第8章 分析和大数据

目录

数据的持续增长

- **互联网/在线数据**
  - 点击量
  - 搜索记录
  - 服务器请求
  - 网络日志
  - 手机通话记录
  - 移动GPS定位
  - 用户生成内容
  - 娱乐内容 (如抖音，腾讯视频，bilibili，...)
- **医疗保健与科学计算**
  - 基因组学, 医学图像, 医疗数据, 账单数据
- **图数据**
  - 电信网络
  - 社交网络 (如微信, 微博 ...)
  - 计算机网络
- **物联网**
  - 射频识别（RFID）
  - 传感器
- **金融数据**

数据的持续增长

- 大型强子对撞机每年产生约30拍字节的数据
- Facebook的数据每月增长8拍字节
- 纽约证券交易所每天产生约4太字节的数据
- 2012年，YouTube大约有80拍字节的存储空间
- 互联网档案馆存储了约19拍字节的数据

1太字节等于1024吉字节（Gigabytes，GB）
1拍字节等于1024太字节（TB）

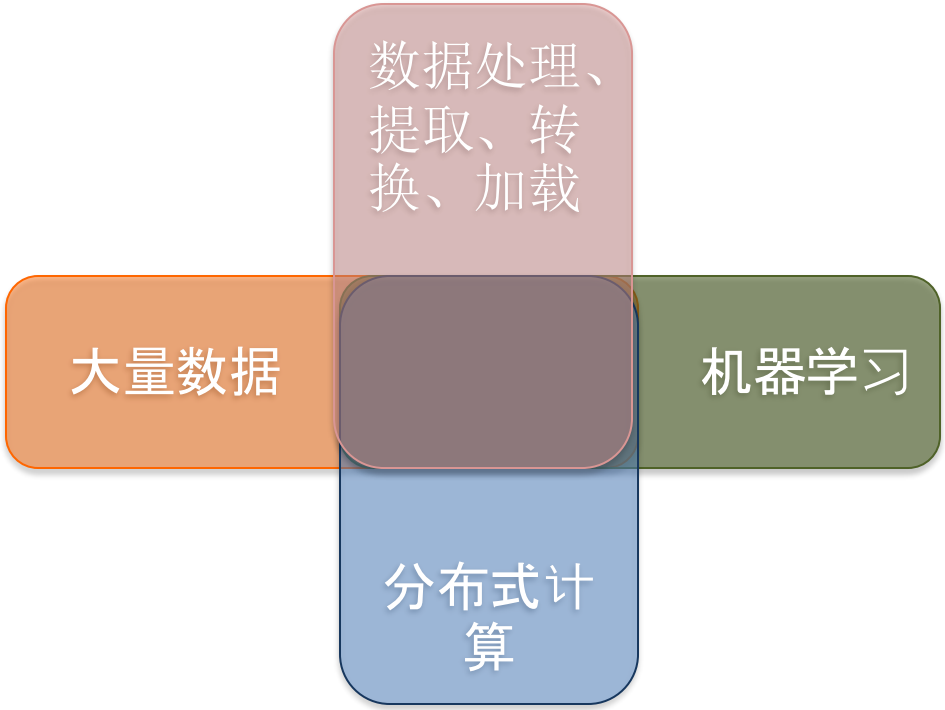
云计算和分布式计算

- 除数据量持续增长以外，另一趋势是云存储和计算资源的普及
  - 用于处理大规模数据集
- 云计算的特点
  - 动态可扩展（云计算平台可以灵活地扩大或缩小服务的能力，包括计算能力、存储空间、网络带宽等，以适应应用程序需求的变化）
  - 按需获取计算资源
  - 按需付费
  - 更经济

数据处理与机器学习方法

- 第三大趋势：数据处理
  - 数据的提取，转换，加载 （ETL）
  - 数据存储（HBase, ……）
  - 处理流式多媒体数据和批量数据的工具

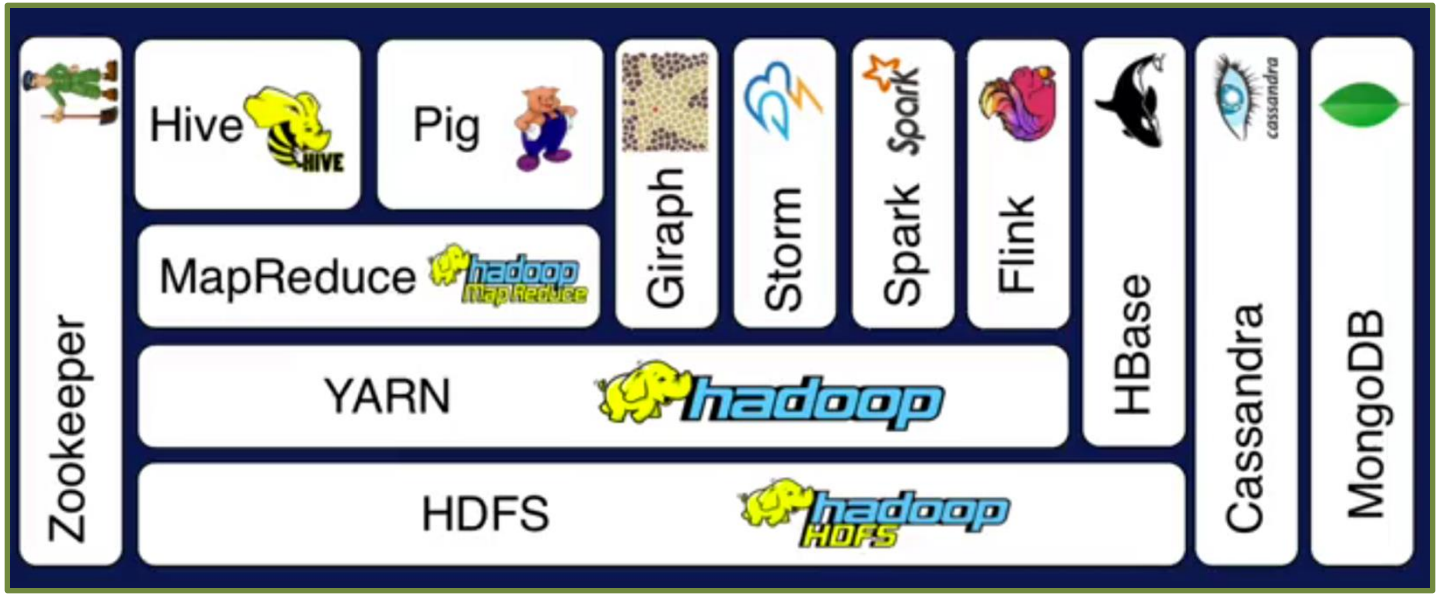
- 第四大趋势：机器学习
  - 分类
  - 回归
  - 聚类
  - 降维



目录

Hadoop 生态系统

- **HDFS（Hadoop分布式文件系统）：**这是基础层，用于存储大数据集。它提供了高吞吐量的数据访问，并能够在廉价的硬件上运行。
- **YARN（Yet Another Resource Negotiator）：**位于HDFS之上，是Hadoop的资源管理层，负责分配系统资源给各个正在运行的应用程序。
- **MapReduce：**这是Hadoop的原生数据处理模型和执行环境，用于在HDFS上执行分布式处理任务。
- **ZooKeeper：**一个集中式服务，用于维护配置信息、命名、提供分布式同步和提供组服务。

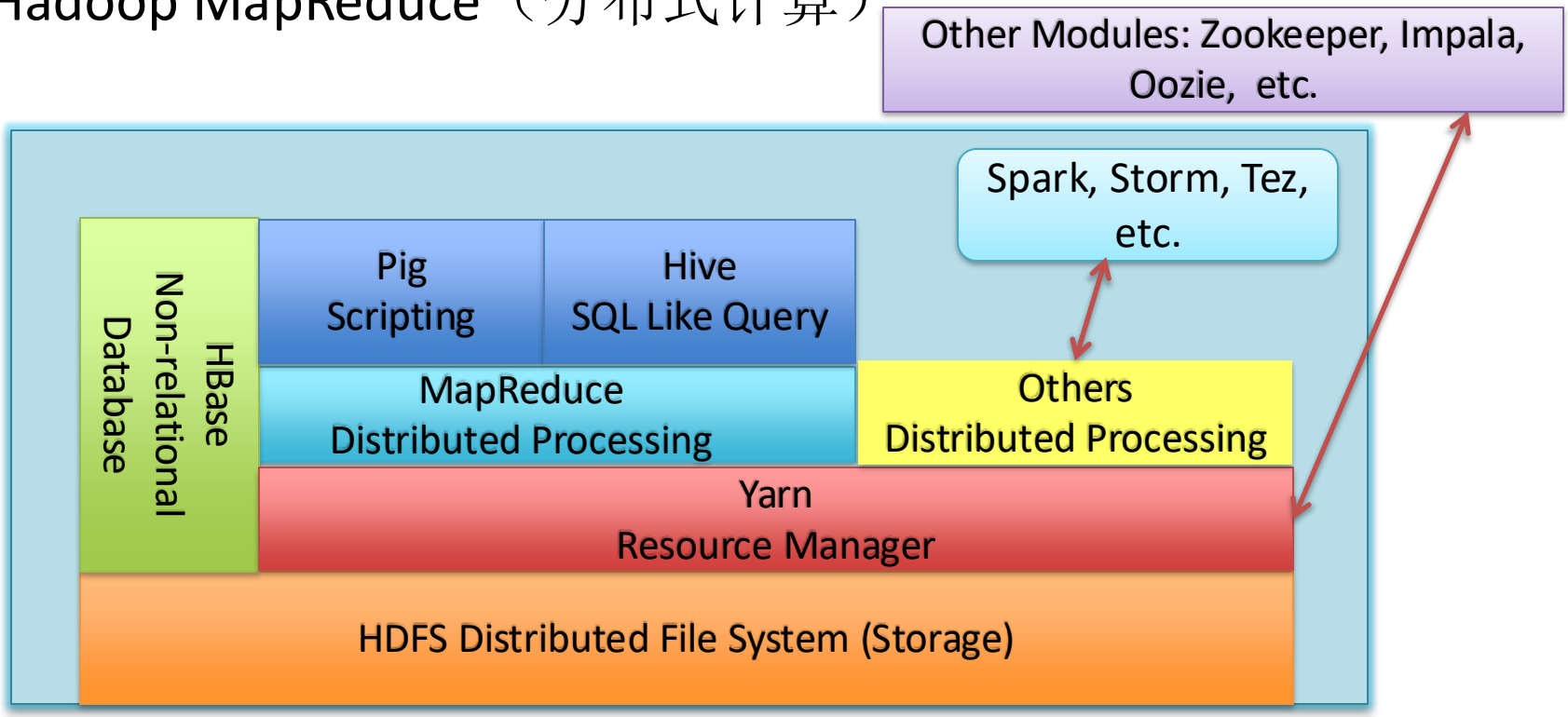


Hadoop 生态系统

- **Hive**：一种数据仓库基础设施，它提供了一种将结构化数据文件映射为一张数据库表的机制，并提供简单的SQL查询功能。
- **Pig**：一个高级平台，用于创建MapReduce程序的复杂数据处理流。
- **Giraph**：用于处理大规模图数据的平台。
- **Storm**：一个实时数据处理系统。
- **Flink**：一个开源流处理框架，用于分布式、高性能、始终可用和准确的数据流处理。
- **HBase**：一个分布式、可伸缩、大数据存储的非关系型数据库。
- **Cassandra**：一个高性能、分布式的NoSQL数据库。
- **MongoDB**：一个面向文档的数据库，用于处理大量的数据集。

