storage across them. Each application has its own executors.

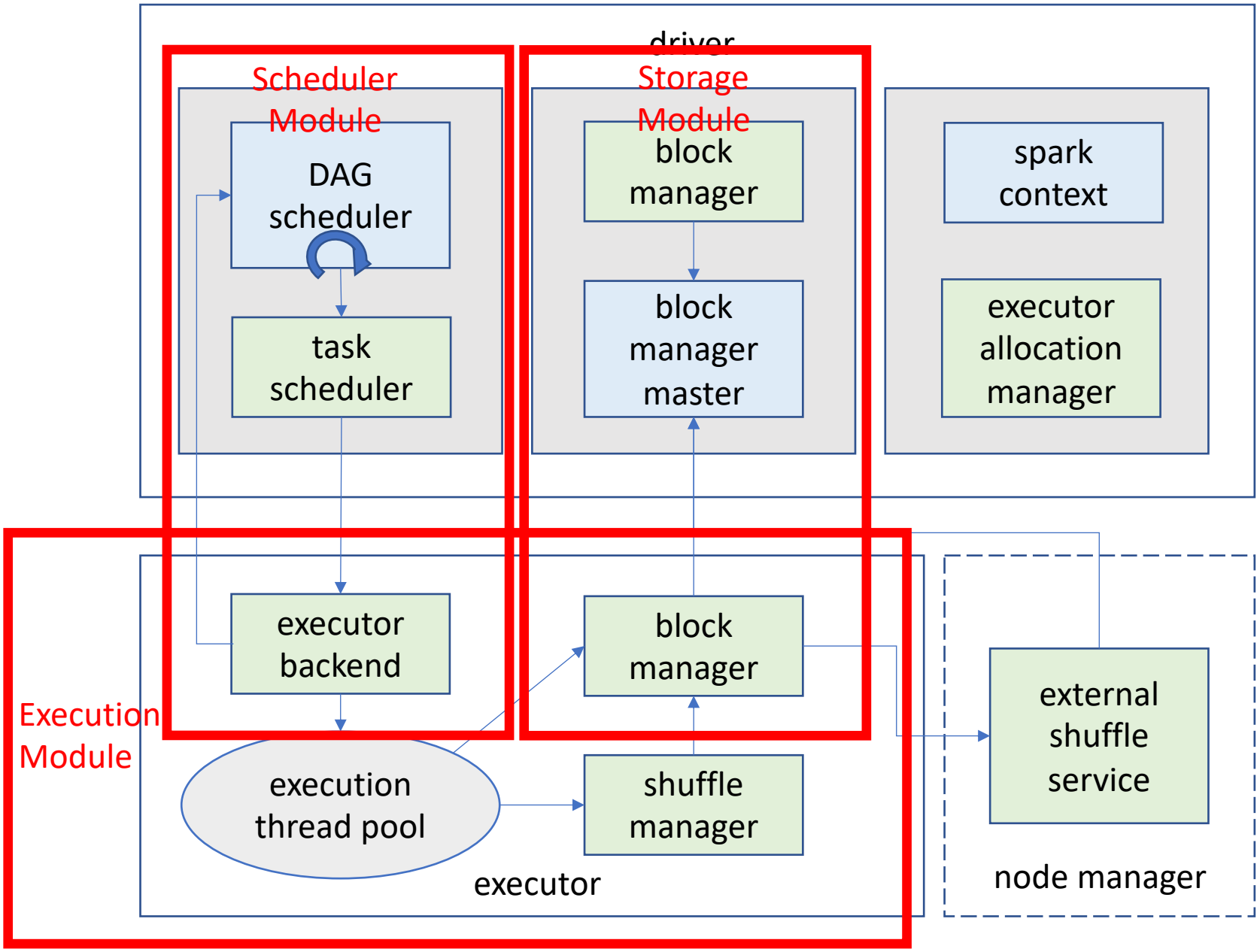
- ***Job:*** A parallel computation consisting of multiple tasks that gets spawned in response to a Spark action (e.g. save, collect).

- ***Stage:*** Each job gets divided into smaller sets of tasks called stages that depend on each other (similar to the map and reduce stages in MapReduce).

- ***Task:*** A unit of work that will be sent to one executor.

BACK

Spark Core - General Architecture



BACK

Spark Core - Infrastructure

Network

- 1.x: **Akka** (Actor model programming framework)
- 2.0 & above: **Netty** (low-level high-performance NIO network programming framework)

Listener Bus

- SparkListenerBus
- StreamingQueryListenerBus

Metrics

- Source (schedulers, blockmanagers, workers…)
- Sink (slf4j, jmx, console…)

