大数据处理系统简介

金熠波

2020年4月

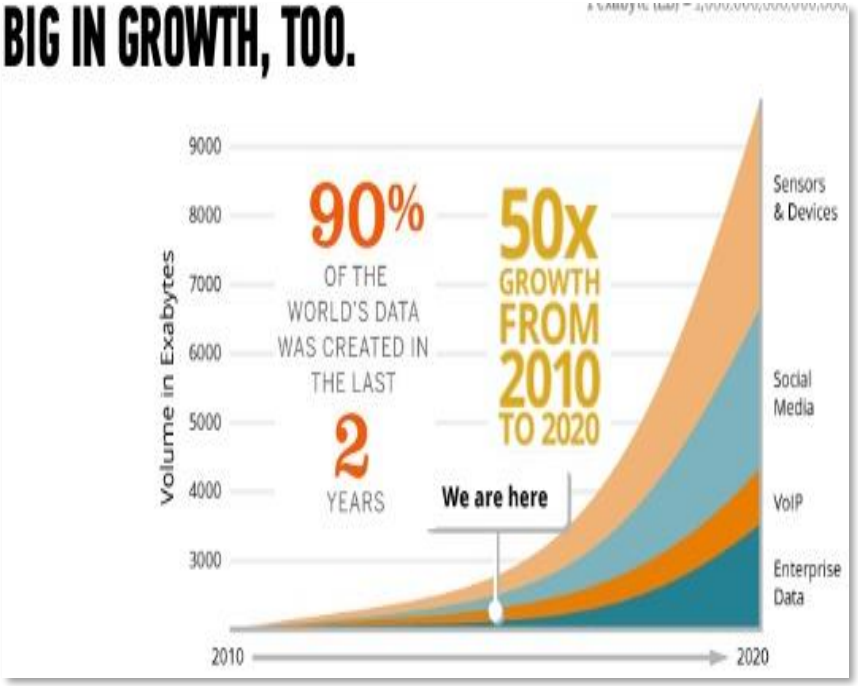
提纲

- 大数据处理系统发展
  - 集群化处理 → 基于内存的数据处理
- 大数据处理系统应用
  - MapReduce型范式 → DAG型范式
- 大数据处理系统剖析
  - Hadoop → Spark
- 异构硬件加速大数据系统

大数据处理系统发展

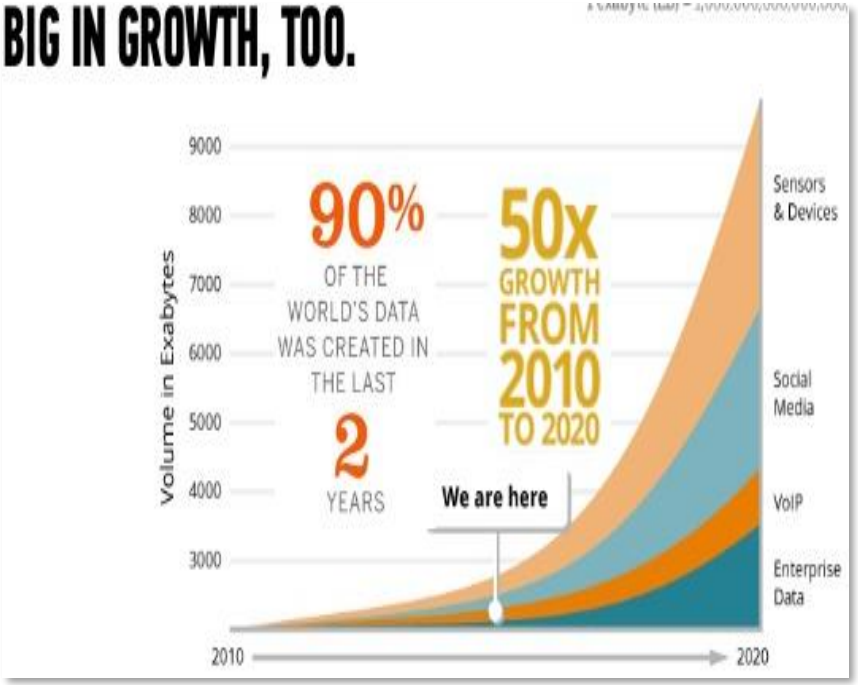
大数据处理需求兴起

- Google：大规模网页数据
  - 需求目标：存储和索引 （Page Rank）



大数据处理需求兴起

- Google：大规模网页数据
  - 需求目标：存储和索引 （Page Rank）

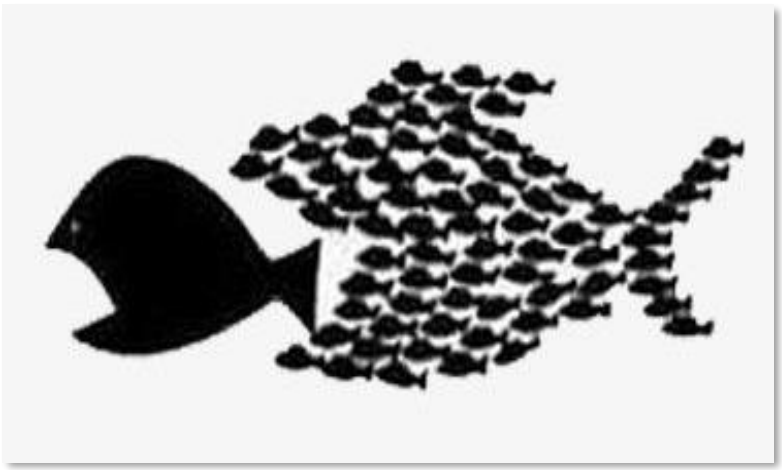


集群化数据处理

- Google：大规模网页数据
  - 需求目标：存储和索引
  - 解决方案：利用集群并行化处理数据 OSDI’04
    - Apache Hadoop ’08：MapReduce

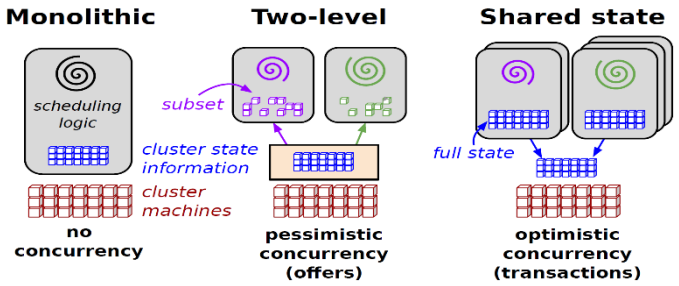
整合集群资源

任务并发执行



集群化数据处理

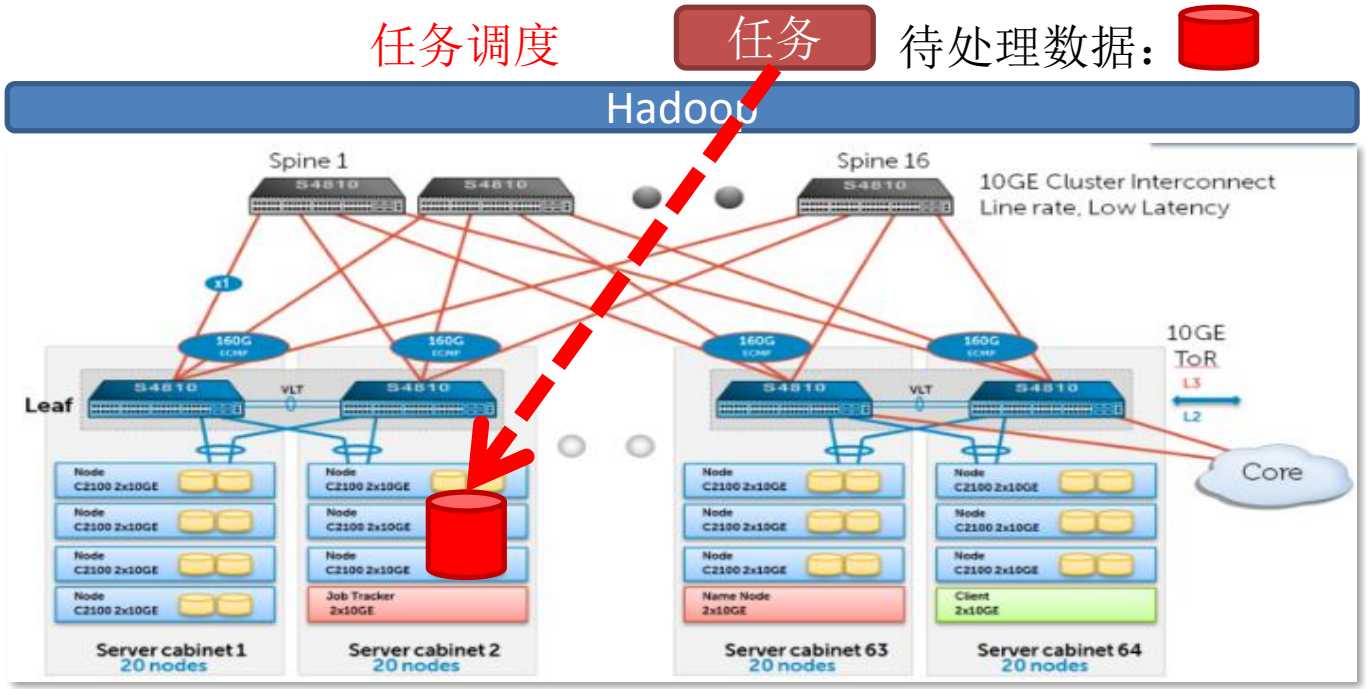
- Apache Hadoop ’08：MapReduce
  - 主从式管理模式
    - » 所有信息（资源+作业）由主节点管理/容错
    - » 百台服务器规模
  - 两层次管理模式：资源管理与任务容错分离开
    - » 支持上千台服务器



Omega: flexible, scalable schedulers for large compute clusters. EuroSys’ 13.

传统集群内大数据处理

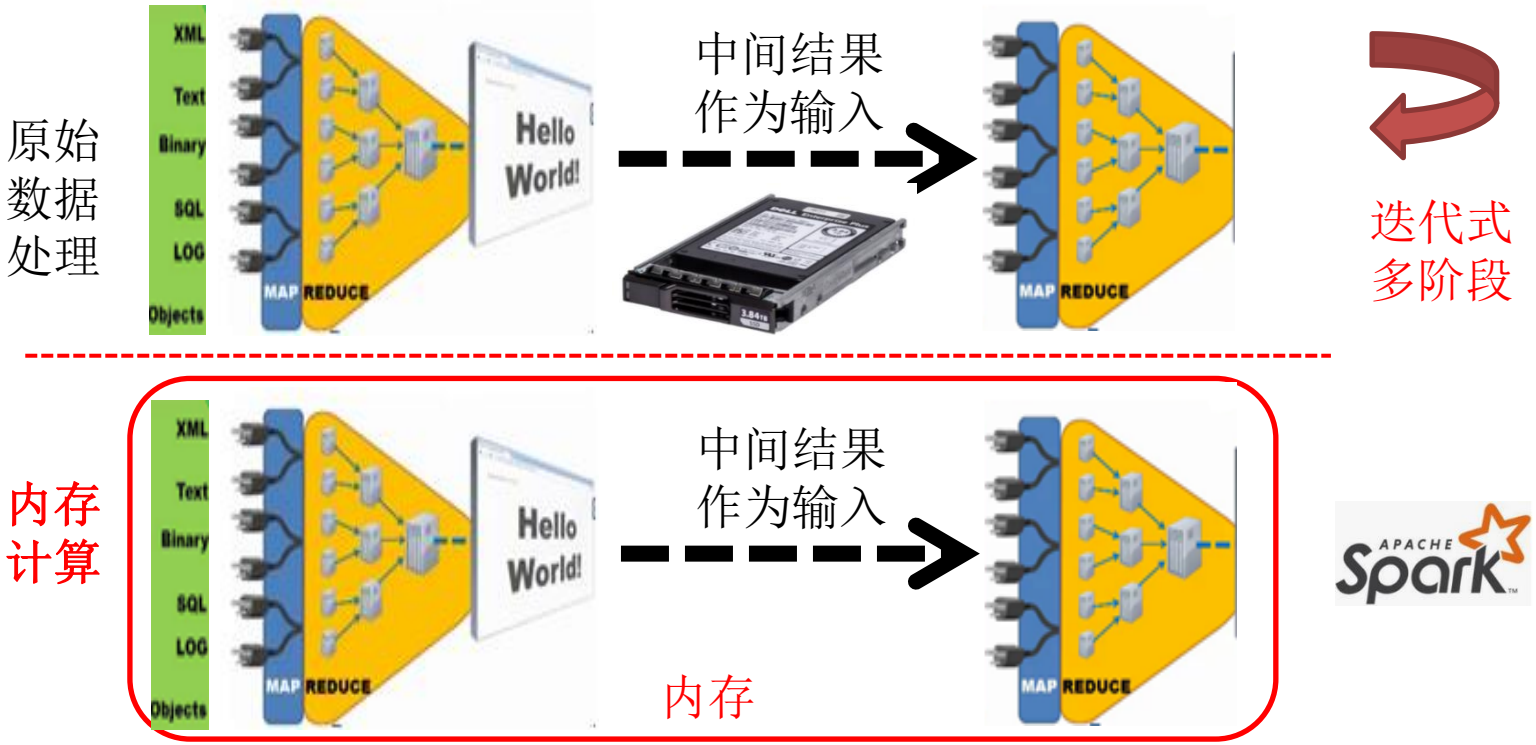
- 核心思想：利用数据本地性进行处理
  - 避免进行大规模数据移动EuroSys’ 10



M Zaharia et. al. Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. EuroSys’ 10.

集群大数据处理发展

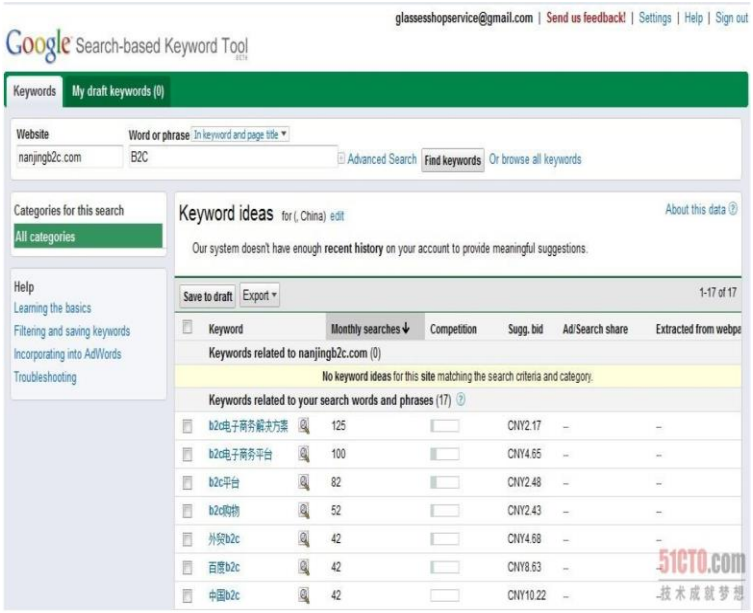
- Spark：避免频繁的磁盘操作
  - 基于内存的分布式弹性数据集RDD, NSDI’ 12

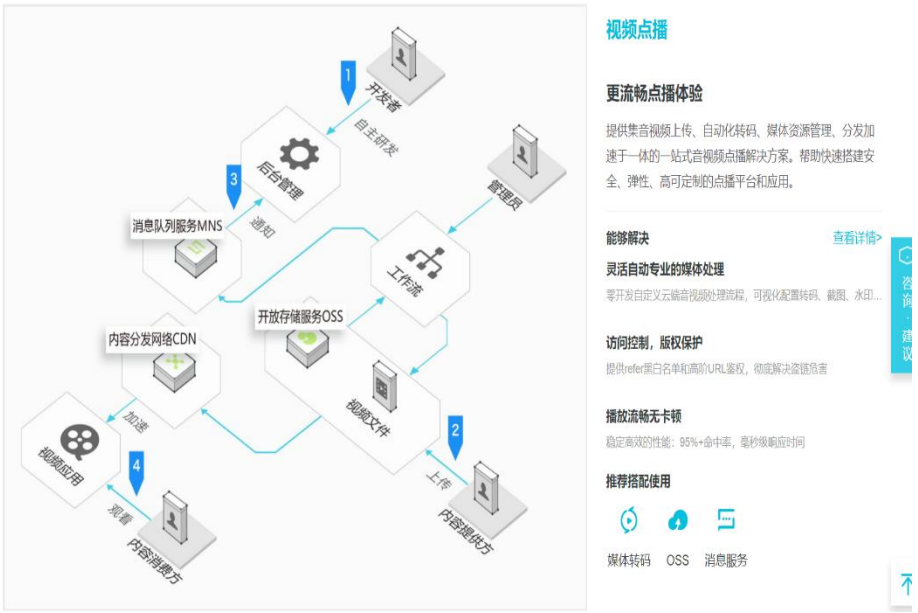


M Zaharia et. al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. NSDI’ 12.

