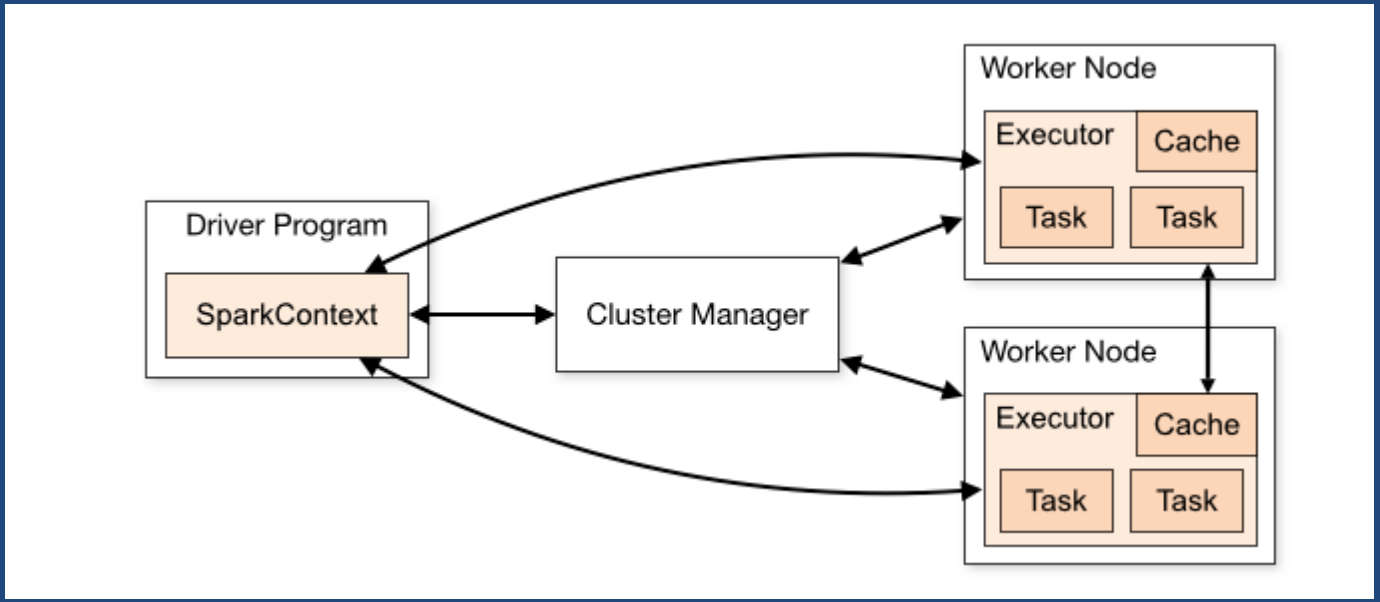
Apache Spark MLLib

Lab04 - Machine Learning

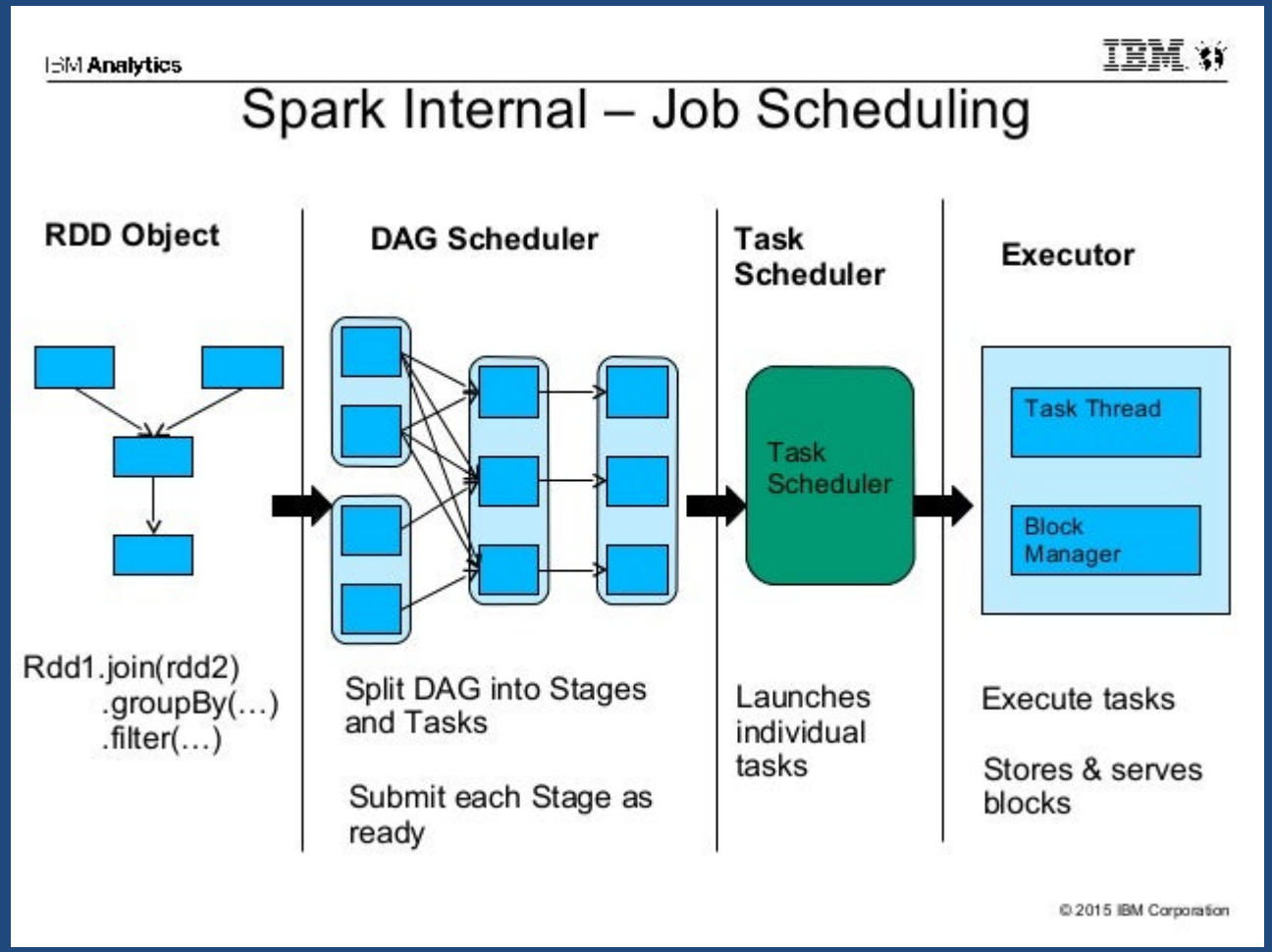
Remember…. What is Spark?

- A Big Data Platform that can divide up data, store it in memory (RAM & Cache) and process it across a cluster in a fault-tolerant way



Motivation for Spark

- Using Spark you can process large datasets that can not fit on a single machine
- You can process the data in parallel
- The code is executed in the different machines where the data is stored



What are Workers, Executers and Cores?

- When working with a Spark standalone cluster, understanding the roles of Workers, Executors, and Cores is crucial for designing efficient cluster operations.

Spark Worker

- In a Spark cluster, a Worker is a node that runs the application code in a distributed manner. Each Worker node has the resources (like CPU, memory, disk) to execute parts of the tasks assigned by the Spark driver. The Worker node communicates with the Spark Manager to get task assignments.

What are Executors inside Worker Nodes?

- An executor is a distributed process launched on Worker nodes. Each Executor is a Java process that runs in the Worker node and includes multiple components such as task threads. Executors perform the actual data processing required by your Spark application. They maintain data in memory across different stages of the job execution, which reduces I/O overhead and speeds up processing.