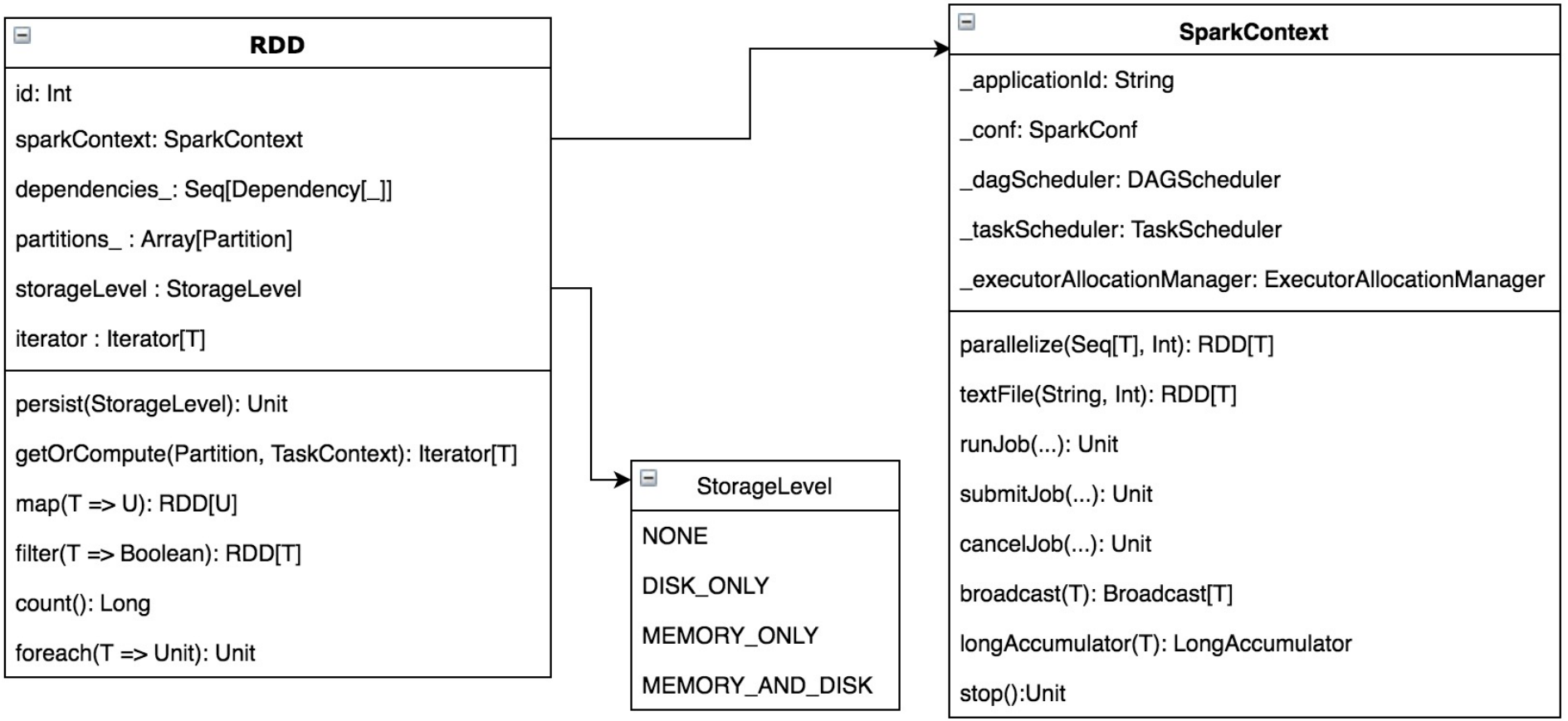
Monitor UI

- web servlet container: **Jetty**

BACK

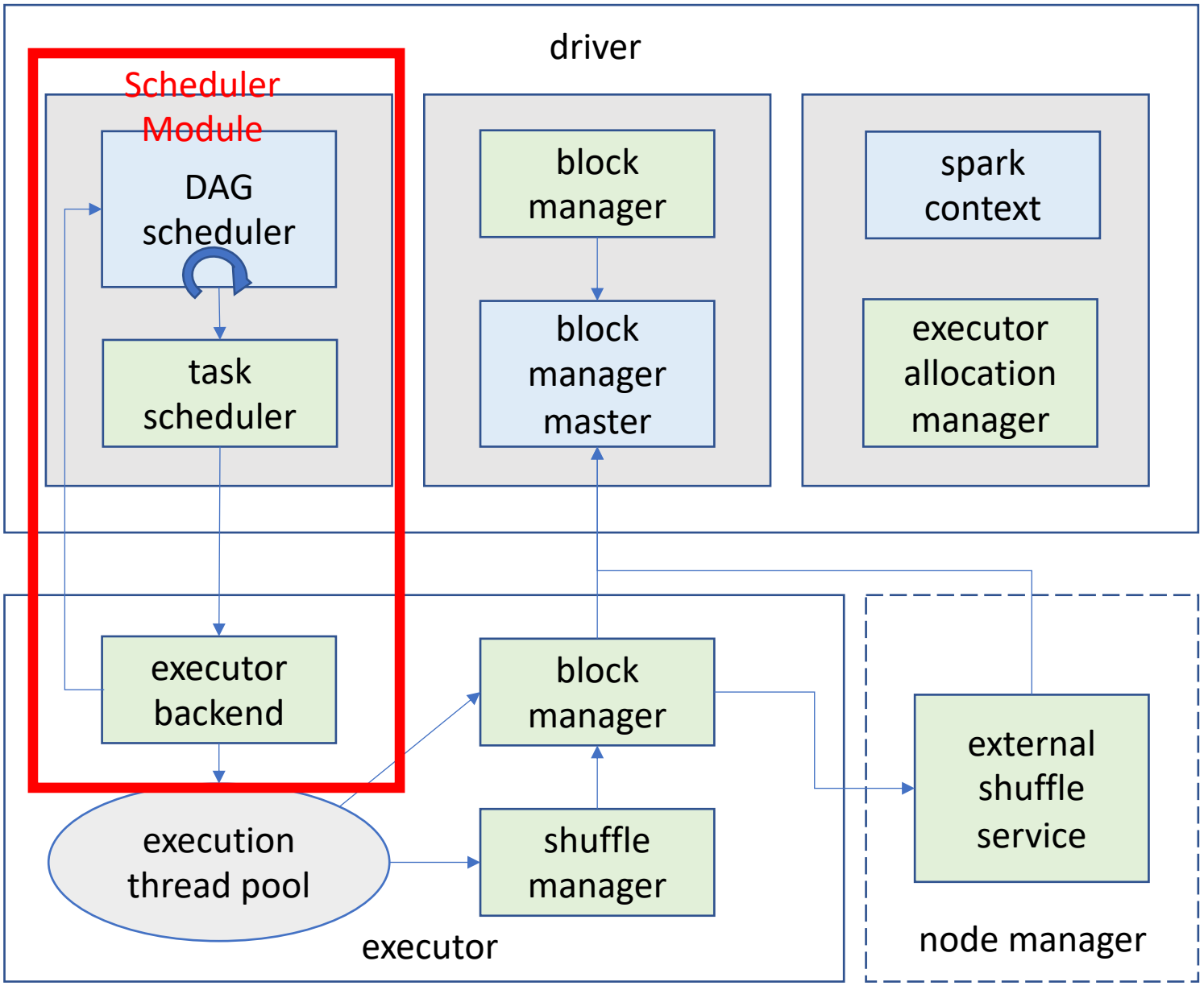
Spark Core – Data Structure

- RDD(Resilient Distributed Datasets)
- SparkContext



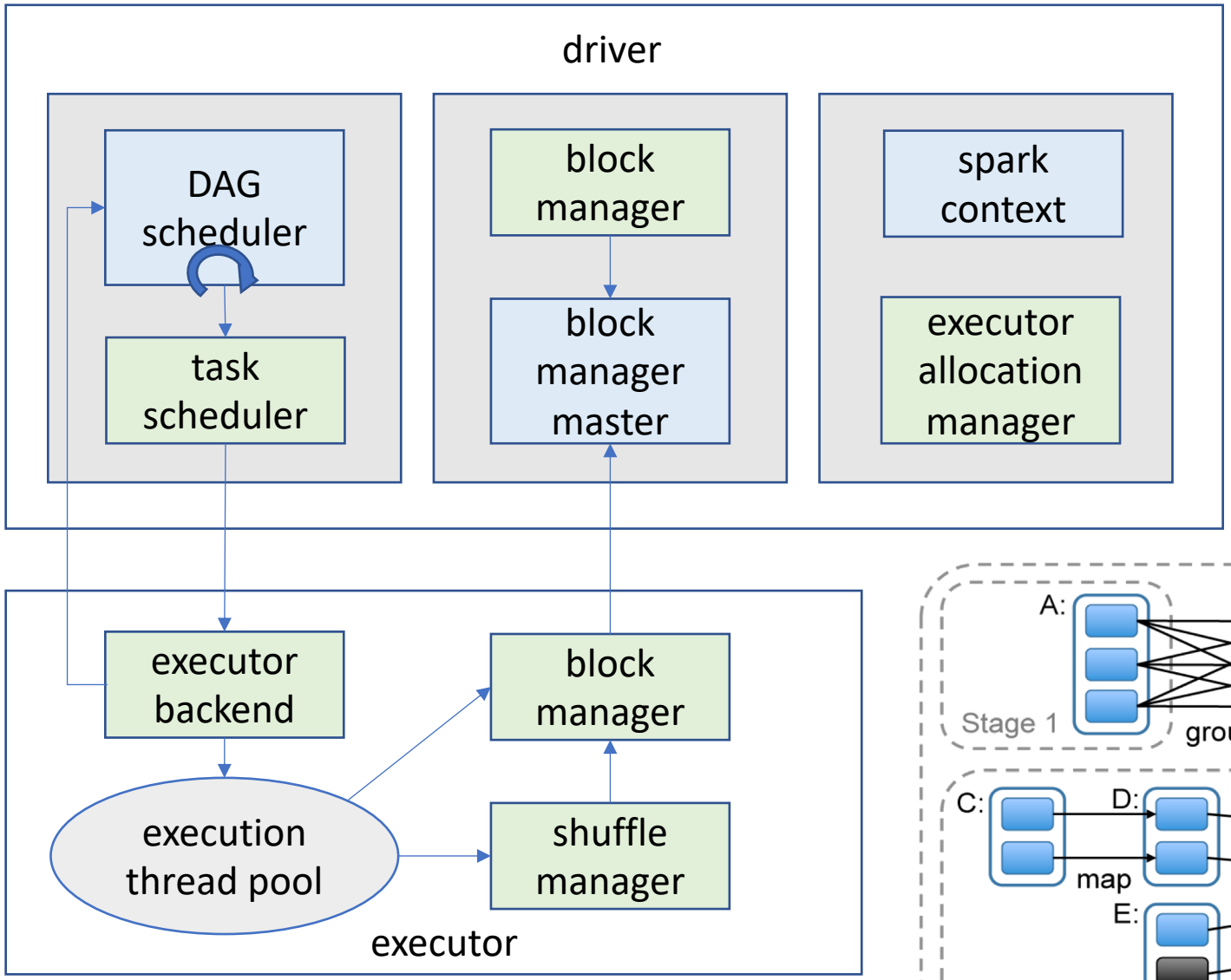
BACK

Spark Core – Scheduler Module



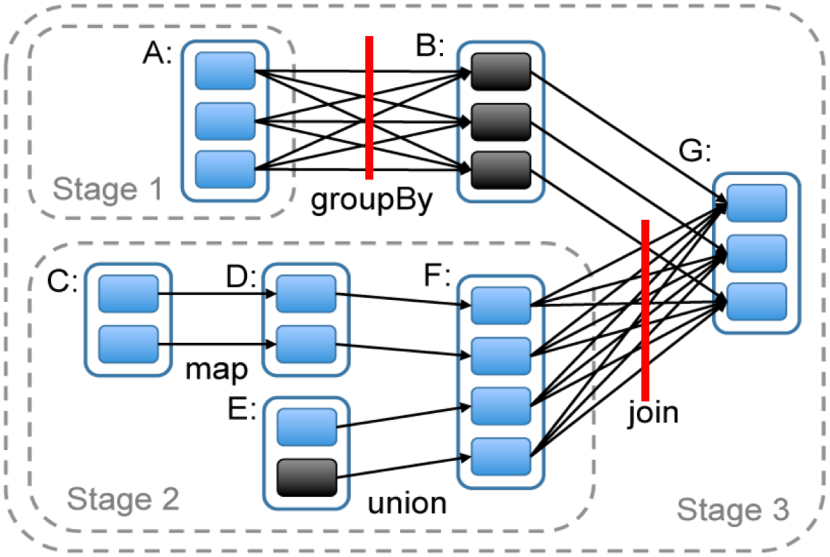
BACK

Spark Core – Scheduler Module



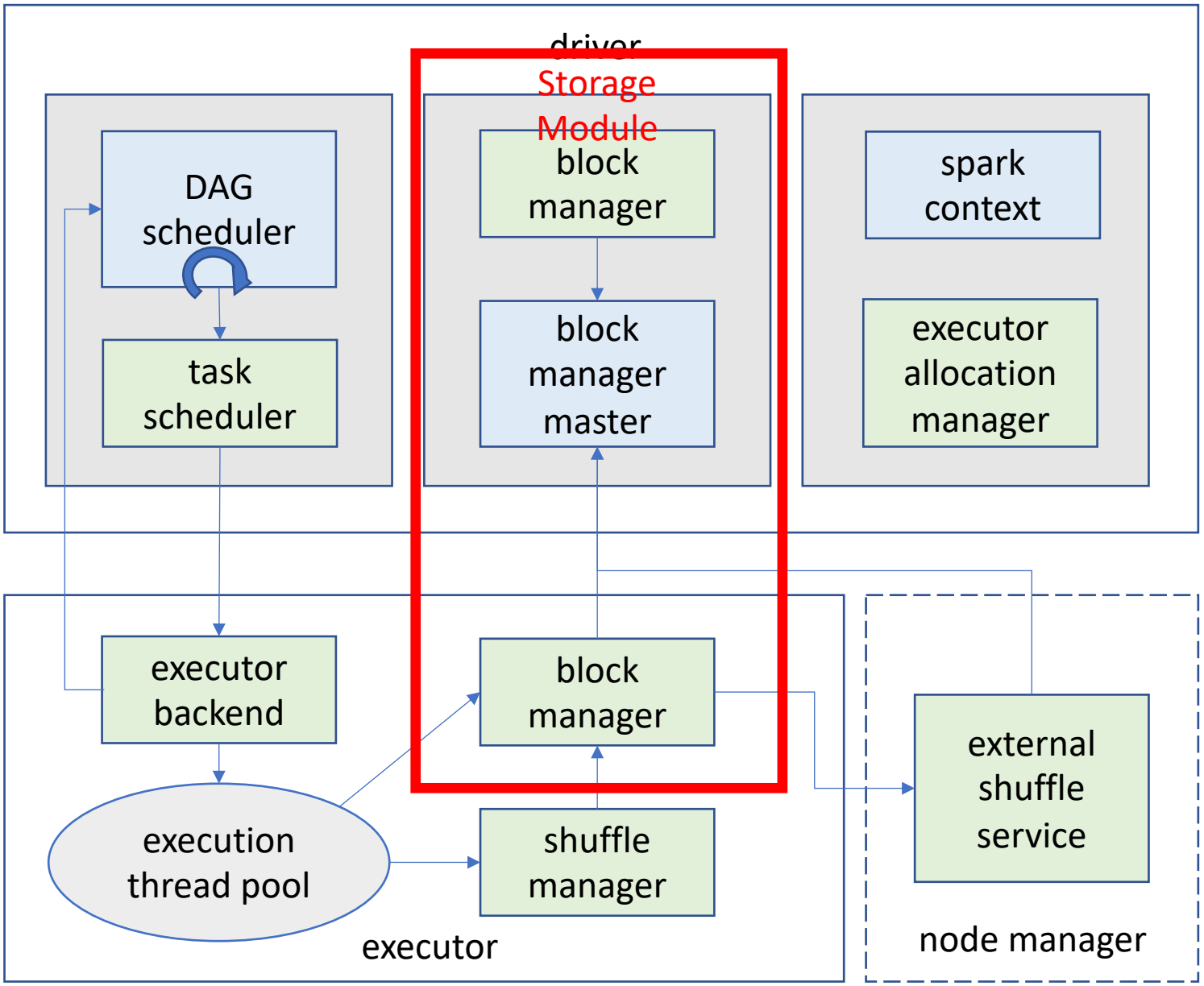
Shuffle: The shuffle is Spark's mechanism for re-distributing data so that it's grouped differently across partitions.