跨域大数据处理需求

- 全球化业务的发展促进跨域数据处理需求
  - 基于区域的广告词竞价：Google AdWords
  - 跨域视频流处理：阿里云、优酷视频点播





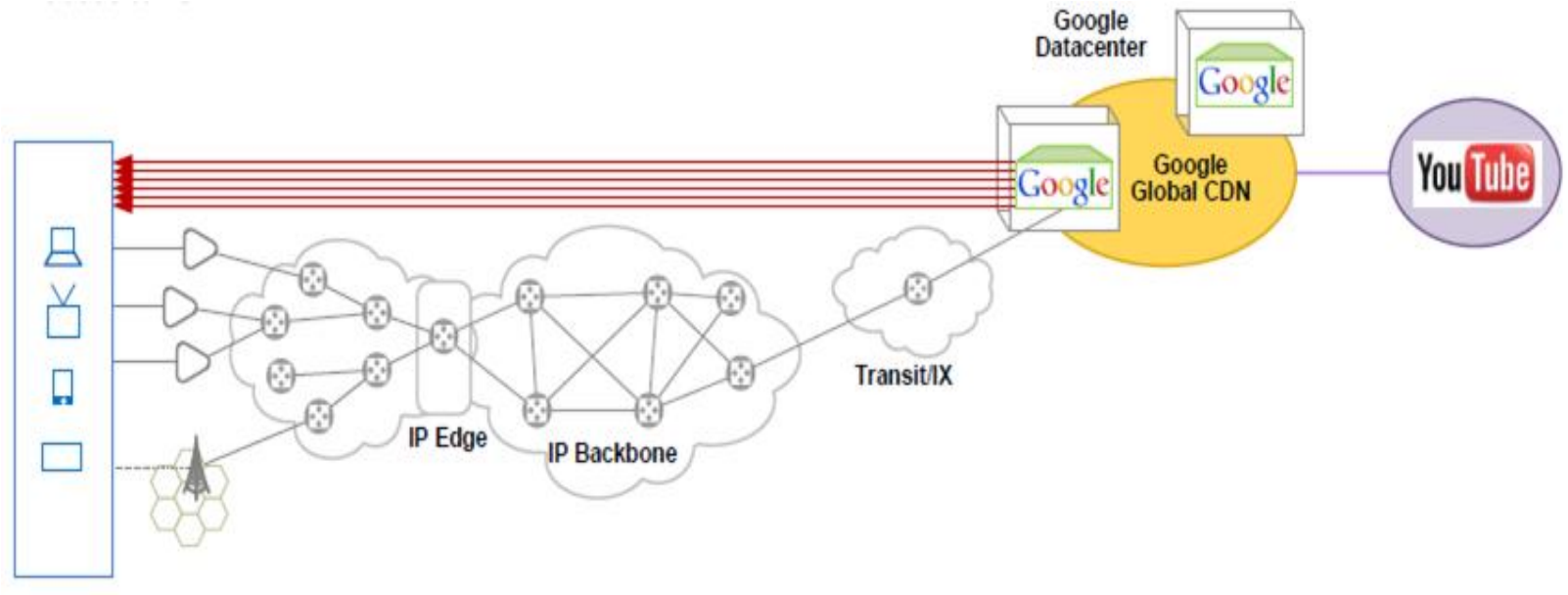
跨域集群现状

- 数据中心向边缘集群的演变

https://www.google.com/about/datacenters/inside/locations/

跨域集群现状

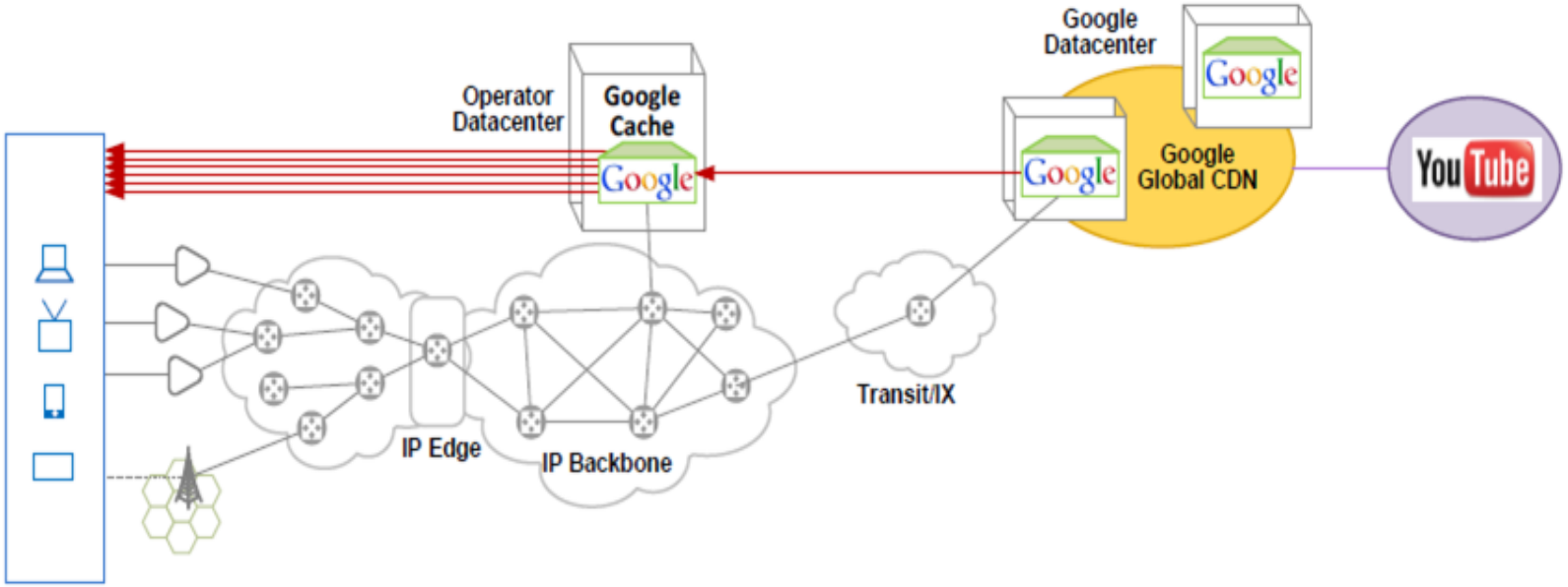
- 数据中心向边缘集群的演变
  - 跨域响应时间长



https://www.netmanias.com/en/post/blog/5921/cdn-google-iptv-netflix-ott-video-streaming-youtube/who-wins-ott-cache-vs-operator-cache

跨域集群现状

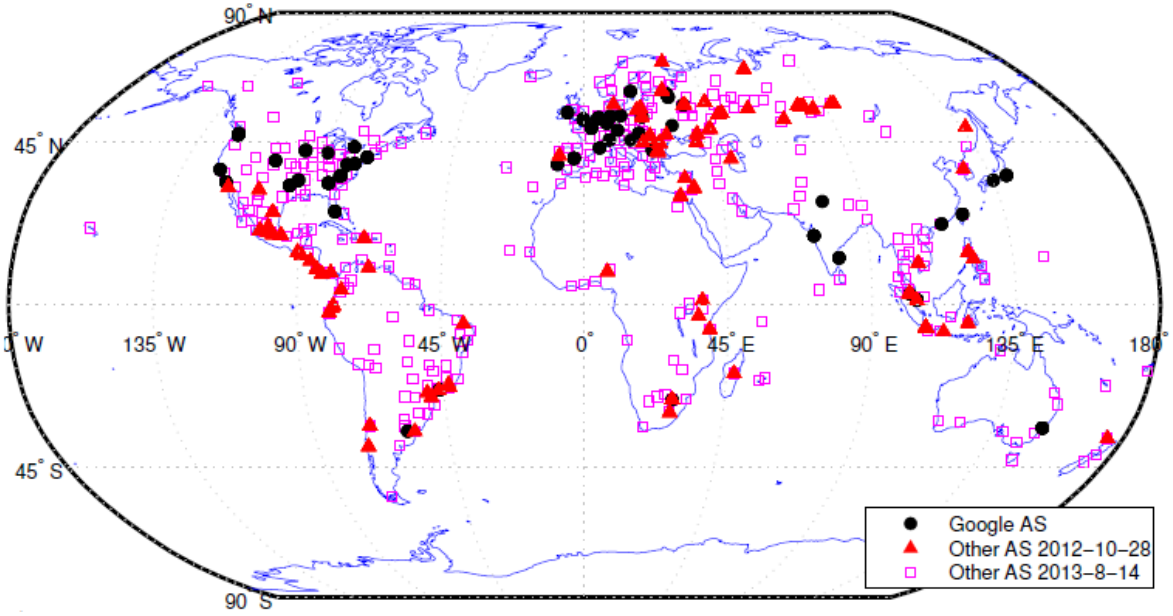
- 数据中心向边缘集群的演变
  - 互联网服务提供商部署边缘处理节点



https://www.netmanias.com/en/post/blog/5921/cdn-google-iptv-netflix-ott-video-streaming-youtube/who-wins-ott-cache-vs-operator-cache

跨域集群现状

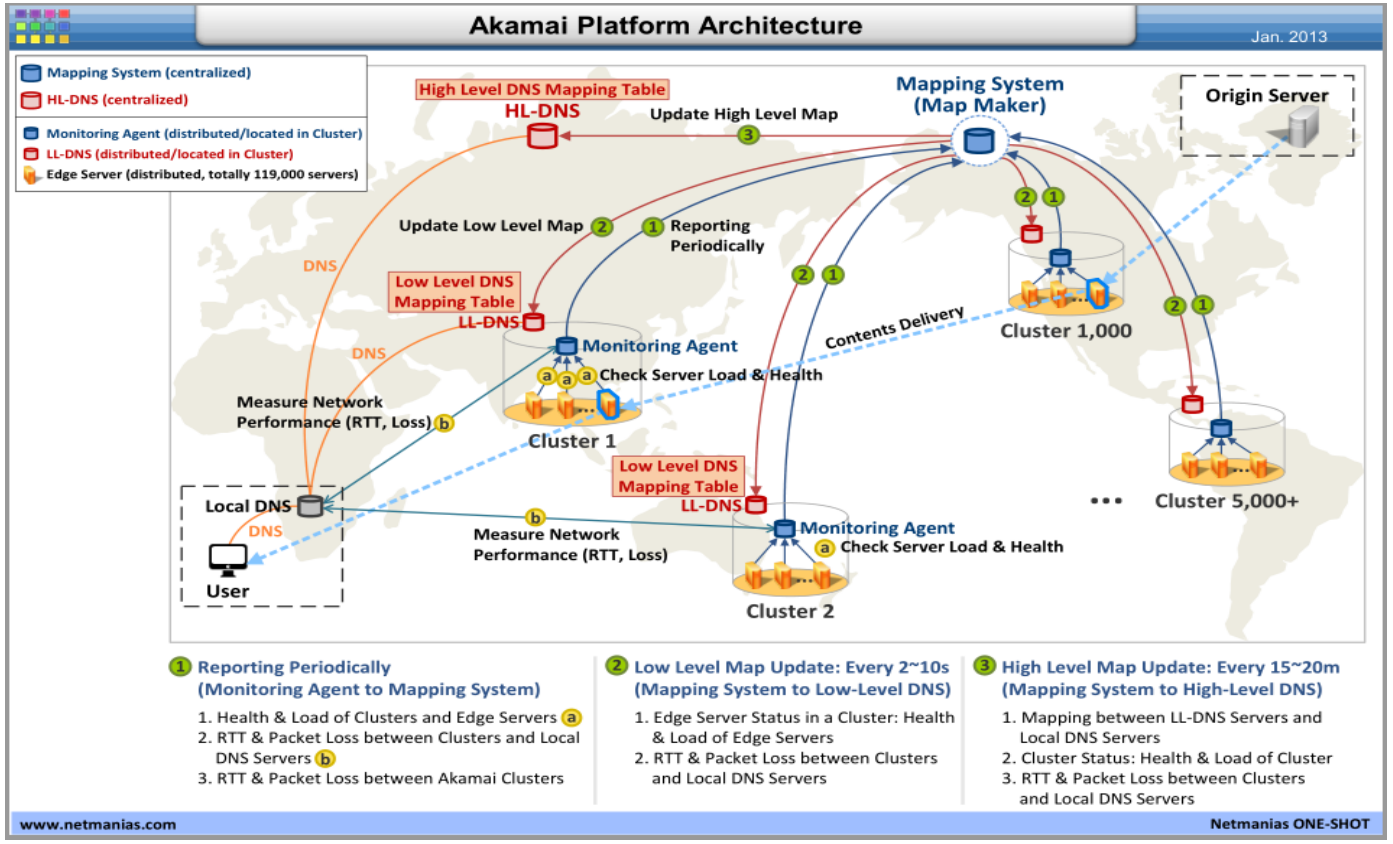
- 数据中心向边缘集群的演变
  - 数以千计的边缘集群协同数据中心提供服务



Calder *et. al*. Mapping the Expansion of Google's Serving Infrastructure. IMC' 13

基于边缘集群的处理框架

- Akamai：边缘集群间的数据收集和分析

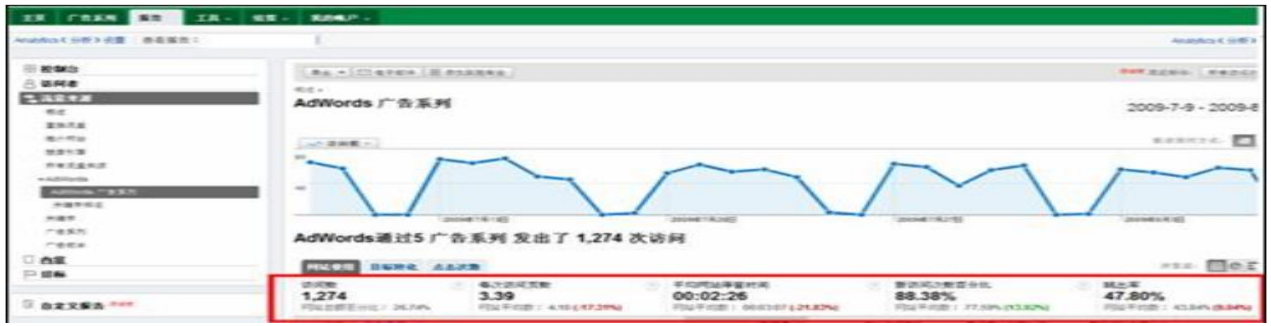


边缘集群产生大量数据

- 例如：用于商业广告竞价的点击数据
  - AdWords：利用该数据进行广告竞价和收费

```
ClickLog(sourceIP,destURL,visitDate,adRevenue,...)
```

```
Q: SELECT sourceIP, sum(adRevenue), avg(pageRank)
   FROM ClickLog cl JOIN PageInfo pi
     ON cl.destURL = pi.pageURL
   WHERE pi.pageCategory = 'Entertainment'
   GROUP BY sourceIP
   HAVING sum(adRevenue) >= 100
```

