CSE 594: Spatial Data Science & Engineering

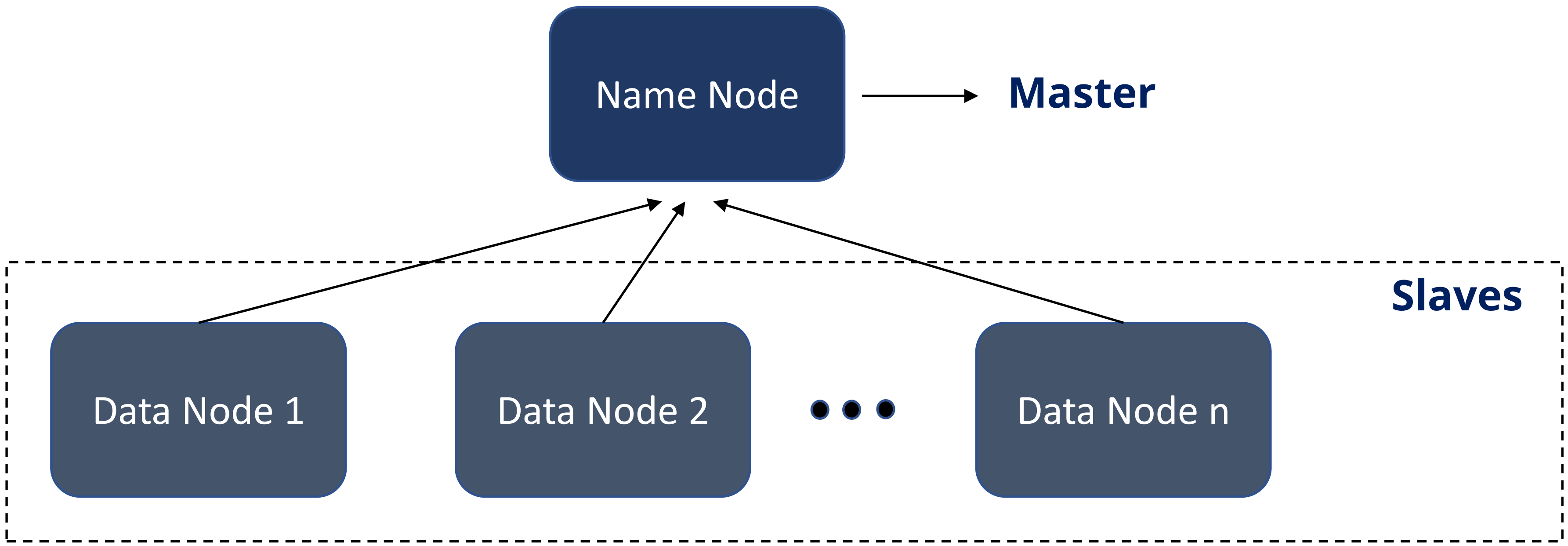
Lecture 9

Big Spatial Data Systems

Apache Hadoop

- A framework that uses distributed storage and parallel processing to store and manage big data
- An ecosystem consisting of three components:
  - ❑ Hadoop HDFS – the storage unit
  - ❑ Hadoop YARN – resource management unit
  - ❑ Hadoop MapReduce – data processing unit