Wide Dependency: transformation have input partitions contributing to many output partitions.



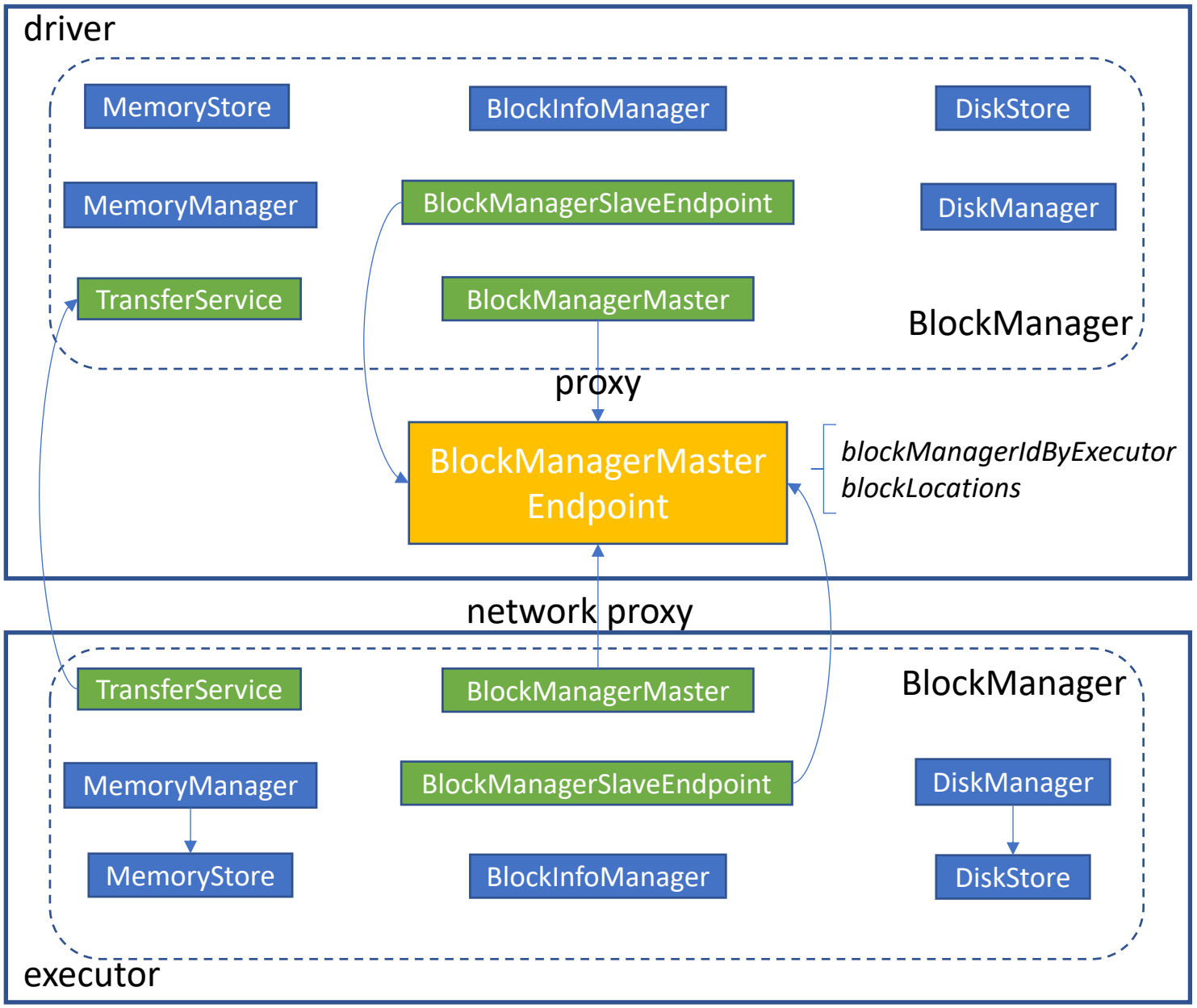
BACK

Spark Core – Storage Module



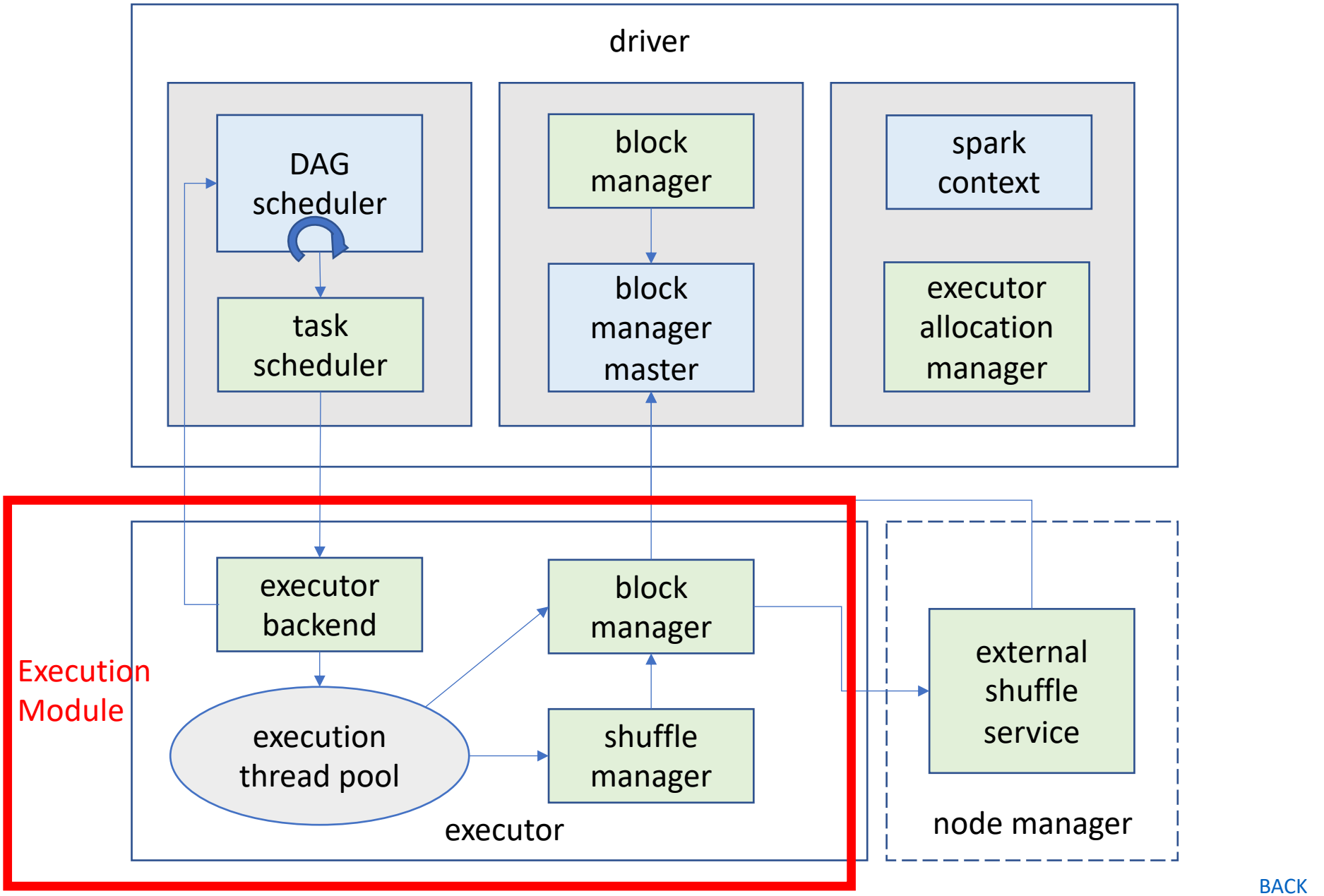
BACK

Spark Core – Storage Module



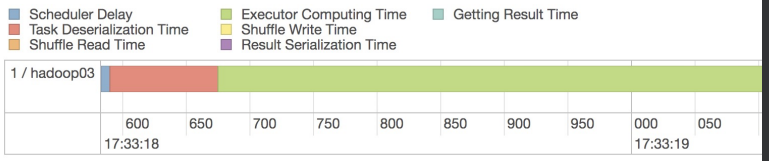
BACK

Spark Core – Execution Module



BACK

Spark Core – Execution Module



Summary Metrics for 1 Completed Tasks

Metric	Min	25th percentile
Duration	1 s	1 s
Scheduler Delay	7 ms	7 ms
Task Deserialization Time	85 ms	85 ms
GC Time	0.2 s	0.2 s
Result Serialization Time	1 ms	1 ms
Getting Result Time	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B
Shuffle Read Blocked Time	0 ms	0 ms
Shuffle Read Size / Records	216.0 B / 1	216.0 B / 1
Shuffle Remote Reads	216.0 B	216.0 B

```
/.../
Executor.taskDeserializationProps.set(taskDescription

updateDependencies(taskDescription.addedFiles, taskDe
task = ser.deserialize[Task[Any]](...)
task.localProperties = taskDescription.properties
task.setTaskMemoryManager(taskMemoryManager)

// Run the actual task and measure its runtime.
taskStartTimeNs = System.nanoTime()
taskStartCpu = if (threadMXBean.isCurrentThreadCpuTim
var threwException = true
val value = Utils.tryWithSafeFinally {
  val res = task.run(
    taskAttemptId = taskId,
    attemptNumber = taskDescription.attemptNumber,
    metricsSystem = env.metricsSystem,
    resources = taskDescription.resources)
  threwException = false
  res
} (...)
```

```
/**...*/
private[spark] abstract class Task[T](
    val stageId: Int,
    val stageAttemptId: Int,
    val partitionId: Int,
    @transient var localProperties: Properties = new Properties,
    // The default value is only used in tests.
    serializedTaskMetrics: Array[Byte] =
      SparkEnv.get.closureSerializer.newInstance().serialize(TaskMetrics.registered).array(),
    val jobId: Option[Int] = None,
    val appId: Option[String] = None,
    val appAttemptId: Option[String] = None,
    val isBarrier: Boolean = false) extends Serializable {

  @transient lazy val metrics: TaskMetrics =
    SparkEnv.get.closureSerializer.newInstance().deserialize(ByteBuffer.wrap(serializedTaskMetrics))

  /**...*/
  final def run(
      taskAttemptId: Long,
      attemptNumber: Int,
      metricsSystem: MetricsSystem,
      resources: Map[String, ResourceInformation]): T = {
    SparkEnv.get.blockManager.registerTask(taskAttemptId)
    ...
    val taskContext = new TaskContextImpl(...)

    context = if (isBarrier) {...} else {...}

    InputFileBlockHolder.initialize()
    TaskContext.setTaskContext(context)
    taskThread = Thread.currentThread()

    if (_reasonIfKilled != null) {...}

    new CallerContext(...).setCurrentContext()

    try {
      runTask(context)
    } catch {
      case e: Throwable =>
        // Catch all errors; run task failure callbacks, and rethrow the exception.
        ...
    } finally {
      try {
        /.../
        context.markTaskCompleted(None)
      } finally {
        try {
          Utils.tryLogNonFatalError (...)
        } finally {
          /.../
          TaskContext.unset()
          InputFileBlockHolder.unset()
        }
      }
    }
  }
```

