小象科技
ChinaHadoop.cn

Spark基础入门

董西成
2016年10月

主要内容

1. Spark概述
2. Spark核心概念
3. Spark运行模式
4. Spark在互联网公司中应用

主要内容

1. Spark概述
2. Spark核心概念
3. Spark运行模式
4. Spark在互联网公司中应用

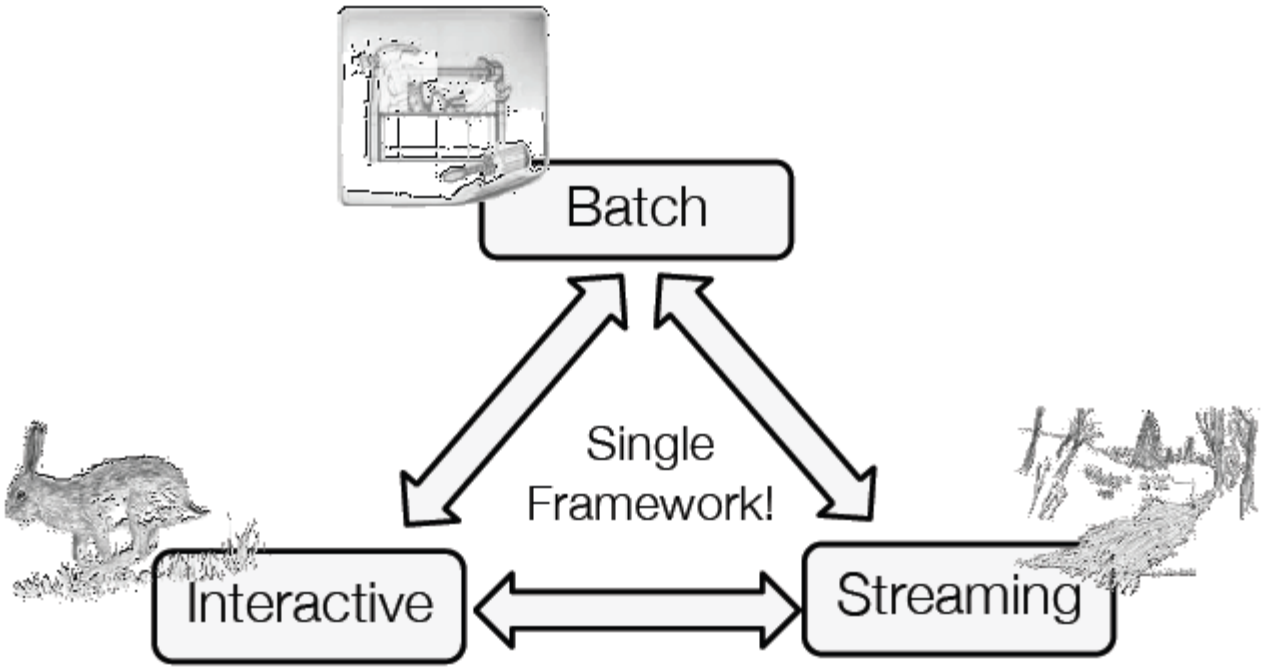
Spark背景： MapReduce局限性

- MapReduce框架局限性
  - ✓ 仅支持Map和Reduce两种操作
  - ✓ 处理效率低效
    - ✓ Map中间结果写磁盘，Reduce写HDFS，多个MR之间通过HDFS交换数据；
    - ✓ 任务调度和启动开销大；
    - ✓ 无法充分利用内存
    - ✓ Map端和Reduce端均需要排序
  - ✓ 不适合迭代计算（如机器学习、图计算等），交互式处理（数据挖掘）和流式处理（点击日志分析）
- MapReduce编程不够灵活
  - ✓ 尝试scala函数式编程语言

背景：框架多样化

- 现有的各种计算框架各自为战
  - ✓批处理：**MapReduce、Hive、Pig**
  - ✓流式计算：**Storm**
  - ✓交互式计算：**Impala**
- 能否有一种灵活的框架可同时进行批处理、流式计算、交互式计算等？