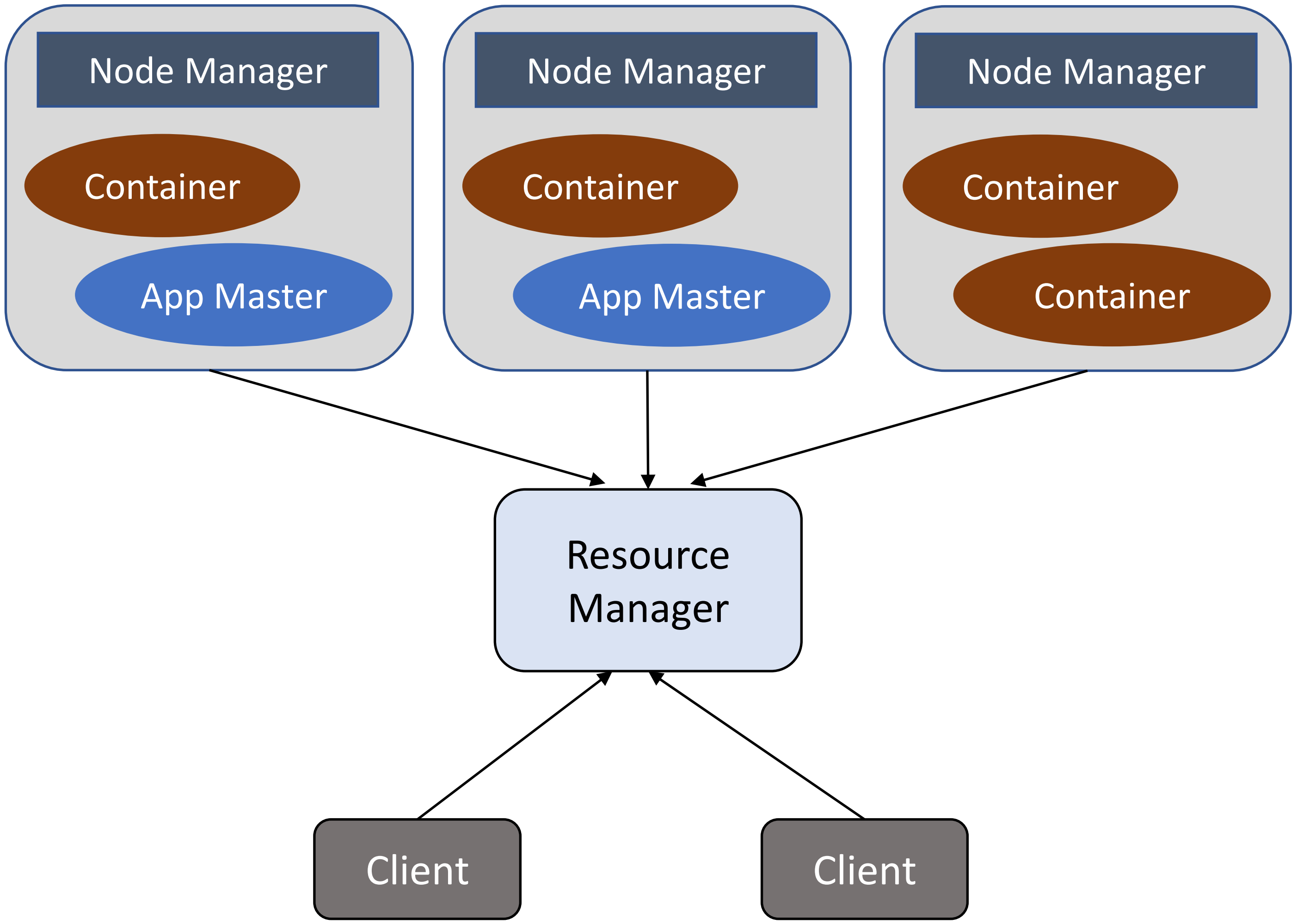
HDFS – Hadoop Distributed File System



- Two components – one name node and multiple data nodes
- Data is divided into 128 MB blocks and distributed across data nodes
- Name node stores the meta data and manages data nodes
- Data nodes read, write, process and replicate the data, and send signals to the name node

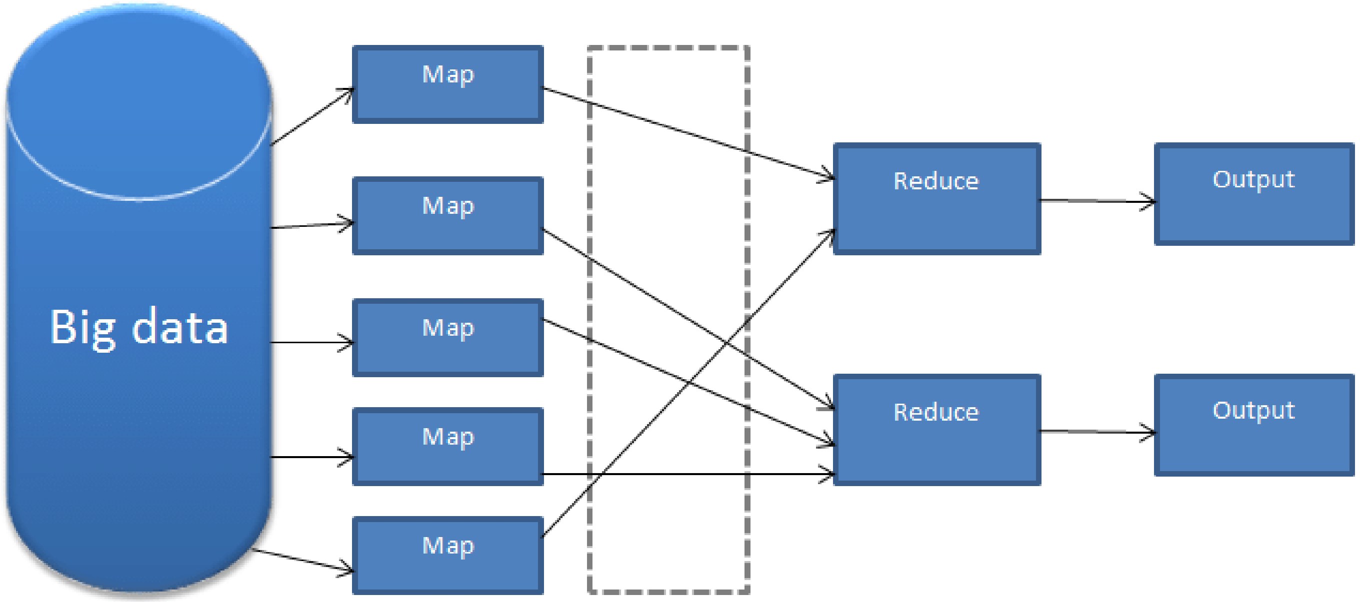
Hadoop YARN – Yet Another Resource Negotiator