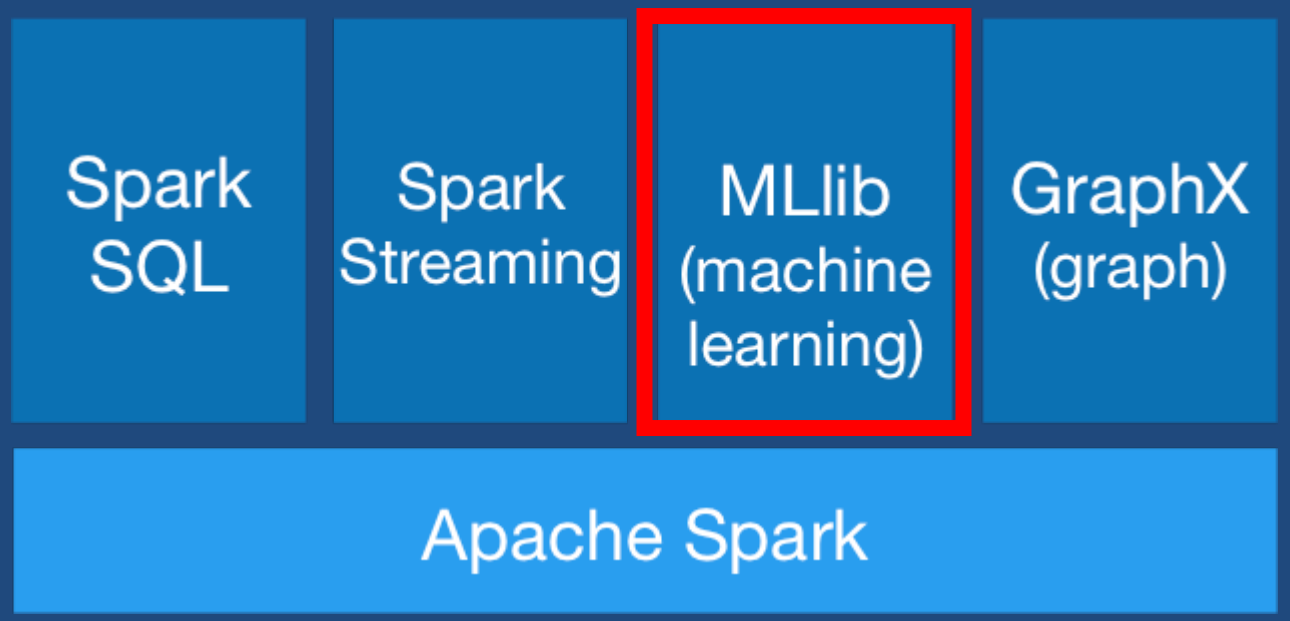
What are the cores inside a worker node?

- Cores refer to the computational capability of a Worker node. They determine the parallelism for task execution. Each task in Spark is a unit of work that will be assigned to a core for execution. The more cores available, the higher the number of tasks that can run simultaneously.

- Suppose you have a Worker node with 4 cores and a Spark application that has 8 tasks to execute. Here, 4 tasks can run in parallel on the Worker node, while the remaining tasks will wait until a core becomes free.

Motivation

- To help the project, we choose to give a lab on pySpark MLLib
- This library is used to distribute ML code on several computers
- It is transparent to the user

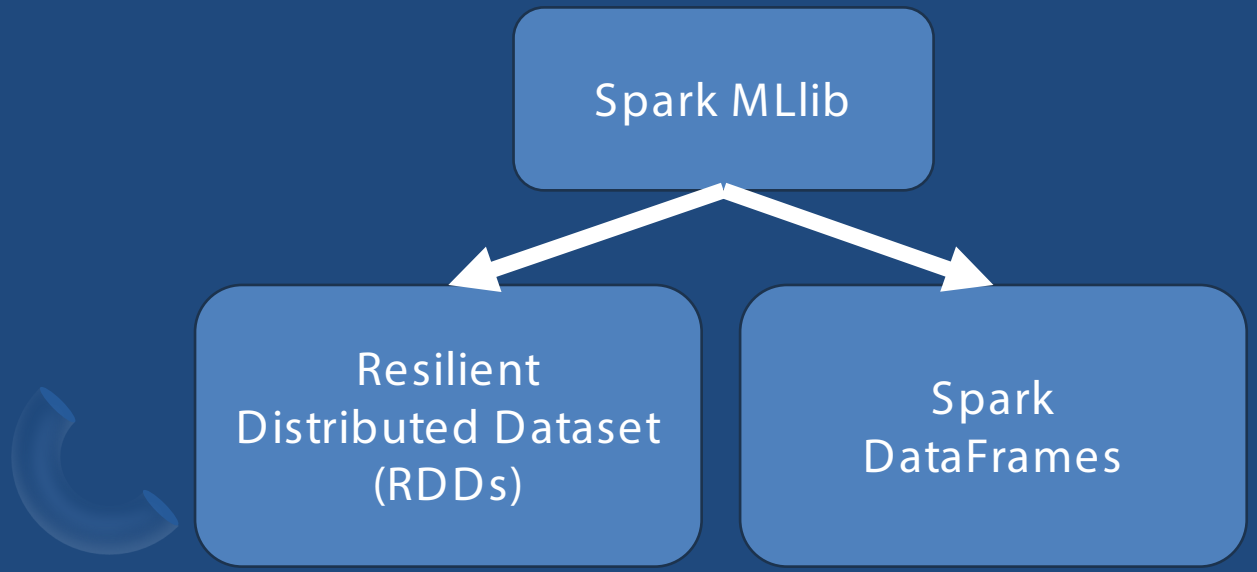


Distributed Library MLLib

- Apache Spark’s MLlib is a machine learning library designed to work with distributed computing. Underneath, it operates using Spark’s core RDD (Resilient Distributed Dataset) model or DataFrame. Here's a breakdown of how MLlib distributes computations:

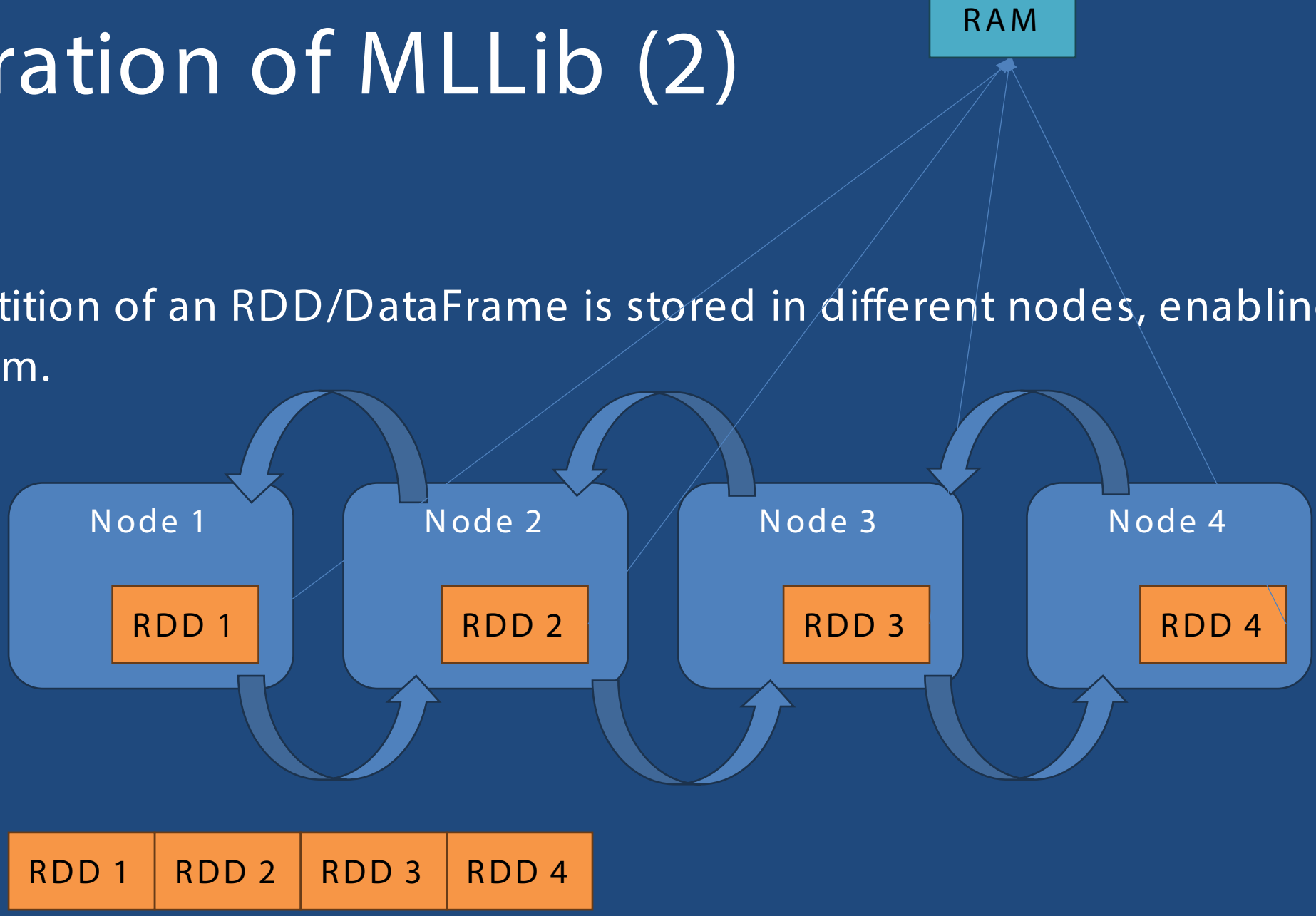
Operation of MLLib (1)

- Spark MLlib operates on RDDs or DataFrames, which are distributed across multiple worker nodes in a cluster.