Apache Hadoop 的基本模块

- Hadoop Distributed File System (HDFS)（分布式文件系统）
- Hadoop YARN（资源管理）
- Hadoop MapReduce（分布式计算）



目录

Hadoop HDFS

- Hadoop分布式文件系统（HDFS）是Hadoop生态系统中大多数工具使用的分布式文件系统，为处理大型数据集提供了良好的可扩展性。HDFS通过冗余存储来保证数据在硬件故障时的可靠性。
- HDFS 适合于:
  - 存储大文件
  - 处理流数据
- 不适合于:
  - 存储大量小文件
  - 需要频繁随机访问文件的场景
  - 对访问延迟有极低要求的应用

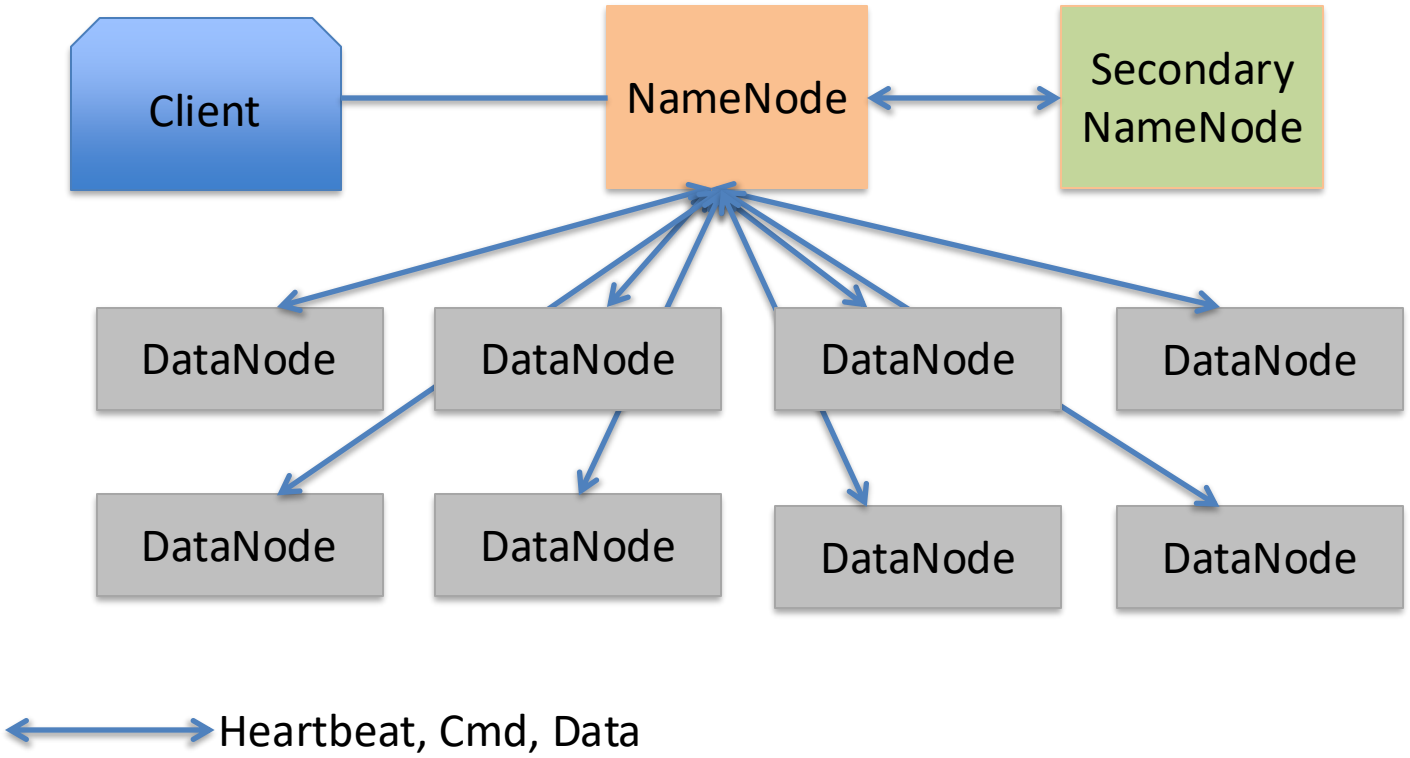
单个Hadoop集群拥有5000台服务器和250拍字节（petabytes）的数据

Hadoop分布式文件系统（HDFS）的设计

- 主从架构
  - 一个NameNode用于管理元数据
  - 多个 DataNodes 用于存储数据用
- 其它
  - Secondary NameNode作为备份使用

HDFS 架构

- NameNode保存元数据、文件名、位置和目录信息。
- DataNode为数据块提供存储空间。

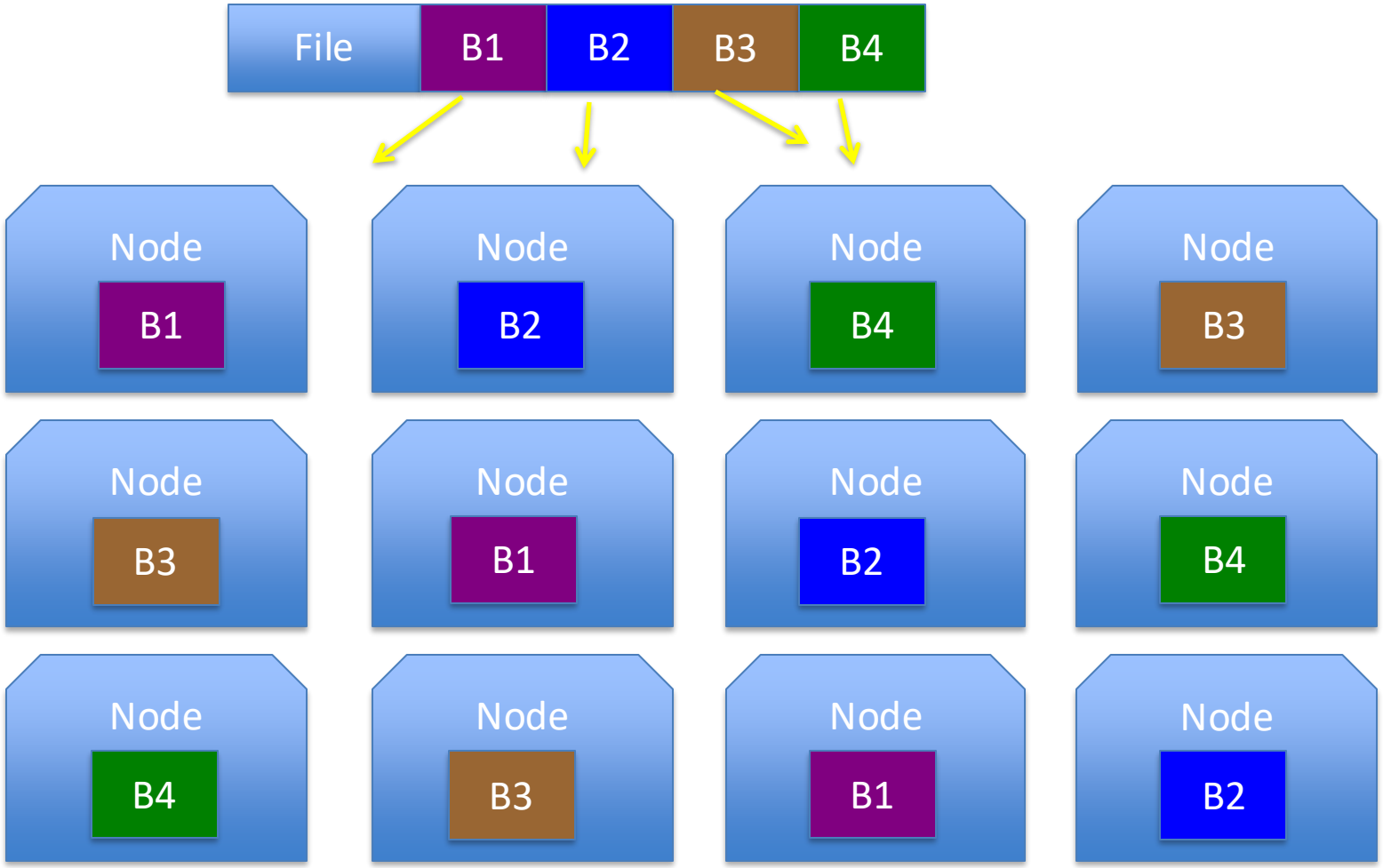


HDFS

- 文件被划分为多个数据块。
  - 数据块是读写操作的基本单位。
  - 默认的数据块大小是64MB，但可以设置得更大
  - 因此，HDFS适合存储较大的文件。
- HDFS中的数据块会被多次复制。
  - 一个数据块会被存储在多个位置，包括不同的机架上（通常复制3次）。
  - 这使得HDFS的存储具有容错性，并且读取速度更快。

HDFS

- 如果HDFS中的节点发生故障会怎样？
- 为了容错，对数据块进行复制。



一些HDFS Shell命令

在HDFS上创建一个目录

- hadoop fs -mkdir /user/godil/dir1

列出某一目录下所有的内容

- hadoop fs -ls /user/godil

从HDFS上传下载一个文件到指定文件夹

- hadoop fs -put /home/godil/file.txt /user/godil/datadir/
- hadoop fs -get /user/godil/datadir/file.txt /home/

查看文件内容：

- Hadoop fs -cat /user/godil/datadir/book.txt

还有许多类似于Unix的其他命令。

HBase

- 基于HDFS之上构建的NoSQL（非关系型数据库）
- 其设计参考了2006年的谷歌BigTable论文
- 能够处理各种类型的数据
- 存储大量数据（TB、PB级别）
- 是一种面向列的数据存储方式
- 因为列存储的读取效率和压缩效率都比较高，能够快速访问和修改存储在数据库中的特定数据项
- 可以进行水平扩展（水平扩展指通过增加节点以增加容量和性能，垂直扩展指的是升级节点硬件）