- Acts like an OS to Hadoop.
- Another file system on top of HDFS
- Responsible for managing cluster resources
- Performs job scheduling
- Allows Hadoop to run different distributed applications simultaneously
- Separates the processing layer from resource management layer

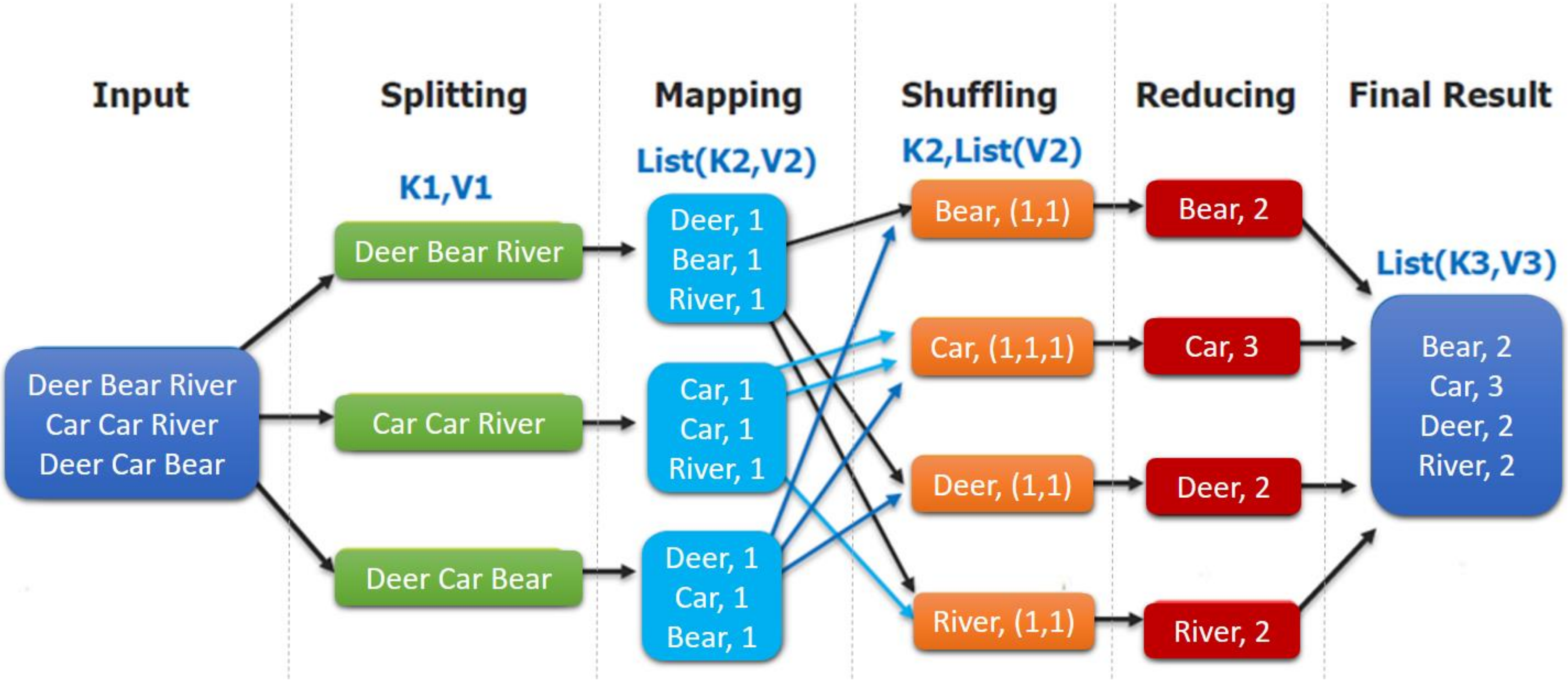


Hadoop MapReduce

- A programming model for parallelizing tasks into multiple nodes
- Consists of two phases – mapping phase followed by reducing phase
- Mapper reads and processes the data to produce key-value pairs
- Reducer takes the outputs from mappers and aggregates the key-value pairs to produce the final output

