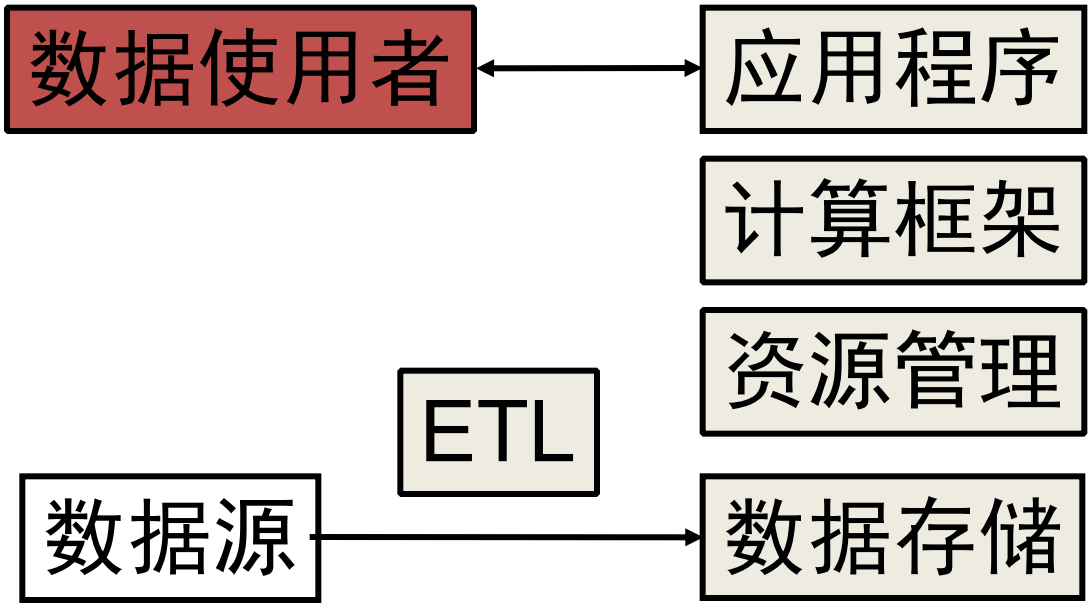
Spark & Shark 初探

杨辰
平台技术中心\数据平台组

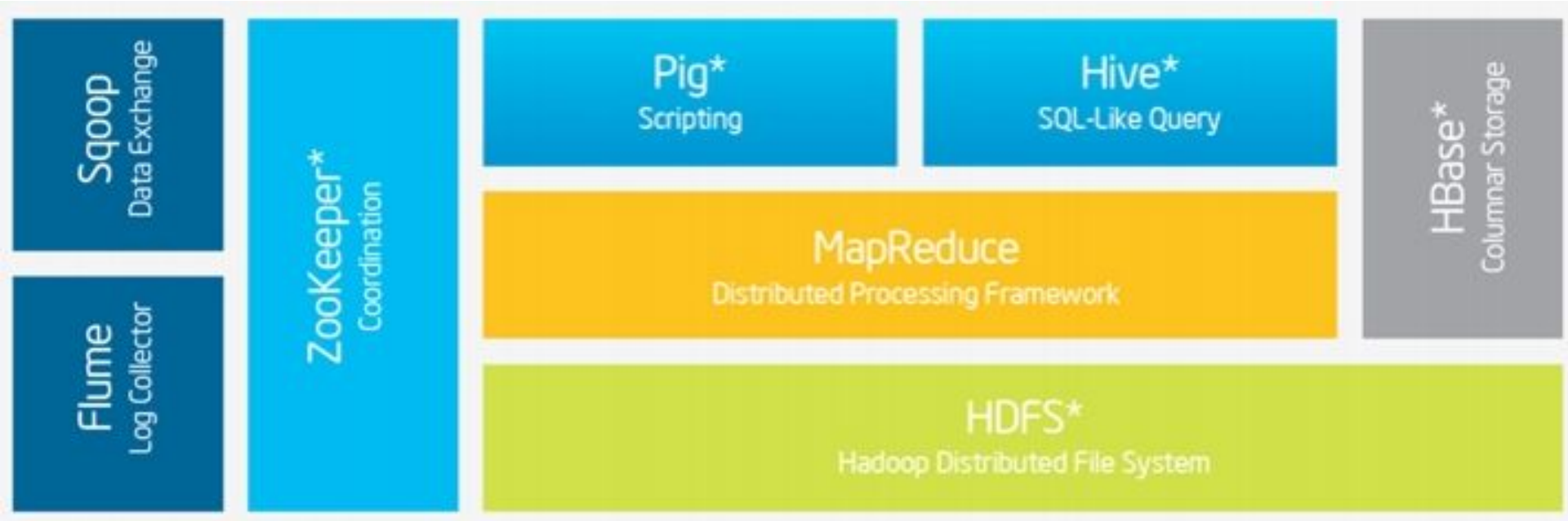
数据有什么用?