Spark Core – Tungsten

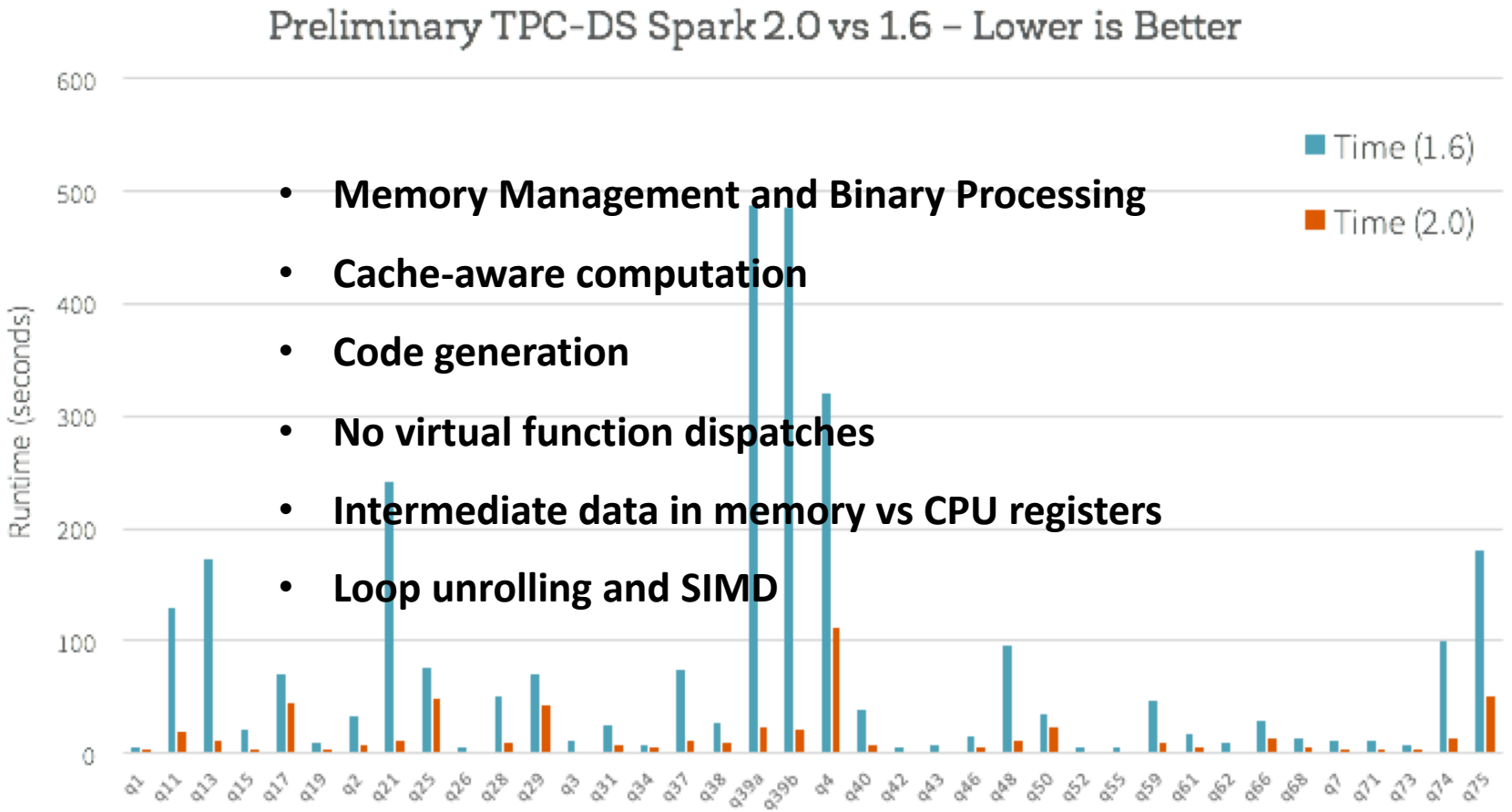
Spark is already pretty fast, but can we push the boundary and make Spark 10X faster?

Cost Per Row (in nanoseconds, single thread)

primitive	Spark 1.6	Spark 2.0
filter	15ns	1.1ns
sum w/o group	14ns	0.9ns
sum w/ group	79ns	10.7ns
hash join	115ns	4.0ns
sort (8-bit entropy)	620ns	5.3ns
sort (64-bit entropy)	620ns	40ns
sort-merge join	750ns	700ns

BACK

Spark Core – Tungsten



- **Memory Management and Binary Processing**
- **Cache-aware computation**
- **Code generation**
- **No virtual function dispatches**
- **Intermediate data in memory vs CPU registers**
- **Loop unrolling and SIMD**

BACK

Spark Core – Tungsten

- **Memory Management and Binary Processing**

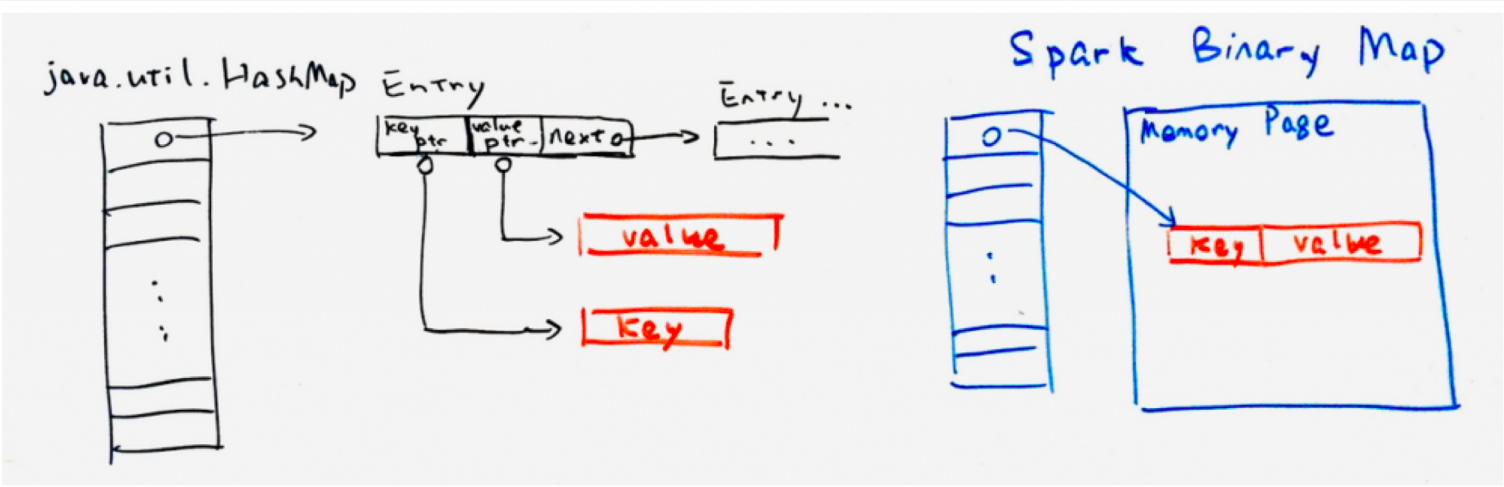
```
java.lang.String object internals:
OFFSET  SIZE   TYPE DESCRIPTION                    VALUE
     0     4        (object header)                ...
     4     4        (object header)                ...
     8     4        (object header)                ...
    12     4 char[] String.value                   []
    16     4    int String.hash                    0
    20     4    int String.hash32                  0
Instance size: 24 bytes (reported by Instrumentation API)
```

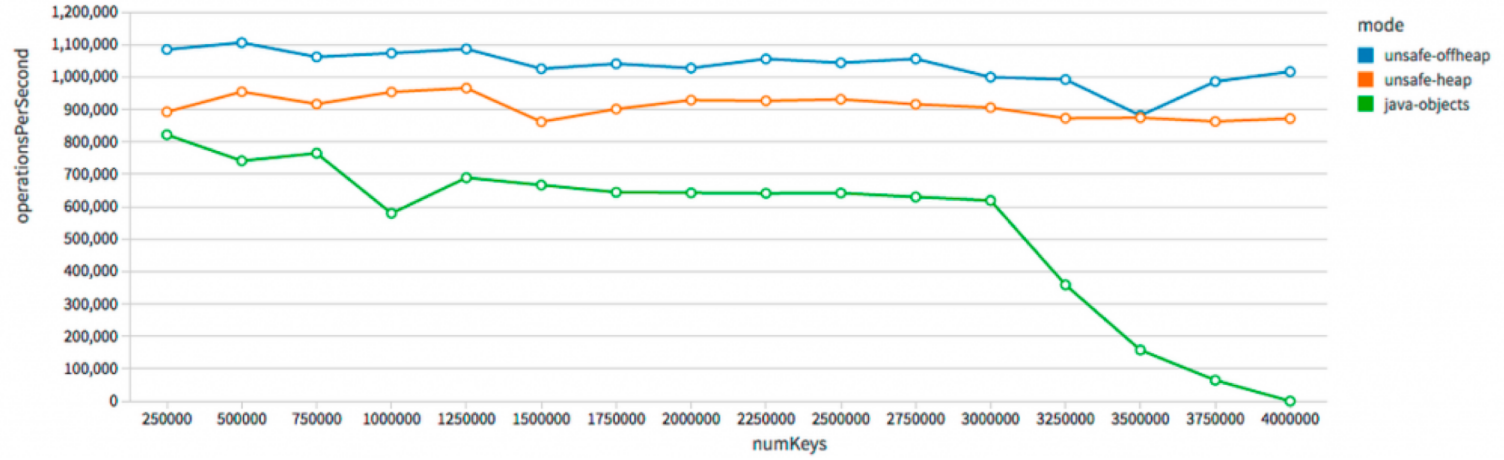
```scala
/**
 * Allocates memory for use by Unsafe/Tungsten code.
 */
private[memory] final val tungstenMemoryAllocator: MemoryAllocator = {
  tungstenMemoryMode match {
    case MemoryMode.ON_HEAP => MemoryAllocator.HEAP
    case MemoryMode.OFF_HEAP => MemoryAllocator.UNSAFE
  }
}
```