HBase

- 与传统的关系数据库模型（RDBMS）相比，Hbase不擅长于：
  - 事务性应用
  - 数据分析
- HBase对于文本搜索和处理也不够高效。

事务性应用指的是那些数据库事务必须精确管理，以确保数据的准确性和完整性的应用。例如，在银行系统中，当一个用户从一个账户向另一个账户转账时，这个过程要么完全发生，要么完全不发生，不能出现只完成一半的情况。此外，这个过程需要隔离，以确保不会与其他正在进行的事务相混淆，最后一旦完成，其结果必须被永久保存下来。

MapReduce：大数据的简单编程模型

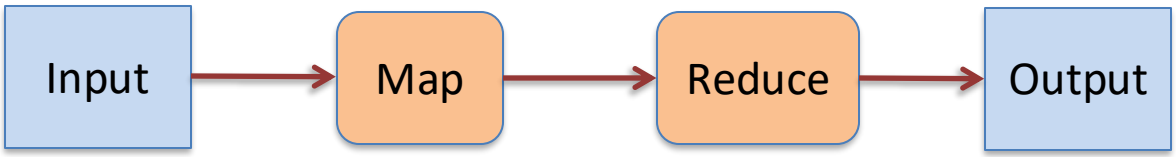
基于谷歌的MapReduce论文（2004年）

- MapReduce是Hadoop生态系统中一种简化的编程范式
- 传统的并行编程需要对不同的计算和系统概念有专业知识
  - 例如多线程、同步机制（锁、信号量和监视器）
  - 使用不当可能会导致程序崩溃、产生错误结果或严重影响性能。
  - 通常情况下，并不容错硬件故障。
- MapReduce编程模型极大简化了并行代码的运行
  - 无需处理上述任何问题
  - 只需创建Map和Reduce函数即可

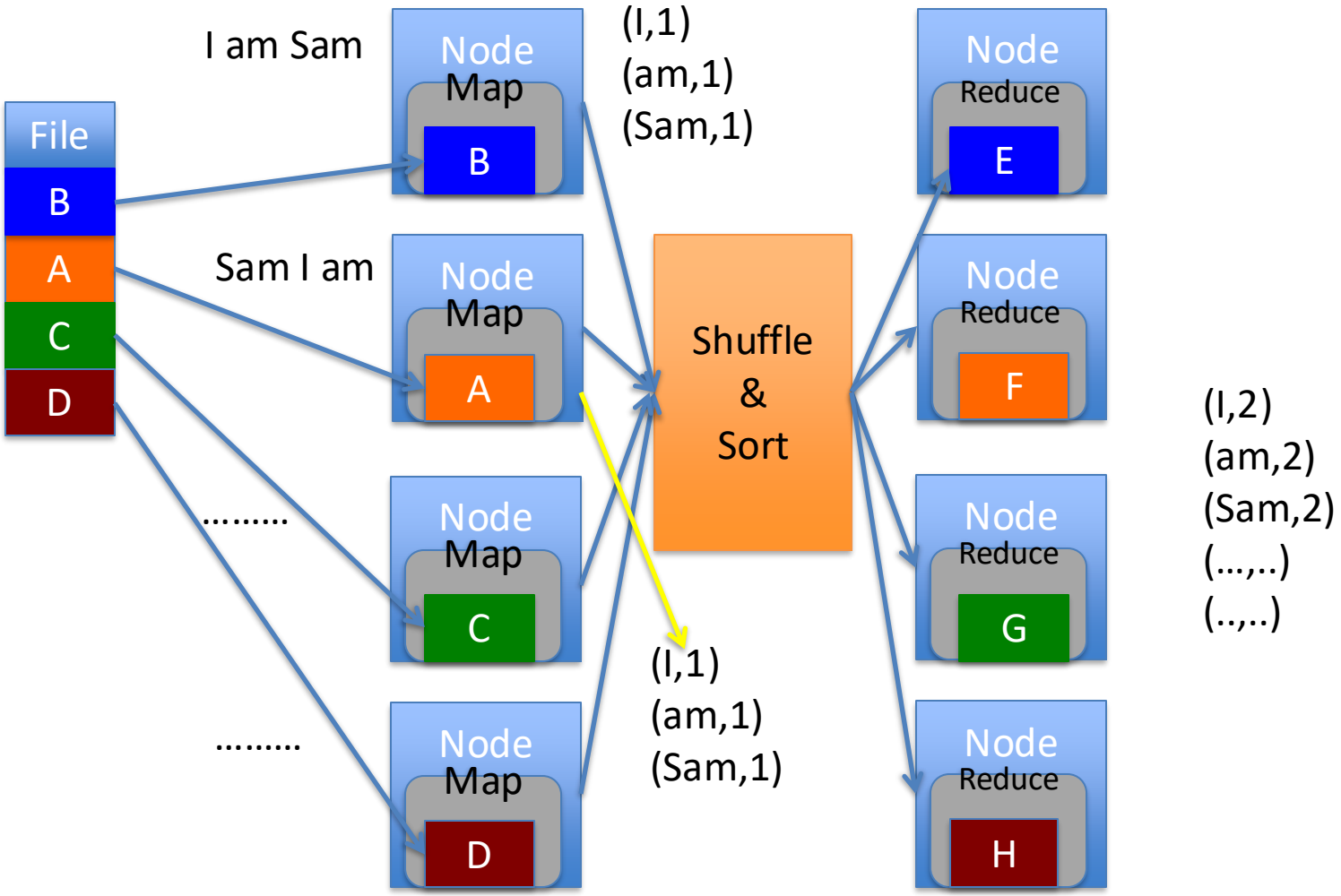
MapReduce 编程范式

- Map和Reduce操作基于函数式编程

Map:	**Reduce:**
将函数应用于列表或集合中每个元素 list1=[1,2,3,4,5]; square x = x * x list2=Map square(list1) print list2 -> [1,4,9,16,25]	将一个函数应用于列表或集合的元素以减少它们到单一的值 list1 = [1,2,3,4,5]; A = reduce (+) list1 Print A -> 15



MapReduce 词频统计的例子



MapReduce的缺点

- 强制数据处理遵循Map和Reduce模型
  - 缺少其他的针对分布式的操作，包括join（连接）、filter（过滤）、flatMap（扁平化映射）、groupByKey（按键分组）、union（合并）、intersection（交集）等。
- 在Map和Reduce之前后都需要读写磁盘
  - 对于迭代任务效率不高，比如机器学习
- 只有Java语言得到原生支持。
  - 需要支持其他语言。
- 只适用于批处理
  - 缺乏交互性和数据流处理。

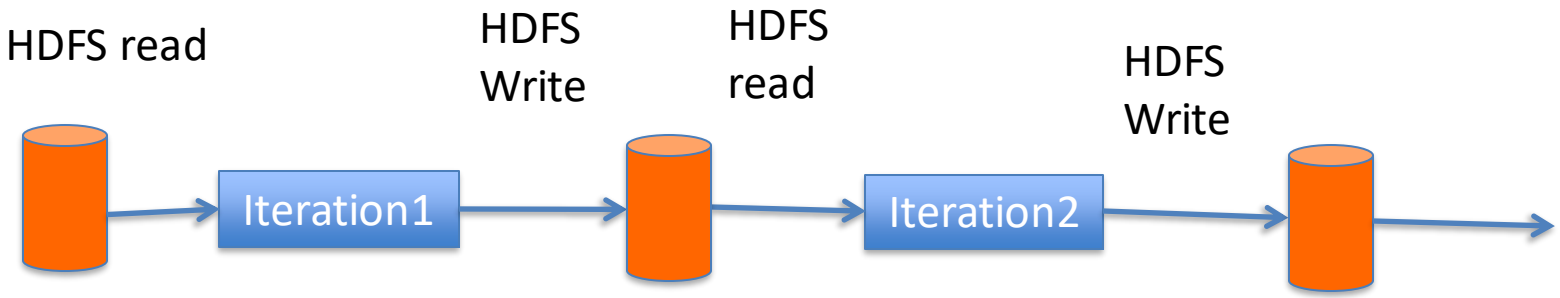
目录

MapReduce的替代方案是Apache Spark

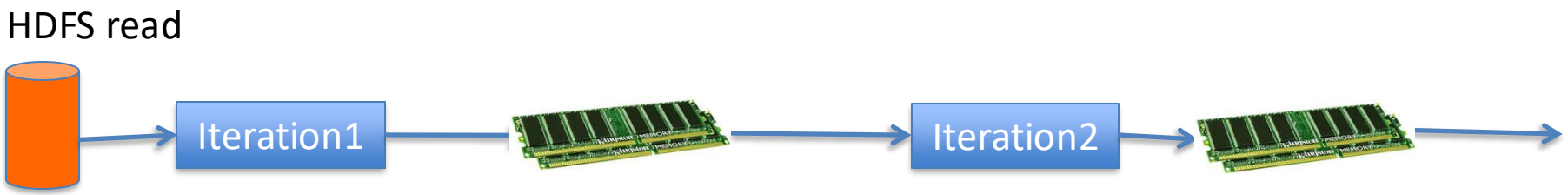
- 一个新的通用框架，解决了MapReduce的许多不足。
- 它能够利用Hadoop生态系统，例如HDFS、YARN、HBase等。
- 具有许多其他操作，例如join（连接）、filter（过滤）、flatMapdistinct、groupByKey（按键分组）、reduceByKey（按键归约）、sortByKey（按键排序）、collect、count、first等（大约30种高效的分布式操作）。
- **内存中数据缓存**（用于迭代算法、图算法和机器学习等）。
- 原生支持Scala、Java、Python和R语言。
- 支持交互式shell，用于探索性数据分析。
- Spark API非常简单易用。 最初在加州大学伯克利分校的AMPLab开发，现由Databricks.com维护。