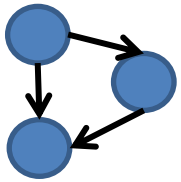


跨域大数据处理框架

- 跨域数据处理架构

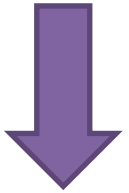
```
Q: SELECT sourceIP, sum(adRevenue), avg(pageRank)
   FROM ClickLog cl JOIN PageInfo pi
     ON cl.destURL = pi.pageURL
   WHERE pi.pageCategory = 'Entertainment'
   GROUP BY sourceIP
   HAVING sum(adRevenue) >= 100
```

数据分析查询

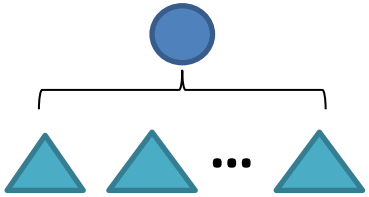


数据分析作业

Frameworks：Spark/Hadoop



跨域数据集



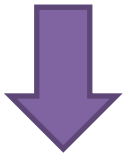
批次数据分析任务

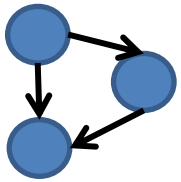
跨域大数据处理框架

- 跨域数据处理架构

```
Q: SELECT sourceIP, sum(adRevenue), avg(pageRank)
   FROM ClickLog cl JOIN PageInfo pi
     ON cl.destURL = pi.pageURL
   WHERE pi.pageCategory = 'Entertainment'
   GROUP BY sourceIP
   HAVING sum(adRevenue) >= 100
```

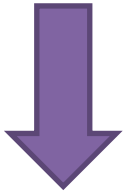
数据分析查询



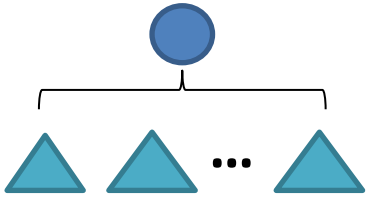


数据分析作业

Frameworks：Spark/Hadoop



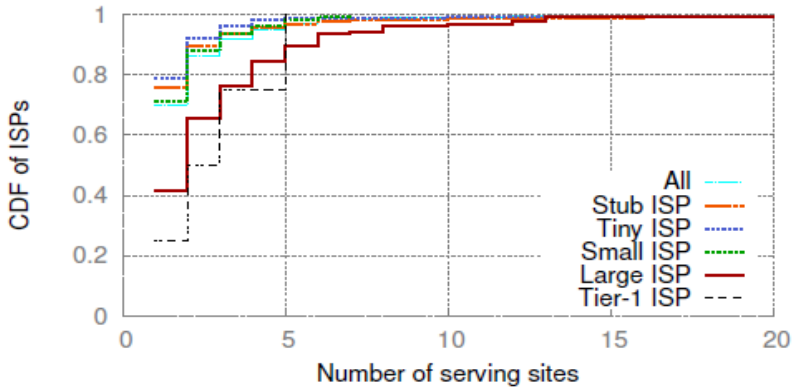
跨域数据集

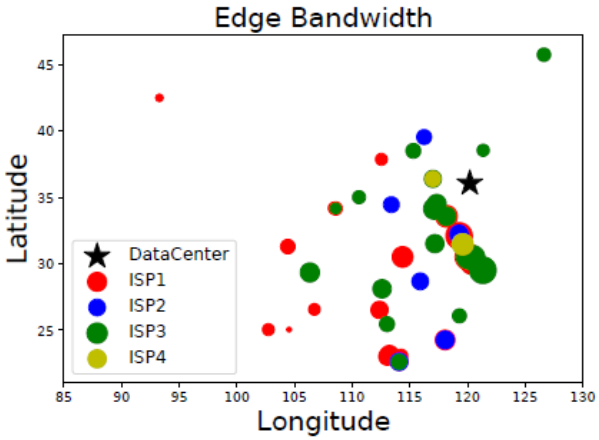


减少批次任务
完成时间

跨域边缘集群的异构性

- 资源异构性
  - 带宽资源
  - 计算资源



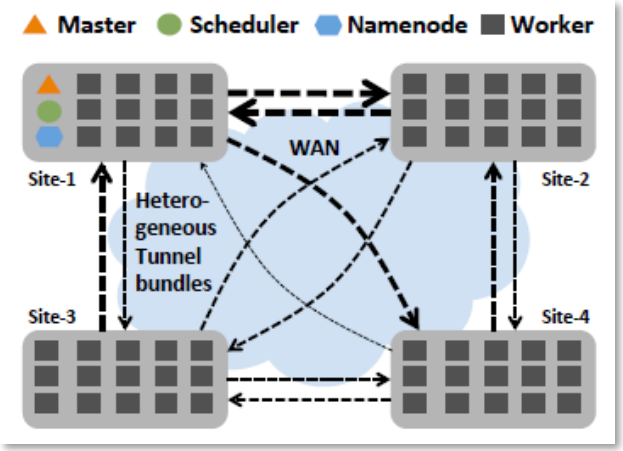


Performance		Worst	Best
Server	Memory	GB	TB, DDR
	CPU (Cores)	Few	Hundreds
	Disk	GB	TB, SSD

Calder *et. al*. Mapping the Expansion of Google's Serving Infrastructure. IMC' 13
Alibaba Cloud Computing, and Chinaz for measurement.

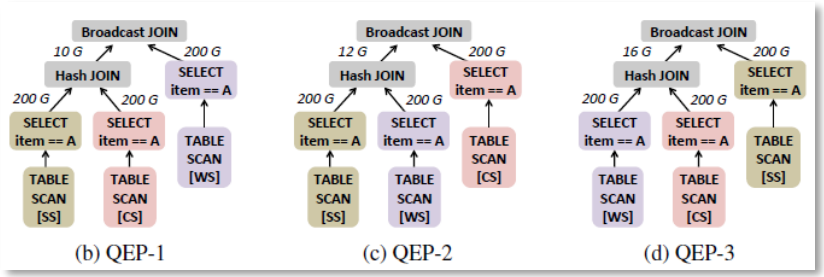
集群间数据处理

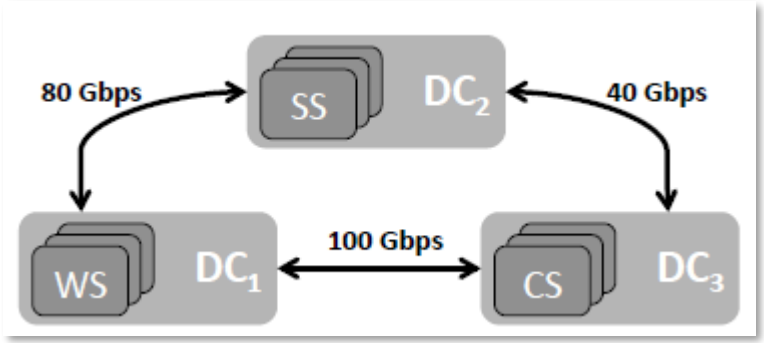
- 跨域资源异构性大
  - 需要综合计算力 & 带宽进行资源部署



Performance		Worst	Best
Server	Memory	GB	TB, DDR3/4
	CPU	Several Cores	Hundreds Cores
	Disk	GB	TB, SSD

(b) Edges with heterogeneous computing capacities.





V Raajay et. al. Clarinet: WAN-Aware Optimization for Analytics Queries. OSDI' 16.

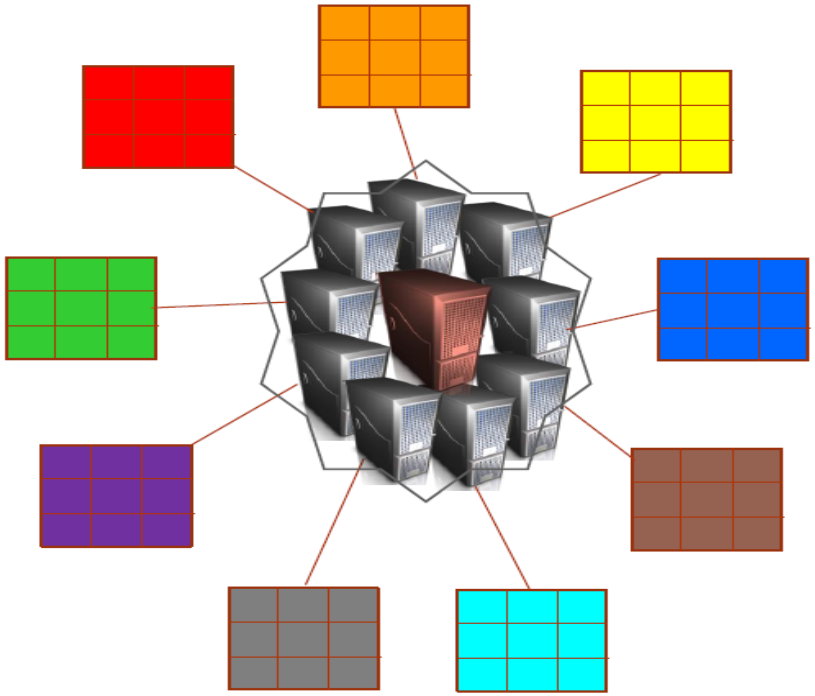
大数据处理生态

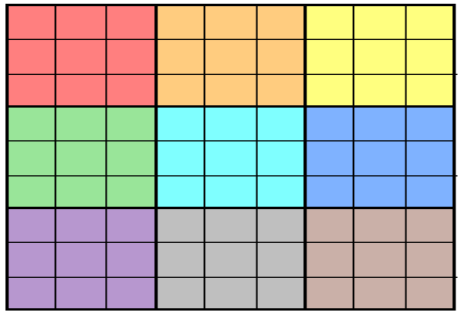

MAPREDUCE (Processing using different languages)
HIVE & DRILL (Analytical SQL-on-Hadoop)
MAHOUT & SPARK MLlib (Machine learning)
PIG (Scripting)
HBASE (NoSQL Database)
ZOOKEEPER & AMBARI (Management & Coordination)
SPARK (In-Memory, Data Flow Engine)
KAFKA & STORM (Streaming)
SOLR & LUCENE (Searching & Indexing)
OOZIE (Scheduling)
Resource Management
YARN
Storage
HDFS
Apache Ambari

大数据处理系统应用

大数据处理应用

- MapReduce范式



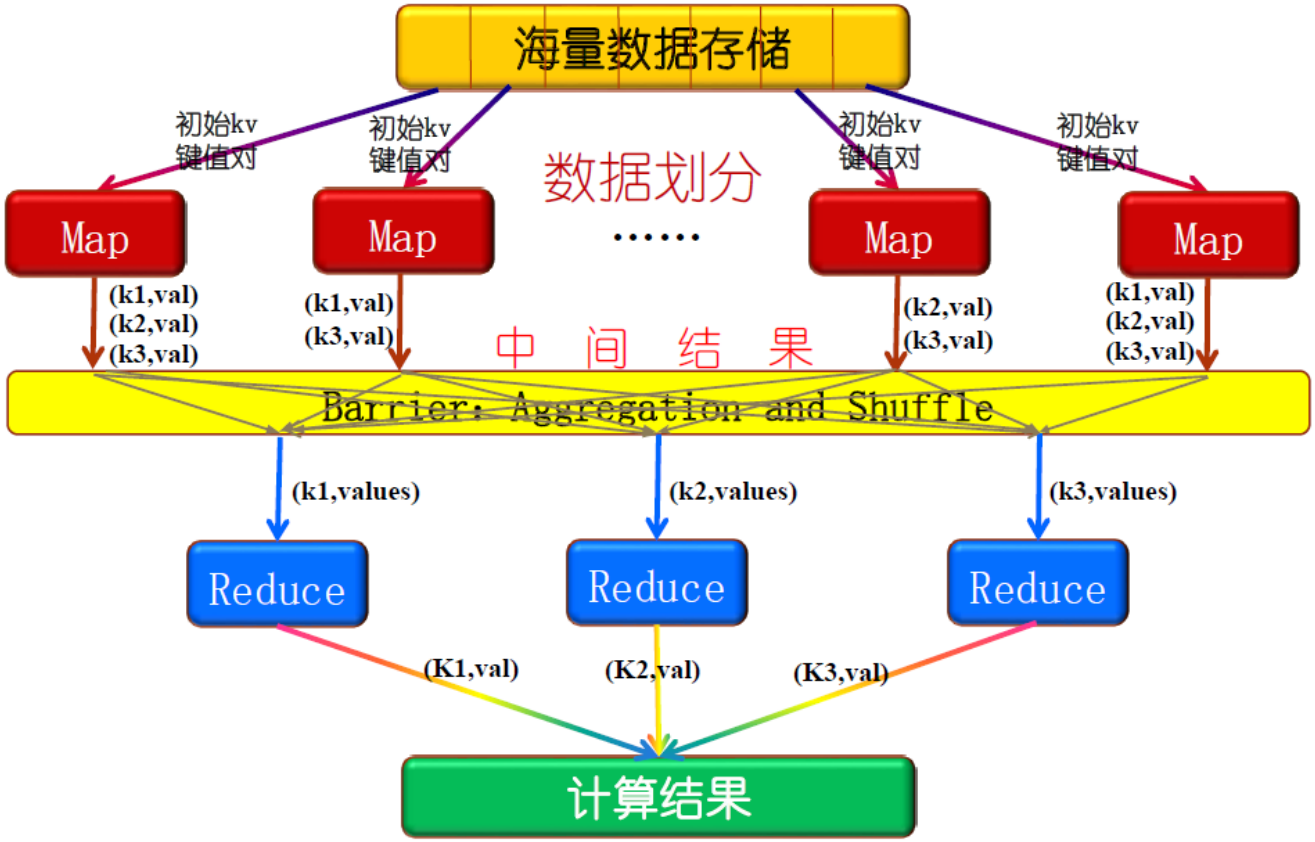


Master：负责划分和分配任务

Workder：负责数据块计算

大数据处理应用

- MapReduce范式



大数据处理应用

基于MapReduce的处理过程示例——文档词频统计：WordCount

设有4组原始文本数据：

Text 1: the weather is good　　Text 2: today is good

Text 3: good weather is good　　Text 4: today has good weather

传统的串行处理方式(Java)：

```
String[] text = new String[]
 { "the weather is good","today is good ","good weather is good "," today has good weather" } ;
HashTable ht = new HashTable();
for(i=0; i<3; ++i)
{  StringTokenizer st = new StringTokenizer(text[i]);
   while (st.hasMoreTokens())
   {  String word = st.nextToken();
      if(!ht.containsKey(word)) {  ht.put(word, new Integer(1));  }
      else { int wc = ((Integer)ht.get(word)).intValue() +1;// 计数加1
             ht.put(word, new Integer(wc));
           }
   }
}
for (Iterator itr=ht.KeySet().iterator();  itr.hasNext(); )
{ String word = (String)itr.next();
  System.out.print(word+ ": "+ (Integer)ht.get(word)+";   ");
}
```