- 业务报表, 决策支持
- 业务监控 (数据实时性)
- 分析解决各种问题, 疑惑 (交互式查询)
- 数据驱动服务 (ML)

数据处理软件栈



Hadoop数据处理软件栈



Hadoop 2.0

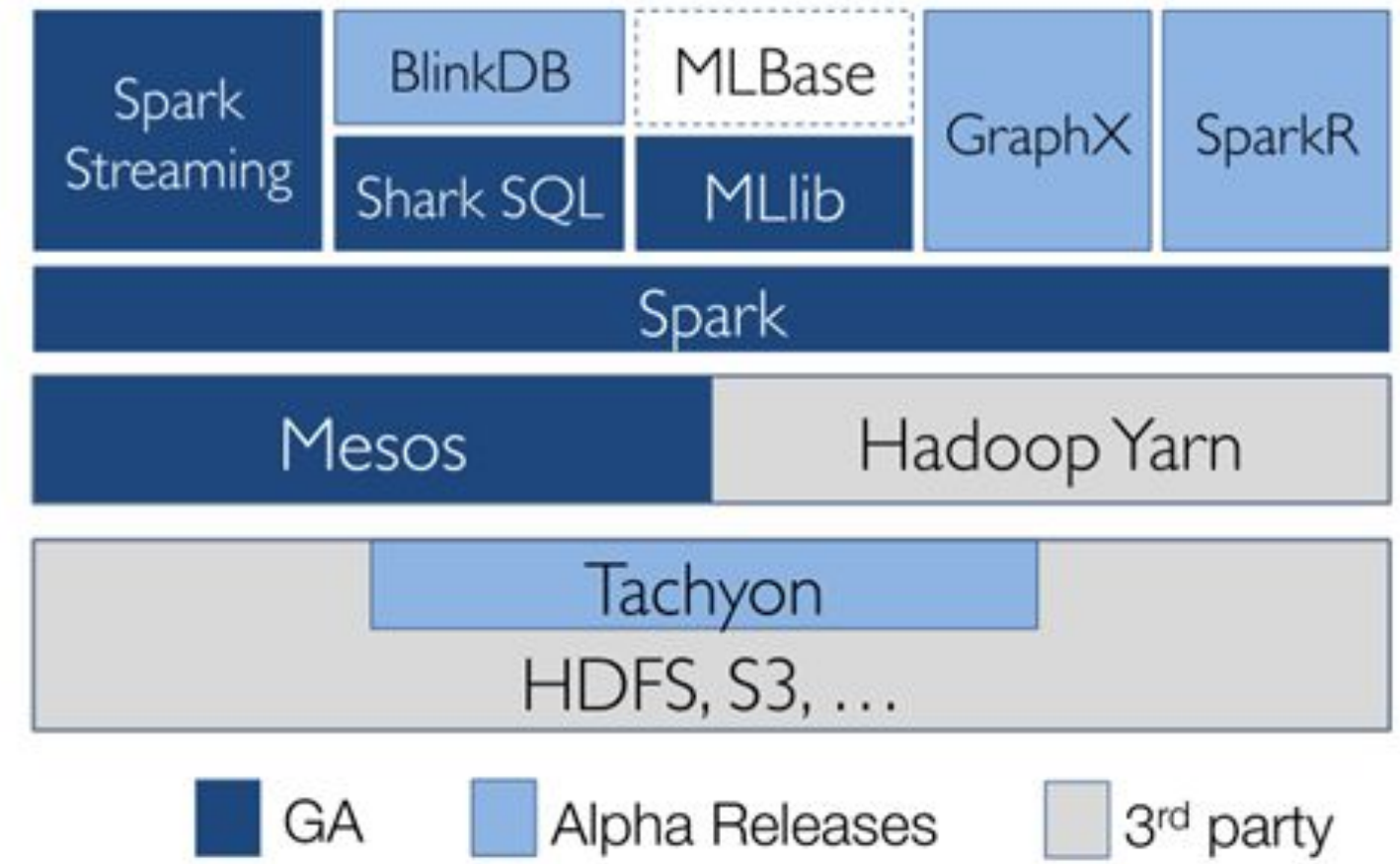
在统一的资源管理上运行不同的计算框架

痛苦和需求

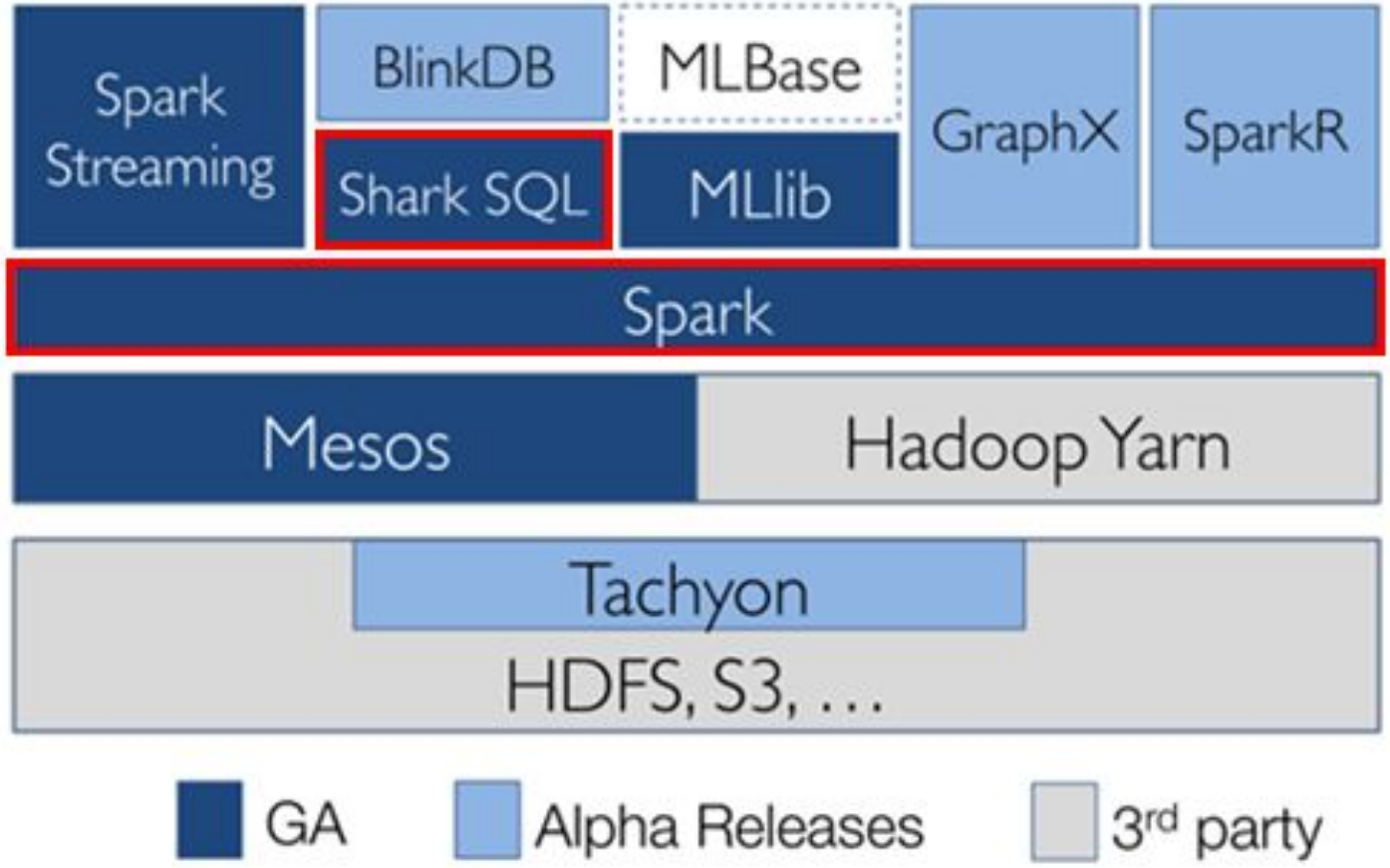
统一的计算框架下:

- 更加实时的查询速度
- 数据流实时处理
- 快速, 易用的数据挖掘(迭代算法)

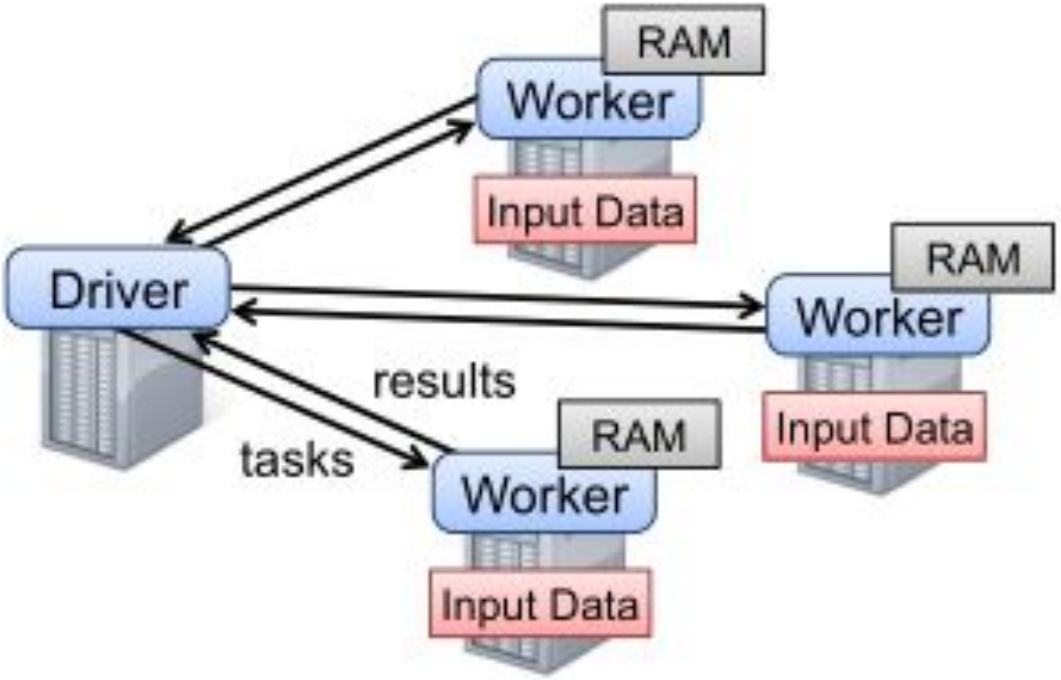
BDAS数据处理软件栈



今天介绍的内容点

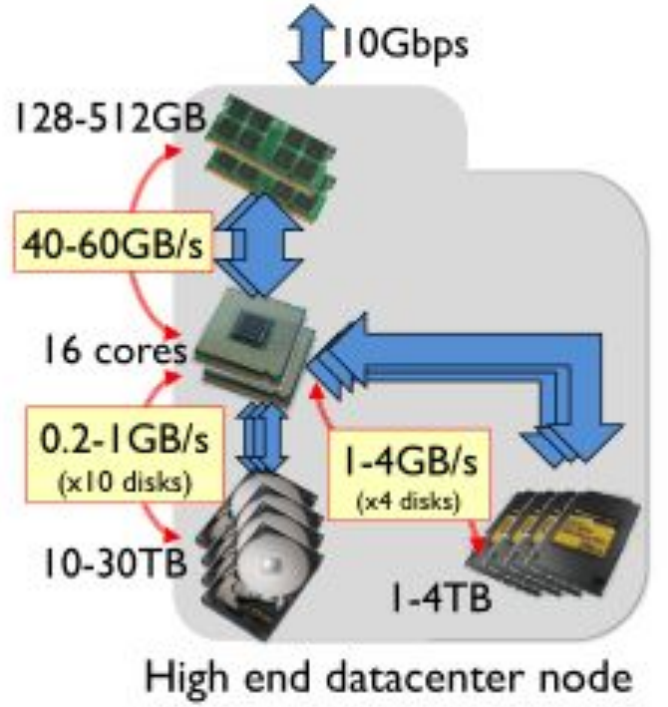


