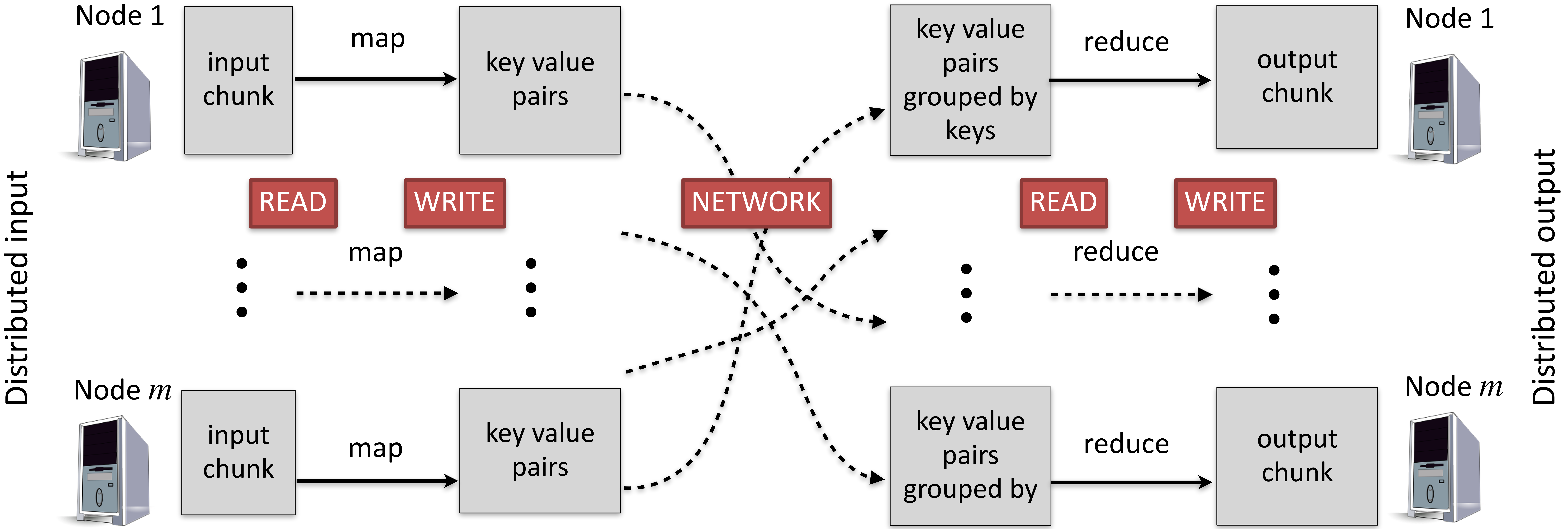
Apache Spark

Debapriyo Majumdar
Indian Statistical Institute
debapriyo@isical.ac.in

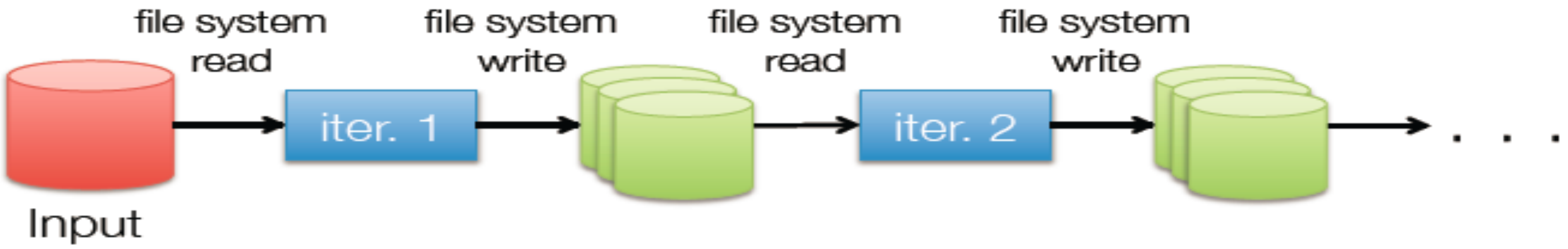
The MapReduce Paradigm

Vanilla MapReduce is based on acyclic data flow



- *Inefficient* for applications that repeatedly reuse a working set of data. For example:
  - Iterative algorithms (machine learning, graphs)
  - Interactive data mining tools (functionality similar to R, Python): with Hadoop, apps need to reload data from stable storage for each query

Limitations of Vanilla MapReduce



- Map-reduce is fine for one-pass computation, but inefficient for multi-pass algorithms
  - Examples: k-means, PageRank
- No efficient mechanism for data sharing
  - State between steps goes to distributed file system
  - Slow due to replication & disk storage
- Not interactive or flexible
  Have to write *map* and *reduce* for any task
- Commonly spend 90% of time doing I/O

Apache Spark

- Goals
  - Extend the MapReduce model to better support two common classes of analytics apps:
    - Iterative algorithms (machine learning, graph)
    - Interactive data mining
  - Enhance programmability

- Approach: **Resilient Distributed Dataset (RDD)**
  - Allow apps to keep working sets in memory as long as possible
  - Retain the advantages of MapReduce (fault tolerance, data locality, scalability)
  - Support a wide range of applications

RDDs

- An RDD is a read-only , partitioned collection of records
- Can only be created by :
  - (1) Data in stable storage
  - (2) Other RDDs (transformation , lineage)
- Each RDD include:
  - 1) A set of partitions (atomic pieces of datasets)
  - 2) A set of dependencies on parent RDDs
  - 3) A function for computing the dataset based on its parents
  - 4) Metadata about its partitioning scheme
  - 5) Data placement
- An RDD has enough information about how it was derived from other datasets(its lineage)
  - Fault tolerance
- Users can control two aspects of RDDs
  - (1) Persistence (in RAM, reuse)
  - (2) Partitioning (hash, range, [<k, v>])
- Transformations are lazy, they don't compute right away. Just remember the transformations applied to datasets(lineage). Only compute when an action require.

Programming languages

- Spark is implemented in Scala
- Allows interactive use from Scala interpreter
- Scala
  - High-level language for JVM
  - Object-oriented + Functional programming
  - Statically typed
  - Comparable in speed to Java
  - No need to write types due to type inference
  - Interoperates with Java
  - Can use any Java class, inherit from it, etc;
  - Can also call Scala code from Java
- Python (Pyspark)
  - Slower than Java / Scala in performance (about 2-3 times)
  - But gaining a lot of popularity because python is used widely nowadays

Memory Management

- Spark provides three options for persist RDDs:
  - In-memory storage as deserialized Java Objects
    - fastest, JVM can access RDD natively
  - In-memory storage as serialized data
    - space limited, choose another efficient representation, lower performance cost
  - On-disk storage
    - RDD too large to keep in memory, and costly to recompute
    - Defaults to (almost) vanilla MapReduce for certain tasks

References and Acknowledgments

- Credits for some of the slides go to Wen Zhiguang and Jiaul Paik