BACK

Spark Core – Tungsten

- **Memory Management and Binary Processing**

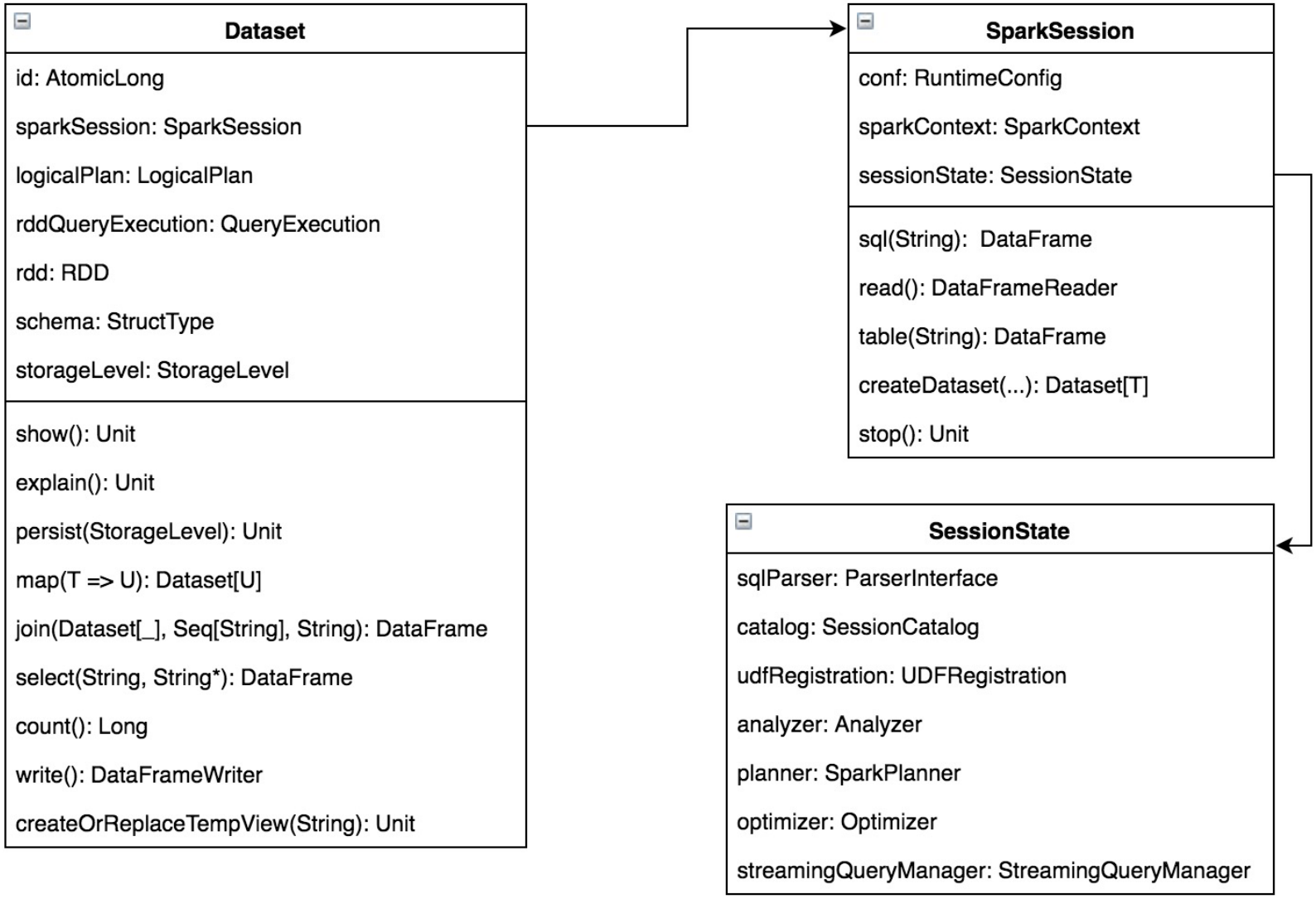




BACK

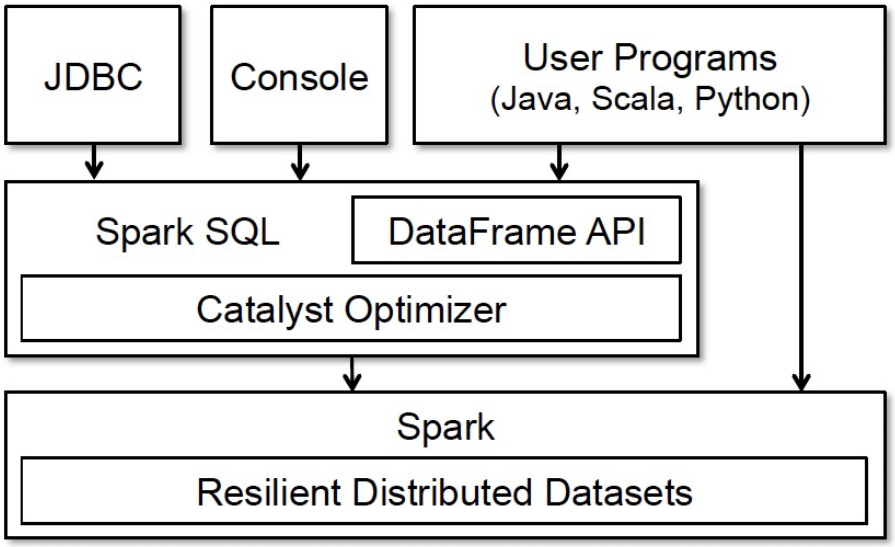
Spark SQL – Data Structure

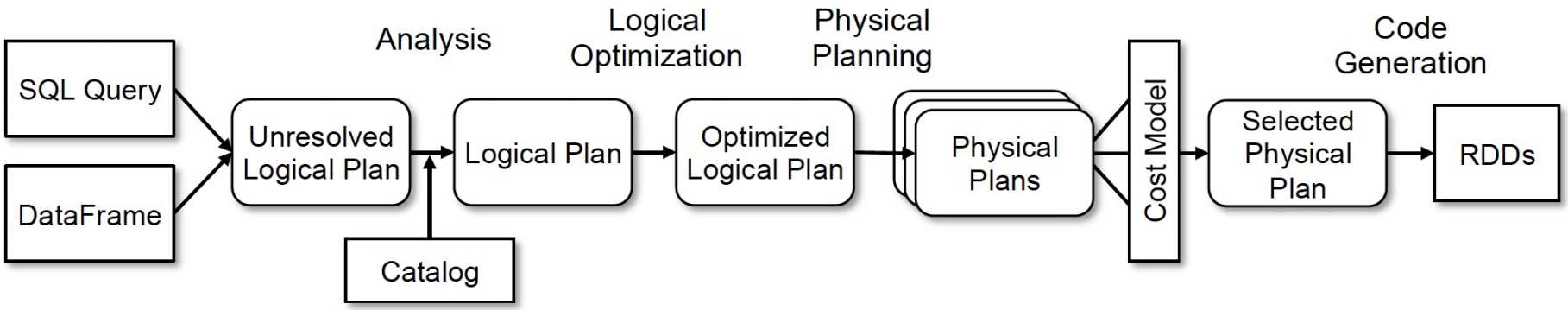
- Dataset[Row]/DataFrame
- SparkSession



BACK

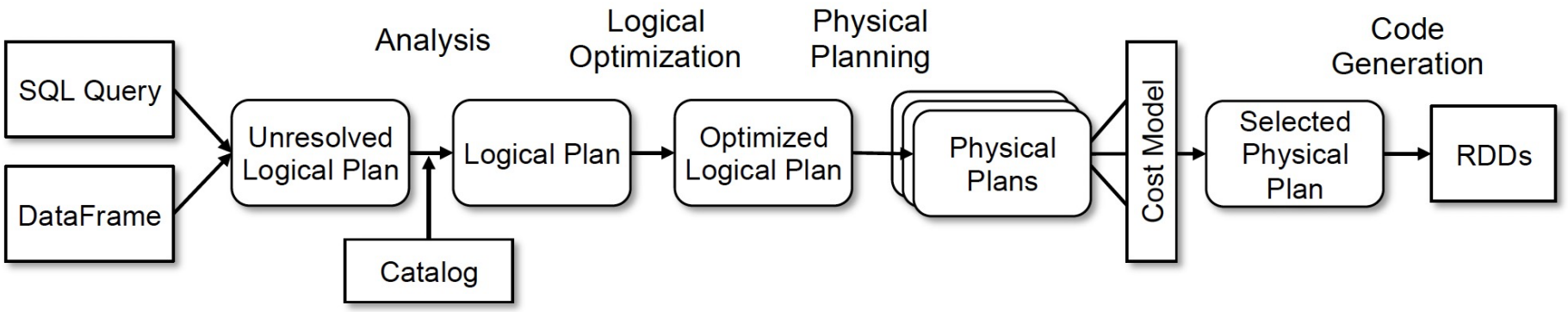
Spark SQL – Catalyst & sql parse pipeline





BACK

Spark SQL – Catalyst & sql parse pipeline



RBO (rule-based optimization)

- push down predicates
- reuse exchange
- column pruning
- …

CBO (cost-based optimization)

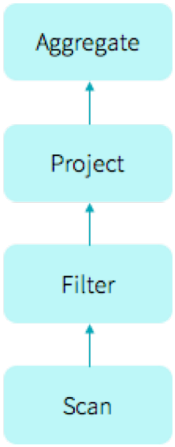
- table/column statistics
- dynamic programming

BACK

Spark SQL – Code Generation

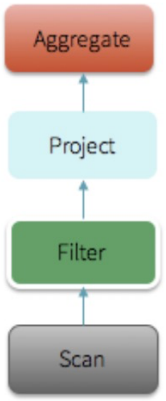
The Past: Volcano Iterator Model

```
select count(*) from store_sales
where ss_item_sk = 1000
```




```
class Filter(child: Operator, predicate: (Row => Boolean))
  extends Operator {
  def next(): Row = {
    var current = child.next()
    while (current != null && !predicate(current)) {
      current = child.next()
    }
    return current
  }
}
```

