Introduction to Spark

Agenda

State of hardware in 2000
State of hardware now
Big data processing needs now
Third generation distributed systems
Apache Spark
Why Spark
RDD - Resilient Distributed Dataset
Spark – RDD Persistence
Spark vs. Hadoop MapReduce
Spark Execution Flow
Terminology

State of hardware in 2000

Hard Disk was cheap.

So disk was primary source of data

Network was costly so data locality

RAM was very costly

Single core machines were dominant

State of hardware now

RAM is the king.

RAM is primary source of data.

We use disk only for fallback.

Network is speedier.

Multi core machines are easily available.

Big data processing needs now

All companies use big data.

Velocity is as much concern as volume.

Needs of real time are as much important as batch processing.

Use cases are not just limited to search.

Third generation distributed systems

Handle both batch processing and real time.

Exploit RAM as much as disk.

Multiple core aware.

Do not reinvent the wheel.

They use …

HDFS for storage.

YARN for distribution.

Plays well with Hadoop

Apache Spark

A fast and general engine for large scale data processing.

Written in Scala.

Licensed under Apache.

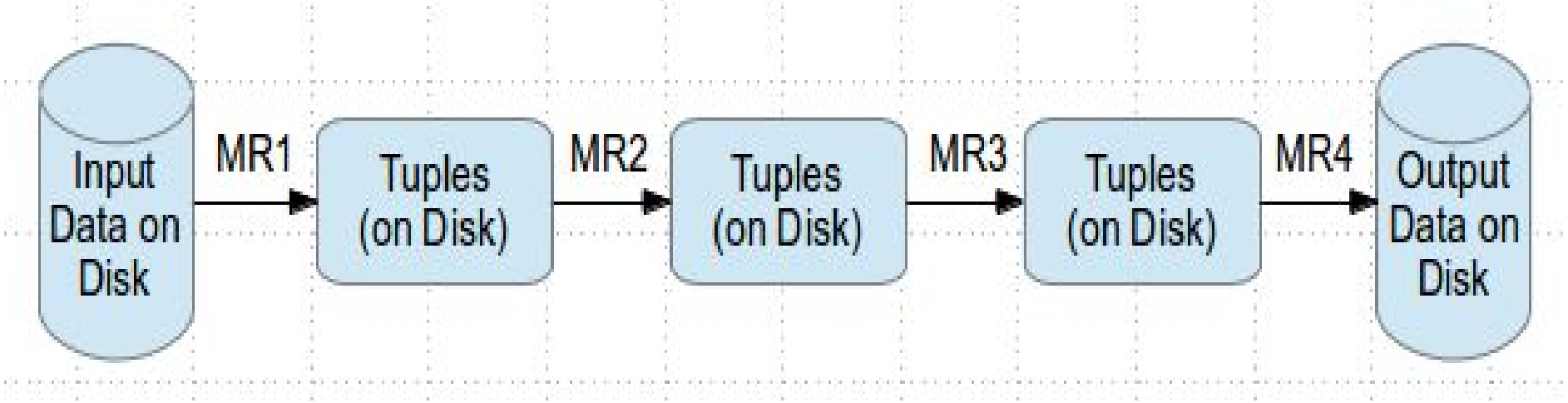
- Processing engine; instead of just "map" and "reduce", defines a large set of *operations* (transformations & actions)
  - Operations can be arbitrarily combined in any order
- Open source software
- Supports Java, Scala and Python
- Key construct: Resilient Distributed Dataset (RDD)

Why Spark

Most of Machine Learning Algorithms are iterative because each iteration can improve the results