输出：　good: 5;　has: 1;　is: 3;　the: 1;　today: 2;　weather: 3

大数据处理应用

构建抽象模型：Map与Reduce

基于MapReduce的处理过程示例——文档词频统计：WordCount

MapReduce处理方式

使用4个map节点：

map节点1:

输入：(text1, “the weather is good”)

输出：(the, 1), (weather, 1), (is, 1), (good, 1)

map节点2:

输入：(text2, “today is good”)

输出：(today, 1), (is, 1), (good, 1)

map节点3:

输入：(text3, “good weather is good”)

输出：(good, 1), (weather, 1), (is, 1), (good, 1)

map节点4:

输入：(text3, “today has good weather”)

输出：(today, 1), (has, 1), (good, 1), (weather, 1)

大数据处理应用

构建抽象模型：Map与Reduce

基于MapReduce的处理过程示例--文档词频统计：WordCount

MapReduce处理方式

使用3个reduce节点：

reduce节点1:

输入：(good, 1), (good, 1), (good, 1), (good, 1), (good, 1)

输出：(good, 5)

reduce节点2:

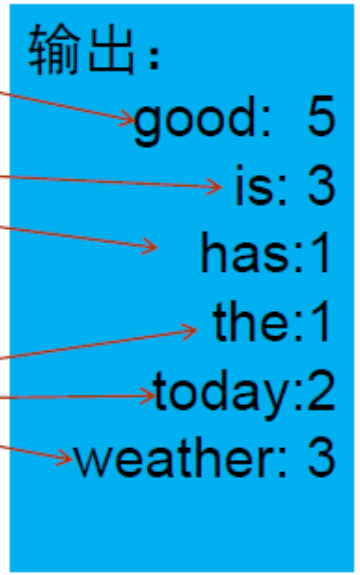
输入：(has, 1), (is,1), (is,1), (is, 1),

输出：(has, 1), (is, 3)

reduce节点3:

输入：(the, 1), (today, 1), (today, 1)
(weather, 1), (weather,1), (weather,

输出：(the, 1), (today, 2), (weather, 3)



大数据处理应用

构建抽象模型：Map与Reduce

基于MapReduce的处理过程示例--文档词频统计：WordCount

MapReduce处理方式

MapReduce伪代码（实现Map和Reduce两个函数）：

```
Class WordCountMapper
  method map(String input_key, String input_value):
    // input_key: text document name
    // input_value: document contents
    for each word w in input_value:
      EmitIntermediate(w, "1");
```

```
Class WordCountReducer
  method reduce(String output_key,
                Iterator intermediate_values):
    // output_key: a word
    // output_values: a list of counts
    int result = 0;
    for each v in intermediate_values:
      result += ParseInt(v);
    Emit(output_key,  result);
```

大数据处理应用

构建抽象模型：Map与Reduce

基于MapReduce的处理过程示例--文档词频统计：WordCount

MapReduce处理方式

使用3个reduce节点：

reduce节点1:

输入：(good, 1), (good, 1), (good, 1), (good, 1), (good, 1)

输出：(good, 5)

reduce节点2:

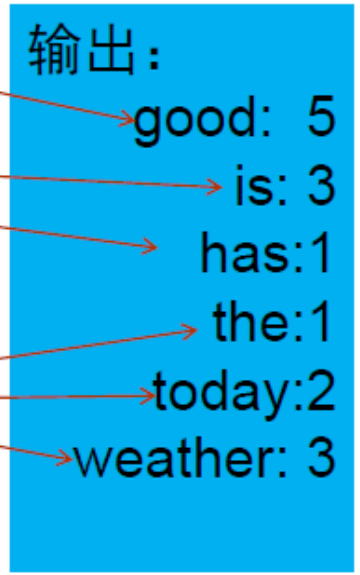
输入：(has, 1), (is,1), (is,1), (is, 1),

输出：(has, 1), (is, 3)

reduce节点3:

输入：(the, 1), (today, 1), (today, 1)
(weather, 1), (weather,1), (weather,

输出：(the, 1), (today, 2), (weather, 3)



has + the 共有几个？

大数据处理应用

- 自定义Reducer划分

定制Partitioner

程序员可以根据需要定制Partitioner来改变Map中间结果到Reduce节点的分区方式，并在Job中设置新的Partitioner

```
Class NewPartitioner extends HashPartitioner<K,V>
{  // override the method
    getPartition(K key, V value, int numReduceTasks)
    {  term = key. toString().split(“,”)[0]; //<term, docid>=>term
        super.getPartition(term, value, numReduceTasks);
    }
}
```

并在Job中设置新的Partitioner：

```
Job. setPartitionerClass(NewPartitioner)
```

大数据处理应用

- 自定义键值对：key-value

doc1：
one fish
two fish

doc2：
red fish
blue fish

doc3：
one red bird

倒排索引：
one: doc1, doc3
fish: doc1, doc2
two: doc1
red: doc2, doc3
blue: doc2
bird: doc3

基于以上索引的搜索结果：
fish → doc1, doc2
red → doc2, doc3
red fish → doc2

大数据处理应用

- 自定义键值对：key-value

简单的文档倒排算法

改进:map输出的key除了文件名，还给出了该词所在行的偏移值:
格式： filename#offset

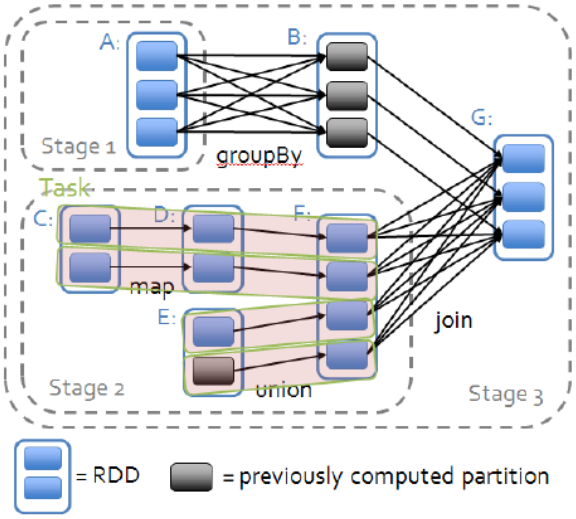
```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class InvertedIndexMapper extends Mapper<Text, Text, Text, Text>
{       @Override
        protected void map(Text key, Text value, Context context)
                        throws IOException, InterruptedException
        // default RecordReader: LineRecordReader; key: line offset; value: line string
        {       FileSplit fileSplit = (FileSplit)context.getInputSplit();
                String fileName = fileSplit.getPath().getName();
                Text word = new Text();
                Text fileName_lineOffset = new Text(fileName+"#"+key.toString());
                StringTokenizer itr = new StringTokenizer(value.toString());
                for(; itr.hasMoreTokens(); )
                {   word.set(itr.nextToken());
                    context.write(word, fileName_lineOffset);
                }
        }
}
```

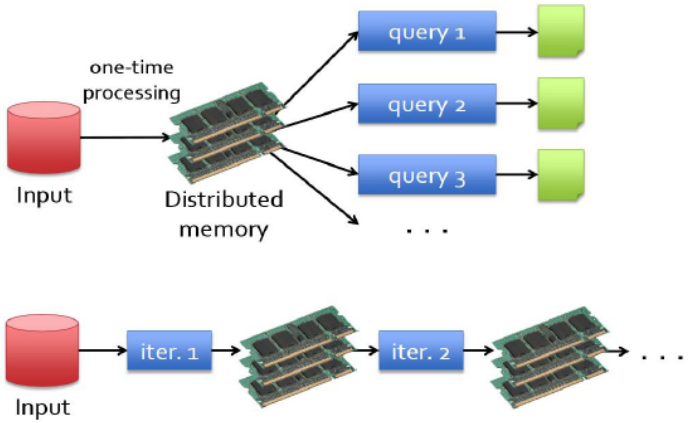
大数据处理应用

- DAG型计算范式：MapReduce的拓展

Spark基于内存计算思想提高计算性能

- Spark提出了一种基于内存的弹性分布式数据集(RDD)，通过对RDD的一系列操作完成计算任务，可以大大提高性能
- 同时一组RDD形成可执行的有向无环图DAG，构成灵活的计算流图
- 覆盖多种计算模式





大数据处理应用

- DAG型计算范式：MapReduce的拓展

Spark的基本编程方法与示例

//在一个存储于HDFS的Log文件中，计算出现ERROR的行数，本程序使用Scala语言编写，这个语言也是Spark开发和编程的推荐语言。

```
def main(args: Array[String]) {   //定义一个main函数
    val conf = new SparkConf().setAppName("Spark Pi")   //定义一个sparkConf，
        提供Spark运行的各种参数，如程序名称、用户名称等
    val sc = new SparkContext(conf)   //创建Spark的运行环境，并将Spark运行的
        参数传入Spark的运行环境中
    val fileRDD=sc.textFile("hdfs:///root/Log")  //调用Spark的读文件函数，从HDFS
        中读取Log文件，输出一个RDD类型的实例：fileRDD。具体类型：
        RDD[String]
    val filterRDD=fileRDD.filter(line=>line.contains("ERROR"))  //调用RDD的filter函
        数，过滤fileRDD中的每一行，如果该行中含有ERROR，保留；否则，删
        除。生成另一个RDD类型的实例：filterRDD。具体类型:RDD[String]
        注：line=>line.contains("ERROR")表示对每一个line应用contains()函数
    val result =  filterRDD.count()  //统计filterRDD中总共有多少行，result为Int类型
    sc.stop() //关闭Spark
```

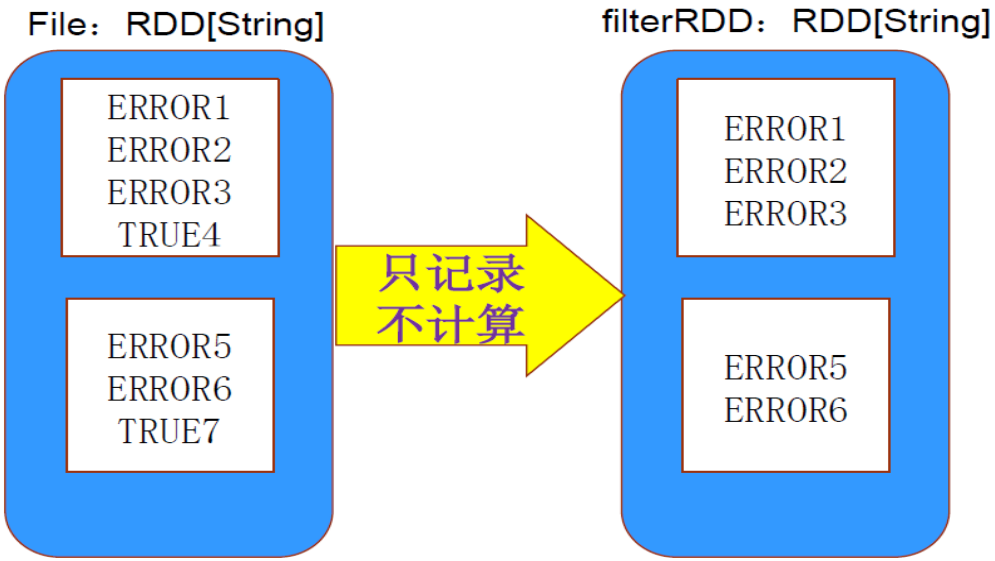
大数据处理应用

- DAG型计算范式：MapReduce的拓展

RDD的transformation示例

例： val filterRDD=fileRDD.filter(line=>line.contains("ERROR"))

设fileRDD中包含以下7行数据：
"ERROR1 ERROR2 ERROR3 TRUE4 ERROR5 ERROR6 TRUE7"

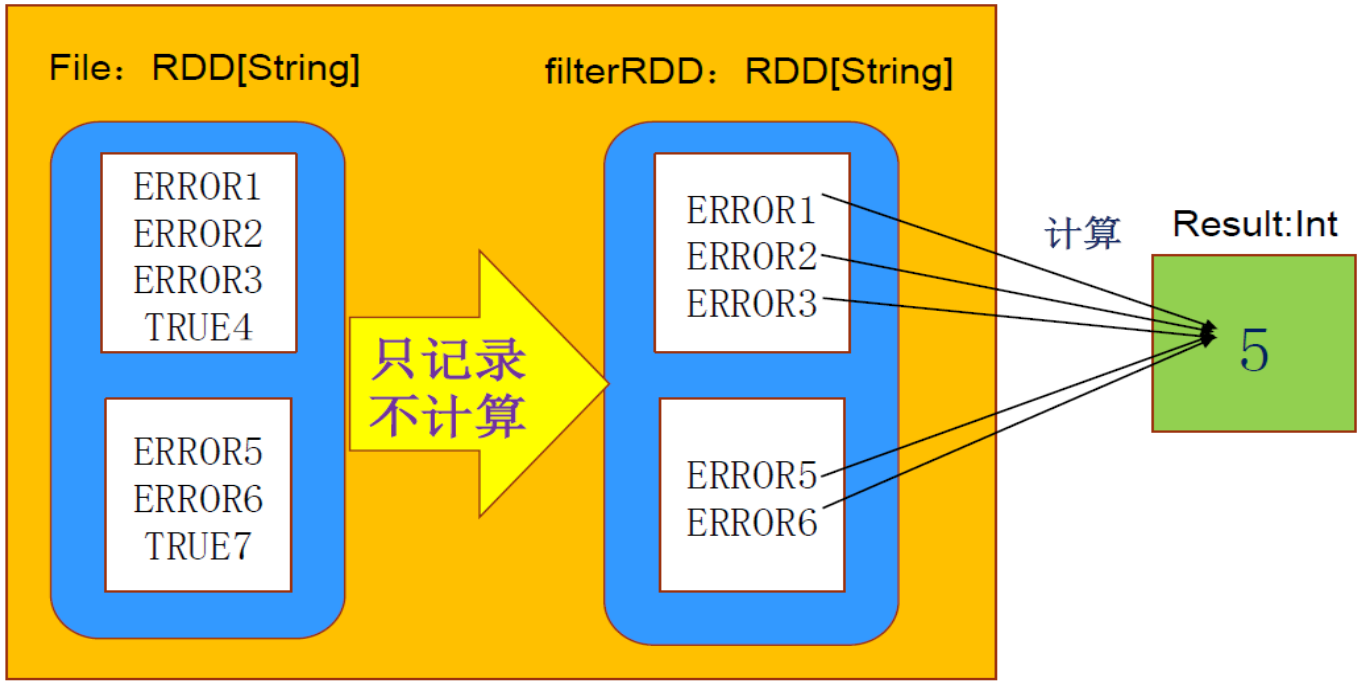


大数据处理应用

- DAG型计算范式：MapReduce的拓展

RDD的action图示

例： val result = filterRDD.count()