使用内存代替磁盘进行数据处理。

Hadoop使用磁盘进行数据共享。



Spark使用内存进行数据共享。



排序比赛

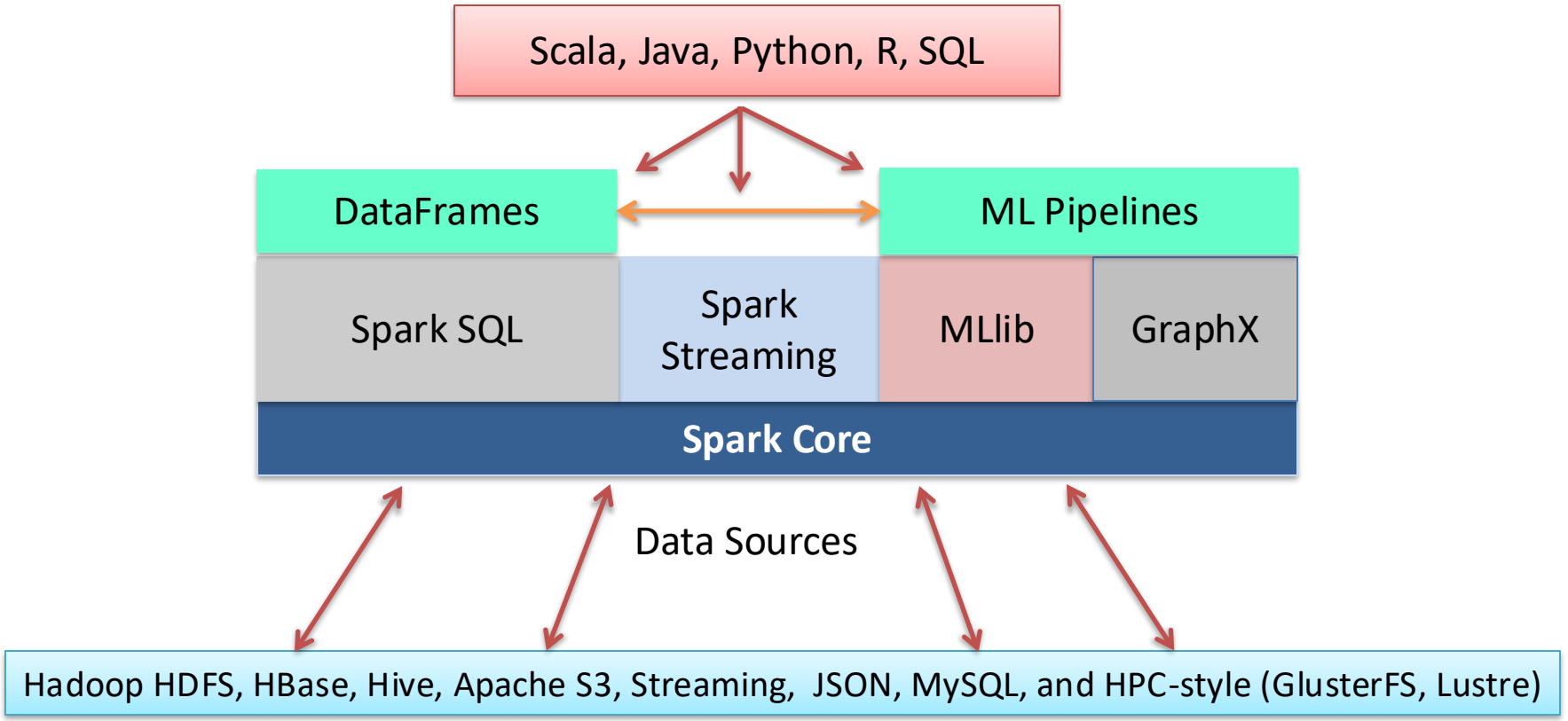
	MapReduce记录 (2013)	Spark 记录(2014)
数据大小	102.5 TB	100 TB
耗时	72 分钟	23 分钟
#节点数	2100	206
#核心数	50400 物理核心	6592 虚拟核心
集群磁盘吞吐量	3150 GB/s	618 GB/s
网络	专用数据中心的网络, 10Gbps	虚拟化网络（AWS EC2），10Gbps
排序速率	**1.42 TB/分钟**	**4.27 TB/分钟**
排序速率/节点	**0.67 GB/分钟**	**20.7 GB/分钟**

Spark，使用节点数仅为十分之一却速度快3倍。

Sort benchmark, Daytona Gray: sort of 100 TB of data (1 trillion records)
http://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html

Apache Spark

Apache Spark 支持数据分析、机器学习、图处理、流数据等。它可以读写多种数据类型，并支持多种语言开发



弹性分布式数据集（RDDs）

- RDD（弹性分布式数据集）是Spark中的数据容器。
- 无论Spark在数据处理的哪个阶段或哪个模块，它都是通过操作RDD来完成的，这使得数据处理过程在Spark中保持了一致性和高效性。
- 由于应用程序共享RDD抽象，所以可以混合使用不同类型的转换来创建新的RDD。
- RDD可以通过并行化一个集合或读取一个文件来创建。
- RDD具有容错性，这意味着即使在分布式系统中的部分节点失败的情况下，也能恢复数据。

DataFrames & SparkSQL

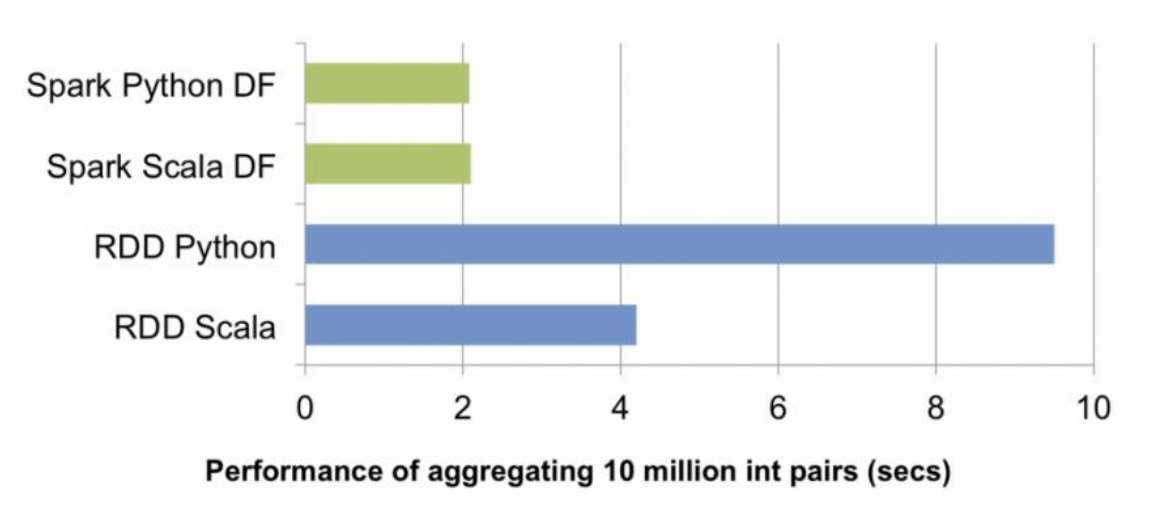
- DataFrame是在RDD的基础上构建的一个更高层次的抽象，它提供了更简单的数据操作方式和更好的性能优化。
- DataFrames（DFs）是另一种以列组织的分布式数据集
- 类似于关系数据库、Python的Pandas DataFrame或R的DataTables
  - 一旦构建，DataFrames是不可变的
  - 能够追踪数据的来源
  - 支持分布式计算
- 构造DataFrames的方法
  - 从文件读取
  - 从现有的DataFrames转换（Spark或者Pandas）
  - 可以将现有的python集合，如list转化为DataFrame

DataFrame 例子

```
//从users DataFrame中筛选出年龄小于21岁的行，创建一个新的students DataFrame
students = users.filter(users.age < 21)

//另外，也可以使用类似Pandas的语法进行相同的操作。
students = users[users.age < 21]

//将学生按性别分组，并计算每个性别组的数量。
students.groupBy("gender").count()

//将学生的数据与名为logs的DataFrame进行左外连接：
students.join(logs, logs.userId == users.userId,
“left_outer")
```

RDDs vs. DataFrames

- RDD（弹性分布式数据集）是Spark的底层数据处理抽象，能够对数据进行更精细化的操作
- DataFrames是建立在RDDs之上的
- DataFrames have a schema（schema指的是类似于关系数据库中的表结构，存储有各列的名称和数据类型）

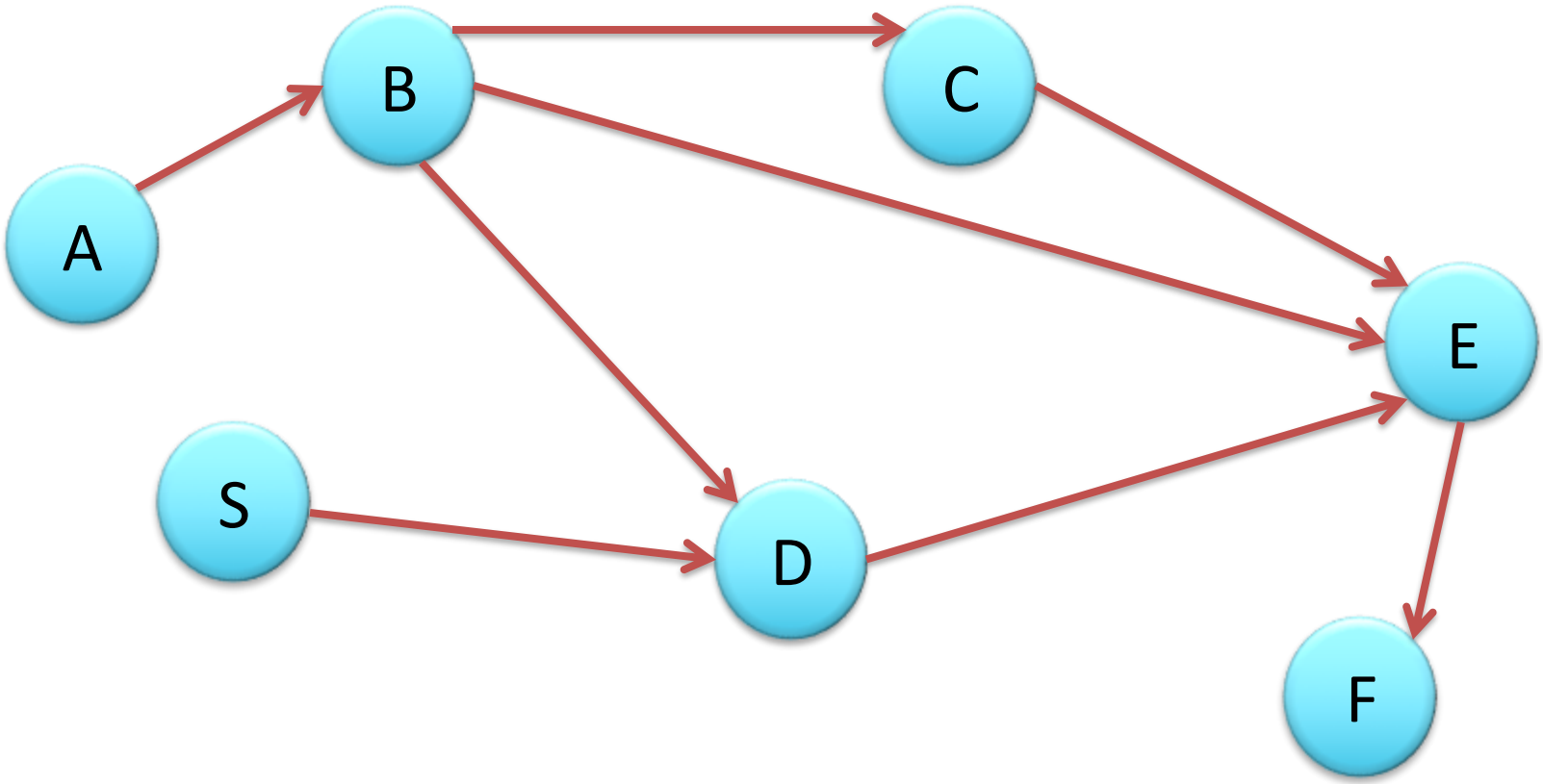


RDD/DataFrames的性能比较

Spark 操作

Transformations (创建一个新的RDD)	Map filter sample groupByKey reduceByKey sortByKey intersection	flatMap union join cogroup cross mapValues reduceByKey
Actions (将结果返回给驱动程序或存储到外部存储系统)	collect Reduce Count takeSample take lookupKey	first take takeOrdered countByKey save foreach