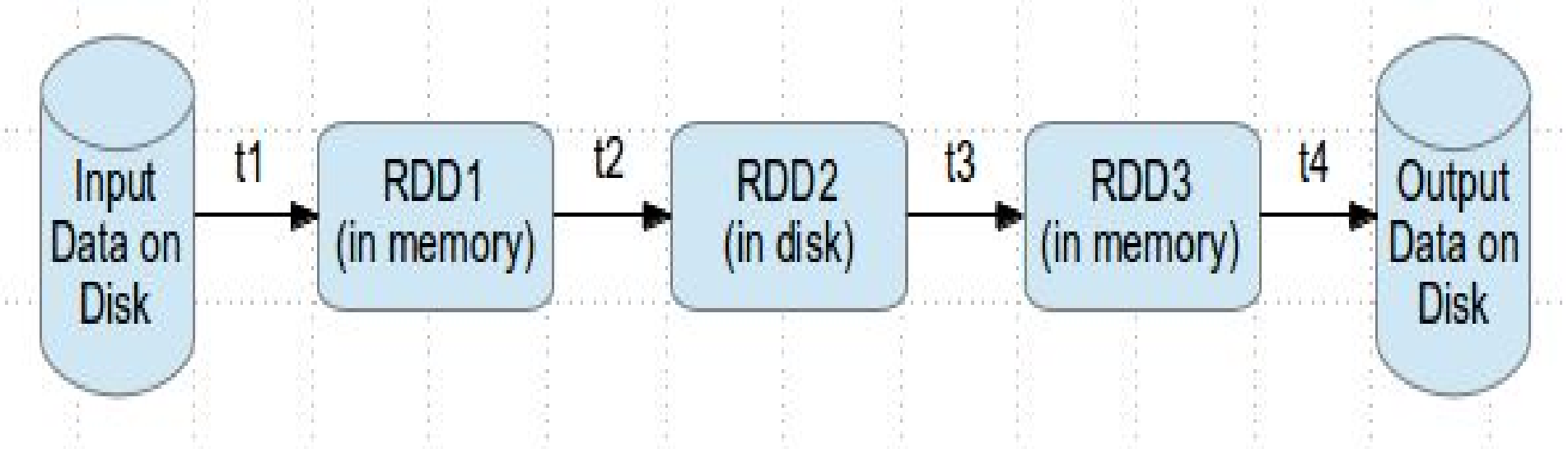
With Disk based approach each iteration's output is written to disk making it slow

Hadoop execution flow



Why Spark

Spark execution flow



Resilient Distributed Dataset (RDD)

- Resilient Distributed Dataset (RDD) is a basic Abstraction in Spark
- Immutable, Partitioned collection of elements that can be operated in parallel
- Basic Operations
  - map
  - filter
  - persist
- RDD main characteristics:
  - A list of partitions
  - A function for computing each split
  - A list of dependencies on other RDDs

RDD - Resilient Distributed Dataset

- RDDs represent data or transformations on data
- RDDs can be created from Hadoop InputFormats (such as HDFS files, or by transforming other RDDs (you can stack RDDs)
- Actions can be applied to RDDs; actions force calculations and return values
- Lazy evaluation: Nothing computed until an action requires it
- RDDs are best suited for applications that apply the same operation to all elements of a dataset