产生背景：大统一系统



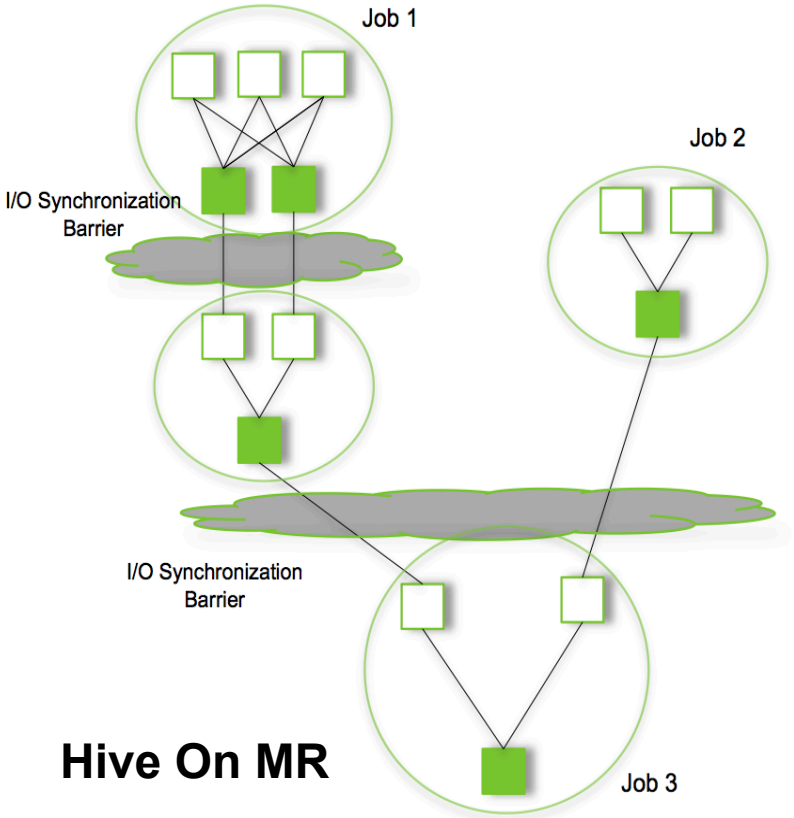
- 在一个统一的框架下，进行批处理、流式计算、交互式计算

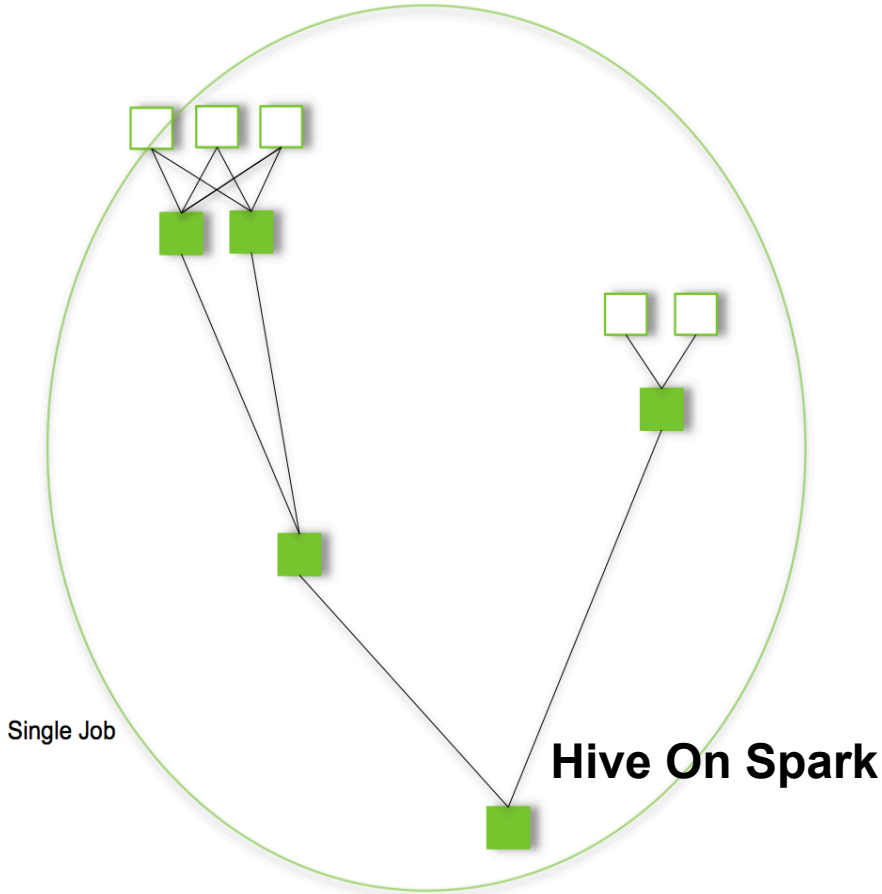
Spark特点

- 高效（比MapReduce快10~100倍）
  - ✓ 内存计算引擎，提供Cache机制来支持需要反复迭代计算或者多次数据共享，减少数据读取的IO开销
  - ✓ DAG引擎，减少多次计算之间中间结果写到HDFS的开销
  - ✓ 使用多线程池模型来减少task启动开稍，shuffle过程中避免不必要的sort操作以及减少磁盘IO操作
- 易用
  - ✓ 提供了丰富的API，支持Java，Scala，Python和R四种语言
  - ✓ 代码量比MapReduce少2~5倍
- 与Hadoop集成
  - ✓ 读写HDFS/Hbase
  - ✓ 与YARN集成

MapReduce与Spark（DAG）

```
SELECT a.state, COUNT(*), AVERAGE(c.price)
FROM a
JOIN b ON (a.id = b.id)
JOIN c ON (a.itemId = c.itemId)
GROUP BY a.state
```





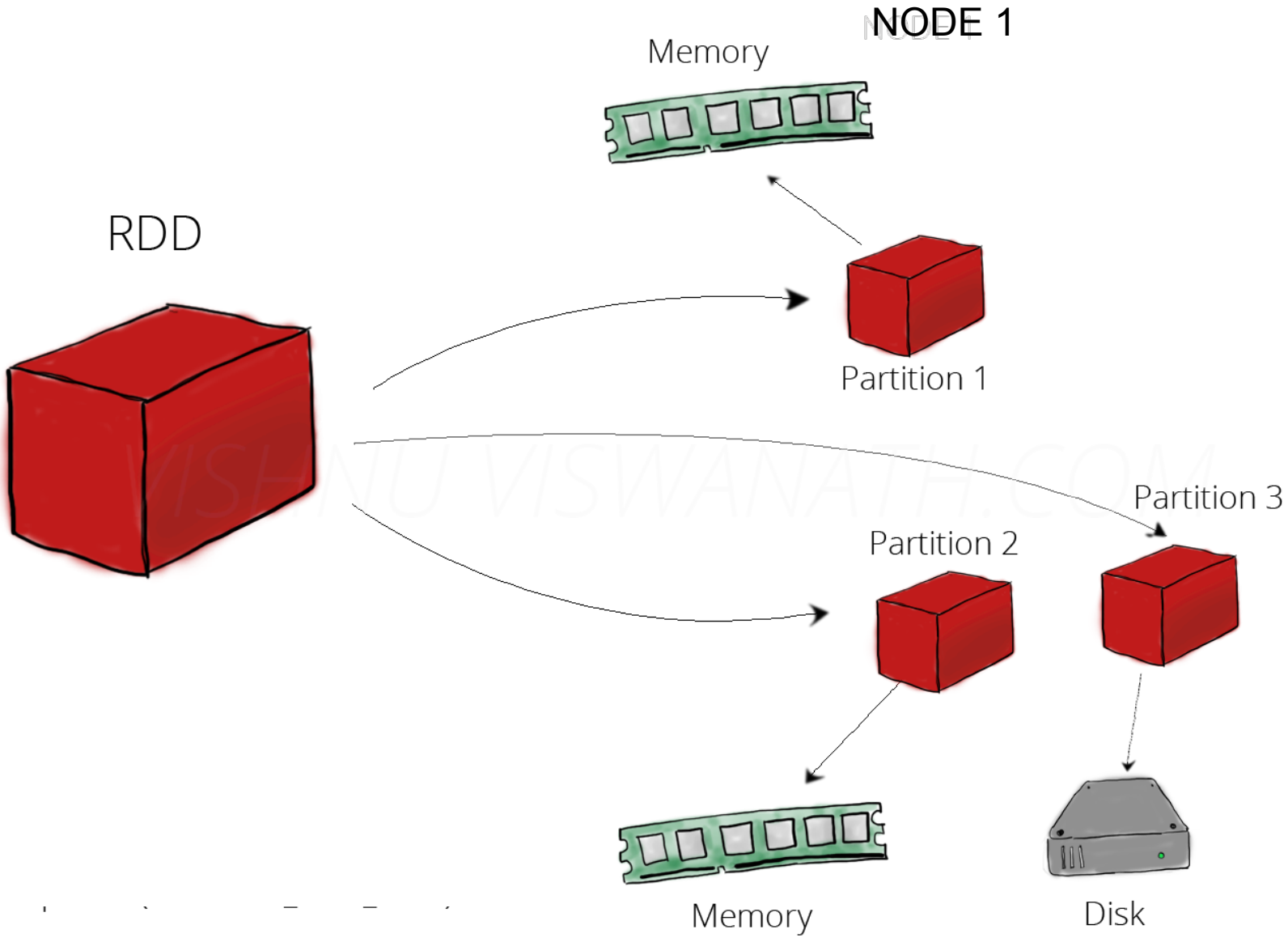
主要内容

1. Spark概述
2. **Spark核心概念**
3. Spark运行模式
4. Spark在互联网公司中应用

Spark核心概念—RDD

- RDD：Resilient Distributed Datasets，弹性分布式数据集
  - ✓分布在集群中的只读对象集合（由多个**Partition**构成）
  - ✓可以存储在磁盘或内存中（多种存储级别）
  - ✓通过并行“转换”操作构造
  - ✓失效后自动重构