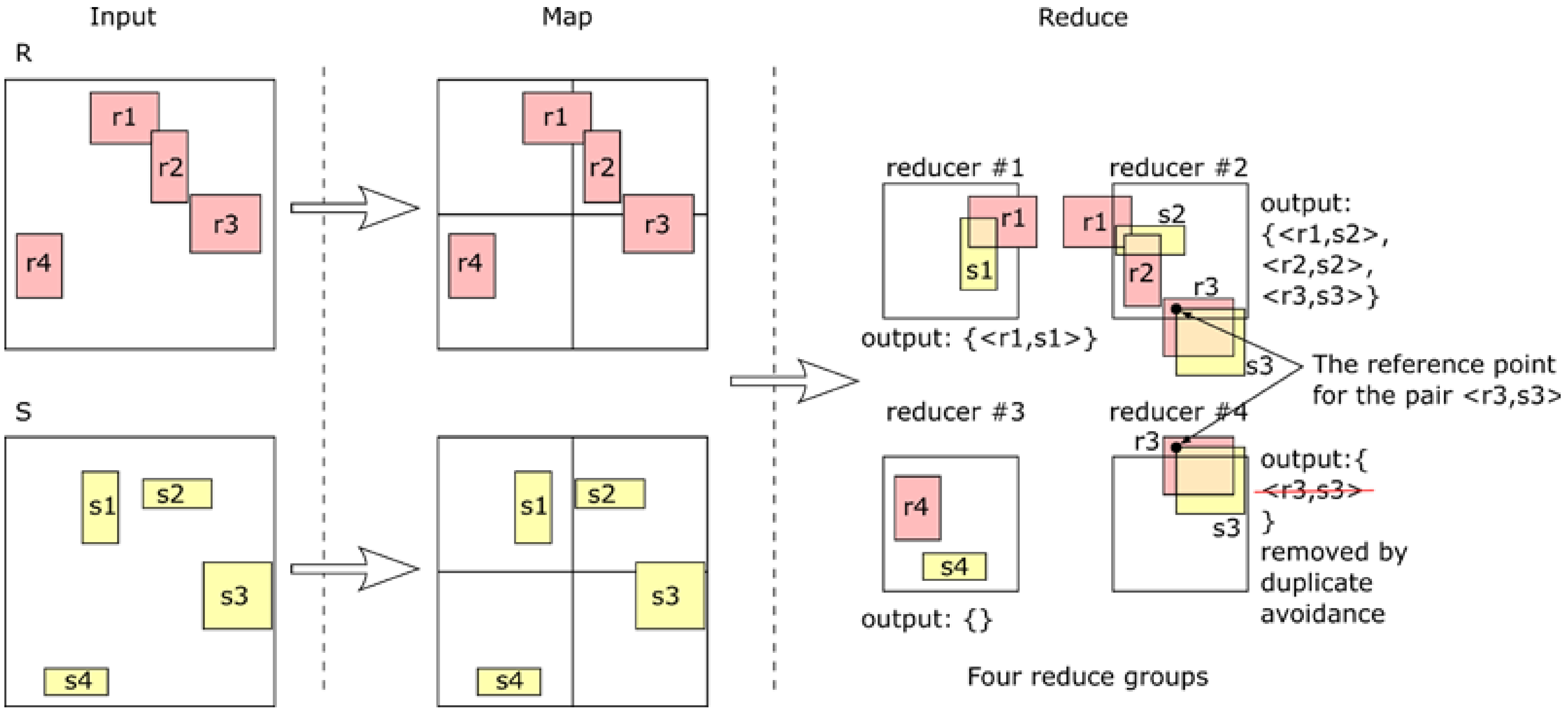


Simple MapReduce Example



MapReduce Algorithm for Word Counting Problem

MapReduce for Spatial Join



MapReduce for Spatial Join

- Partitions the input rectangles according to a fixed sized grid
- Store the records in the corresponding cells
- Records overlapped with multiple cells need to be replicated in all overlapping cells
- Perform the join in each machine in parallel
- A mapper reports the pairs of cell ids and spatial objects in the corresponding partition
- A reducer collects the pairs from mappers and performs the join operation followed by a duplicate removal step

Removing Duplicates

- Compute the reference point (top left corner of the intersection) for a candidate joining pair
- Report a pair only if the reference point lies within the grid cell

MapReduce for Spatial Operations

- Some spatial operations may need only one of either mapper or reducer in practice