Current: Whole-stage Code Generation



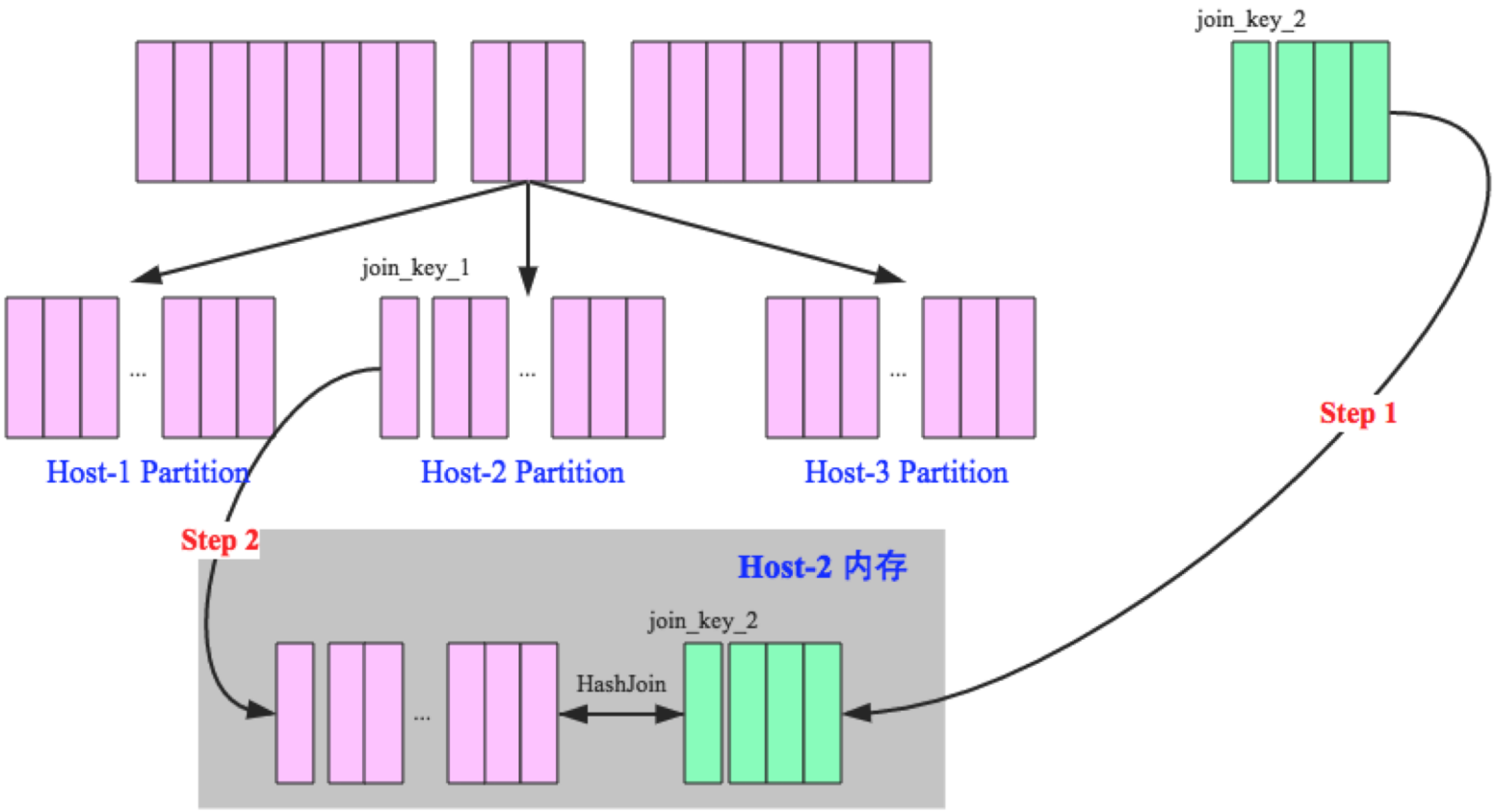
```
long count = 0;
for (ss_item_sk in store_sales) {
  if (ss_item_sk == 1000) {
    count += 1;
  }
}
```

- **No virtual function dispatches**
- **Intermediate data in memory vs CPU registers**
- **Loop unrolling and SIMD**

BACK

Spark SQL - Join

Implementations of Join

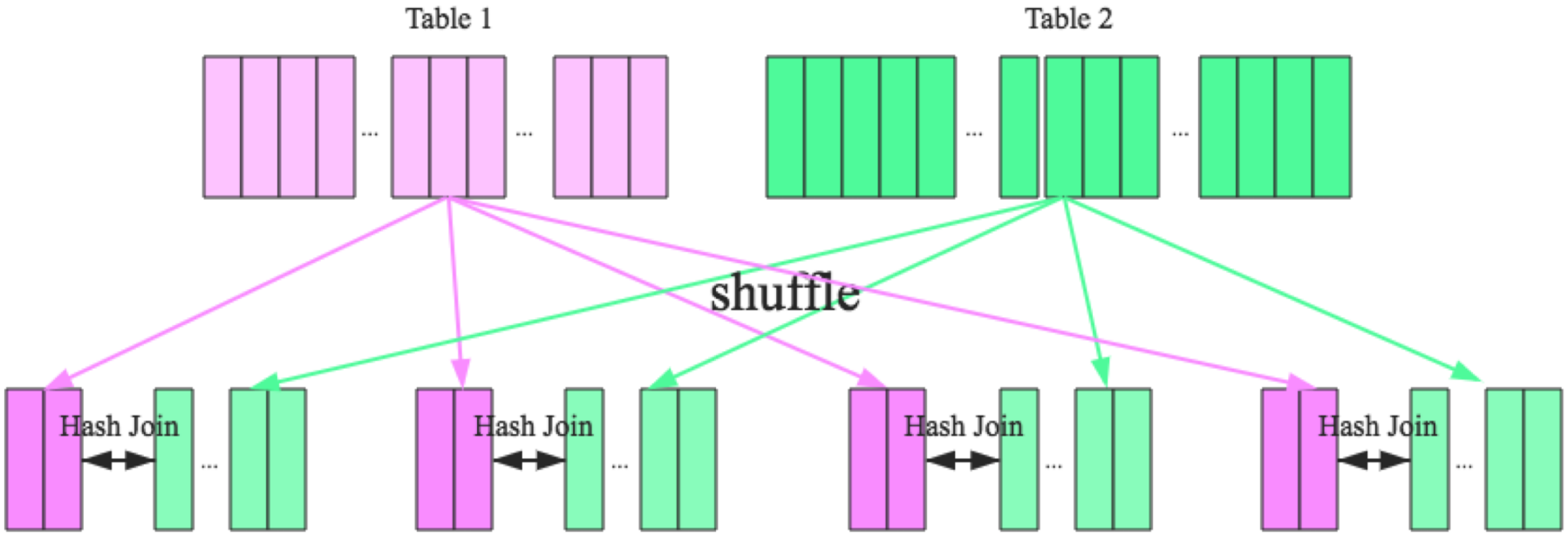


broadcast hash join

BACK

Spark SQL - Join

Implementations of Join

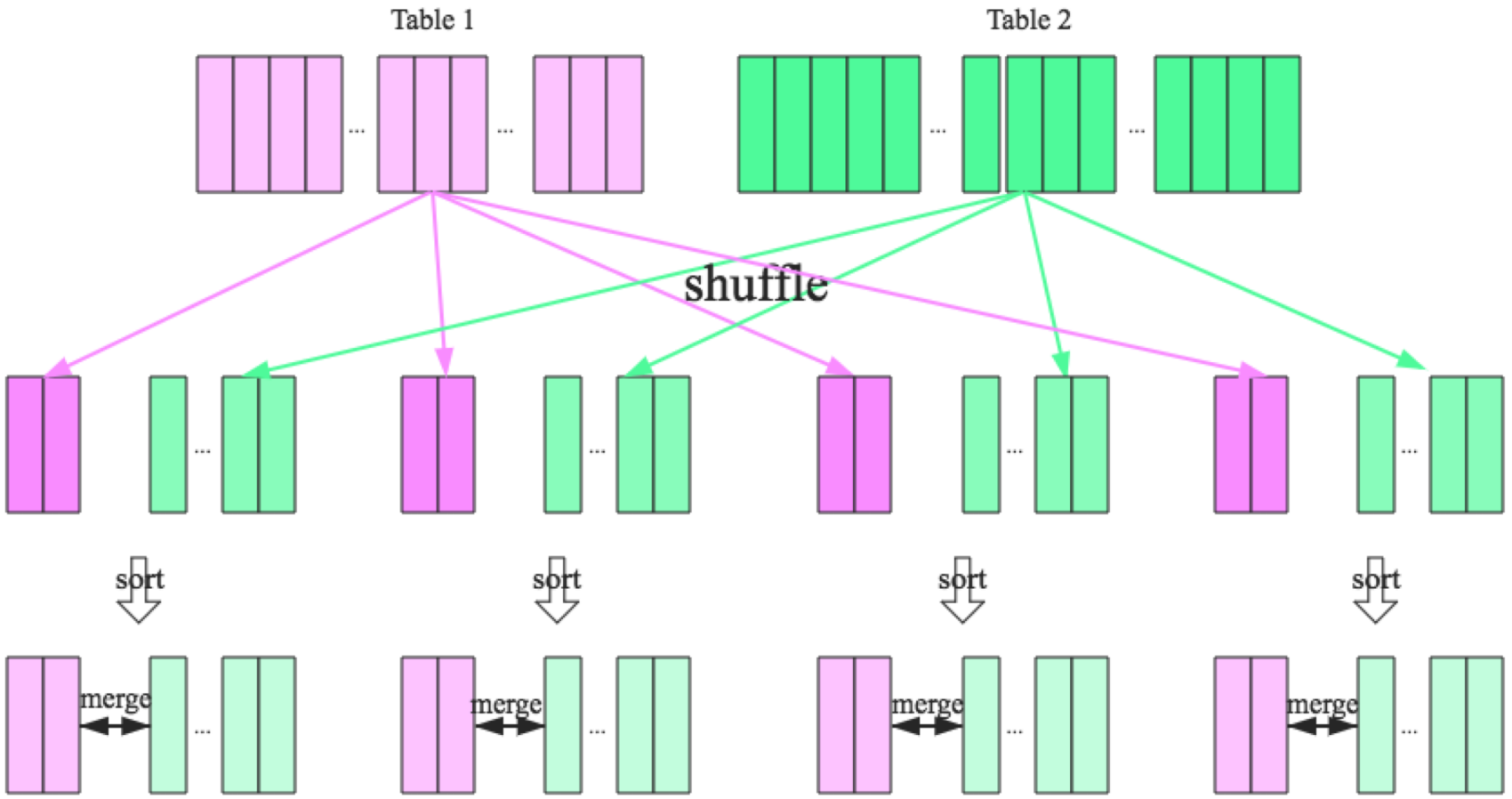


shuffle hash join

BACK

Spark SQL - Join

Implementations of Join



sort merge join

BACK

Deployment & Intergration – Storage Systems

Distributed File Systems

- HDFS
- s3
- oss
- JindoFS

```
val dataset = sparkSession.read
  .option("header", "true")
  .option("timestampFormat", "yyyy/MM/dd HH:mm:ss")
  .csv( path = "hdfs://hadoop01:8020/input/file.csv")
```