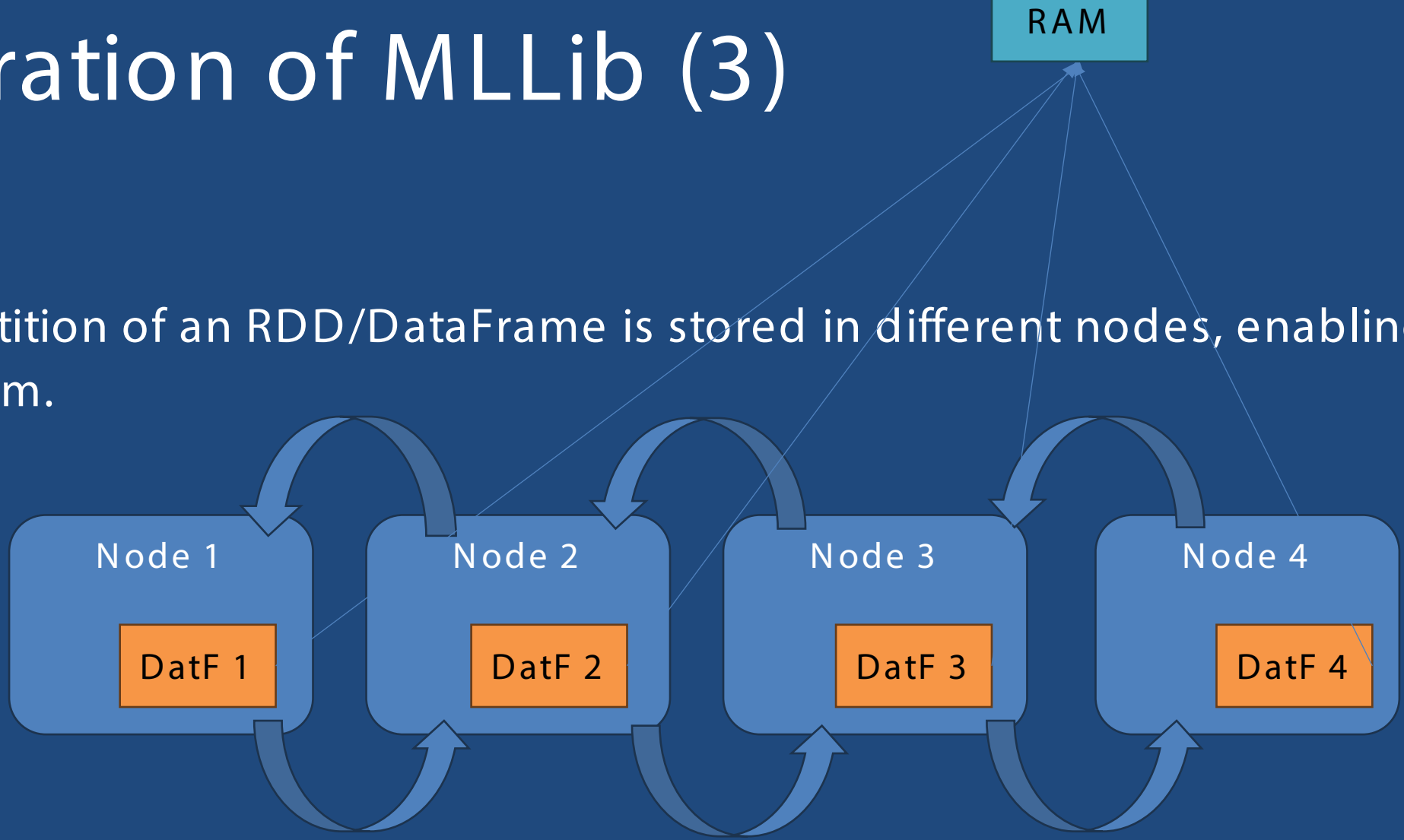
Operation of MLLib (2)