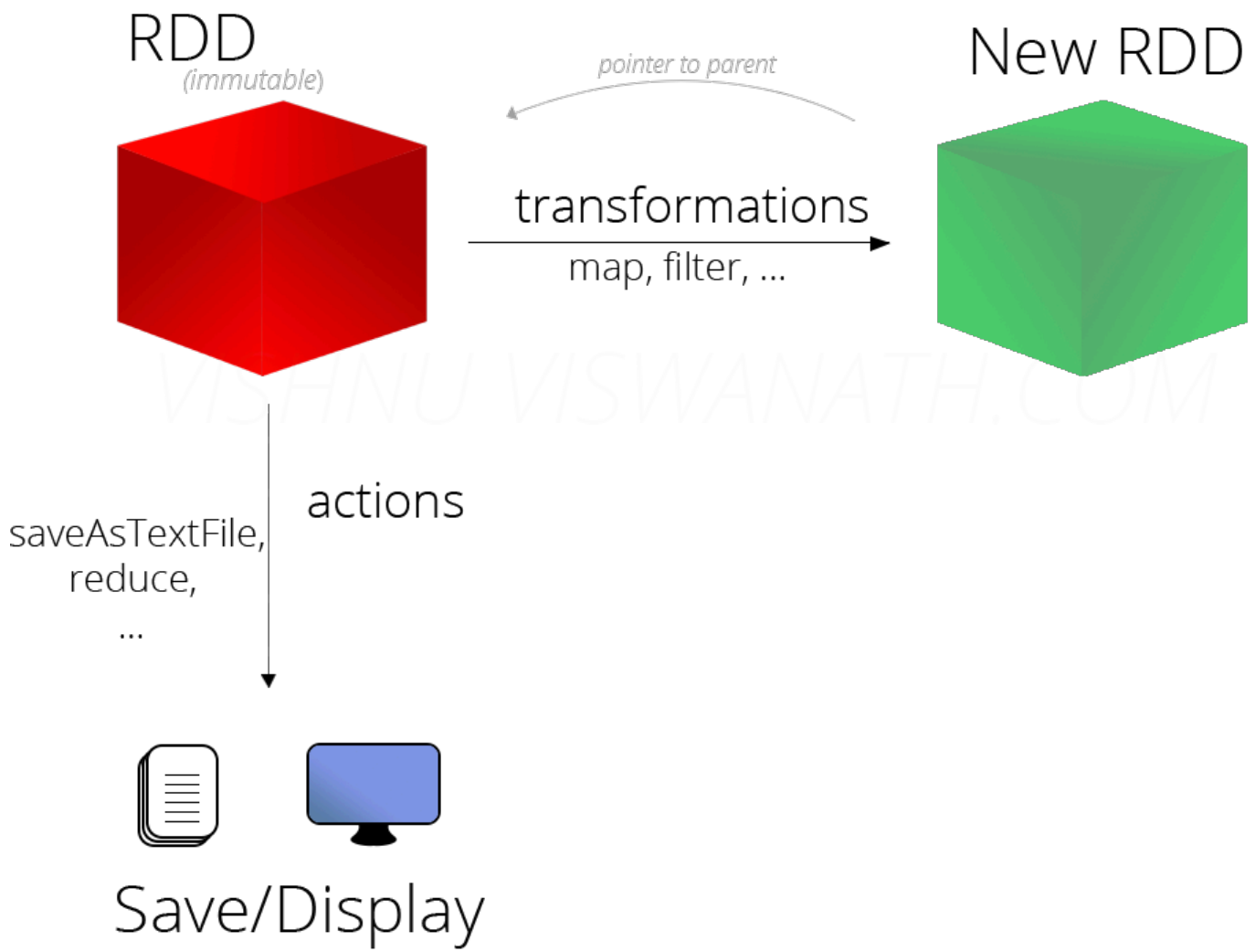
Spark核心概念—RDD



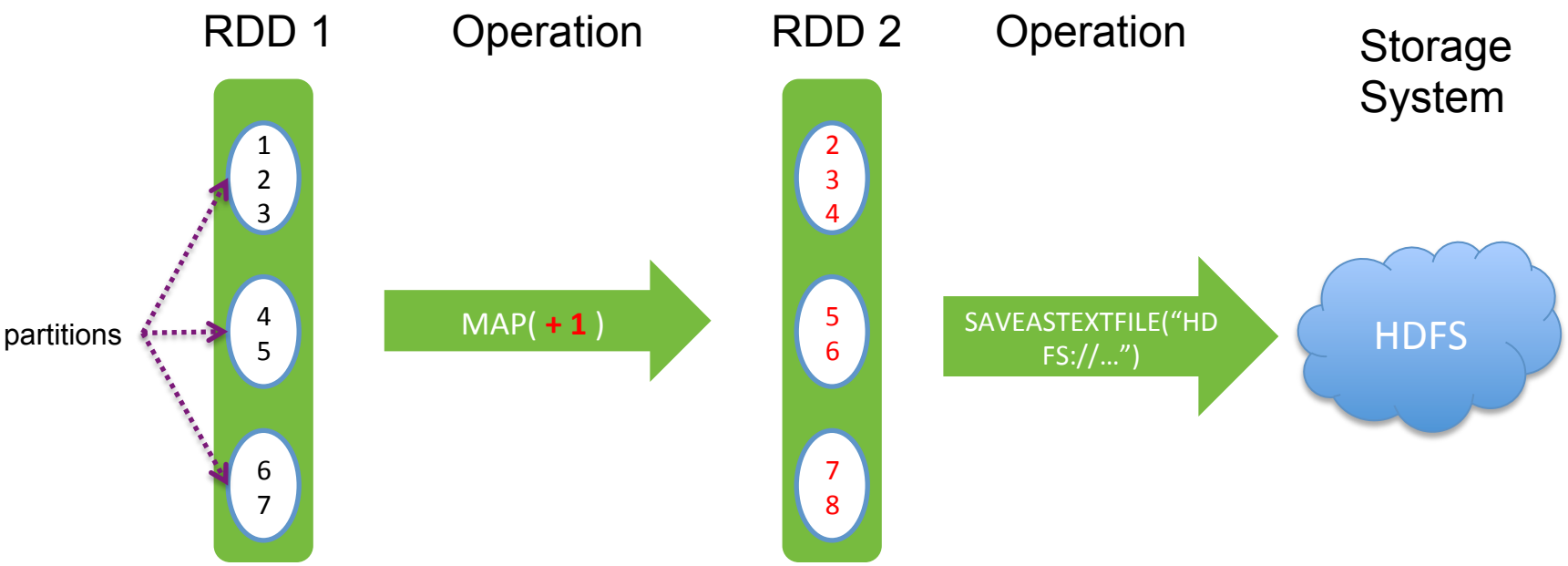
RDD基本操作（operator）

- **Transformation**
  - ✓ 可通过Scala集合或者Hadoop数据集构造一个新的RDD
  - ✓ 通过已有的RDD产生新的RDD
  - ✓ 举例：map， filter，groupBy，reduceBy
- **Action**
  - ✓ 通过RDD计算得到一个或者一组值
  - ✓ 举例：count，reduce，saveAsTextFile