大数据处理应用

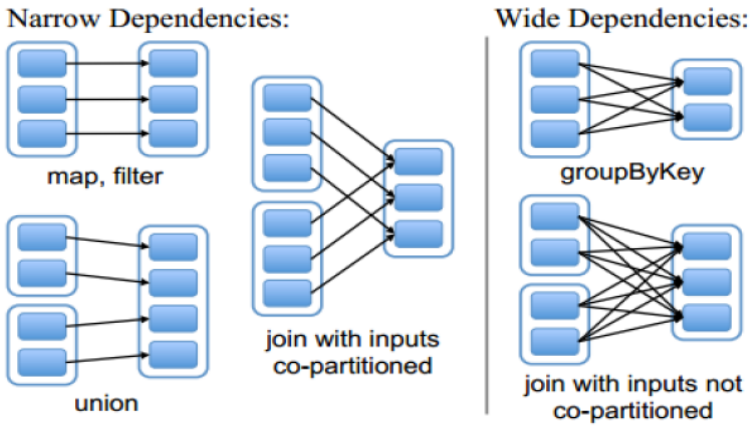
- DAG型计算范式：MapReduce的拓展

RDD之间的依赖关系

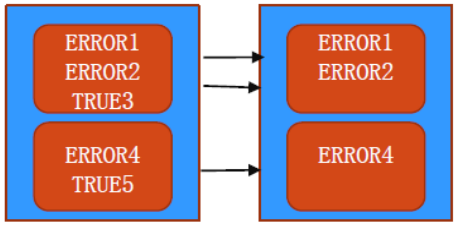
- 在Spark中存在两种类型的依赖：

➢ 窄依赖：父RDD中的一个Partition最多被子RDD中的一个Partition所依赖。

➢ 宽依赖：父RDD中的一个Partition被子RDD中的多个Partition所依赖。



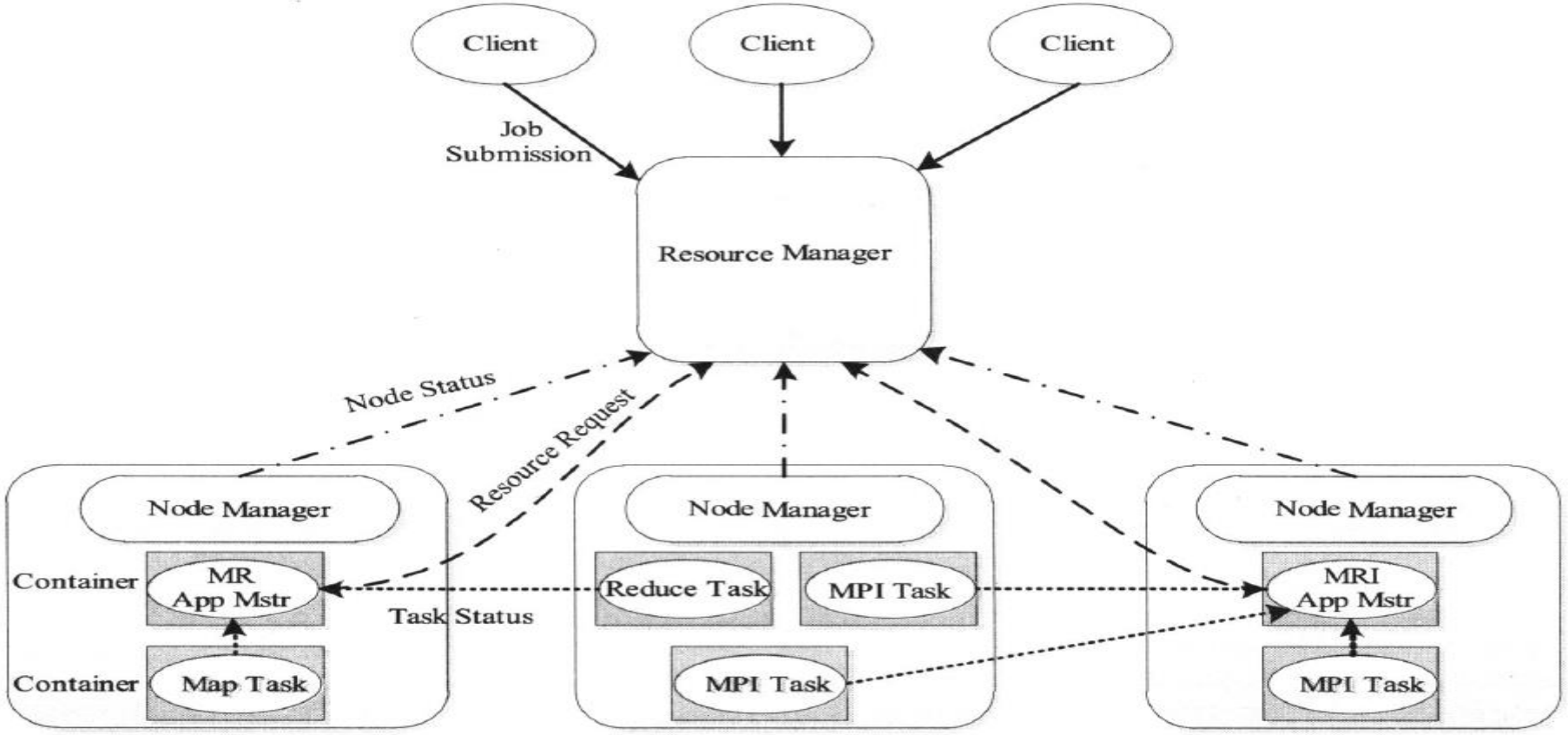
例子：
val filterRDD=fileRDD.filter
(line=>line.contains("ERROR"))
是一种窄依赖