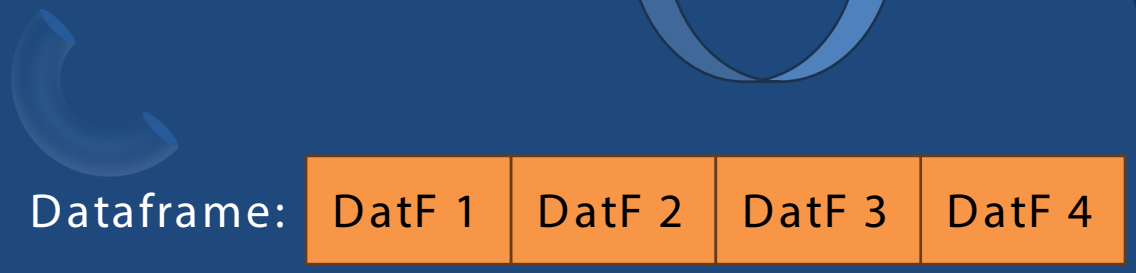
- Each partition of an RDD/DataFrame is stored in different nodes, enabling parallelism.



Operation of MLLib (3)

- Each partition of an RDD/DataFrame is stored in different nodes, enabling parallelism.





Data Partitioning in the RAM

The number of partitions is determined based on:

- The size of the dataset. (if its small choose more partitions)
- The number of available cores in the cluster. (use the full power of the cores)
- Configurations like spark.sql.shuffle.partitions.

```
rdd = spark.sparkContext.parallelize(range(1000), numSlices=4) # Creates 4 partitions
```

The dataset is divided into 4 partitions. Each partition is loaded into RAM of different worker nodes.

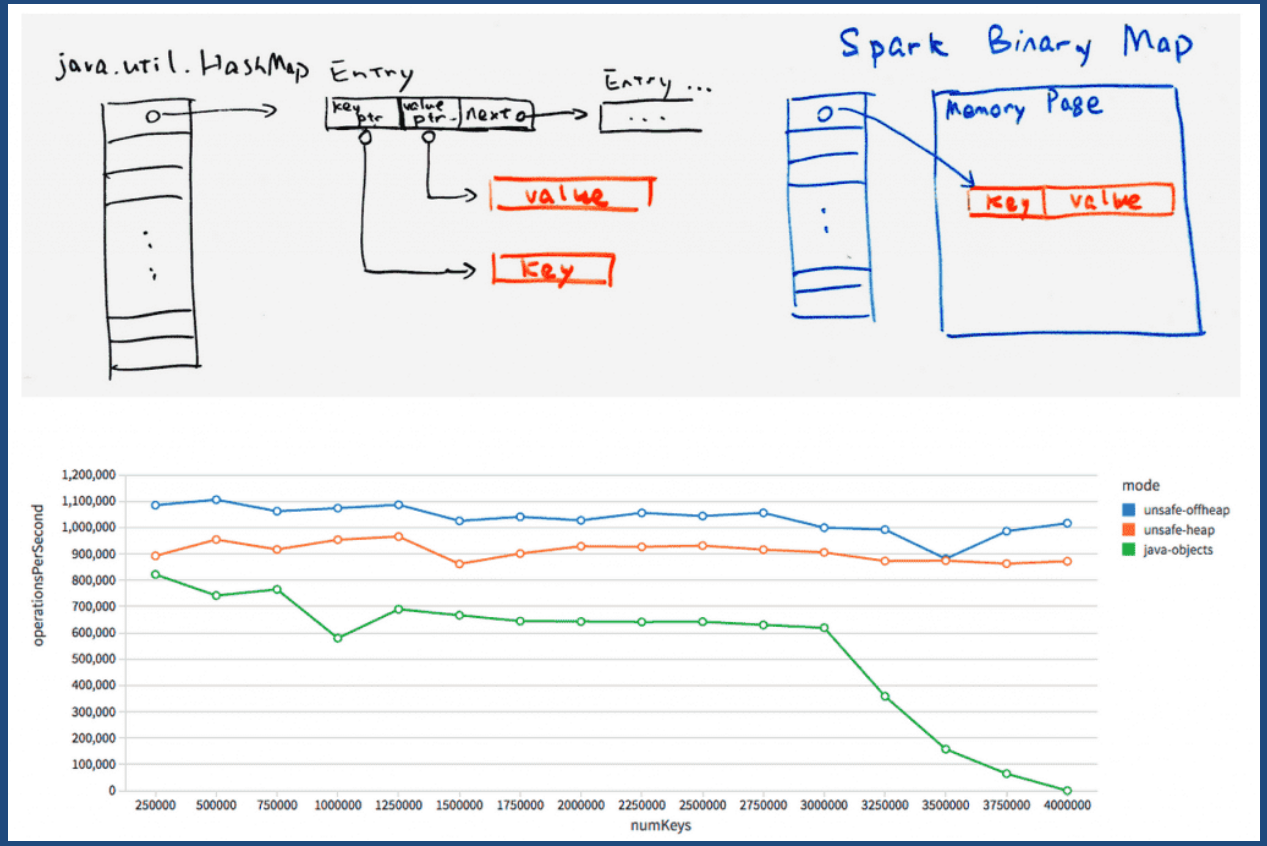
How Data is Stored in RAM

- Spark uses the Tungsten Execution Engine and Project Tungsten to optimize RAM usage.