RDD基本操作（operator）



Operator示例



作用在**RDD**上的**operation**

Spark提供的Transformation与Action实现

Transformations	*map*(*f* : T ⇒ U)	:	RDD[T] ⇒ RDD[U]
	filter(*f* : T ⇒ Bool)	:	RDD[T] ⇒ RDD[T]
	flatMap(*f* : T ⇒ Seq[U])	:	RDD[T] ⇒ RDD[U]
	sample(*fraction* : Float)	:	RDD[T] ⇒ RDD[T] (Deterministic sampling)
	groupByKey()	:	RDD[(K, V)] ⇒ RDD[(K, Seq[V])]
	reduceByKey(*f* : (V, V) ⇒ V)	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	union()	:	(RDD[T], RDD[T]) ⇒ RDD[T]
	join()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (V, W))]
	cogroup()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (Seq[V], Seq[W]))]
	crossProduct()	:	(RDD[T], RDD[U]) ⇒ RDD[(T, U)]
	mapValues(*f* : V ⇒ W)	:	RDD[(K, V)] ⇒ RDD[(K, W)] (Preserves partitioning)
	sort(*c* : Comparator[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	partitionBy(*p* : Partitioner[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
Actions	*count*()	:	RDD[T] ⇒ Long
	collect()	:	RDD[T] ⇒ Seq[T]
	reduce(*f* : (T, T) ⇒ T)	:	RDD[T] ⇒ T
	lookup(*k* : K)	:	RDD[(K, V)] ⇒ Seq[V] (On hash/range partitioned RDDs)
	save(*path* : String)	:	Outputs RDD to a storage system, *e.g.*, HDFS

Spark RDD cache/persist

- **Spark RDD Cache**
  - ✓ 允许将RDD缓存到内存中或磁盘上，以便于重用
  - ✓ Spark提供了多种缓存级别，以便于用户根据实际需求进行调整

```
val NONE = new StorageLevel(false, false, false, false)
val DISK_ONLY = new StorageLevel(true, false, false, false)
val DISK_ONLY_2 = new StorageLevel(true, false, false, false, 2)
val MEMORY_ONLY = new StorageLevel(false, true, false, true)
val MEMORY_ONLY_2 = new StorageLevel(false, true, false, true, 2)
val MEMORY_ONLY_SER = new StorageLevel(false, true, false, false)
val MEMORY_ONLY_SER_2 = new StorageLevel(false, true, false, false, 2)
val MEMORY_AND_DISK = new StorageLevel(true, true, false, true)
val MEMORY_AND_DISK_2 = new StorageLevel(true, true, false, true, 2)
val MEMORY_AND_DISK_SER = new StorageLevel(true, true, false, false)
val MEMORY_AND_DISK_SER_2 = new StorageLevel(true, true, false, false, 2)
val OFF_HEAP = new StorageLevel(false, false, true, false)
```

- **RDD cache使用**

```
val data = sc. textFile("hdfs://nn:8020/input")
data.cache() //实际上是data.persist(StorageLevel.MEMORY_ONLY)
//data.persist(StorageLevel.DISK_ONLY_2)
```

一个完整的实例：wordcount

```
import org.apache.spark._
import SparkContext._

object WordCount {
  def main(args: Array[String]) {
    if (args.length != 3 ){
      println("usage is org.test.WordCount <master> <input> <output>")
      return
    }
    val sparkConf = new SparkConf().setAppName(”WordCount")
    val sc = new SparkContext(sparkConf)
    val rowRdd= sc.textFile(args(1))
    val resultRdd = rowRdd.flatMap(line => line.split("\\s+"))
       .map(word => (word, 1)).reduceByKey(_ + _)
    resultRdd.saveAsTextFile(args(2))
  }
}
```

应用程序名称

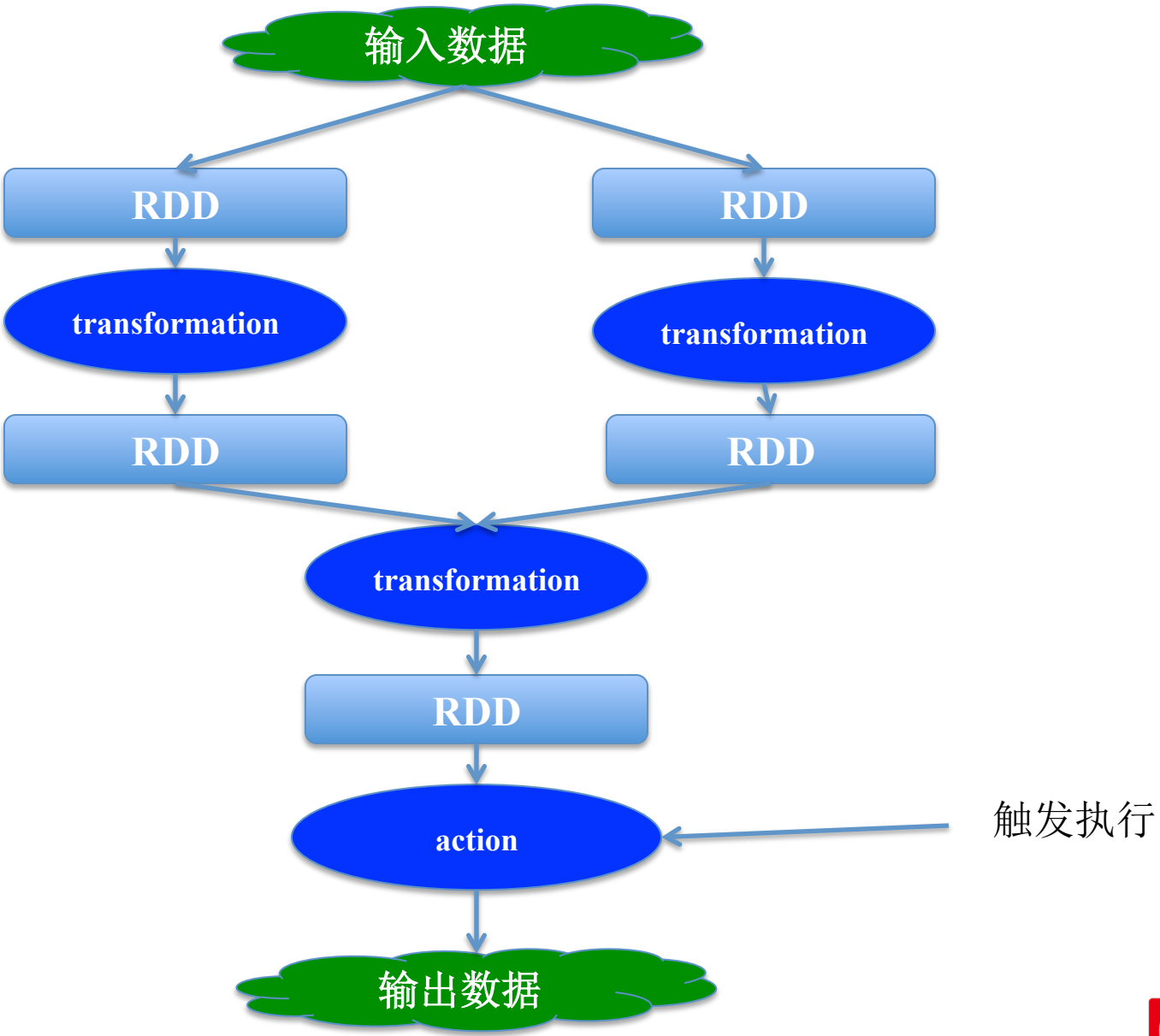
输入数据所在目录，比如：
hdfs://host:port/input/data