大数据处理系统剖析

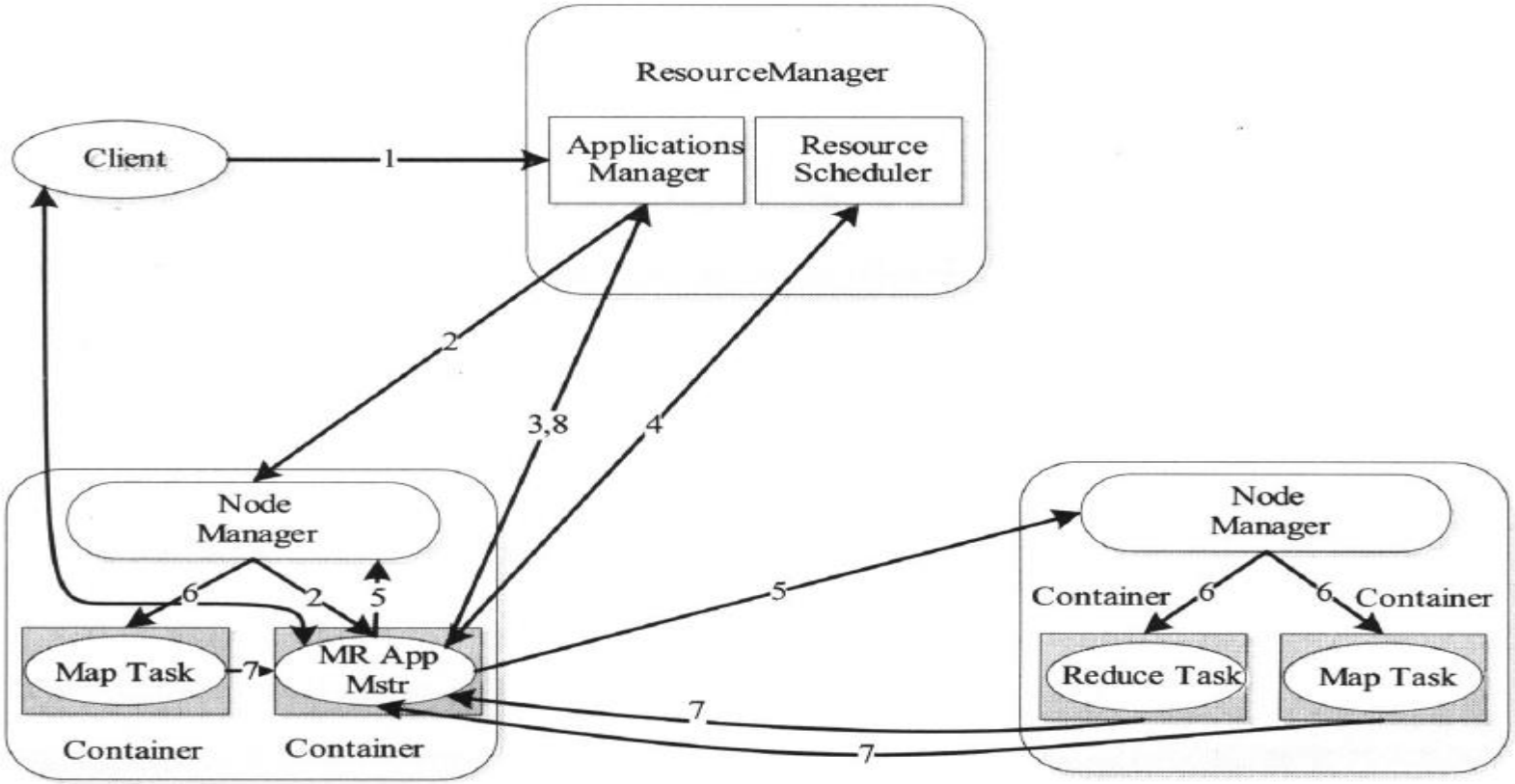
大数据处理系统剖析

- 集群化资源管理Hadoop-Yarn



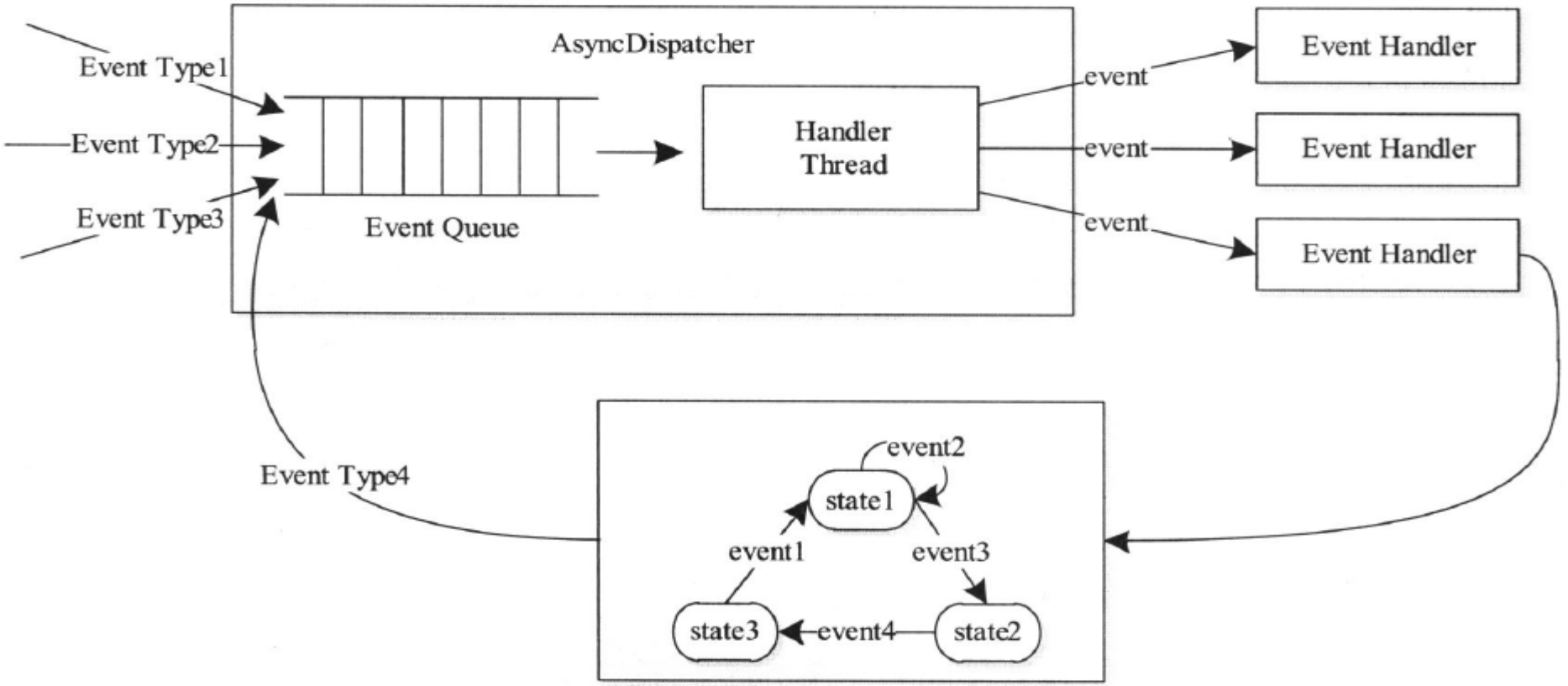
大数据处理系统剖析

- Hadoop资源分配流程



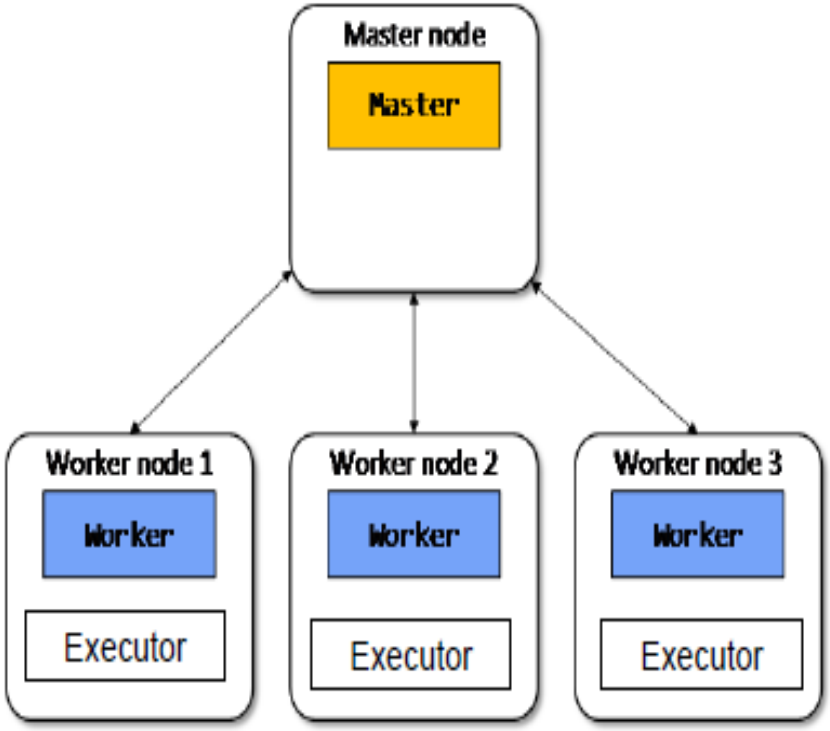
大数据处理系统剖析

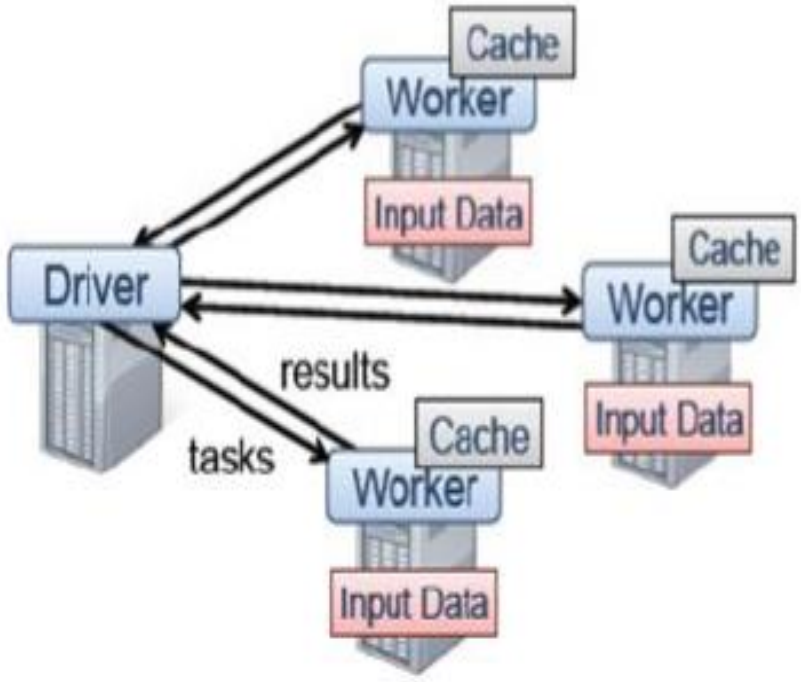
- Hadoop事件处理模块



大数据处理系统剖析

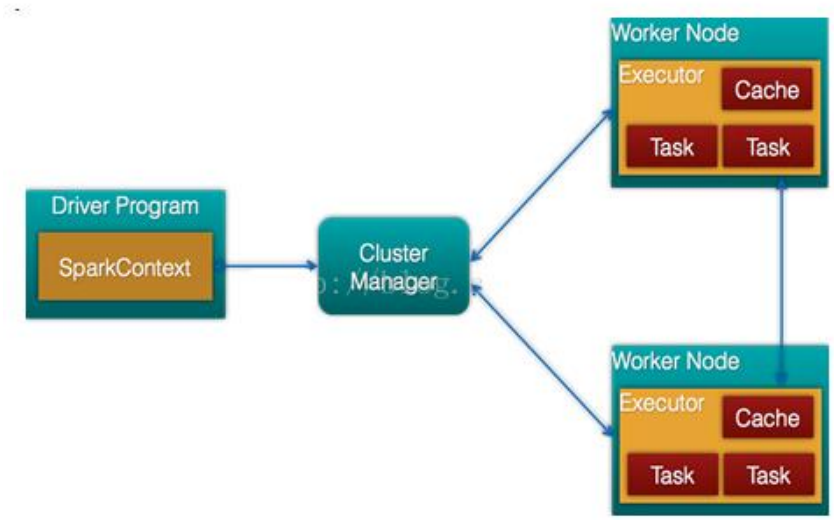
- 集群化资源管理Spark

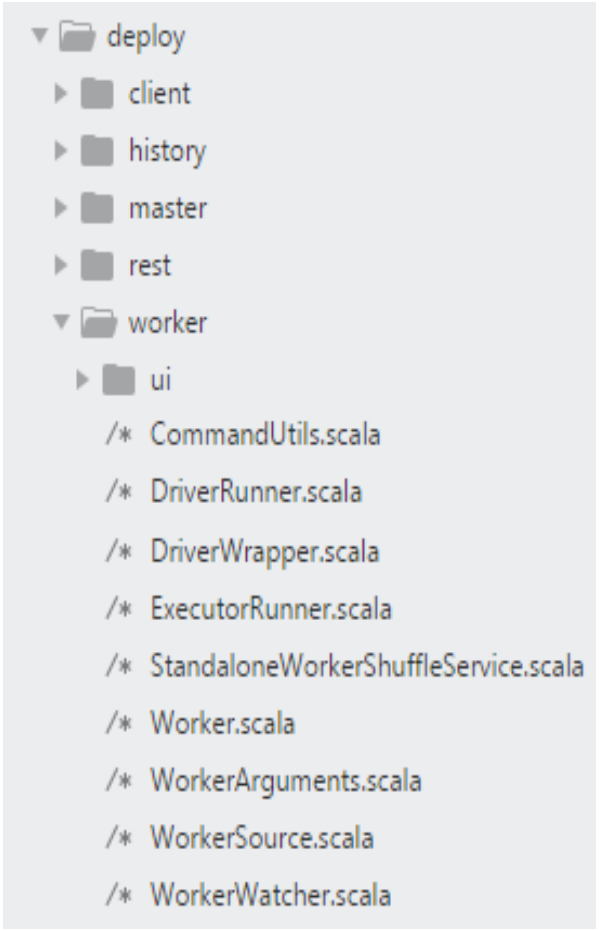




大数据处理系统剖析

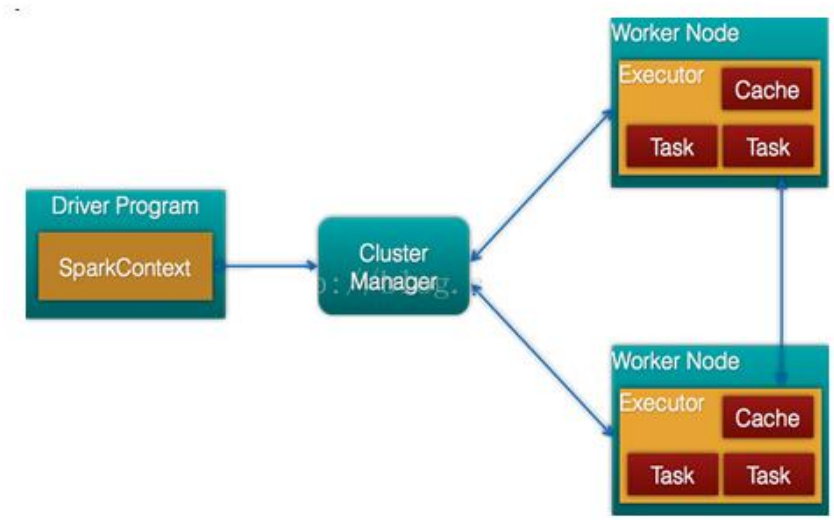
- 集群化资源管理Spark
  - 组件 + 通信

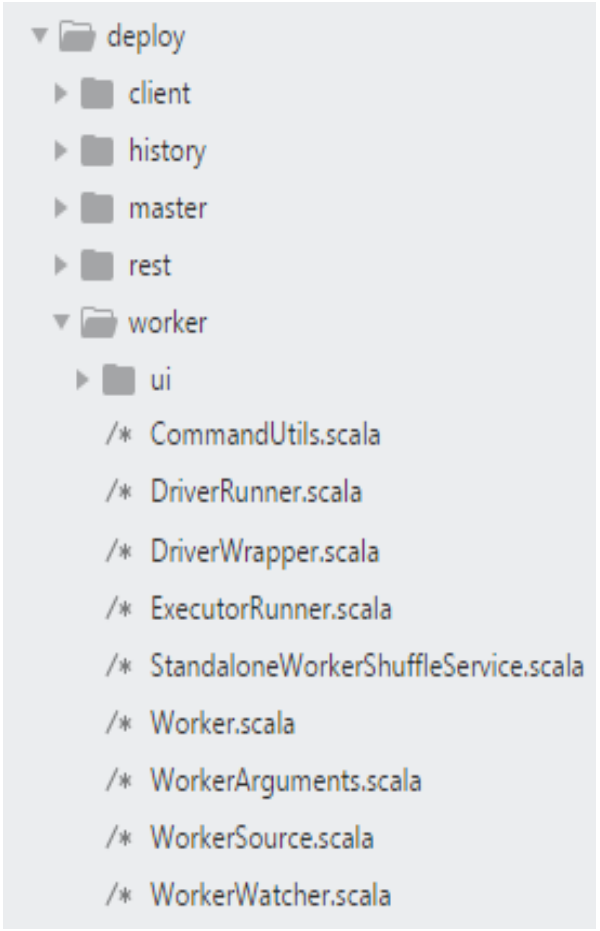




大数据处理系统剖析

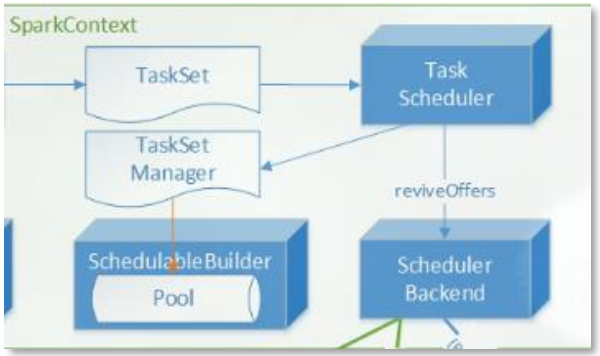
- 集群化资源管理Spark
  - 组件 + 通信





大数据处理系统剖析

- 集群化资源管理Spark
  - 组件 + 通信



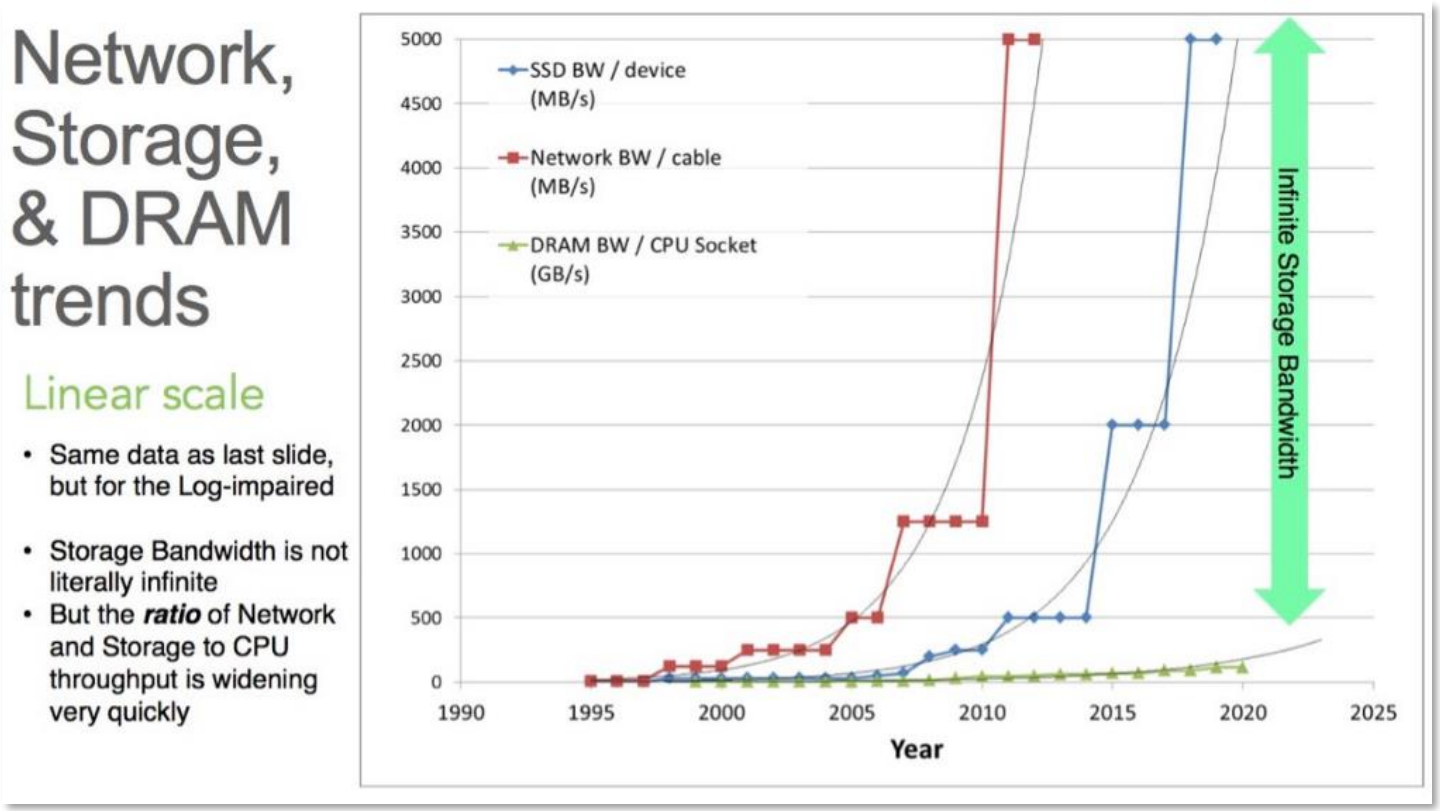

```
88      def receiveWithLogging = {
89        case RegisterExecutor(executorId, hostPort, cores, logUrls) => ...
115
116       case StatusUpdate(executorId, taskId, state, data) => ...
129
130       case ReviveOffers =>
131         makeOffers()
132
133       case KillTask(taskId, executorId, interruptThread) => ...
141
142       case StopDriver => ...
145
146       case StopExecutors => ...
152
153       case RemoveExecutor(executorId, reason) => ...
156
157       case DisassociatedEvent(_, address, _) => ...
160
161       case RetrieveSparkProps =>
162         sender ! sparkProperties
163     }
```