Tables

- Hive
- Hbase
- Kudu

```
sparkSession.catalog.setCurrentDatabase("cdp_test")

val dataset = sparkSession.table( tableName = "dynamic_list_member")
```

```
val kuduContext = new KuduContext(KUDU_MASTER, sparkSession.sparkContext)
val kuduTableList: java.util.List[String] = kuduContext.syncClient.getTablesList.getTablesList

kuduTableList.toList.filter(_.contains(KUDU_DATABASE)).foreach(it => {
  sparkSession.read.format( source = "org.apache.kudu.spark.kudu")
    .option("kudu.master", KUDU_MASTER)
    .option("kudu.table", it)
    .load.createOrReplaceTempView(it.substring(it.lastIndexOf( str = ".") + 1))
})
```

Sources/Sinks

- kafka
- file
- console(debugging)
- memory(debugging)

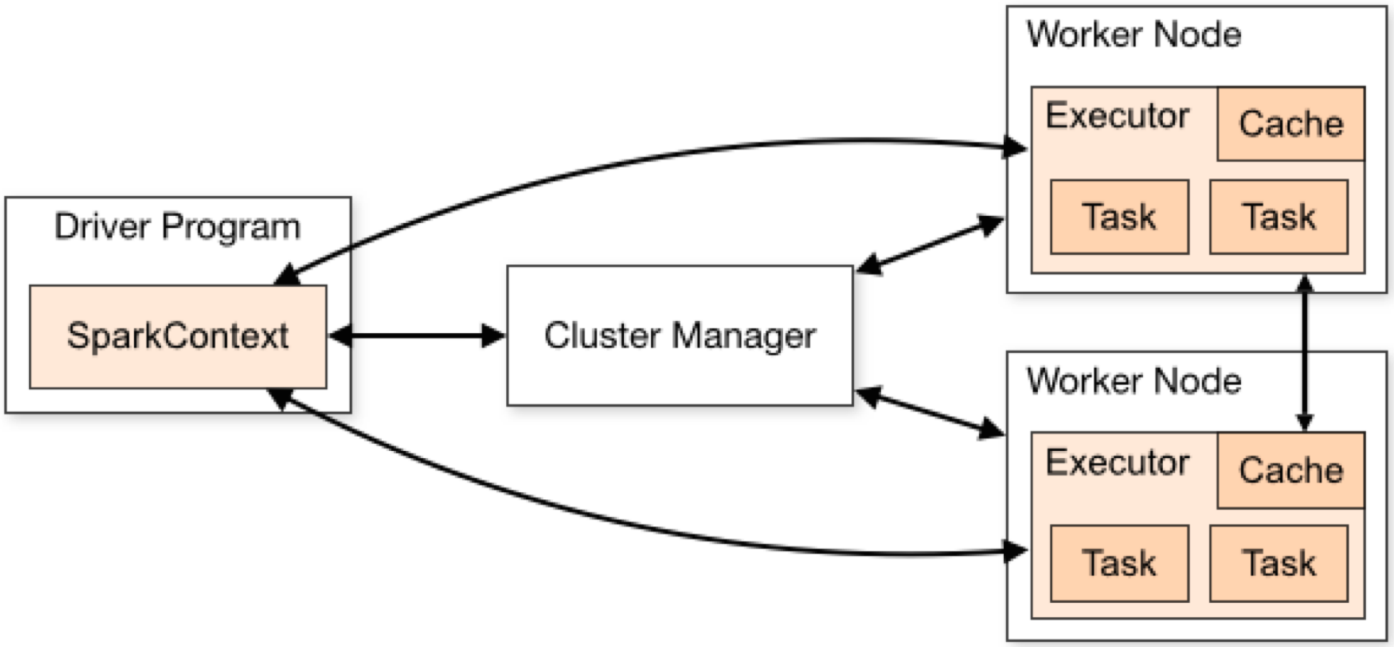
```
val scan = new Scan()
scan.setRowPrefixFilter(tenantId.toString.getBytes())
scan.addFamily(columnFamily)

val rdd = hbaseContext.hbaseRDD(
  TableName.valueOf( name = s"$namespace:customer_profile"),
  scan).map(it =>
  Row(new String(it._1.get),
    it._2.getValueAsByteBuffer(columnFamily, "statistics".getBytes)
  ))

val schema = StructType(Seq(
  StructField("customer_id", LongType),
  StructField("value", DoubleType)
))
val dataset = sparkSession.createDataFrame(rdd, schema)
```

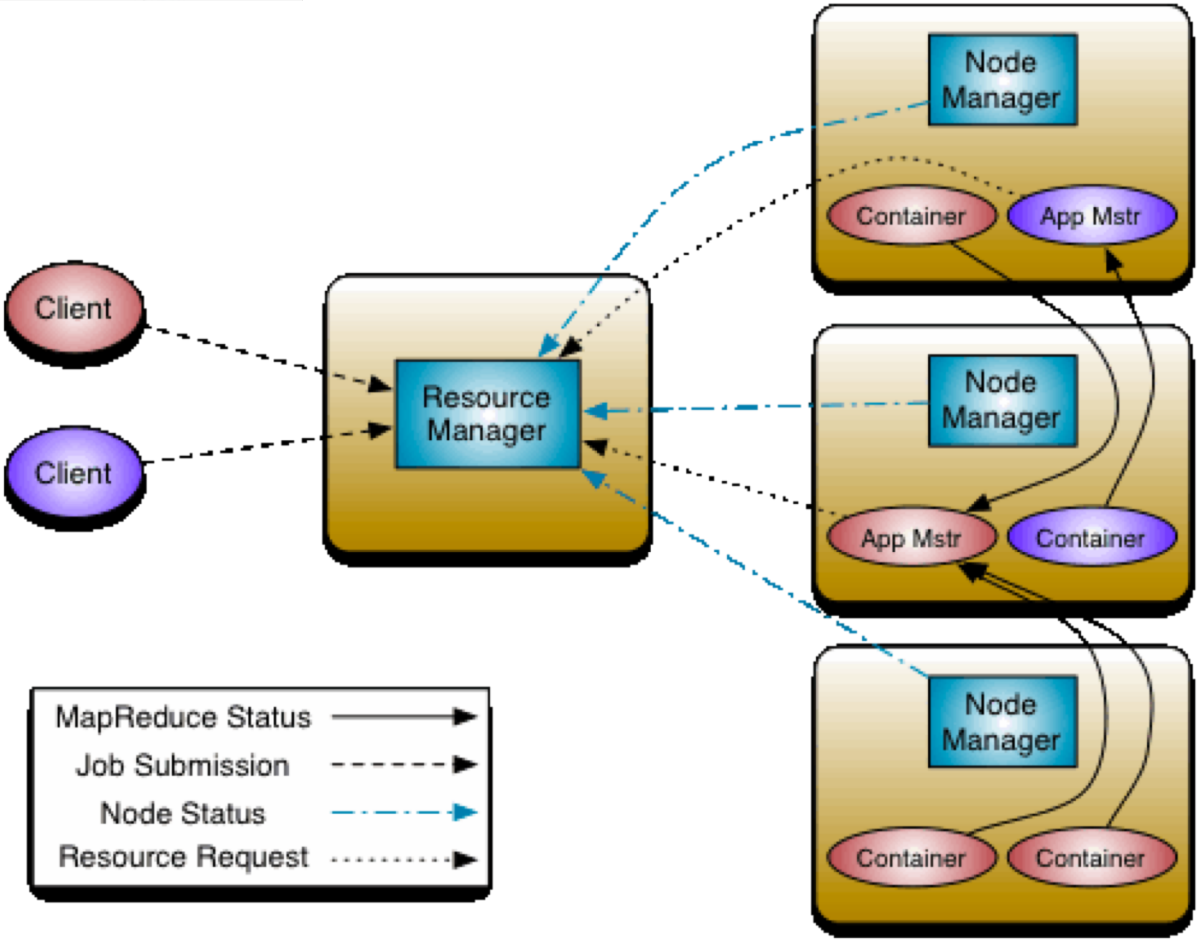
BACK

Deployment & Intergration – Resource Manager Systems



BACK

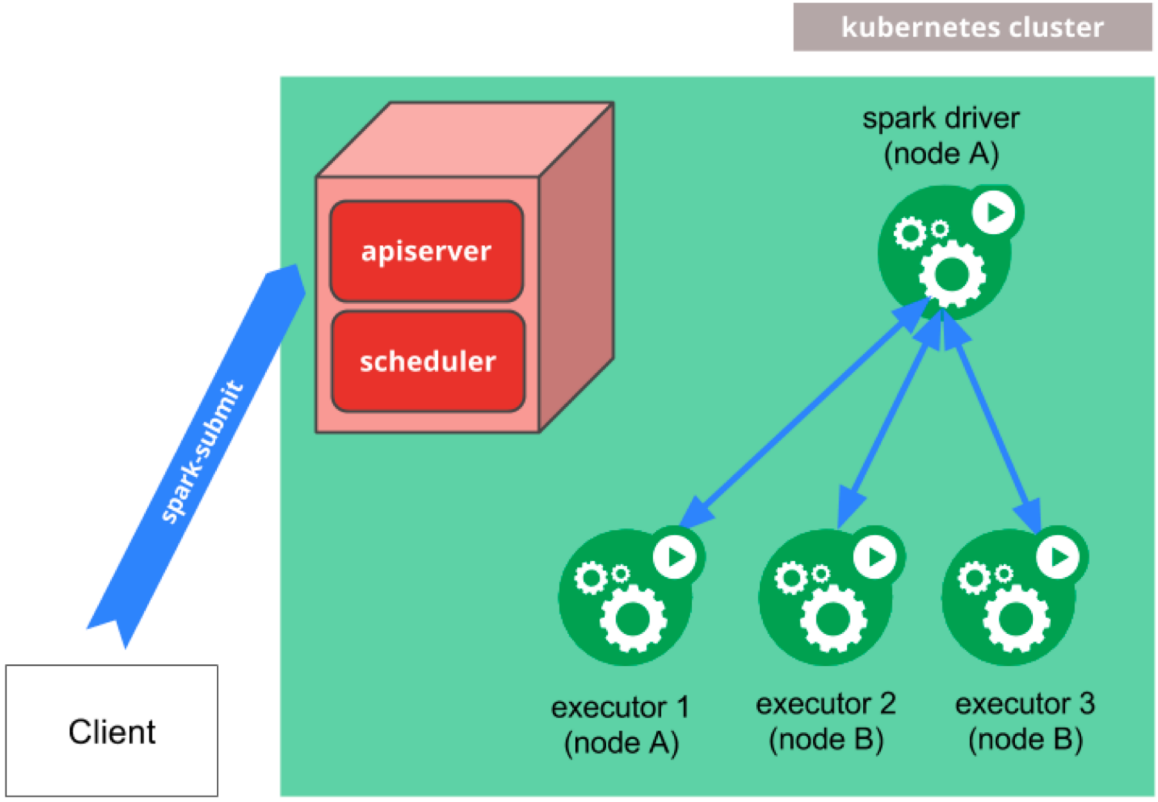
Deployment & Intergration – Resource Manager Systems