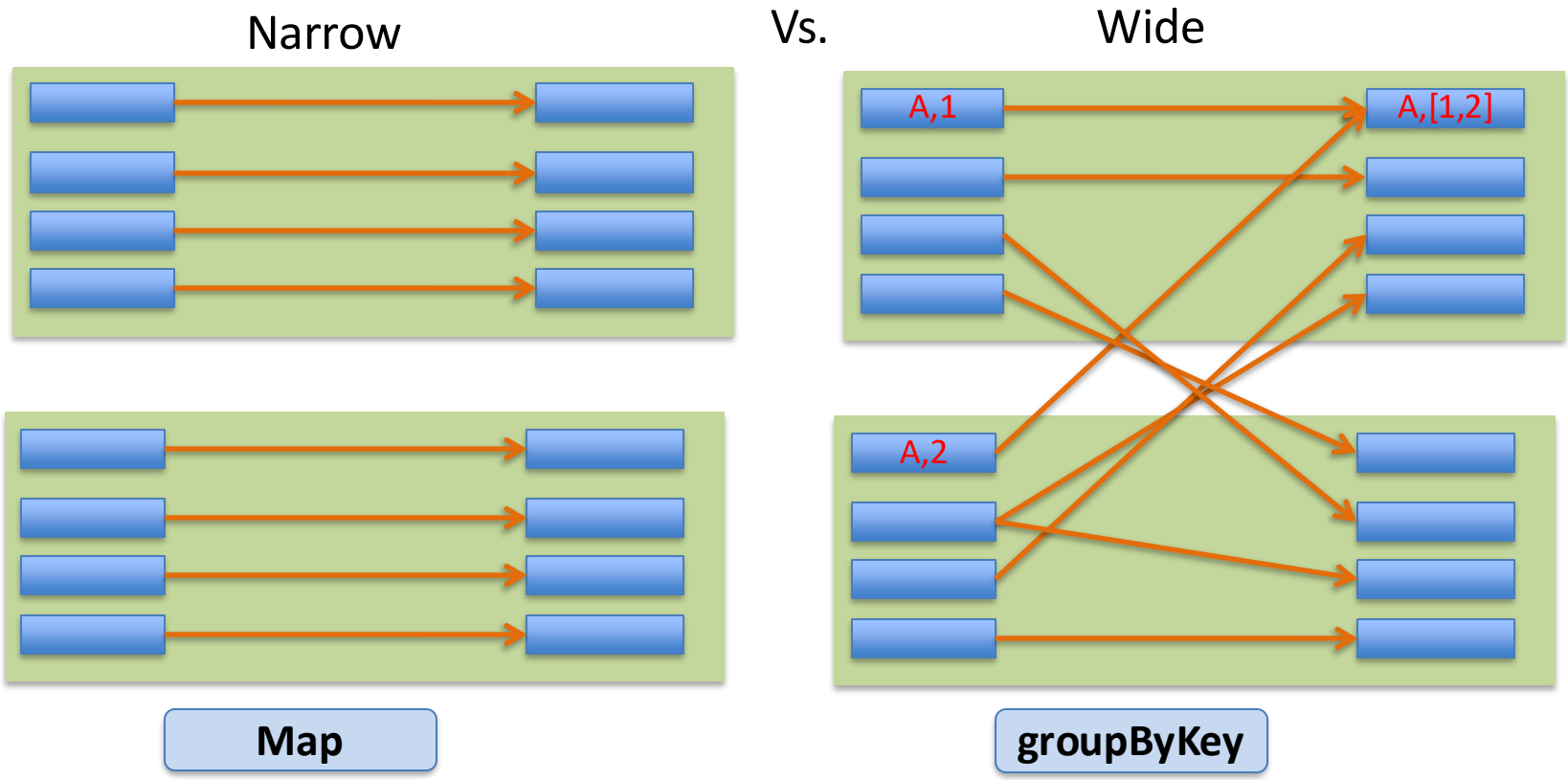
有向无环图 (DAG)



在Apache Spark中，有向无环图（DAGs）用于跟踪节点（RDD）间依赖关系

- 每个节点代表一个RDD
- 箭头表示对这些RDDs进行的transfromations操作

Narrow Vs. Wide transformation

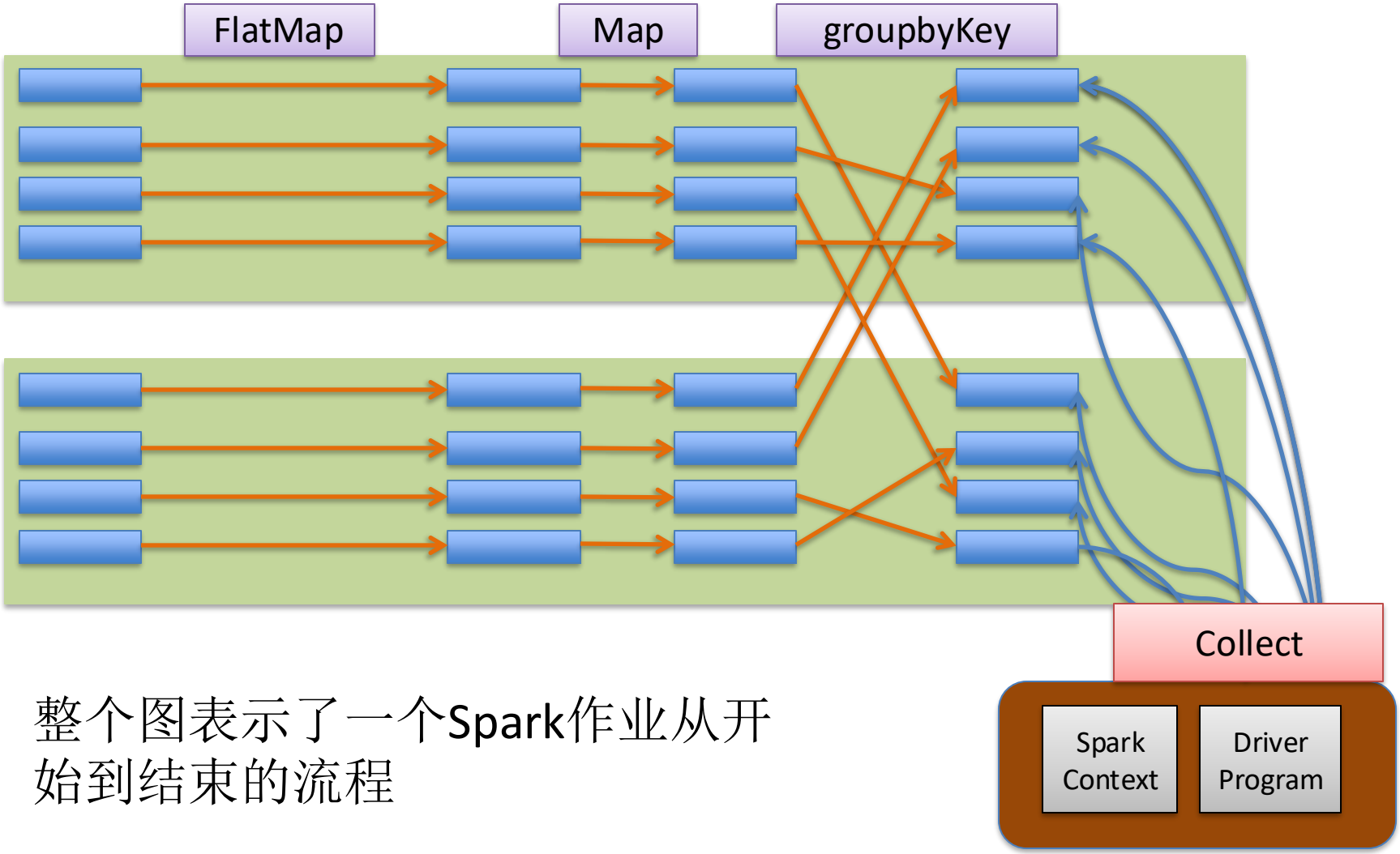


这个图表展示了在分布式数据处理框架（如Apache Spark）中两种不同类型的数据转换：窄转换（Narrow Transformation）和宽转换（Wide Transformation）。

Actions

- 什么是action
  - 工作流程的最后阶段
  - 触发DAG（有向无环图）的执行
  - 将结果返回给驱动程序或将数据写入HDFS（Hadoop分布式文件系统）或文件

Spark 工作流



整个图表示了一个Spark作业从开始到结束的流程

Python RDD API 例子

- Word count

```
text_file = sc.textFile("hdfs://usr/godil/text/book.txt")
counts = text_file.flatMap(lambda line: line.split(" ")) \
          .map(lambda word: (word, 1)) \
         .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://usr/godil/output/wordCount.txt")
```

- Logistic Regression

```
# Every record of this DataFrame contains the label and
# features represented by a vector.
df = sqlContext.createDataFrame(data, ["label", "features"])
# Set parameters for the algorithm.
# Here, we limit the number of iterations to 10.
lr = LogisticRegression(maxIter=10)
# Fit the model to the data.
model = lr.fit(df)
# Given a dataset, predict each point's label, and show the results.
model.transform(df).show()
```

例子来自 http://spark.apache.org/

RDD 持久化及其移除

- 在Apache Spark中，RDD持久化（或缓存）是一种优化技术，用于存储经常访问的RDDs，以便快速访问。
- 当一个RDD被标记为持久化时，Spark会在第一次计算它时，保留其元素在内存或磁盘中的副本。
- 这样，在后续的动作中使用这个RDD时，Spark可以直接使用这个副本，而不需要重新计算整个RDD。
- 持久化有多种级别，包括仅内存、内存和磁盘、仅磁盘等，具体取决于数据集的大小和计算资源。
- 当RDD不再需要时，可以使用unpersist()方法将其从缓存中移除，以释放资源。

Broadcast Variables 与 Accumulators (Shared Variables)

- Shared Variables指的是可以被集群中不同节点共享的变量。共享变量允许在多个任务和节点之间共享和传递数据。
- 广播变量（ Broadcast variables ）使得每个工作节点可以保存一份共享的、只读的数据副本。

```
>broadcastV1 = sc.broadcast([1, 2, 3,4,5,6]) #创建一个包含列表 [1, 2, 3, 4, 5, 6] 的广播变量。
>broadcastV1.value #访问广播变量的值，在此例中返回 [1, 2, 3, 4, 5, 6]。
[1,2,3,4,5,6]
```