谢谢大家！

异构硬件加速大数据系统

异构硬件数据处理背景

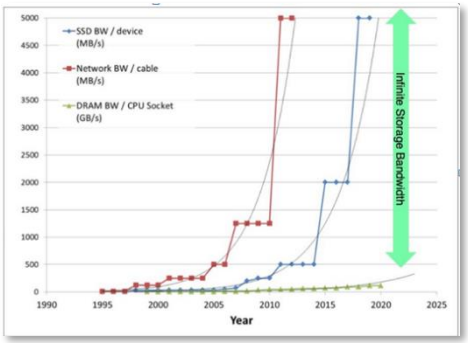
- 异构硬件突飞猛进

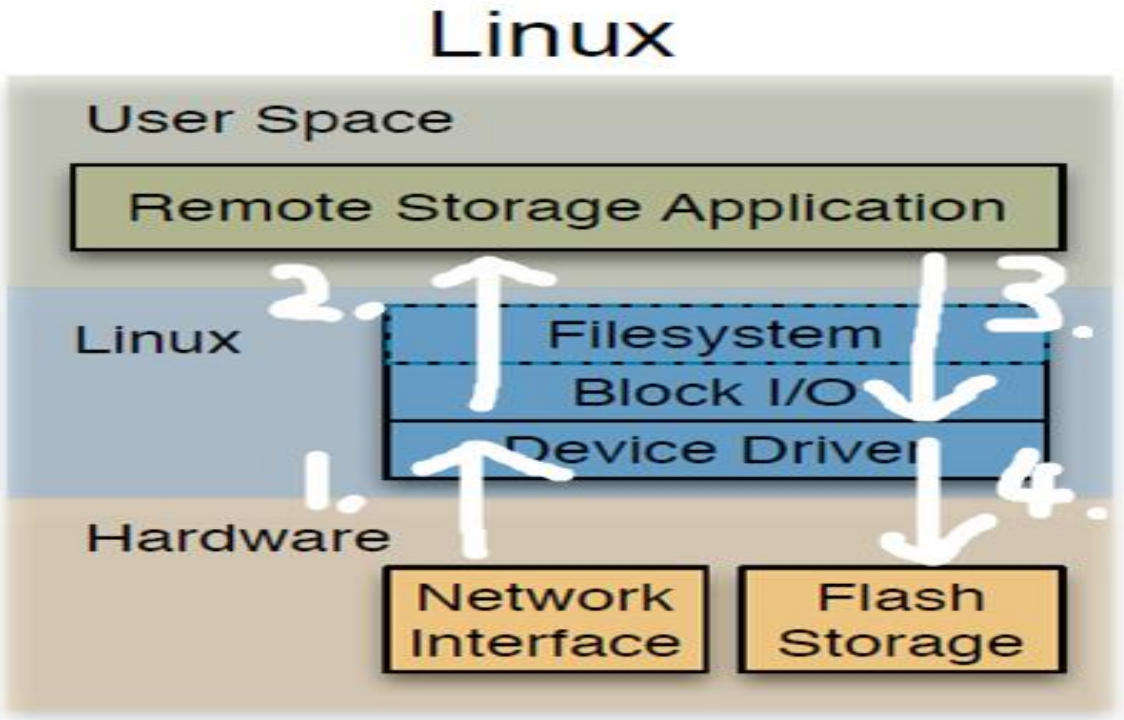


https://blog.westerndigital.com/cpu-bandwidth-the-worrisome-2020-trend/

异构硬件数据处理背景

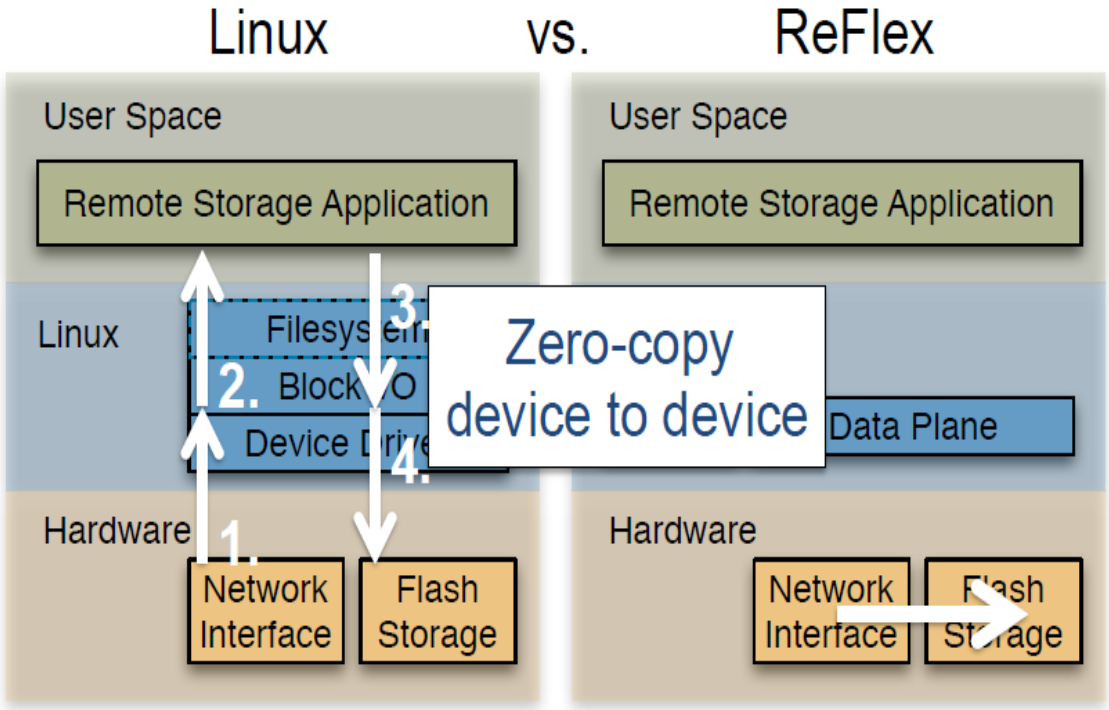
- 传统：中断式数据处理模式
  - 数据会经过多次拷贝





异构硬件数据处理背景

- 传统：中断式数据处理模式
  - 目标：数据零拷贝



A Klimovic et. al. ReFlex: Remote Flash == Local Flash. SIGOPS' 17

目标：零拷贝数据处理

- 基于异构硬件的数据处理

内存

内存计算

Spark

异构硬件支持

Linux Kernel(LK)