- Tungsten is the codename for the umbrella project that made changes to Apache Spark's execution engine that focuses on substantially improving the efficiency of memory and CPU for Spark applications, to push performance closer to the limits of modern hardware.

Tungsten Project Ideas (some)

- Memory Management and Binary Processing: leveraging application semantics to manage memory explicitly and eliminate the overhead of JVM object model and garbage collection
- Cache-aware computation: algorithms and data structures to exploit memory hierarchy
- Intermediate data in memory vs CPU registers: Tungsten Phase 2 places intermediate data into CPU registers. This is an order of magnitudes reduction in the number of cycles to obtain data from the CPU registers instead of from memory



Difference RDD vs DataFrame

- Data can be stored in two primary ways:

1. Serialized (Compressed in Memory)
    1. Uses less RAM but requires deserialization when accessed.
    2. Default format for RDD caching.
2. Deserialized (Direct Access)
    1. Faster access but consumes more RAM.
    2. Default format for DataFrames.

Creating a RDD

```python
from pyspark import SparkContext

sc = SparkContext("local", "RDD Example")
data = [1, 2, 3, 4, 5]
rdd = sc.parallelize(data)
rdd.collect()
```

[1,2,3,4,5]

DataFrame Creation

```python
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("DataFrame Example").getOrCreate()
data = [("Alice", 1), ("Bob", 2)]
columns = ["Name", "ID"]
df = spark.createDataFrame(data, columns)
df.show()
```

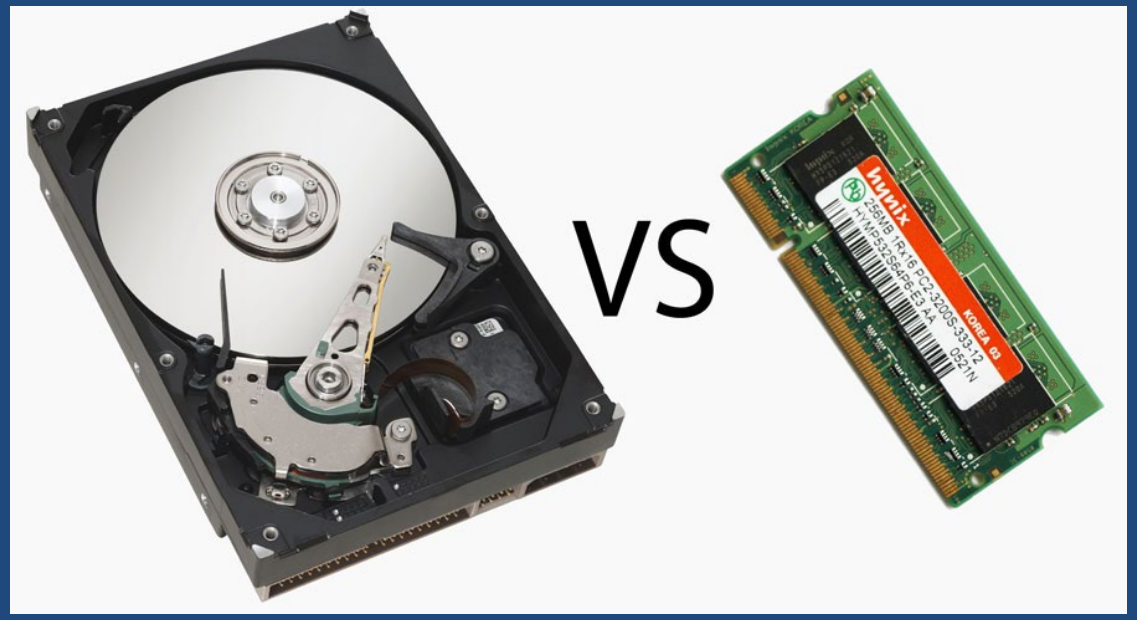
What happens when RAM is full?