Computing Dataset Boundary

- Needs only reduce operation
- Calculate the merged rectangular boundary of spatial objects two by two until all objects are aggregated
- After calculating the boundary for each partition, aggregate the boundaries of partitions

Transforming Reference System

- Needs only mapping operation
- For each partition, traverse the objects in the corresponding partition and convert their SRS

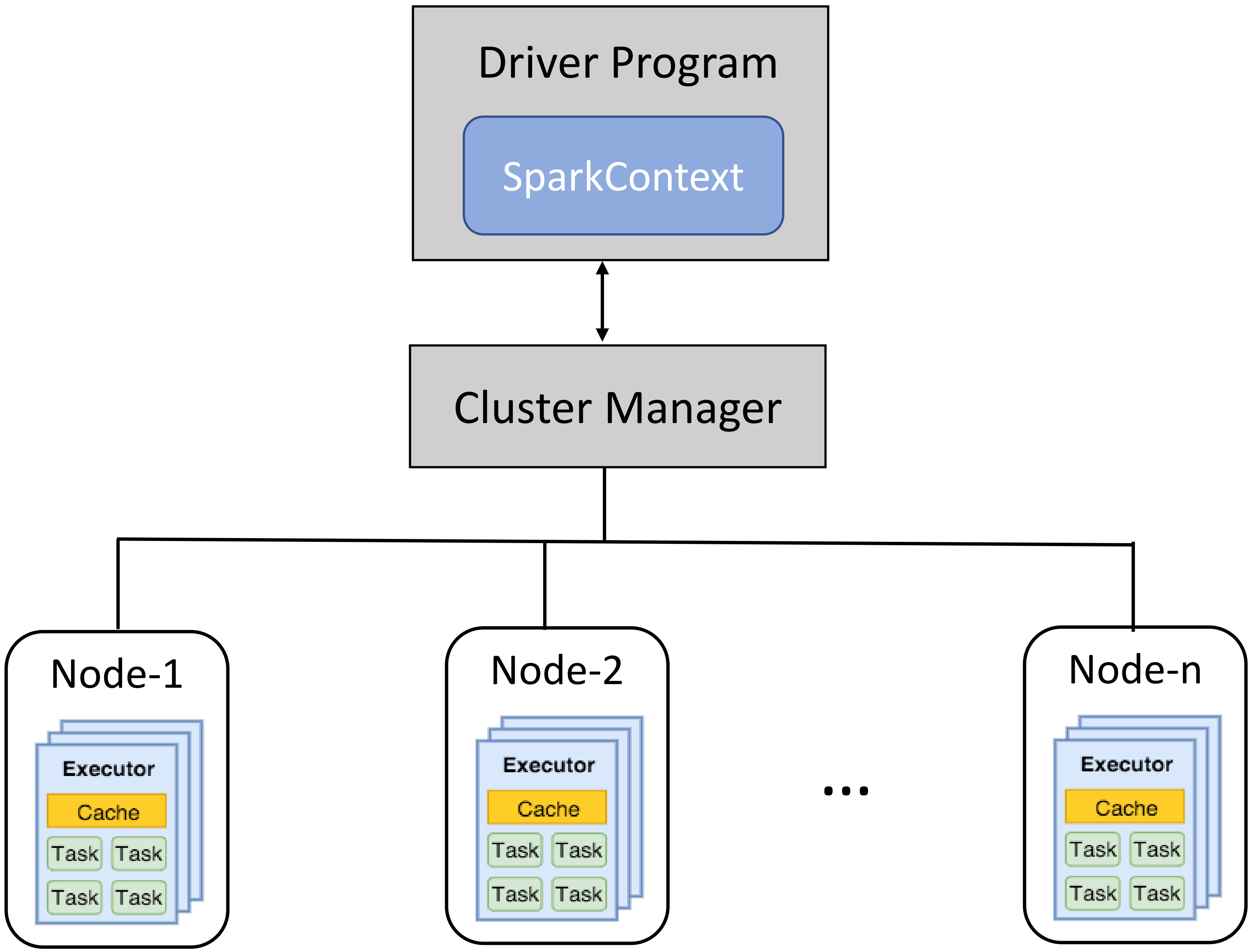
Apache Spark

- Distributed Data Processing Framework
- Can perform processing tasks on very large datasets
- Distribute data processing tasks across multiple computers
- Supports in-memory data caching and reuse across computations
- Supports majority of the data formats and various storage systems

Spark Key Concepts

- Job
- Stages
- Tasks
- DAG
- Executor

Spark Components



Spark Driver Components

Spark RDD

- Resilient Distributed Dataset
- Spark representation of a set of data

Resilient → Fault tolerant, able to recompute during failure

Distributed → Data resides on multiple nodes

Dataset → Records of data. Loaded from JSON, CSV, etc.