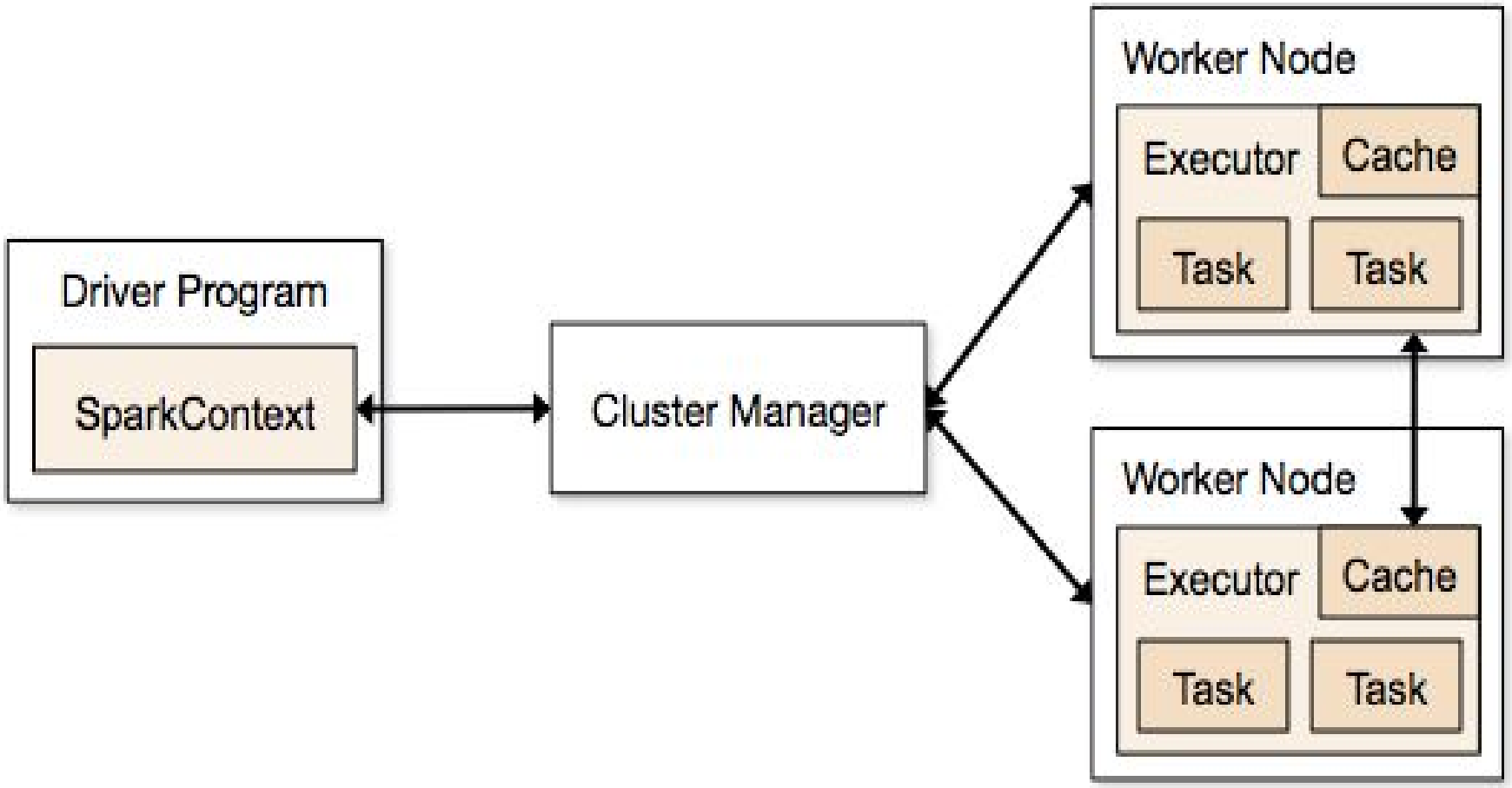
Spark – RDD Persistence

- You can persist (cache) an RDD
- When you persist an RDD, each node stores any partitions of it that it computes in memory and reuses them in other actions on that dataset (or datasets derived from it)
- Allows future actions to be much faster (often >10x).
- Mark RDD to be persisted using the persist() or cache() methods on it. The first time it is computed in an action, it will be kept in memory on the nodes.
- Cache is fault-tolerant – if any partition of an RDD is lost, it will automatically be recomputed using the transformations that originally created it
- Can choose storage level (MEMORY_ONLY, DISK_ONLY, MEMORY_AND_DISK, etc.)
- Can manually call unpersist()

Spark vs. Hadoop MapReduce

- Performance: Spark normally faster but with caveats
  - Spark can process data in-memory; Hadoop MapReduce persists back to the disk after a map or reduce action
  - Spark generally outperforms MapReduce, but it often needs lots of memory to do well; if there are other resource-demanding services or can't fit in memory, Spark degrades
- Ease of use: Spark is easier to program
- Data processing: Spark more general

Spark Execution Flow



Terminology

- **Worker Node :** Node that run the application program in cluster
- **Executor**
  - Process launched on a worker node, that runs the Tasks
  - Keep data in memory or disk storage
- **Task :** A unit of work that will be sent to executor
- **Job**
  - Consists multiple tasks
  - Created based on a Action
- **Stage :** Each Job is divided into smaller set of tasks called Stages that is sequential and depend on each other
- **SparkContext :**
  - represents the connection to a Spark cluster, and can be used to create RDDs, accumulators and broadcast variables on that cluster.

…Thanks…