Spark架构



为什么基于内存?

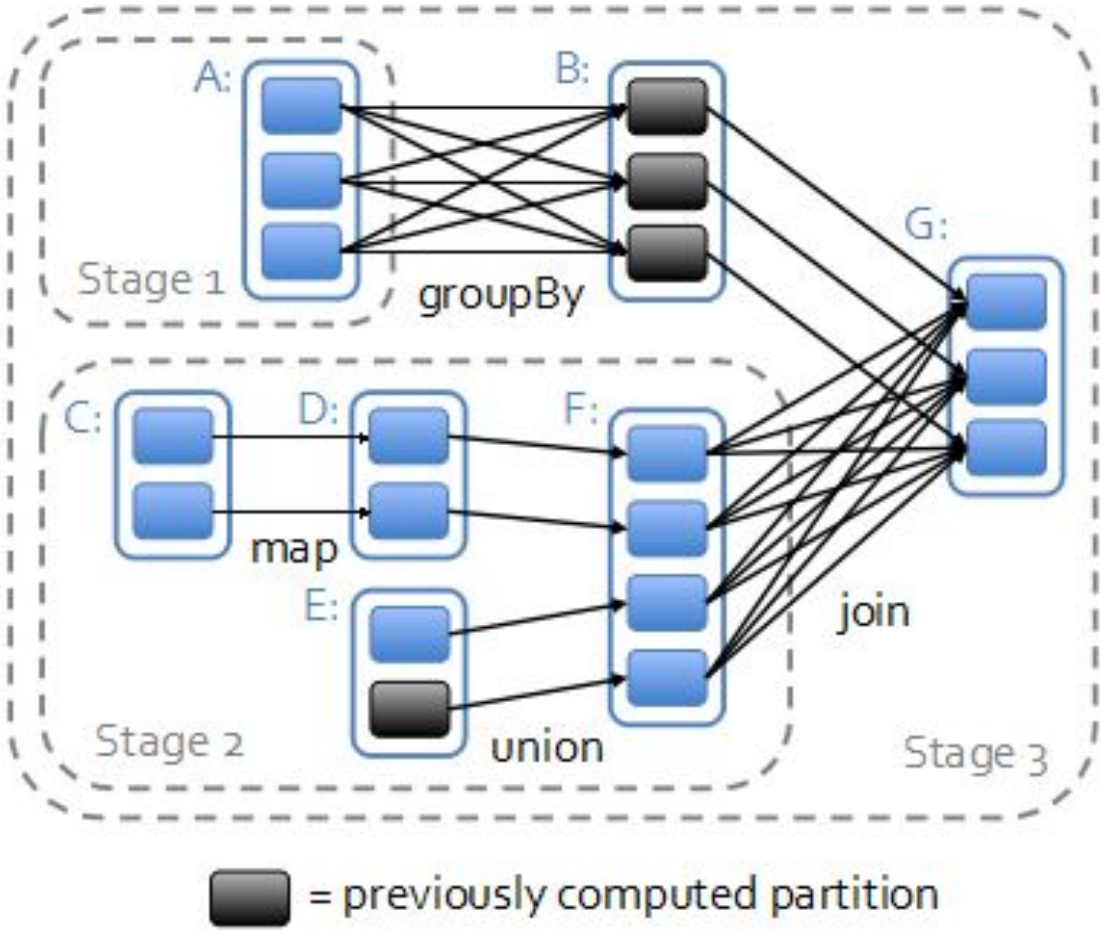
- 观察：热数据很小，（基本）可以加载到内存上
- IO速度优势
- 避免序列化/反序列化