BACK

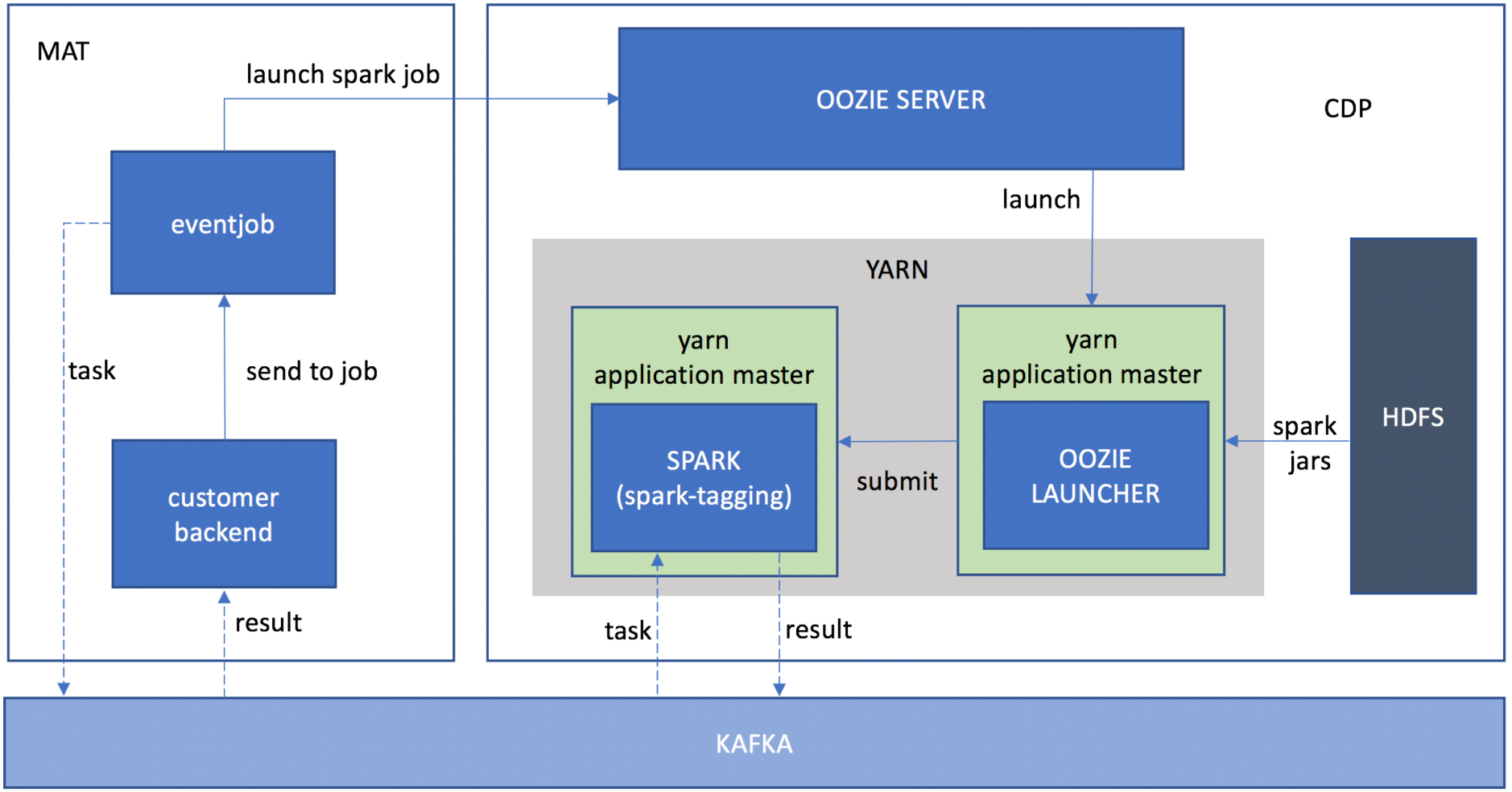
Deployment & Intergration – Resource Manager Systems

kubernetes



BACK

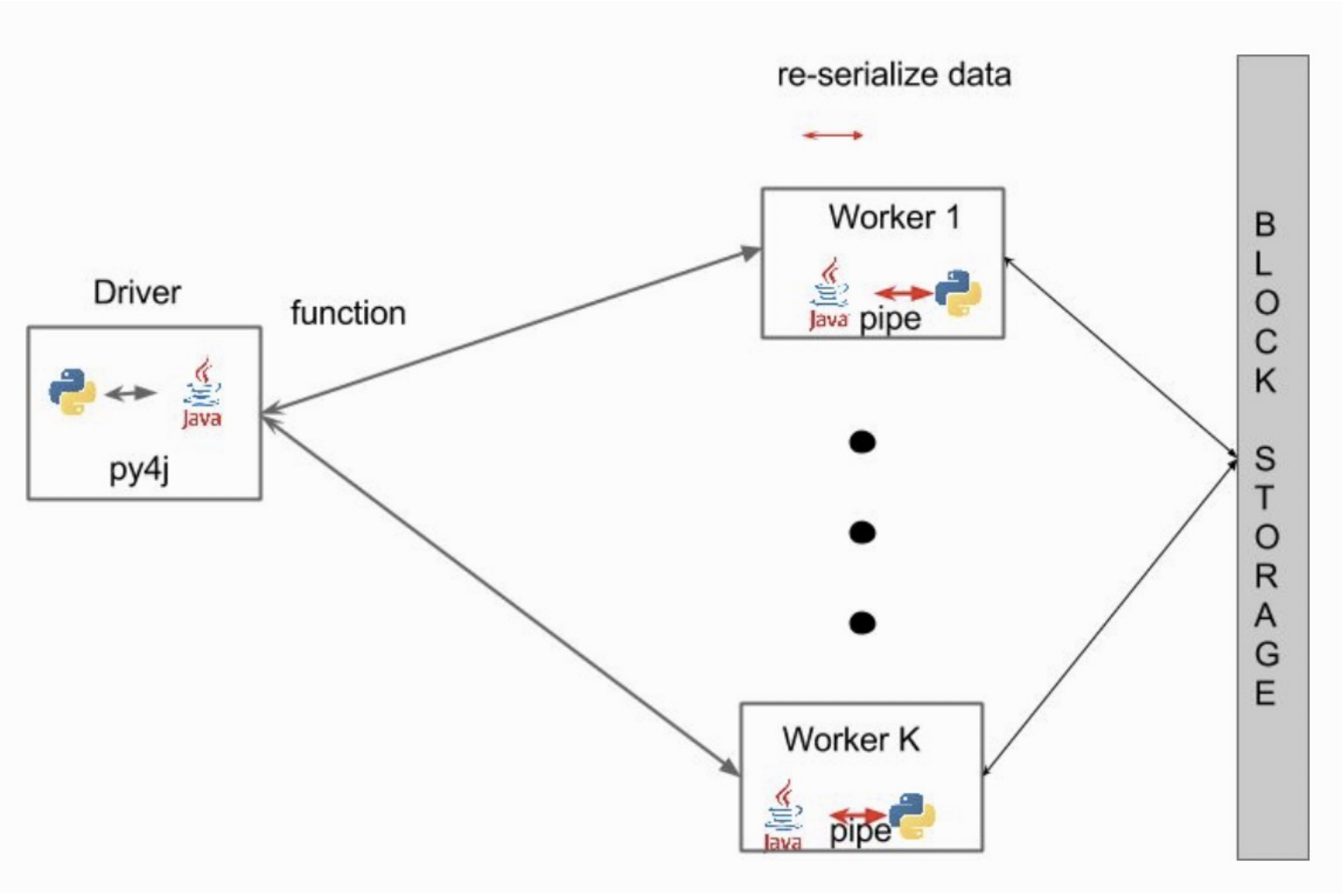
Deployment & Intergration – SparkDriver Framework



BACK

More – Beyond JVM

PySpark / SparkR



BACK

More – Beyond Batch Processing

Streaming101:The world beyond batch

	Strom	Spark Streaming	Flink
Streaming Model	Native	Micro-batching	Native
Guarantees	At-Least-Once	Exactly-Once	Exactly-Once
Back Pressure	No	Yes	Yes
Latency	Very Low	Medium	Low
Throughput	Low	High	High
Fault Tolerance	Record ACKs	RDD Based Check Pointing	Check Pointing
Stateful	No	Yes (DStream)	Yes (Operators)

BACK

References

- 耿嘉安. Spark内核设计的艺术：架构与实现(2017). 机械工业出版社
- Holden Karau, Rachel Warren. High Performance Spark(2017). O'Reilly Media, Inc.
- Michael Armbrusty, Reynold Xin. Spark SQL: Relational Data Processing in Spark(2015). DataBricks, Inc.
- Reynold Xin, Josh Rosen. Project Tungsten: Bringing Apache Spark Closer to Bare Metal(2015). DataBricks, Inc.
- Sameer Agarwal, Davies Liu and Reynold Xin. Apache Spark as a Compiler: Joining a Billion Rows per Second on a Laptop: Deep dive into the new Tungsten execution engine(2016). DataBricks, Inc.
- Reynold Xin. Technical Preview of Apache Spark 2.0: Easier, Faster, and Smarter(2016). DataBricks, Inc.
- 周志明.深入理解java虚拟机(第二版)(2013).机械工业出版社
- Tyler Akidau, Slava Chernyak, Reuven Lax. Streaming Systems(2018). O'Reilly Media, Inc.

Q&A

BACK