An RDD is an immutable distributed collection of objects. Each RDD is split into multiple partitions and can be computed on different nodes

Properties of Spark RDD

- In-memory computation
- Lazy evaluation
- Fault tolerance
- Immutable
- Partitioning
- Persistence
- Coarse-grained operations

Spark RDD Operations

- **Transformations**

  Takes an RDD as input and produce one or more RDDs as output

- **Actions**

  RDD operations that produce non-RDD values. One of the ways to send results from executors to the driver.

  - Collect()
  - Count()

Advantages of Apache Spark

- Speed
  - Spark’s in memory data engine can perform tasks up to hundred times faster than Hadoop map-reduce
  - Very efficient for tasks that require frequent write operations
  - Even if data cannot completely be contained in memory it is 10 times faster than map-reduce

- Developer friendly API
- Advanced DAG execution engine
- Numerous storage engines: HDFS, Cassandra, HBase, etc.

Challenges of Handing Spatial Objects with Spark RDDs

- Heterogeneous data sources
  - ❑ Spatial data is stored in a variety formats – CSV, GeoJSON, WKT, shape files, GeoTIFF, GeoParquet
- Complex geometrical shapes
  - ❑ Different objects in a dataset may have a variety of shapes – concave/convex polygon, multi-polygon, line, point
- Spatial partitioning
  - ❑ Default partitioner in Spark does not preserve spatial proximity
- Spatial index support
  - ❑ Spatial indexes are different from regular indexes used by spark

Apache Sedona

- A cluster computing system for processing large-scale spatial data, initially released as GeoSpark
- Add spatial support to Spark by solving the challenges – heterogeneous data sources, complex geometry shapes, spatial partitioning, and spatial index support
- Extends Apache Spark with a set of Spatial RDDs that efficiently load, process, and analyze large-scale spatial data across machines
- Facilitate to create spatial analytics and data mining applications and run them in any Spark environments

Why Apache Sedona?

- **High Speed**

  2X – 10X faster than other Spark-based geospatial data systems on computation intensive query workloads

- **Low Memory Consumption**

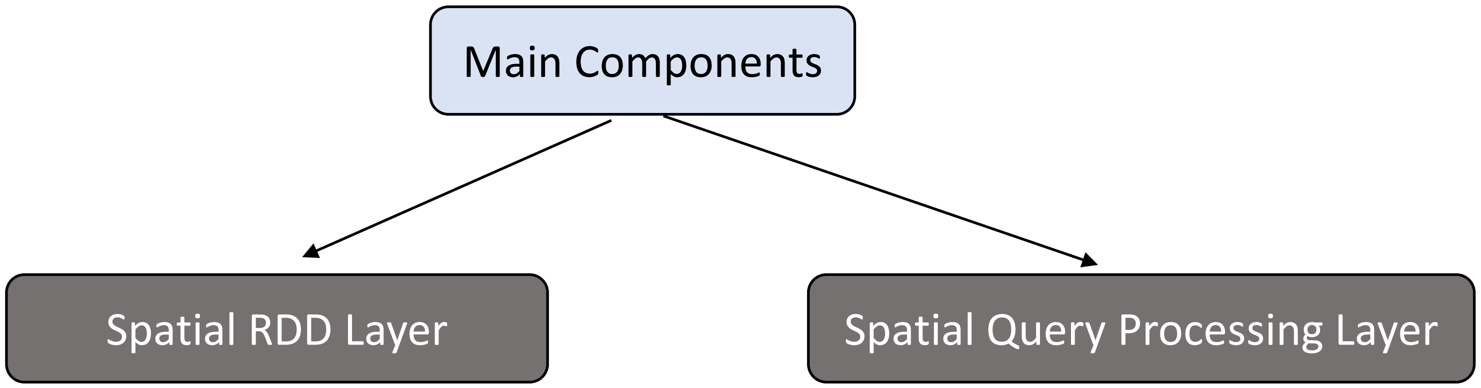
  50% less peak memory consumption than other Spark based geospatial data systems

- **Ease of Use**

  Spatial SQL API has supports for Scala, Java and Python. API is very easy to use

What Makes Apache Sedona Efficient?