数据输出目录，比如：
hdfs://host:port/output/data

RDD Transformation & Action

- 接口定义方式不同
  - ✓ **Transformation： RDD[X] → RDD[Y]**
  - ✓ **Action: RDD[X] → Z (Z不是一个RDD，可能是基本类型，数组等)**
- 惰性执行（Lazy Execution）
  - ✓ **Transformation只会记录RDD转化关系，并不会触发计算**
  - ✓ **Action是触发程序执行（分布式）的算子**

RDD 惰性执行

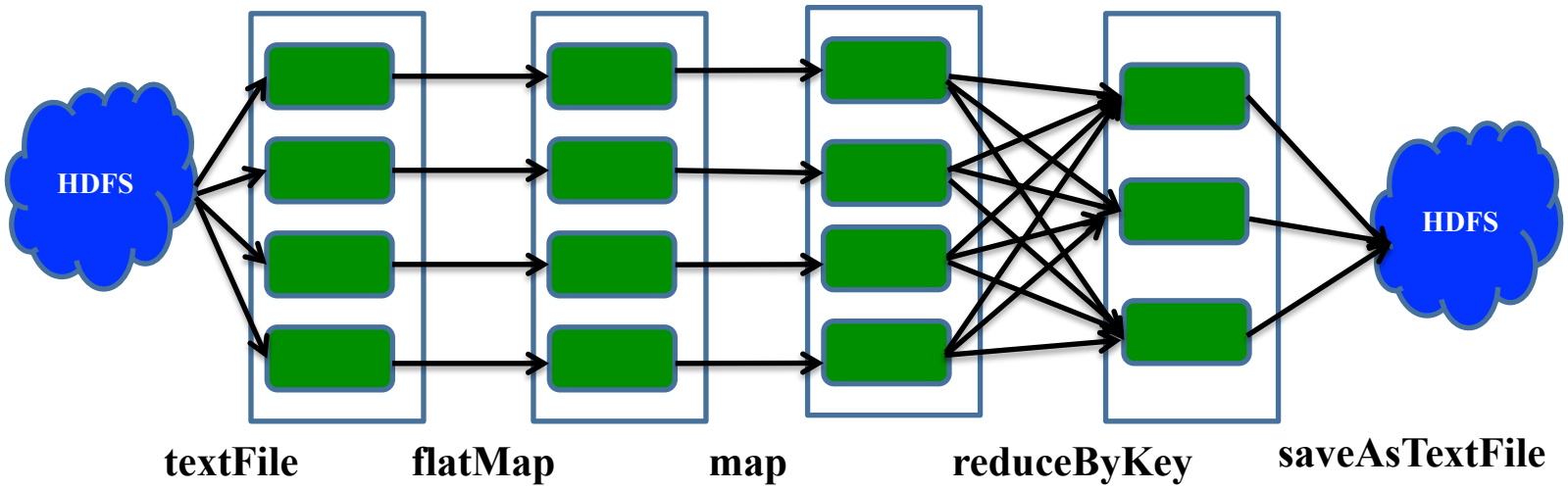


主要内容

1. Spark概述
2. Spark核心概念
3. **Spark运行模式**
4. Spark在互联网公司中应用

程序执行流程

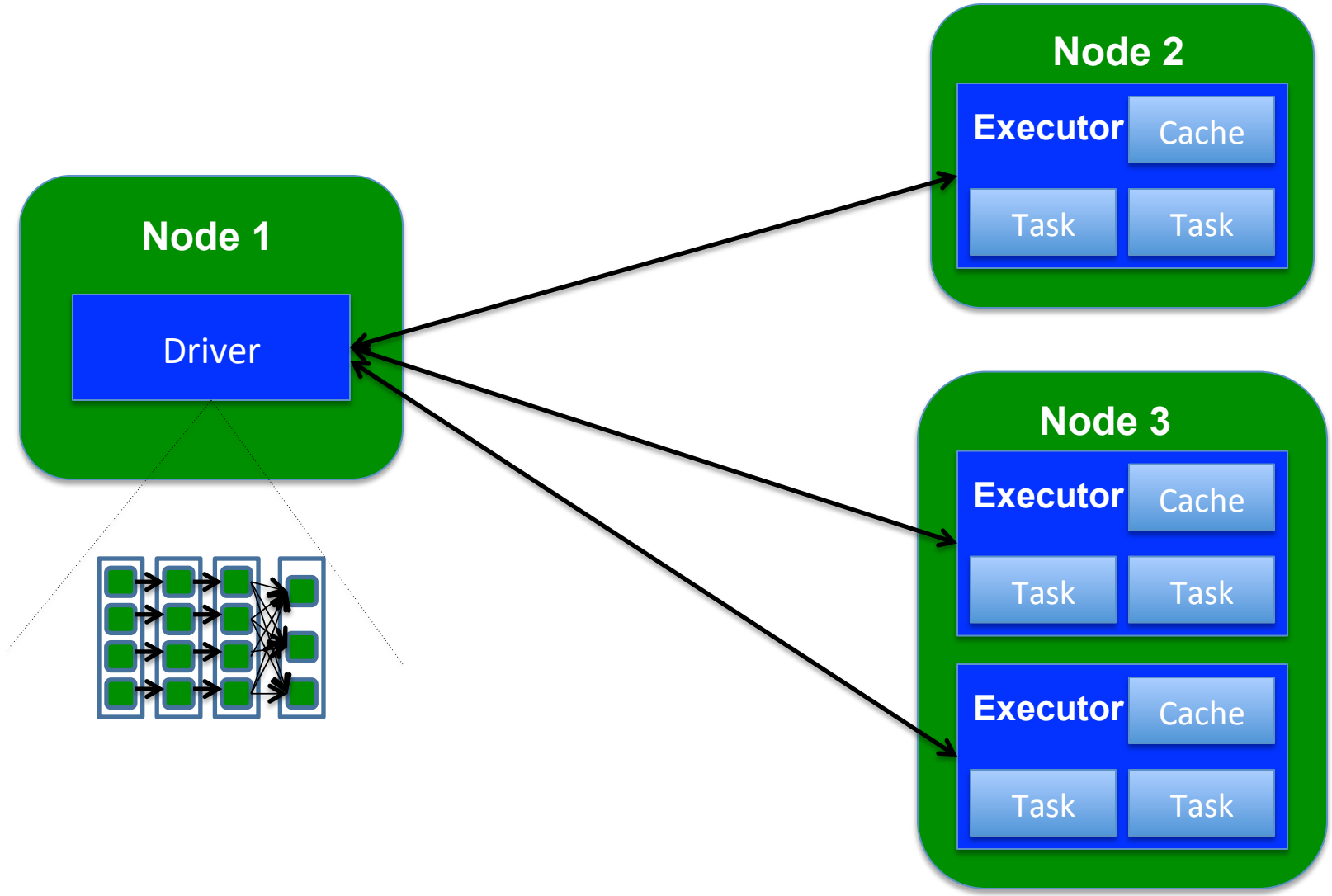
```
val rowRdd = sc.textFile(args(1))
val resultRdd = rowRdd.flatMap(line => line.split("\\s
+")).map(word => (word, 1)).reduceByKey(_ + _)
resultRdd.saveAsTextFile(args(2))
```