- 累加器（Accumulators）是Spark中的一个特殊类型的变量，它专门用于在多个任务间安全地执行累加（如求和、计数）操作，适合在分布式环境下进行并行处理时使用。

```
accum = sc.accumulator(0)
accum.add(x)
accum.value
```

目录

Spark的主要用例

- **流数据处理**：实时处理和分析数据流。
- **机器学习**：使用MLlib库进行机器学习算法的开发和训练。
- **交互式分析**：快速查询和数据探索。
- **数据仓库**：进行大规模数据的ETL操作和数据管理。
- **批处理**：处理大批量数据集。
- **探索性数据分析**：数据挖掘和模式识别。
- **图数据分析**：使用GraphX库进行图形结构的数据分析。
- **空间（GIS）数据分析**：处理和分析地理空间数据。
- 以及更多其他应用。

现实生活中的Spark

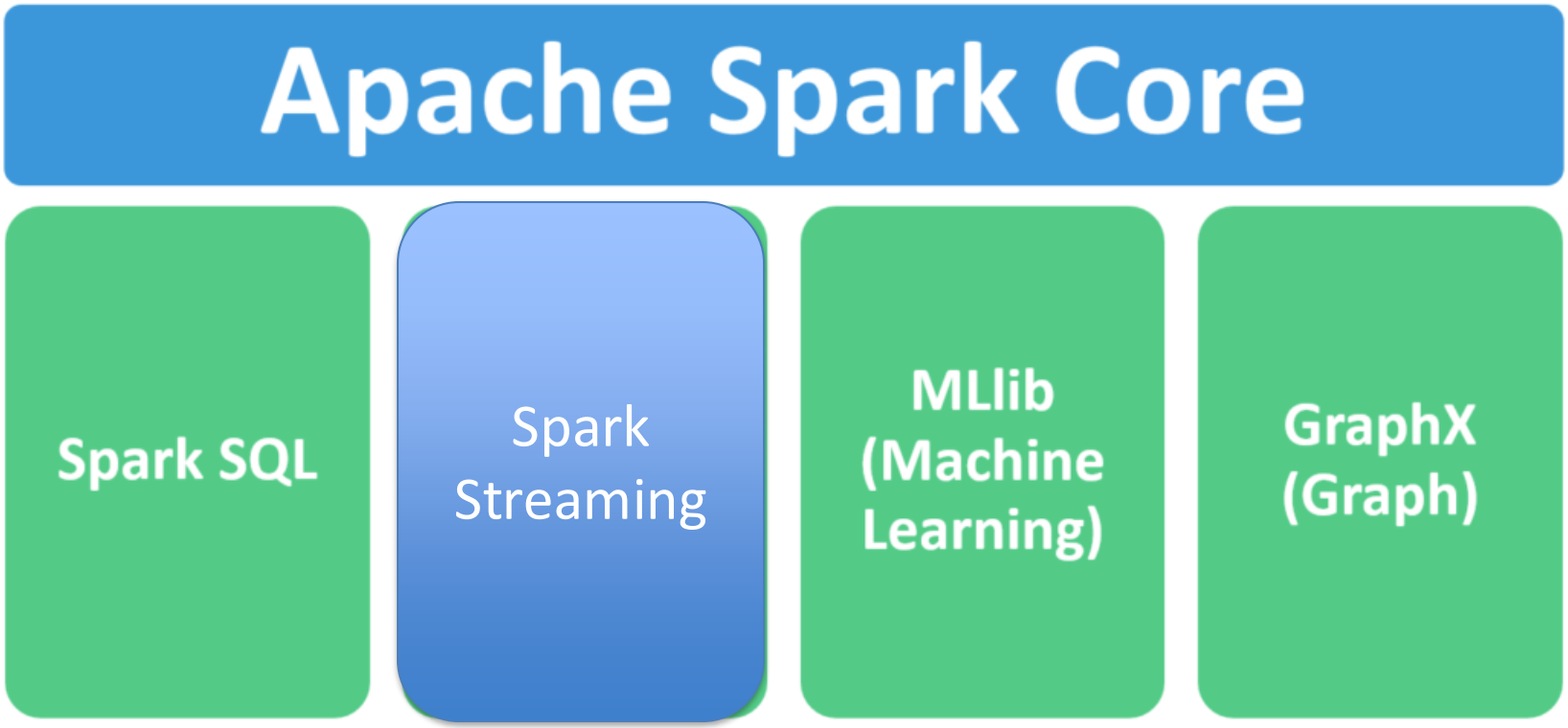
- 滴滴-每天从用户收集数以太字节的事件数据。这些数据量巨大且无结构，来自于滴滴app内的各种交互操作。
  - 持续的 ETL Pipeline：他们使用 Kafka、Spark Streaming 和 HDFS 技术来建立一个持续的 ETL（提取、转换、加载）流程。
  - 这个流程将原始的无结构事件数据转换为结构化数据。
  - 这样处理后的数据被用于更复杂的分析，帮助滴滴优化其运营。

- 阿里巴巴–利用数据科学进行市场和消费者分析：
  - 阿里巴巴利用大数据和机器学习技术深入分析市场趋势和消费者行为。
  - 根据分析结果，阿里巴巴能够开发和优化其金融产品，例如支付宝的信用服务。
  - 同时，他们使用这些技术来识别和降低金融交易中的欺诈风险。

Apache Spark可能不适合以下情况：

- 虽然Spark具有很多优点，但它的优秀的内存处理能力并不总是适用于所有情况：
  - 对于许多简单的用例，Apache MapReduce和Hive可能是更合适的选择。
  - Spark并不是为多用户环境设计的。当多个用户（或作业）要求同时运行时，因为Spark并没有内建的机制来有效隔离每个用户或作业的资源，所以用户需要手动管理和协调他们的资源（内存，CPU时间，存储空间等）使用。

Spark streaming



什么是Stream Processing?

批处理（Batch Processing）

- 也被称为“静态数据处理”（Data at Rest）。
- 批处理适合于那些不要求实时分析的情况，可以在数据集完整后进行全面的处理。
- 它处理有限大小的数据集，一旦所有数据被处理完毕，处理过程就终止。

流数据处理（Stream Processing）

- 又被称为“动态数据处理”（Data in Motion）。
- 它是一个持续的计算过程，能够实时处理无限量的数据流。
- 随着新数据的到来，它能够不断更新答案。
- 通常在较小的时间窗口内操作，以便快速响应最新数据的变化。

为什么需要Stream Processing?

流数据处理与批处理（Batch Processing）相比

- **（近）实时性：**流数据处理能够在极短的时间内完成，从秒到毫秒级别，实现了（近）实时地数据分析和处理。
- **快速响应：**流数据处理支持更快速的数据响应。它不仅能够实时检测数据中的模式，还能及时设置警报，从而及时应对潜在问题或者机遇。
- **处理无界数据：**流数据处理特别适用于处理某些本质上无界的数据，如连续产生的传感器数据。这类数据源不断生成新数据，没有固定的结束点。
- **资源限制下的高效处理：**在存储,计算资源或者带宽有限的情况下，stream processing会选择性地保留最关键和最有用的信息以高效地处理大量数据。
- **连续性处理：**与批处理不同，流数据处理是一个持续不断的过程，没有明确的开始和结束，能够不间断地处理数据流。

流处理的应用

- **计算领域**
  - 日志分析
  - 攻击检测
- **监控与安全**
  - 欺诈检测（信用卡)
  - 入侵检测（监控）
- **传感器数据处理**
  - 天气监测
  - 交通运输
  - 病人健康监测
- **社交媒体和市场趋势**
  - 趋势分析
- **工业**
  - 流程优化（监控生产线数据，提高效率和产量）
  - 预测性维护
- **广告和促销**
  - 根据用户行为定制广告
  - 根据地理位置定制广告
- **金融交易**
  - 算法交易
  - 风险分析
- …

流数据处理的难点

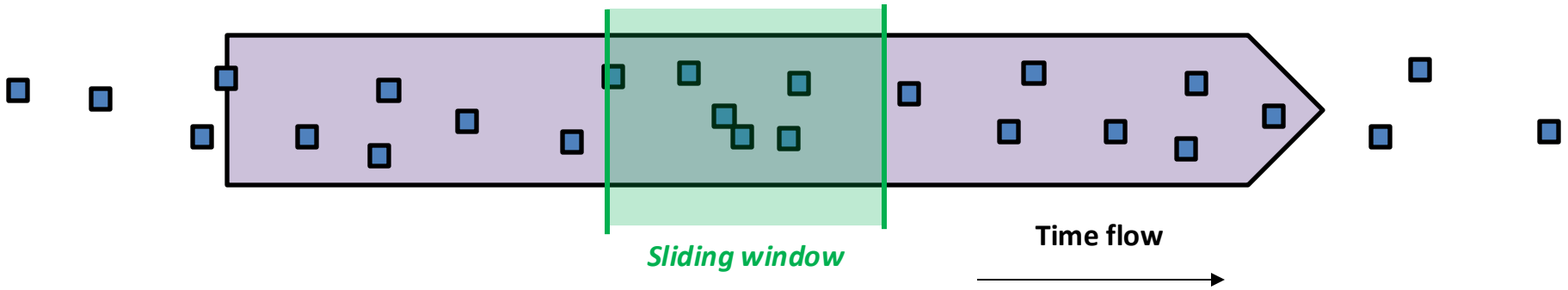
输入：

- **时间限制**：流数据处理通常有严格的时间约束。数据需要在极短的时间内被处理，以保持数据流的连续性和实时性。
- **数据类型多样化**：输入的数据元素可能包括各种类型，如数值、文本、图像等，且这些数据通常是动态变化的。
- **无界性**：流数据是无界的，意味着数据源不断生成数据，没有固定的结束点。这要求系统能够持续处理不断到来的数据。
- **无序性**：输入的数据可能是无序的，尤其是当它来自多个不同的数据源时。
- **不完整性**：由于各种原因（如传输错误、数据损坏等），接收到的数据可能是不完整的，这要求系统能够处理不完整或有缺陷的数据输入。