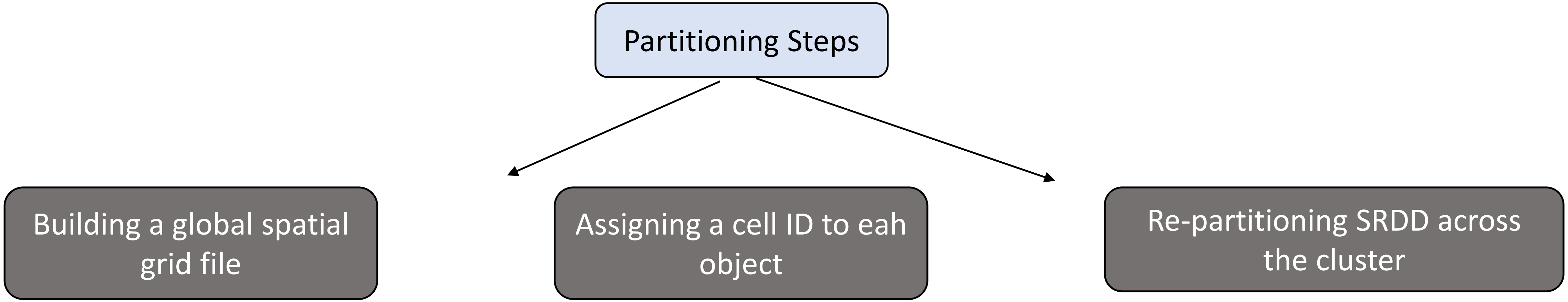
- Introduces Spatial Resilient Distributed Dataset (SRDD) which provides in-built support for geometrical and distance operations
- Spatial RDD can accommodate heterogeneous spatial objects which are very common in a GIS area
- Introduces spatial query processing layer which is optimized for spatial query operators such as range filter, distance filter, spatial k-nearest neighbors, range join and distance join
- When executing a spatial SQL query, the optimizer takes into account the execution time and produces a good query execution plan



Spatial RDD Layer

- Supports loading data from any all popular spatial formats – CSV, WKT/WKB, GeoJSON, Shape file, GeoTIFF, GeoParquet
- Supports heterogeneous geometry shapes – point, multi-point, linestring, multi-linestring, polygon, multi-polygon, circle, and geometry collection
- Provides a built-in library for executing geometrical computation on spatial RDDs
- Provides native support for many common geometrical operations