提交Spark程序（运行在YARN上）

```
export YARN_CONF_DIR=/opt/hadoop/yarn-client/etc/hadoop
bin/spark-submit \
  --master yarn-cluster \
  --class com.hulu.examples.SparkPi \
  --name sparkpi \
  --driver-memory 2g \
  --driver-cores 1 \
  --executor-memory 3g \
  --executor-cores 2 \
  --num-executors 2 \
  --queue spark \
  --conf spark.pi.iterators=500000 \
  --conf spark.pi.slices=10 \
  $FWDIR/target/scala-2.10/spark-example-assembly-1.0.jar
```

程序架构



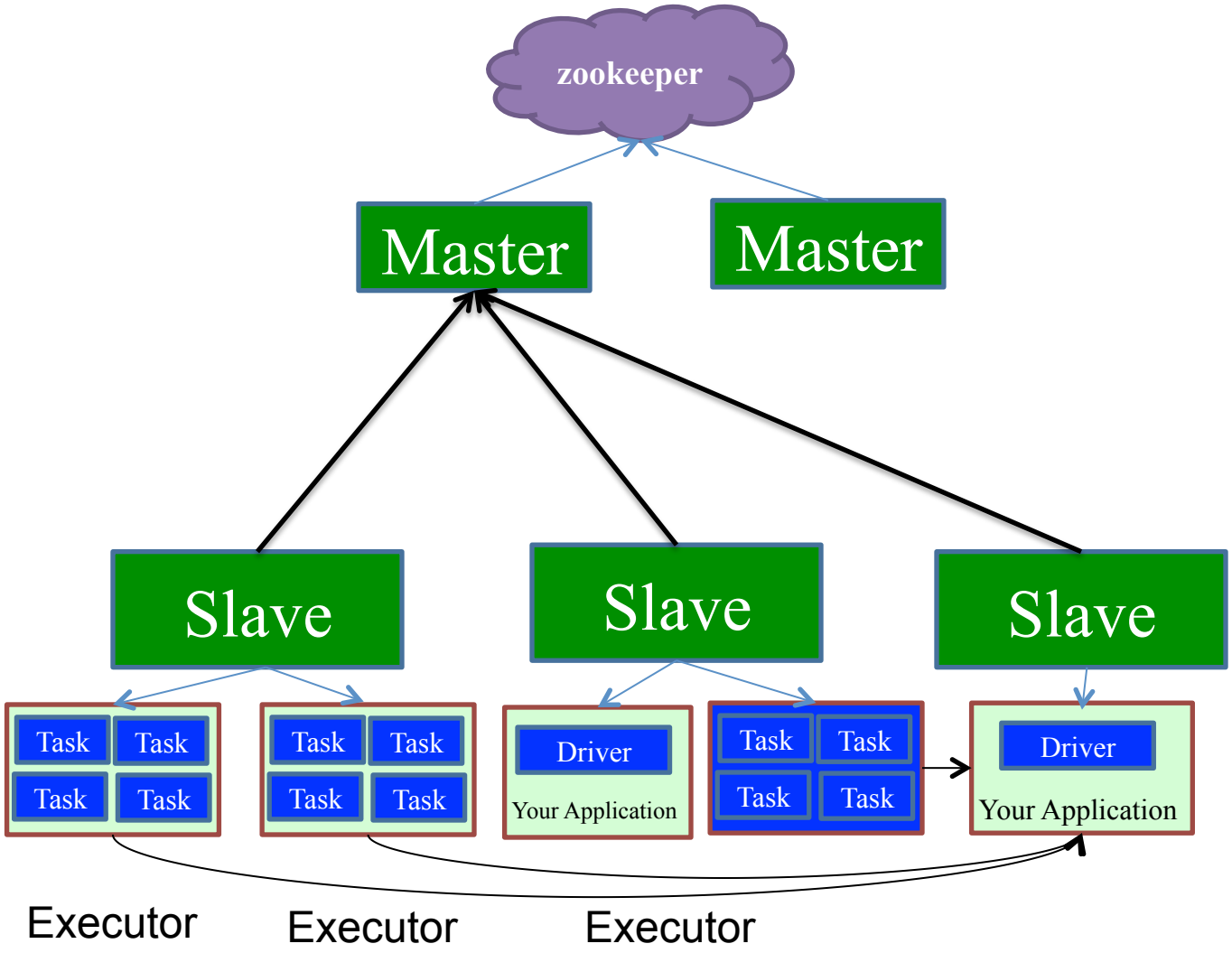
Spark运行模式

- **local（本地模式）**
  - ✓ 单机运行，通常用于测试。
- **standalone（独立模式）**
  - ✓ 独立运行在一个集群中
- **YARN/mesos**
  - ✓ 运行在资源管理系统上，比如YARN或mesos
  - ✓ Spark On YARN存在两种模式
    - ✓ yarn-client
    - ✓ yarn-cluster