RDD (**R**esilient **D**istributed **D**atasets)

- Spark计算中间结果数据,基于内存,冗余,分布式存储
- DAG依赖关系，通过失效重算达到容错
- 可以指定cache到内存
- lineage:
  - source data (external files)
  - transformations (map, filter, groupBy, join)
  - actions (count, collect, save)

RDD(Resilient Distributed Datasets)



Powered by Scala

- Scala: Erlang on JVM / Java in concurrent world
- 优势:
  - 语法简洁
  - 类型推断
  - 支持OO等更多的现代语言特性
  - java互通

- functional style:
  - immutable variable
  - lazy evaluation
  - data centric processing

Spark shell

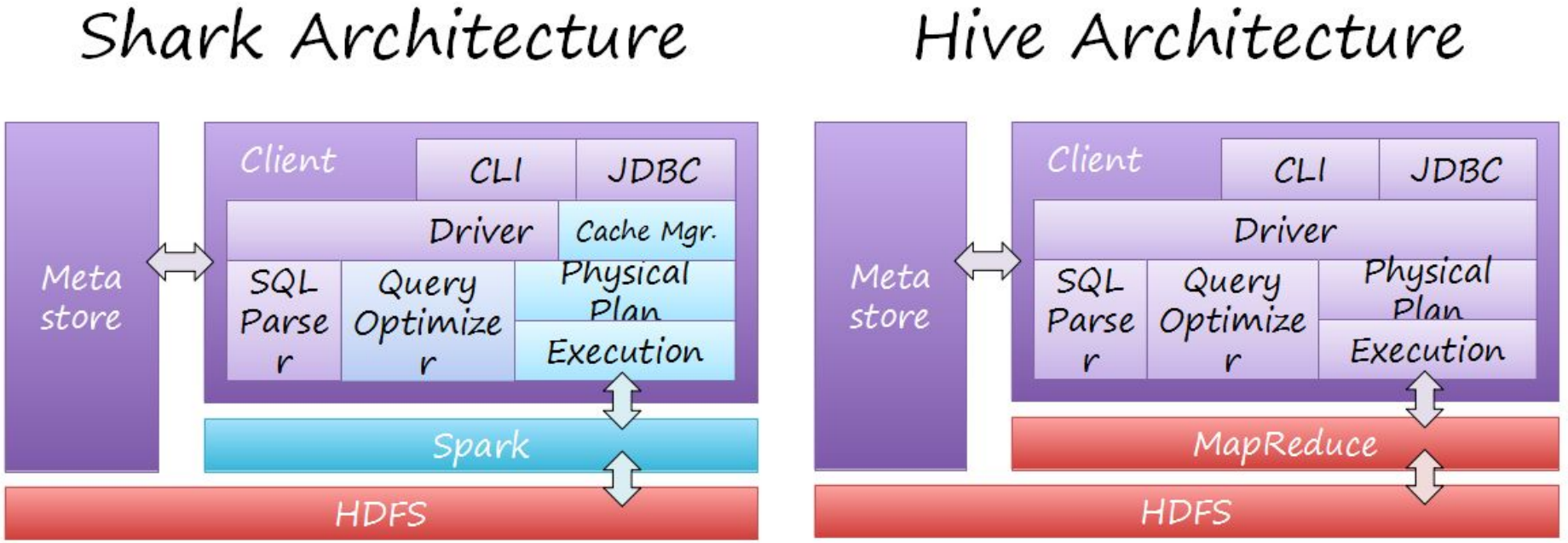
```
// 源数据
val file = sc.textFile("hdfs://namenode.demo.4399dataplat.com:54310/tmp/paylog100m.csv")

//// 数据转换
// 每行数据分割成{玩家, 充值}元组
val pairs = file.map(line => {val tuple = line.split("\t"); (tuple(0), tuple(1).toFloat)})
// 按玩家做聚合
val totals = pairs.reduceByKey(_+_)

totals.filter(_._2 > 100).count // 充值超过100的玩家数目
//// 数据结果采集
// 得到充值最高玩家
val top = totals.reduce((a, b) => if (a._2 > b._2) a else b)

totals.cache // 缓存totals到内存
```