Spatial RDD Partitioning



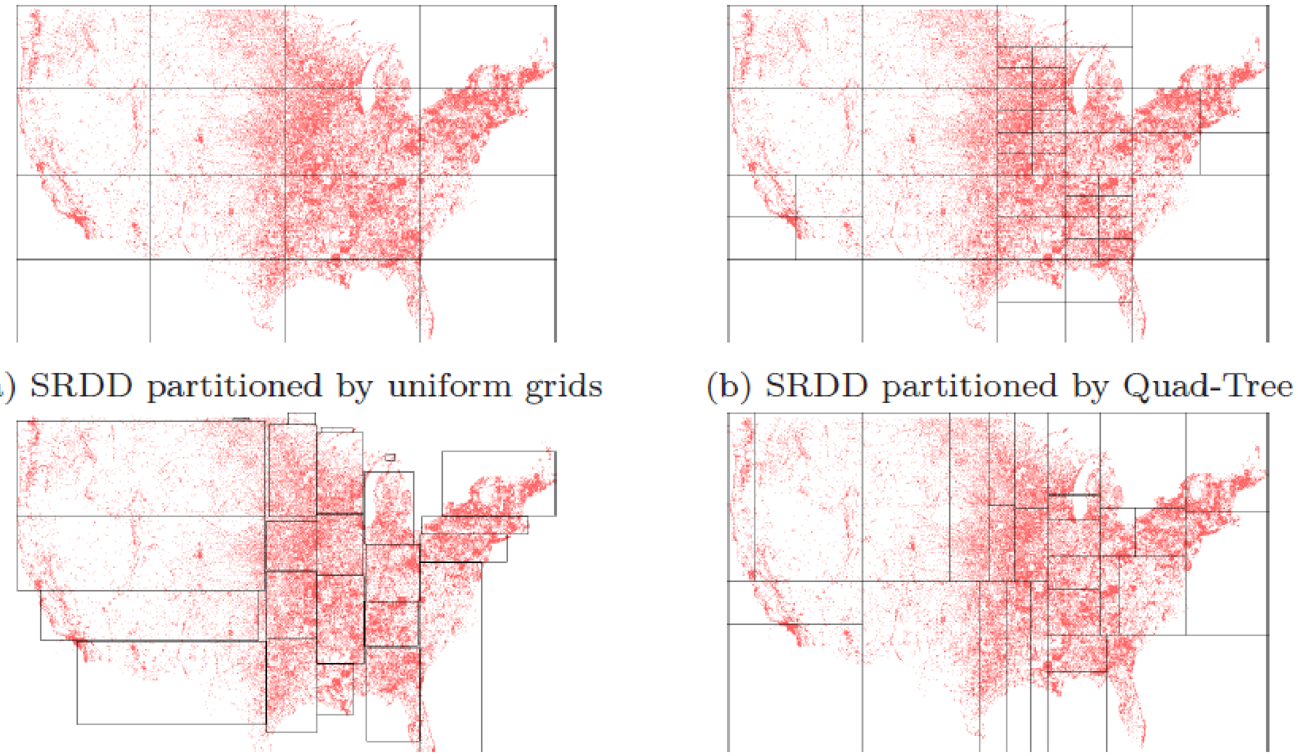
Building a Global Spatial Grid File

- Samples a subset of spatial data preserving the spatial distribution
- Create the partitions on the sampled data
- Partitioning on sampled data is used for entire dataset

Spatial RDD Partitioning

Building a Global Spatial Grid File

- Supports grid creation with following approaches
  - ❑ Uniform Grid
  - ❑ KDB-Tree
  - ❑ Quad-Tree
  - ❑ R-Tree



(a) SRDD partitioned by uniform grids
(b) SRDD partitioned by Quad-Tree
(c) SRDD partitioned by R-Tree
(d) SRDD partitioned by KDB-Tree

Spatial RDD Partitioning

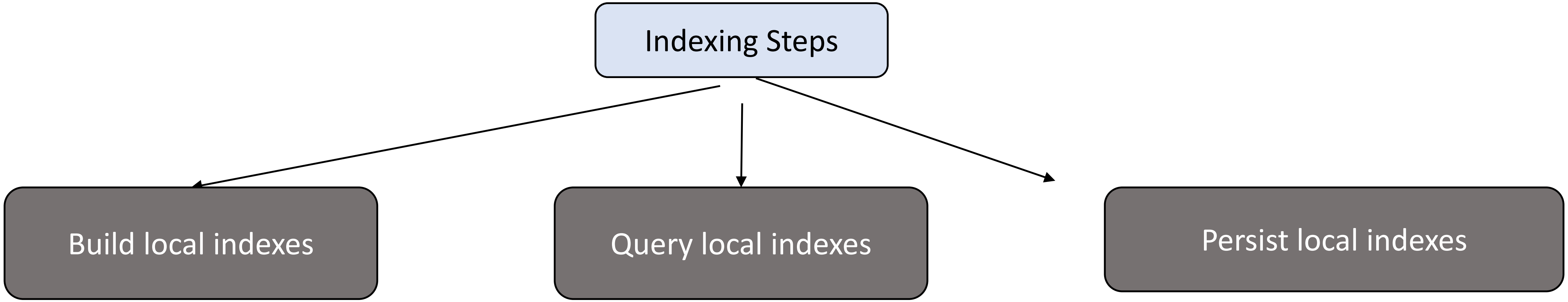
Assigning a Cell ID to Each Object

- Duplicates the grid files and broadcasts the copies to each spatial RDD partition
- After receiving broadcasted grid file, original spatial RDD partitions check each internal object against the grid file
- Results are stored in a new RDD of key-value pairs, where key is the grid ID and object is the spatial object having intersection with the corresponding grid

Re-partitioning SRDD Across the Cluster

- Spatial objects that have the same grid cell ID are grouped into the same partition
- Needs lots of shuffle across the cluster

Spatial RDD Indexing



Spatial RDD Custom Serializer

- Spark default serializer is efficient for regular data types
- Improves over spatial data types for faster data transfer across the cluster

Spatial Query Processing Layer

Processing Spatial Range and Distance Queries

- Applies filter and refinement strategy
- If spatial index exists, use the query window’s MBR to query the spatial index and return candidate results
- If no spatial index, filters spatial objects from all objects in a partition using query window
- Check the spatial relation between query window and candidate objects

Processing K Nearest Neighbor Queries

- 2 phases – selection phase and sorting phase
- Selection phase selects K-nearest objects from each partition using filter and refinement model
- Sort all objects selected by various partitions and return to k

Spatial Query Processing Layer

Processing Spatial Join Queries

- Zip partitions from both dataset
- Perform partition level local join
- Perform aggregation of outputs from local joins
- Remove duplicates using reference point approach
- Also support broadcast join for small datasets

Some Supported Spatial Operations

- Spatial Range Query
- Spatial Join
- Spatial KNN Query
- Spatial Indexing

Some Applications

- **Region heat map:** visualize taxi trip pickup points distribution in an entire city on a map
- **Spatial aggregation:** show the taxi trip pickup count distribution per taxi zone
- **Spatial co-location pattern mining:** Are taxi trip pickup points co-located with area landmarks such as airports, museums, hospitals, etc.?