网卡

NVMe SSD

SAS磁盘

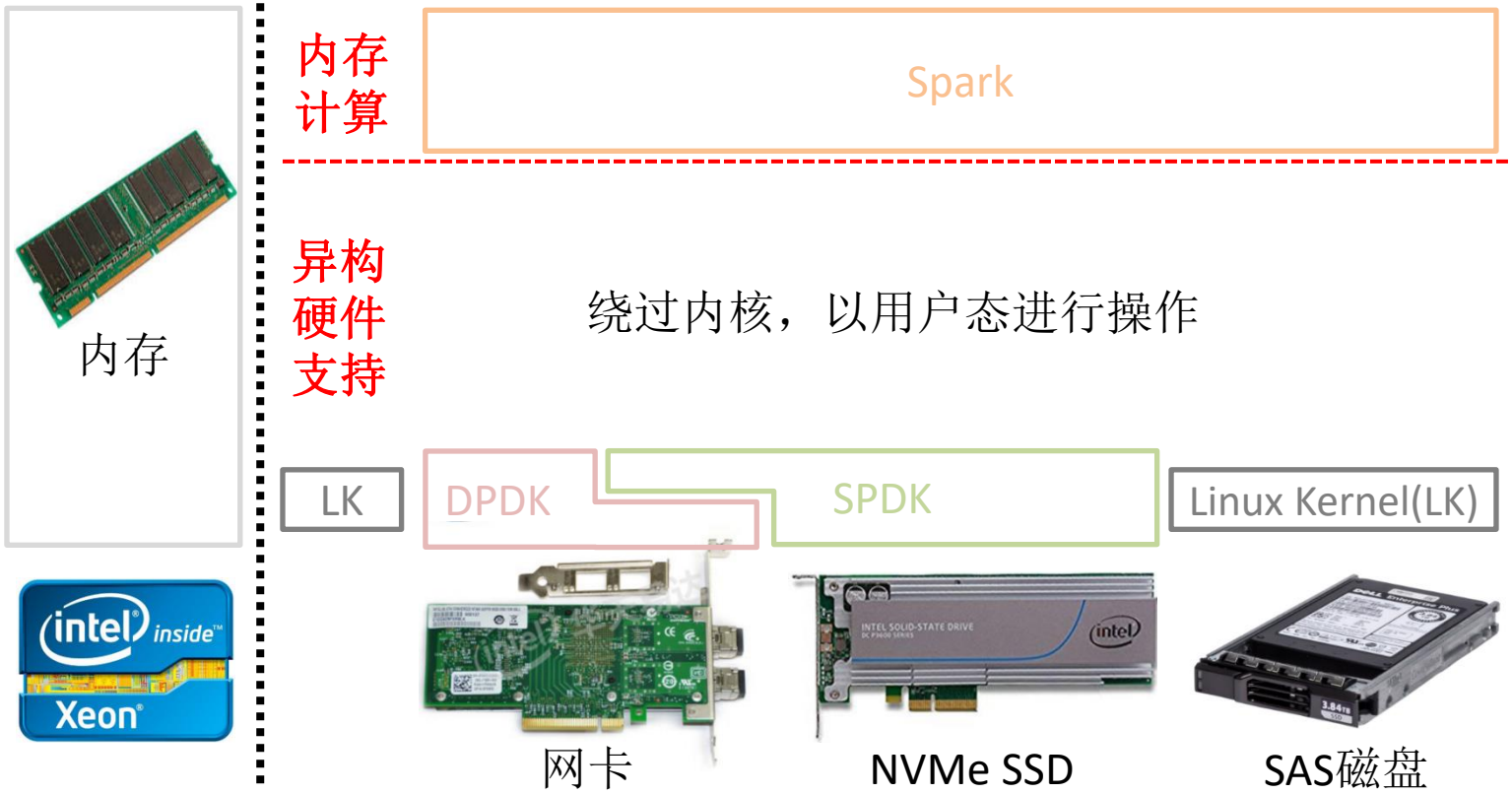
目标：零拷贝数据处理

- 基于异构硬件的数据处理

Intel, DPDK 2010. https://www.dpdk.org

目标：零拷贝数据处理

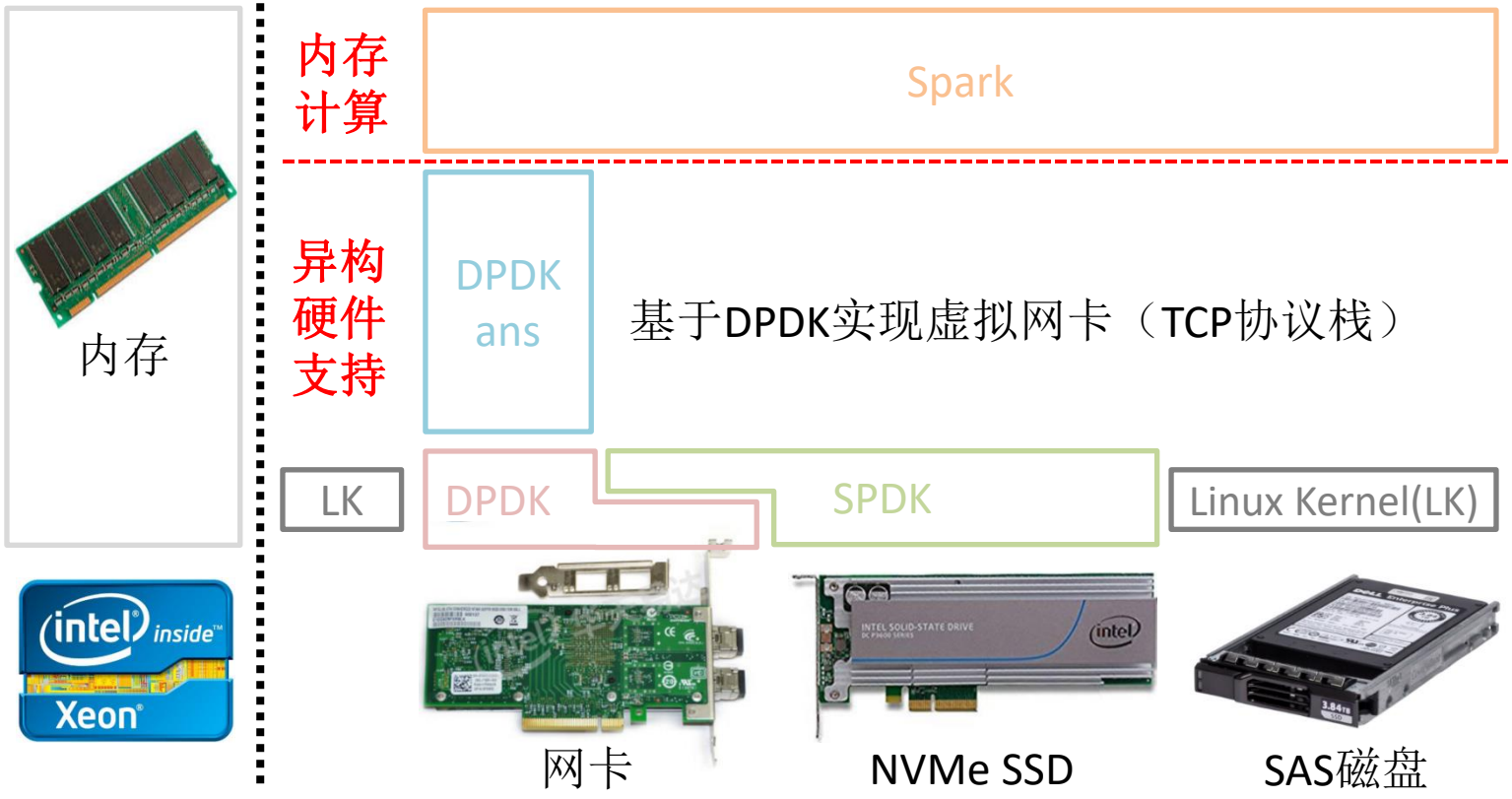
- 基于异构硬件的数据处理



Intel, SPDK 2015. https://github.com/spdk/spdk

目标：零拷贝数据处理

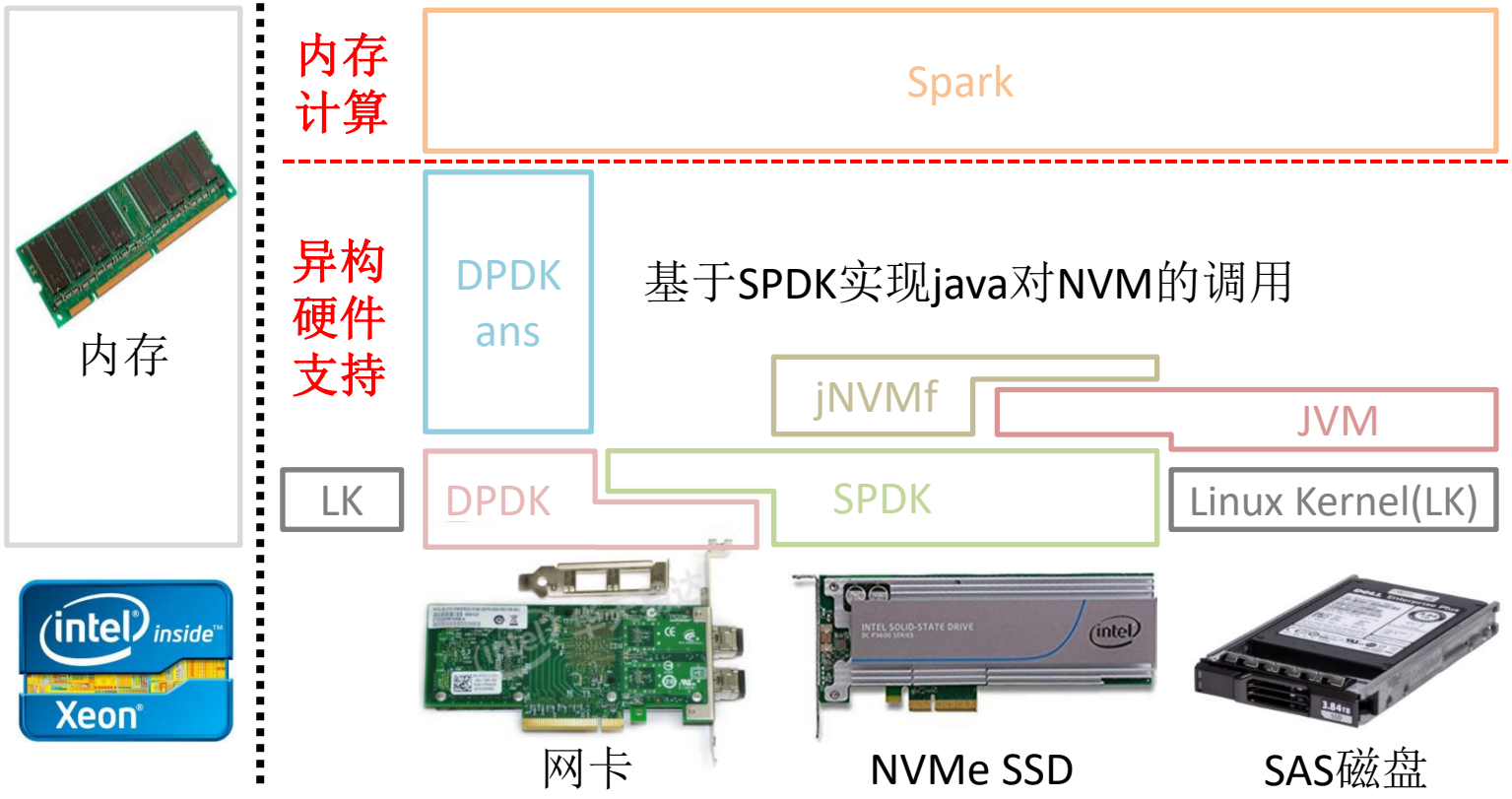
- 基于异构硬件的数据处理



UC Berkeley, DPDK-ans 2016. https://github.com/ansyun/dpdk-ans

目标：零拷贝数据处理

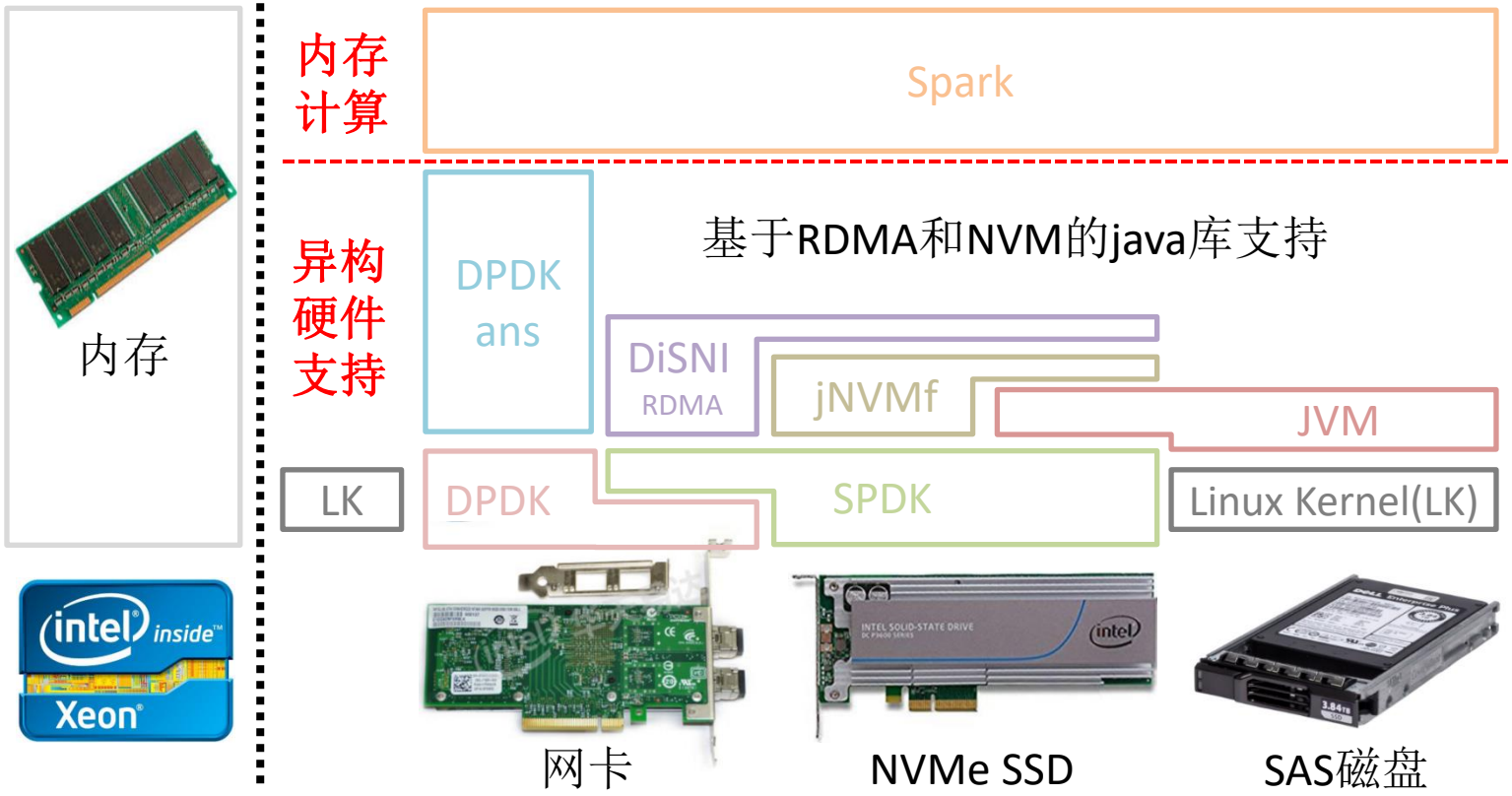
- 基于异构硬件的数据处理



IBM, jNVMf 2018. https://github.com/zrlio/jnvmf

目标：零拷贝数据处理

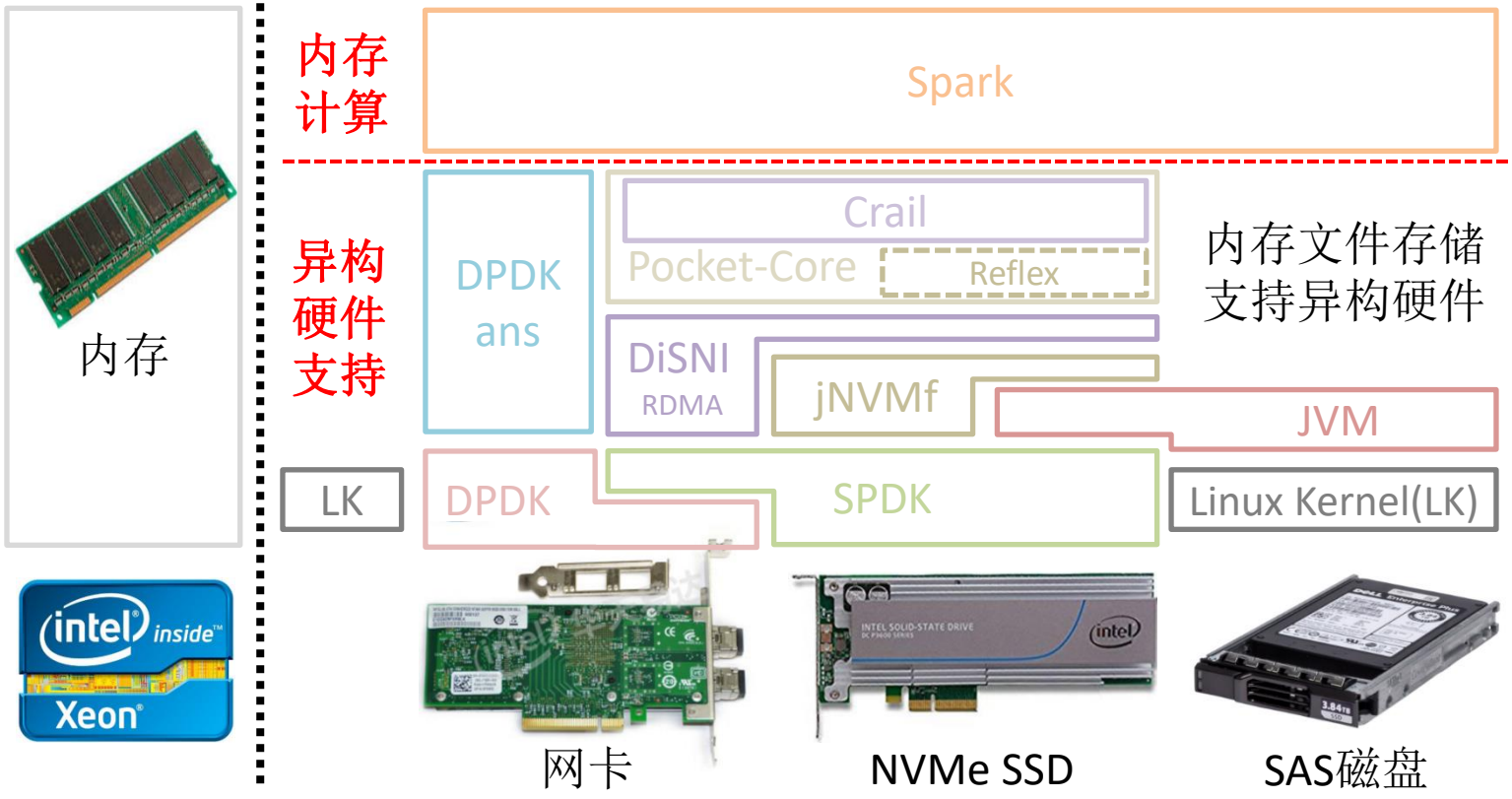
- 基于异构硬件的数据处理



IBM, DiSNI 2018. https://github.com/zrlio/disni

目标：零拷贝数据处理

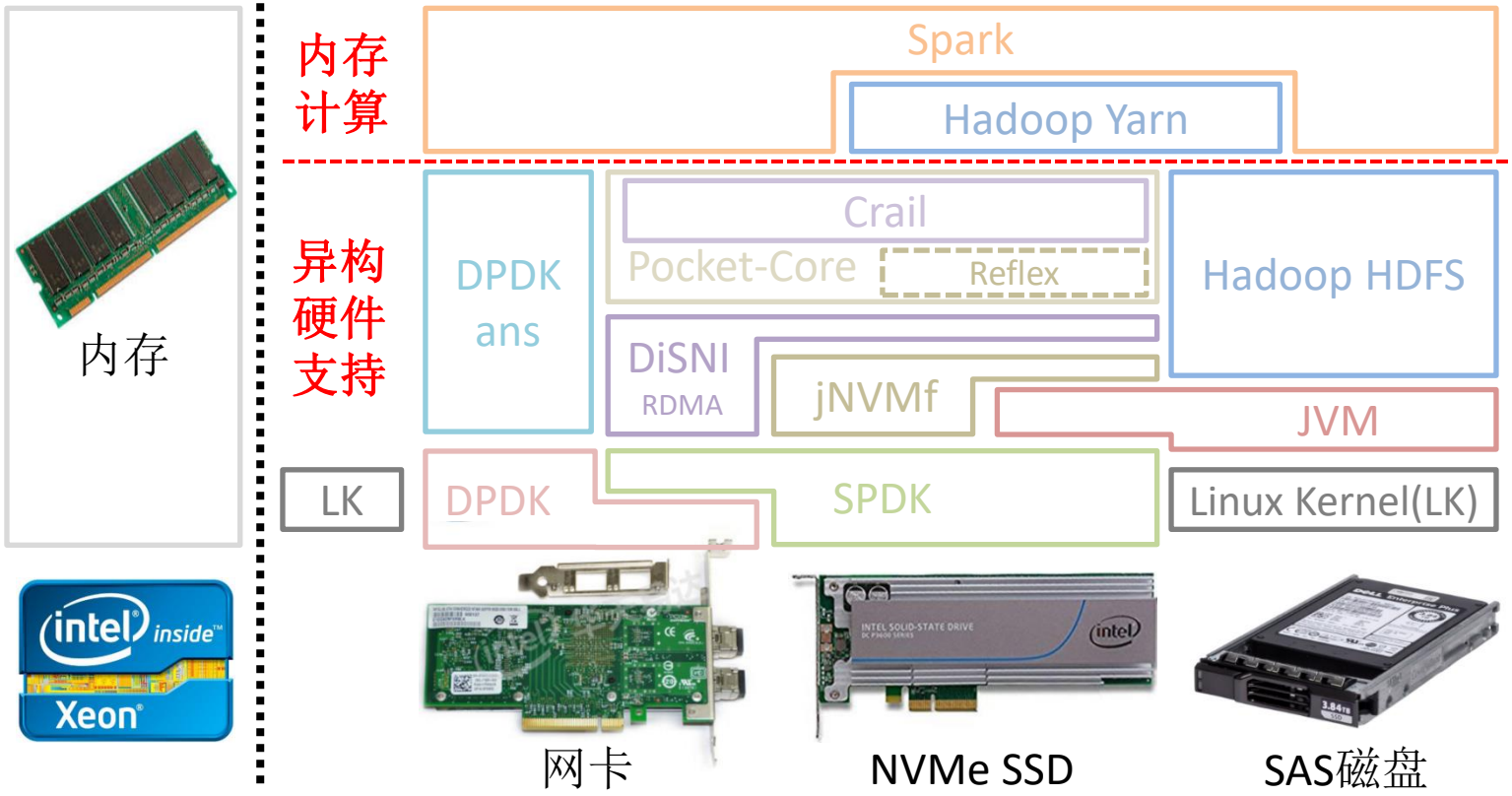
- 基于异构硬件的数据处理



IBM & Stanford, Pocket-Core 2018. https://github.com/stanford-mast/pocket-core

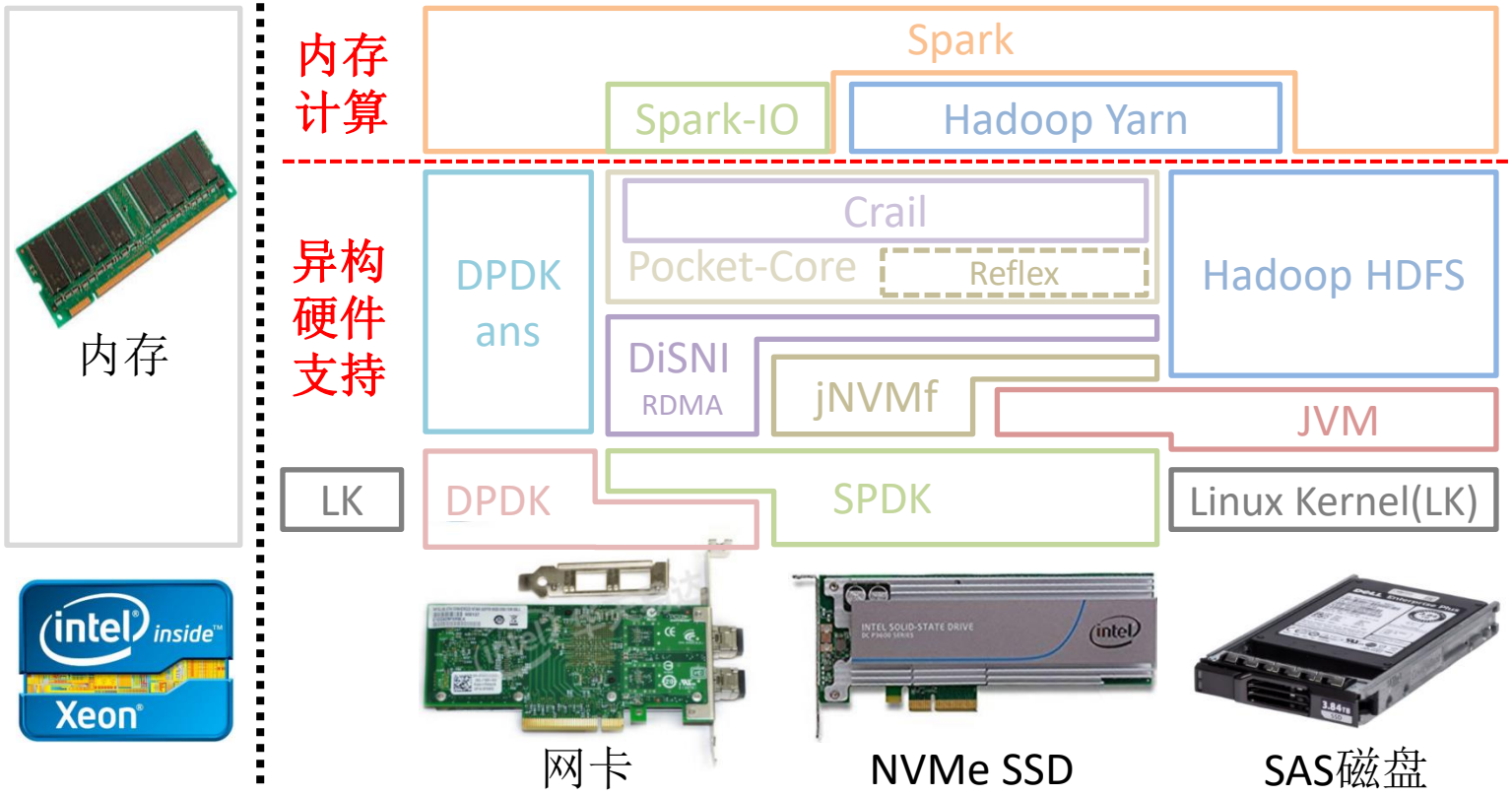
目标：零拷贝数据处理

- 基于异构硬件的数据处理



零拷贝数据处理架构

- 基于异构硬件的数据处理



Stanford, Spark-IO 2018. https://github.com/zrlio/crail-spark-io

谢谢大家！

金熠波

2020年4月