输出：

- **近似答案**：由于流数据处理的实时性和输入数据的不完整性，输出的结果往往是近似值，而不是精确答案。系统会根据到达的数据不断更新这些近似答案。

滑动窗口（Sliding Window）



在流数据处理中，使用**滑动窗口**用于数据的实时监控和分析。

相关的概念

- **事件时间 vs 处理时间**

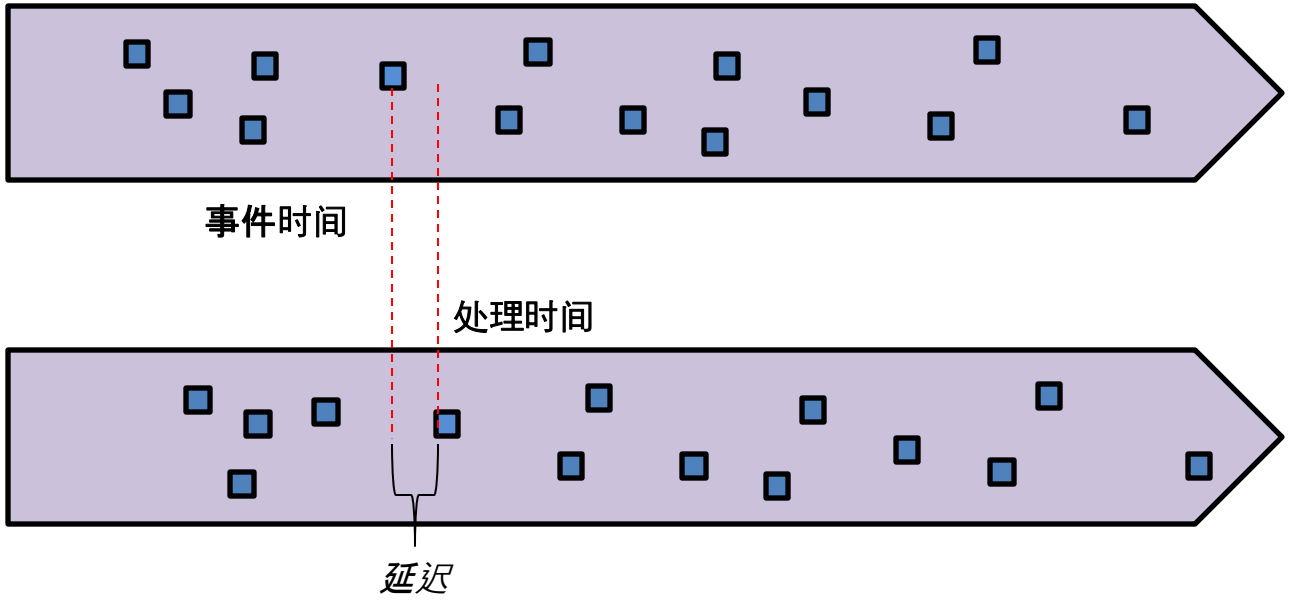
事件时间：

- 事件时间是**数据生成的实际时间**，即数据源记录事件发生的时间。
- 在流数据处理中，事件时间用于精确地反映事件发生的顺序和时间。

处理时间：

- 处理时间是**数据到达处理系统并被处理的时间**。
- 这通常与事件时间不同，因为数据传输和处理可能会有延迟。

相关的概念



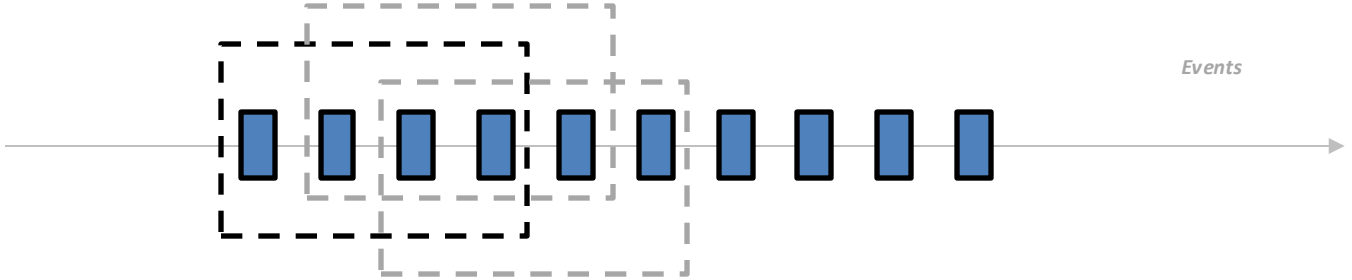
相关的概念

窗口类型

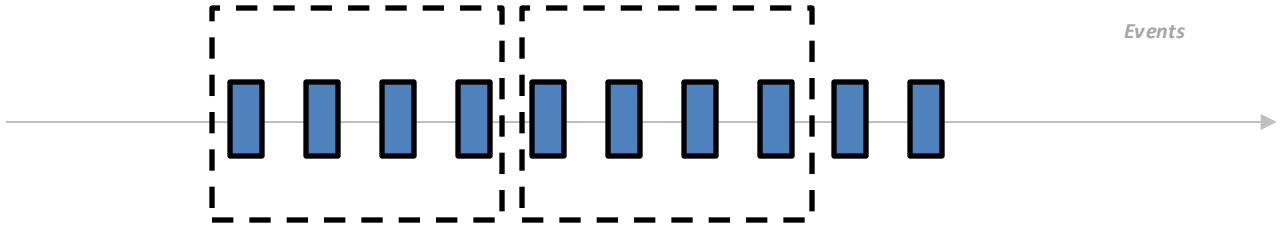
1. **滑动窗口（Sliding Windows）：**
   - 这种窗口在固定的时间间隔内滑动，例如每分钟分析过去5分钟的数据。
   - 窗口和窗口间一般有重叠。
2. **滚动窗口（Tumbling Windows）：**
   - 这种窗口是时间间隔不重叠的固定大小窗口，例如每5分钟一个窗口用于分析过去5分钟的数据。
   - 滚动窗口不会像滑动窗口那样重叠。
3. **基于计数的窗口：**
   - 基于计数的窗口是根据数据元素的数量定义的，例如每处理1000个数据元素一个窗口。

相关的概念

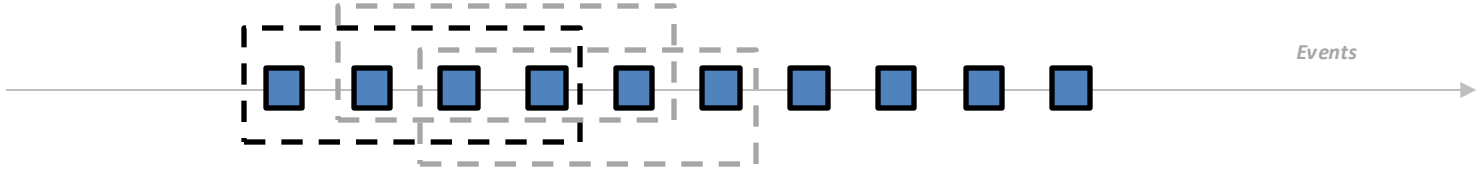
滑动窗口



滚动窗口



基于计数的窗口（每4个数据元素一个窗口）



相关的概念

窗口操作（转换）

- 窗口操作是对窗口内的数据执行的转换，这些操作可以提供窗口内数据的实时分析。

聚合（Aggregations）
聚合操作涉及对窗口内数据的统计分析，包括：求和（Sums），平均值（Averages），计数（Counts），最大值（Maximum）：，如最小值、标准差、中位数等。

过滤（Filtering）
排序（Sorting）
连接（Joining）
。。。

相关的概念

有状态操作 vs无状态操作

有状态操作 :有状态操作指的是在数据处理过程中需要保留之前的记录或部分结果的操作。这些操作依赖于之前的数据状态，以便进行当前的数据处理。常见的有状态操作包括：

- **累积聚合：**如连续计算最小值、最大值和平均温度。这需要记住前面所有记录的信息来计算当前的值。
- **窗口聚合的连续处理：**如滑动窗口中的数据聚合，需要记住前一个窗口的数据状态以快速计算当前窗口的聚合结果。
- **模式和趋势识别：**识别长期的数据模式或趋势，需要对历史数据的持续跟踪和分析。

相关的概念

有状态操作 vs无状态操作

无状态操作（Stateless Operations）无状态操作则是不需要记住之前的数据或状态，仅依赖于当前窗口内的数据进行处理的操作。这些操作每次只处理当前窗口的数据，而不依赖于历史数据。常见的无状态操作包括：

- **窗口内聚合：**如计算过去5分钟内的平均温度。这类操作仅基于当前窗口内（窗口为5分钟）的数据进行计算。
- **过滤和转换：**根据当前窗口内数据的属性进行过滤或对数据进行简单转换。
- **简单计数和统计：**如统计当前窗口内的数据点数量或计算基本统计量。

Stream Processing - Tools

Stream Processing – 工具

Kafka与Spark Streaming

Kafka和Spark Streaming通常一起搭配使用，它们各自在数据处理管道中扮演不同的角色：

- **Apache Kafka**：Kafka是一个分布式流媒体平台，主要用于**高吞吐量**的消息队列和实时数据流的处理。它作为一个强大的消息代理，可以**收集和分发大量的实时数据**。

- **Spark Streaming**：Spark Streaming专门用于处理实时数据流。它可以接收来自各种源的数据流，包括Kafka，并对这些数据**执行复杂的处理和分析**。

当一起使用时，Kafka通常作为数据的生产者和传输者，负责收集和传递数据，而Spark Streaming作为数据的订阅者，接收Kafka中的数据流并进行处理。

Kafka的应用举例

假设有一个在线购物平台，该平台需要实时处理**用户的点击**、**浏览**、**购买**等行为数据。这些数据量巨大，且需要实时分析以提供**个性化的产品推荐**和**监控系统性能**。

- 在这种场景中，Kafka可以用来收集来自网站和应用的用户行为数据。当用户在网站上浏览、点击或购买产品时，这些行为被发送到Kafka中作为消息。Kafka作为消息代理，能够高效处理这些大量的数据流，并将它们分发给不同的系统进行处理，例如：
  - 将用户行为数据发送到**实时分析系统**，这些系统可以立即处理数据以生成个性化的产品推荐。
  - 将数据发送到**监控系统**，以实时监控网站的性能和用户的行为模式。
  - 。 。 。

Spark streaming的应用举例

1.实时数据处理和分析：

- Spark Streaming 可以从 Kafka 接收用户行为数据流。
- 这些数据可以用来计算某个时间段内最受欢迎的产品，或者实时跟踪用户的点击路径。

2.个性化产品推荐：

- 利用 Spark 的机器学习库（如 MLlib），可以基于用户的历史行为和实时行为数据来生成个性化的产品推荐。
- 这些推荐可以即时反馈给用户，增加购买概率。