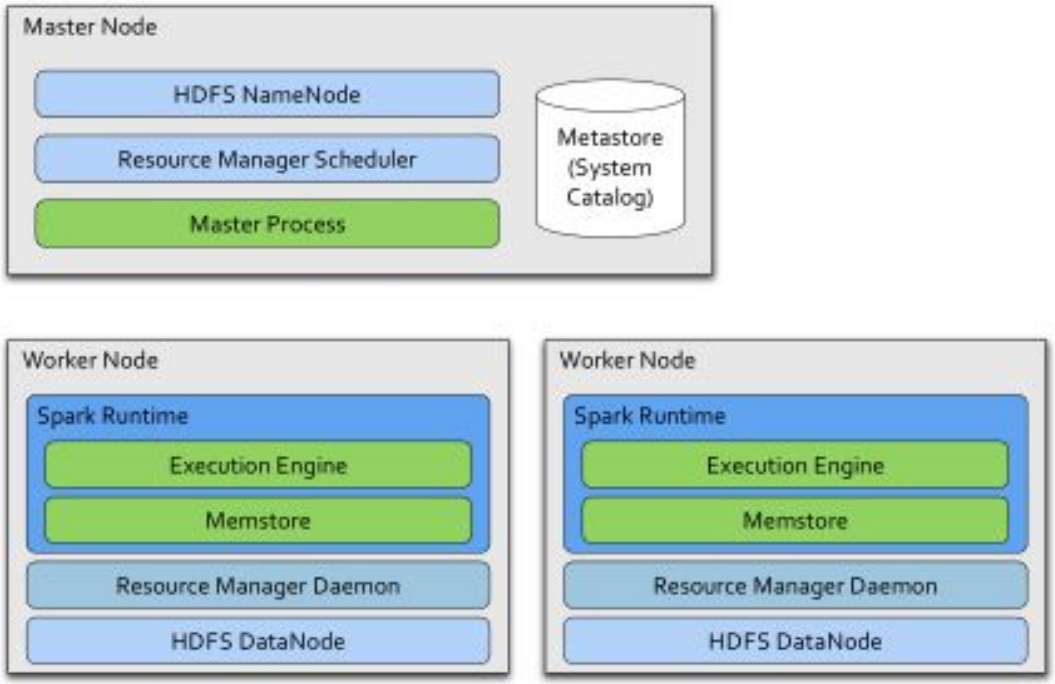
Shark: Hive over Spark



- 元数据存储和Hive互通
- 内存数据列存储
- 比Hive快5-10倍；如果源数据缓存在内存中，快100倍

Shark架构

- Spark + HiveMetastore + Execution Engine
- 也有server模式,共用内存缓存



Shark内存缓存

```
CREATE TABLE xxoo
TBLPROPERTIES ("shark.cache" = "true")
AS SELECT * FROM ...
```

或

```
CREATE TABLE xxoo_cached
AS SELECT * FROM ...
```

可以按照hive分区加载(待研究)

Shark DEMO

End

遇到的问题:

- Spark 调度机制, (Spark standalone调度机为制FIFO)
- Spark单个任务计算内存不足时OOM异常
- 待深入研究和经验积累...