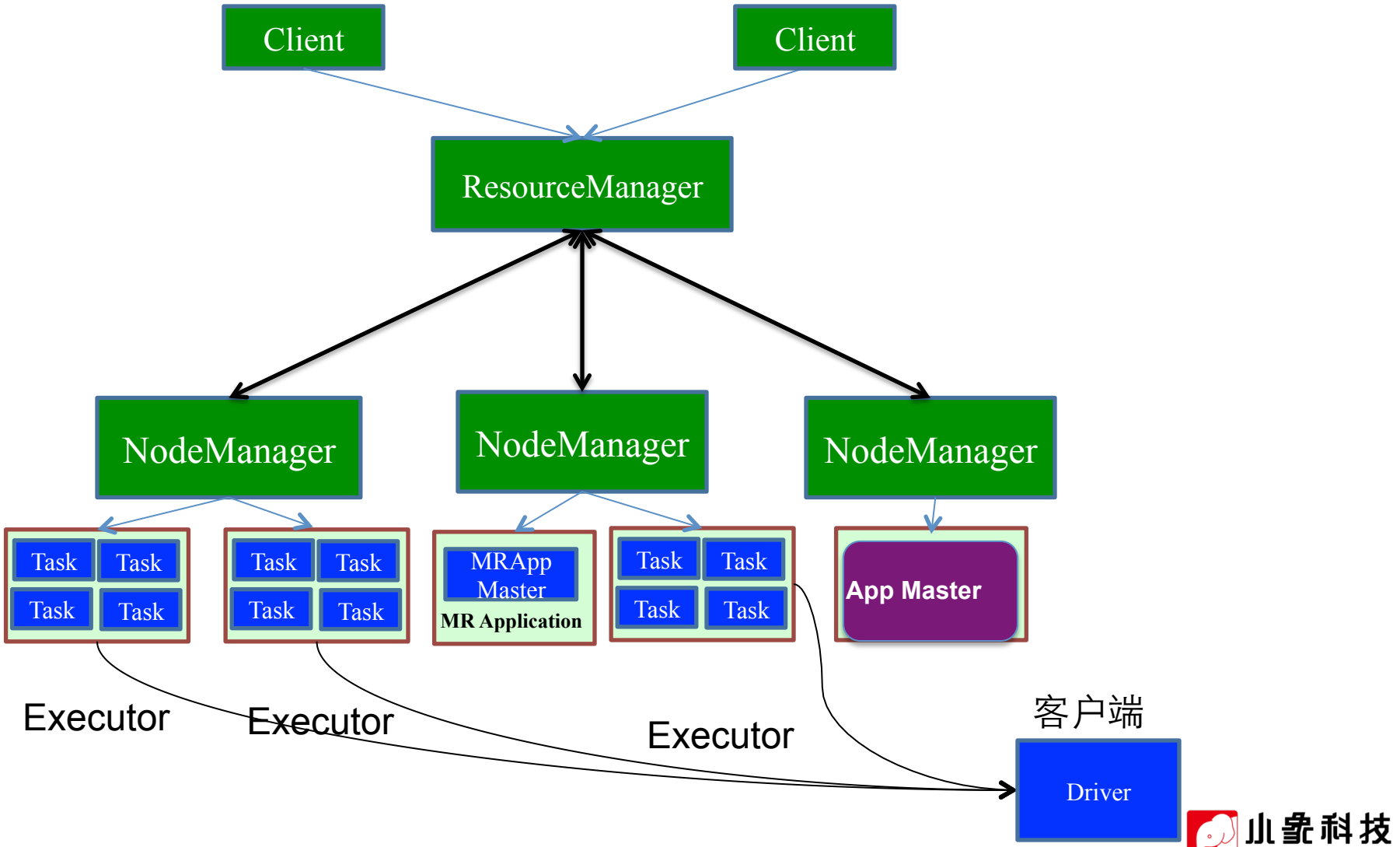
Spark运行模式：本地模式

- 什么是本地模式
  - ✓ 将Spark应用以多线程方式，直接运行在本地，便于调试
- 本地模式分类
  - ✓ **local：只启动一个executor**
  - ✓ **local[K]：启动K个executor**
  - ✓ **local[*]：启动跟cpu数目相同的executor**