3.系统性能监控：

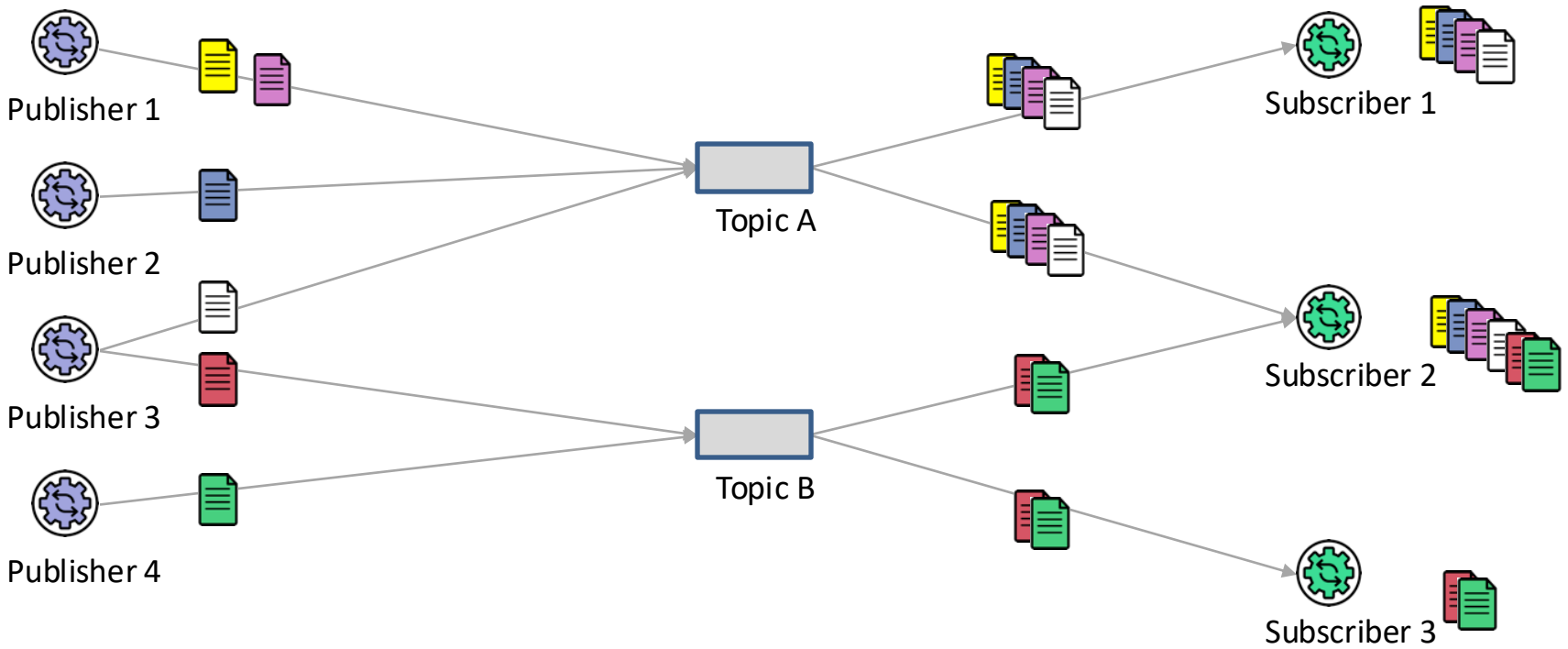
- Spark Streaming 可以实时分析用户行为数据，从中识别出异常模式，如不寻常的流量峰值，可能表明有潜在的系统性能问题。
- 实时监控可以帮助及时发现并解决这些问题，保证网站稳定运行。

4.长期数据趋势分析：

- 虽然 Spark Streaming 重点在于实时处理，但它也可以将数据存储到数据库中（如 Hadoop HDFS、Amazon S3）。
- 存储的数据可以用于后续的批量处理和深度分析，比如分析用户行为的长期趋势，优化营销策略等。

Kafka

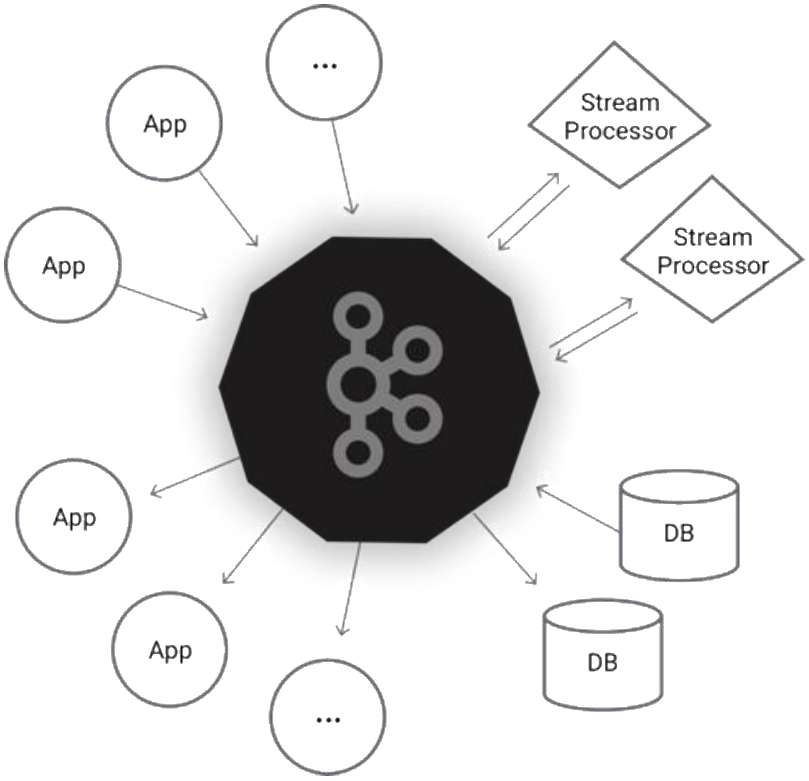
- 收集/分发消息机制



生产者发布消息，消息在Kafka分区中存储，然后消息的订阅者按照需要读取这些消息。

Kafka的特点总结

- 高吞吐量的消息系统（Messaging system）
- 消息收集和分发（Publish & Subscribe）
- 分布式（Distributed ）
- 容错性（Fault Tolerant）
- 可扩展性（Scalable）
- 实时（Real-time）
- 低延（Low Latency）

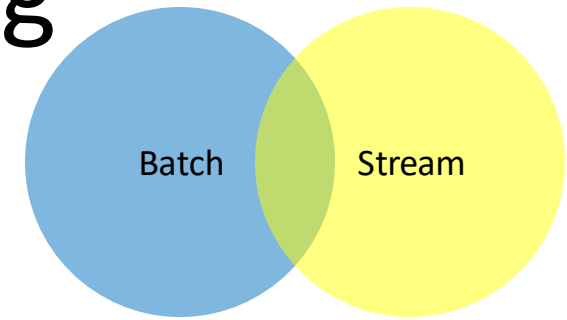


Spark Streaming

- **Spark Streaming**是Apache Spark的一个扩展组件，它增加了对流数据处理的能力。这允许Spark不仅可以处理静态数据集（批处理），还可以处理实时数据流。与Spark核心API集成。
- 可扩展、容错
- 可以从多个数据源读取数据，
- 可以将机器学习模型直接应用于数据流

Image credits: https://spark.apache.org/docs/latest/streaming-programming-guide.html

Spark Streaming



Micro-batch

Spark streaming是如何工作的？

微批处理（Micro-batches processing）： Spark Streaming通过将实时数据流分割成小的数据块或“微批”（micro-batches）来处理数据。这些微批数据在设定的短时间间隔内（如每几秒）被收集，并作为一个批处理任务进行处理。

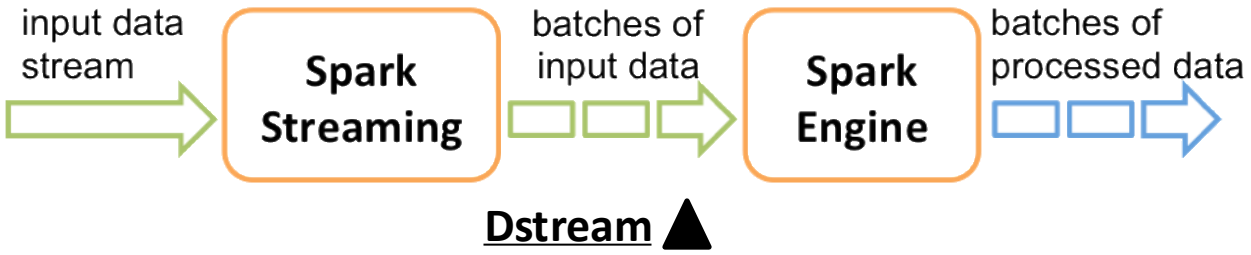
Dstream

Image credits: https://spark.apache.org/docs/latest/streaming-programming-guide.html

Spark Streaming

DStream（离散流）：

- DStream（Discretized Stream）是Spark Streaming中的核心概念，代表一个连续的数据流。它可以从各种输入源（如Kafka、Flume、Kinesis等）创建，或者由其他DStreams衍生而来。
- Dstream本质上是一系列的RDD（弹性分布式数据集）。
- 通过转化为一系列的RDD允许Spark Streaming利用Spark的快速、分布式处理能力。
- DStream支持许多类似于RDD的转换操作，如map（映射）、count（计数）、join（连接）等。



案例分析

案例背景：实时社交媒体数据分析：
假设一个公司希望分析来自社交媒体的实时数据，以监测品牌声誉、用户参与度，并从中提取有价值的信息。

---

1. **数据收集（使用Kafka）：**
   - 社交媒体数据（如推文、评论）通过各种API实时收集。
   - 这些数据被发布到Kafka主题中，Kafka作为消息队列处理这些实时流数据。

2. **数据处理和分析（使用Spark Streaming）：**

- Spark Streaming从Kafka中订阅数据流。
- 使用Spark的强大数据处理能力，比如：
  - 对数据进行清洗和转换。
  - 应用自然语言处理来分析用户情绪。
  - 通过实时聚合计算用户参与度指标。

案例分析

案例背景：实时社交媒体数据分析：
假设一个公司希望分析来自社交媒体的实时数据，以监测品牌声誉、用户参与度，并从中提取有价值的信息。

---

3. **数据存储和进一步分析：**
   - 经过初步处理的数据可以存储到HDFS或其他数据存储系统中，用于进一步的批处理分析或长期存储。
   - 使用Spark SQL进行复杂的查询和深入分析。
4. **实时dashboard和报告：**
   - 处理后的数据用于生成实时dashboard，提供即时的品牌监控和市场信息。
   - 生成定期报告，用于战略决策和市场策略调整。

课堂思考

请简述：在电商平台的背景下，讨论一个实时推荐系统可能如何利用Kafka和Spark Streaming来处理数据。

课堂思考

1. **数据收集：**使用Kafka收集用户行为数据，比如页面浏览、商品点击、购买等。
2. **数据处理：**Spark Streaming作为Kafka的订阅者，实时处理流入的数据。Spark Streaming会实施窗口操作，以便对最近的用户行为进行分析，比如分析用户在过去几分钟内的浏览和点击行为。
3. **特征提取与推荐模型：**Spark Streaming可以实现实时特征提取，将原始用户行为转换为推荐算法所需的输入特征。使用预先训练好的机器学习模型（例如，基于内容的推荐模型等），Spark Streaming可以实时预测用户可能感兴趣的商品。

课堂思考

4. **推荐结果输出：**计算出的推荐可以即时推送到用户的当前会话中，例如，通过网页或者app界面更新推荐商品列表。同时，推荐结果也可以被存储起来，供后续的分析或者用于改进推荐模型。

5. **反馈循环：**用户对推荐结果的响应（比如点击推荐商品）也会被实时捕获，并送回Kafka，形成一个闭环系统。这个实时反馈可以用来进一步优化推荐算法，使推荐更加精准。