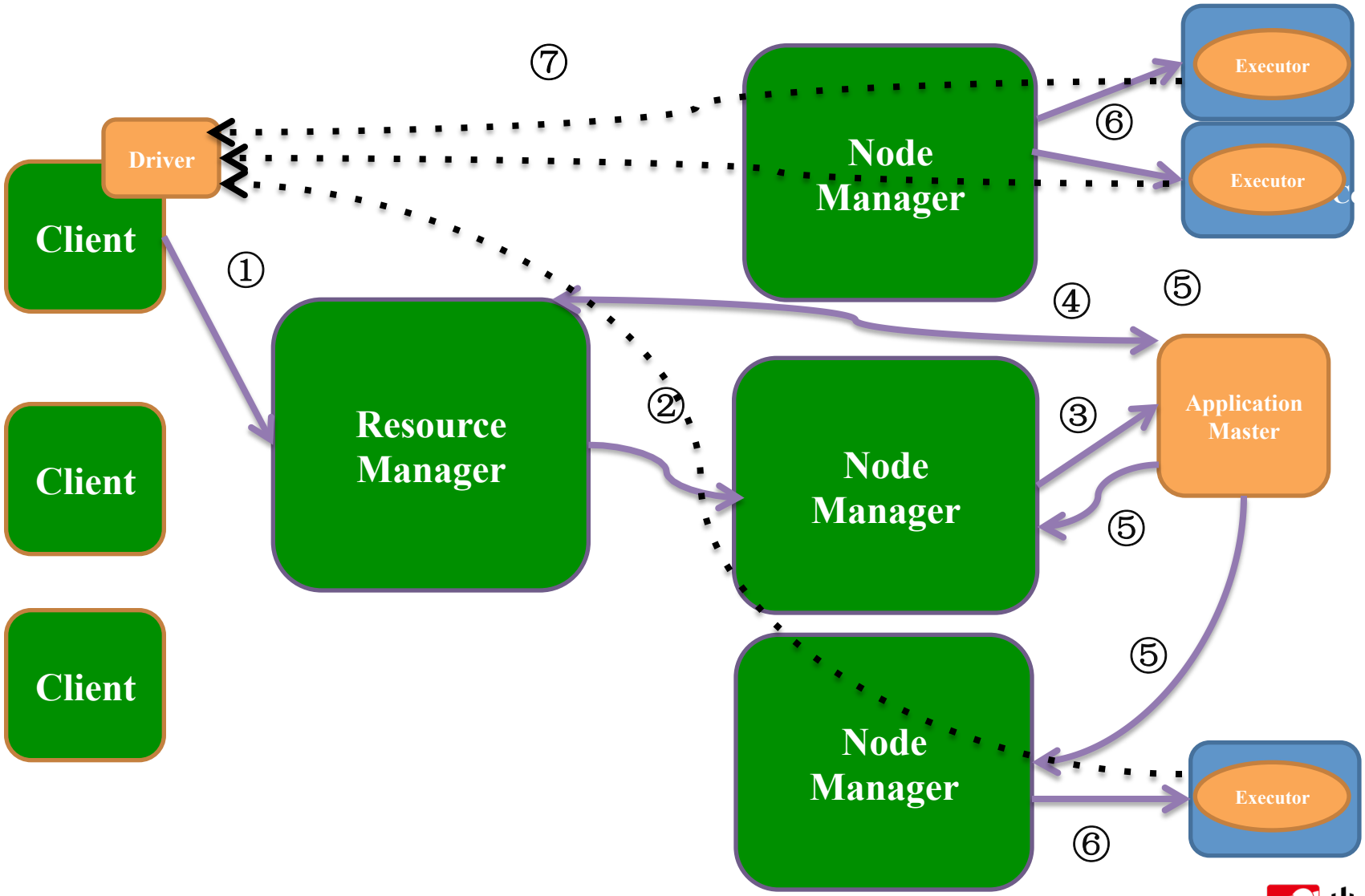
程序运行模式：独立（Standlone）模式



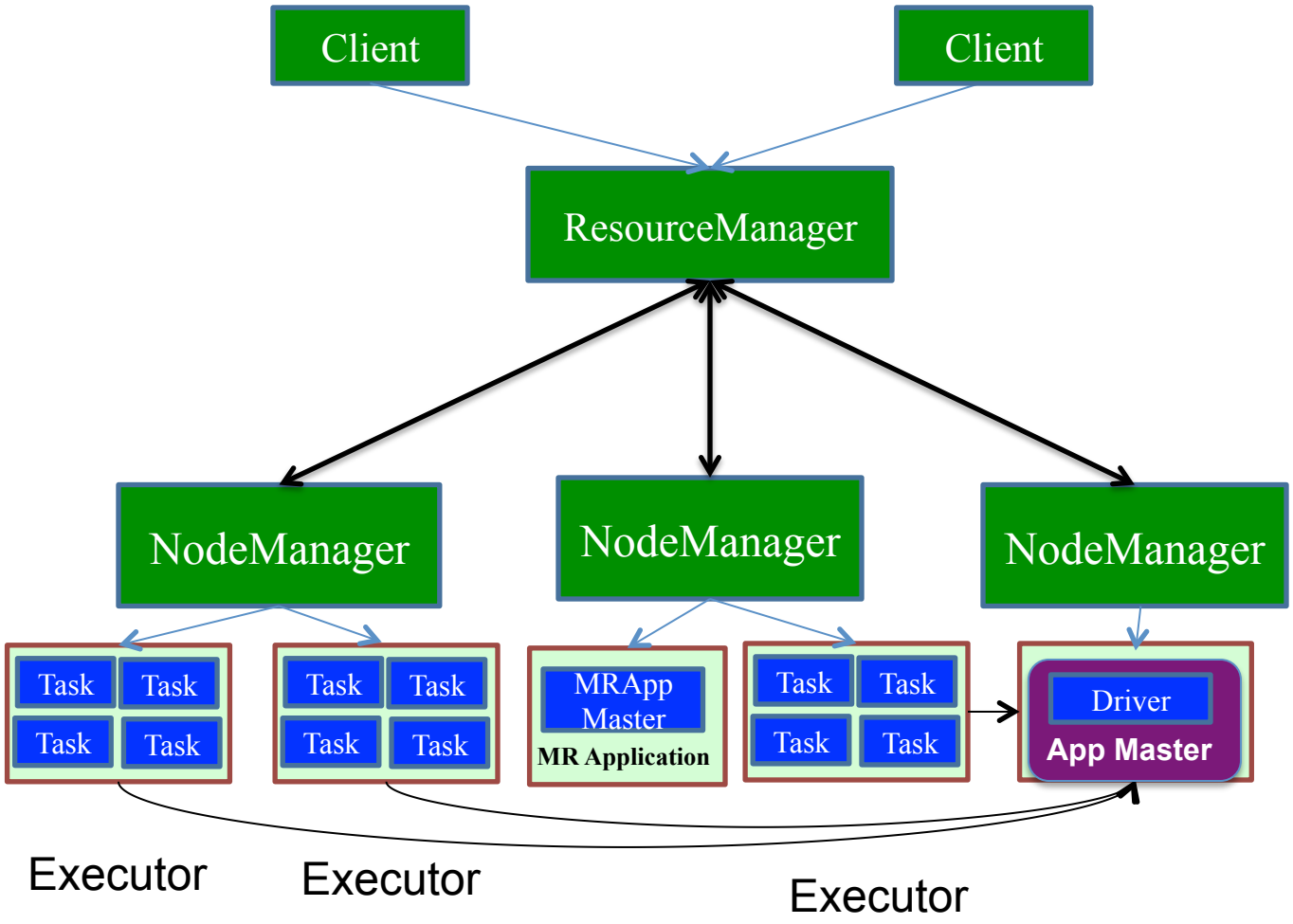
程序运行模式：YARN分布式模式(yarn-client)



程序运行模式：YARN分布式模式(yarn-client)



程序运行模式：YARN分布式模式(yarn-cluster)



主要内容

1. Spark概述
2. Spark核心概念
3. Spark运行模式
4. **Spark在互联网公司中应用**

Spark在腾讯中应用场景

- 广点通
  - ✓ 腾讯大数据精准推荐借助Spark快速迭代的优势，围绕“数据+算法+系统”这套技术方案，实现了在“数据实时采集、算法实时训练、系统实时预测”的全流程实时并行高维算法。
- 基于日志数据的快速查询系统业务
  - ✓ 构建于Spark之上的SparkSQL，利用其快速查询以及内存表等优势，承担了日志数据的即席查询工作。
- 典型算法的Spark实现
  - ✓ 预测用户的广告点击概率；
  - ✓ 计算两个好友间的共同好友数；
  - ✓ 用于ETL的SparkSQL和DAG任务；

Spark在阿里巴巴中应用场景

- 搜索和广告业务
  - ✓ 最初使用Mahout或者自己写的MR来解决复杂的机器学习，导致效率低而且代码不易维护；
  - ✓ 改进：使用Spark来解决多次迭代的机器学习算法、高计算复杂度的算法等，将Spark运用于淘宝的推荐相关算法上。
- 图算法
  - ✓ 利用Graphx解决了许多生产问题，实现的算法包括：
    - 基于度分布的中枢节点发现；
    - 基于最大连通图的社区发现；
    - 基于三角形计数的关系衡量；
    - 基于随机游走的用户属性传播等。

Spark在优酷土豆中应用场景

- 优酷土豆使用Hadoop MapReduce发现的问题
  - ✓ 商业智能BI方面，分析师提交任务之后需要等待很久才得到结果；
  - ✓ 大数据量计算，比如进行一些模拟广告投放之时，计算量非常大的同时对效率要求也比较高；
  - ✓ 机器学习和图计算的迭代运算也是需要耗费大量资源且速度很慢
- 使用Spark解决以上问题
  - ✓ 交互查询响应快，性能比Hadoop提高若干倍；
  - ✓ 模拟广告投放计算效率高、延迟小（同hadoop比延迟至少降低一个数量级）；
  - ✓ 机器学习、图计算等迭代计算，大大减少了网络传输、数据落地等，极大的提高的计算性能。

讨论

- **Spark在任何情况下均比MapReduce高效吗？**
  - ✓ **如果否，请举例。**
- **Spark号称“内存计算框架”，它将所有数据写到内存吗？**
- **当前存在很多DAG引擎，包括Spark，Tez，Flink，为何大家都在讨论Spark？**

推荐阅读资料

- Spark官方文档
  - ✓ http://spark.apache.org/docs/1.6.2/programming-guide.html
- Spark快速大数据分析
  - ✓ 几乎为Spark官方文档的中文翻译
- Spark在线练习
  - ✓ http://ampcamp.berkeley.edu/
  - ✓ http://ampcamp.berkeley.edu/big-data-mini-course/

推荐博客

- Databricks技术博客
  - ✓ https://databricks.com/blog/category/engineering
- Hortonworks技术博客
  - ✓ http://hortonworks.com/blog/
- Cloudera技术博客
  - ✓ http://blog.cloudera.com/

下一讲内容预告

- Spark程序设计与实战
  - ✓ 从头编写一个Spark程序
  - ✓ 常见的transformation和action使用方法和技巧，比如join，reduceByKey，foreach，count等
- 问题
  - ✓ 输入目录大小为1GB，spark为何产生了8个或100个任务？
  - ✓ Spark程序的Reduce task为何是200个，如何减小？
  - ✓ Spark如何访问hbase？
  - ✓ Spark cache如何使用？

小象科技
ChinaHadoop.cn

联系我们：

- 新浪微博：ChinaHadoop
- 微信公号：ChinaHadoop

让你的数据产生价值！