If data exceeds available RAM, Spark:

1. Spills data to disk (disk-based persistence).
2. Evicts old cached data to free up space.
3. Uses memory compression to store more data in RAM.

Memory Spill

- If RAM is insufficient, Spark spills to disk.
- Memory spill in Apache Spark is the process of transferring data from RAM to disk, and potentially back again. This happens when the dataset exceeds the available memory capacity of an executor during tasks that require more memory than is available.



What is MLLib?

Apache Spark's scalable machine learning library.

Can be coded and accessed in Java, Scala, Python, and R

Shipped with Spark since version 0.8

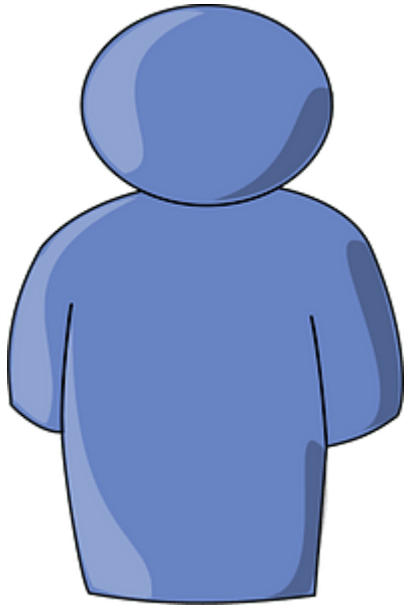
Its goal is to make practical machine learning scalable and easy

What is in MLLib?

- At a high level, it provides tools such as
  - ML Algorithms: common algorithms such as classification, regression, clustering, and others
  - Featurization: feature extraction, transformation, dimensionality reduction, and selection
  - Pipelines: tools for constructing, evaluating, and tuning ML Pipelines
  - Persistence: saving and load algorithms, models, and Pipelines
  - Utilities: linear algebra, statistics, data handling, etc.

Distributed Computing

MLlib is built on RDDs and supports DataFrame-based APIs in Spark (using spark.ml). But it is essentially transparent to the user!



It provides ML Pipelines for easy preprocessing, training, and deployment. [Next Lecture]

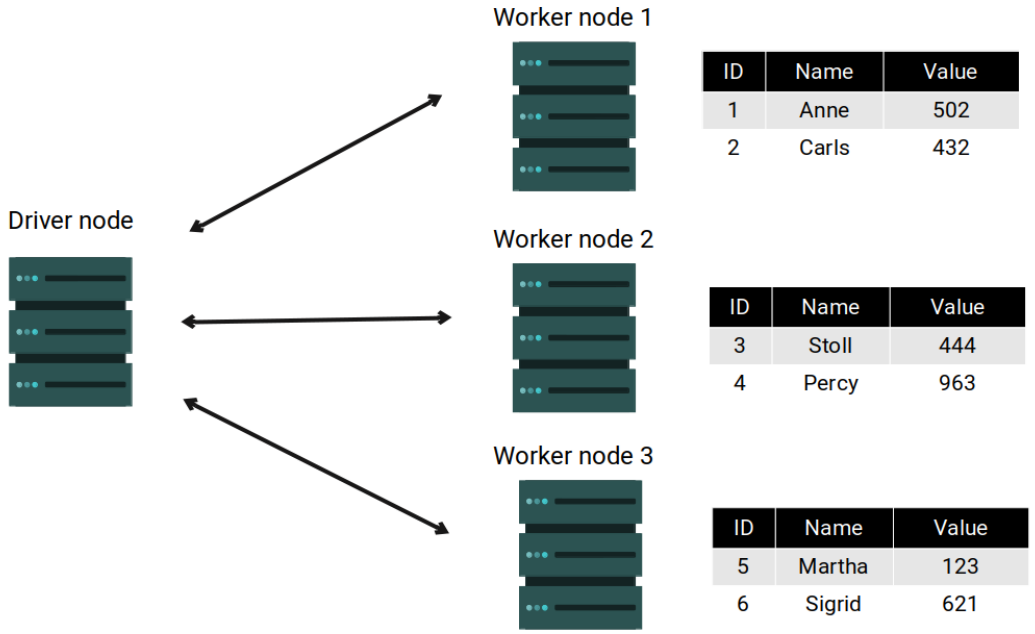
If you are working with Spark MLlib, the preferred API is spark.ml (DataFrame-based API) instead of spark.mllib (RDD-based API), as the latter is being deprecated.

Main concepts

- DataFrame: flexible data type from Spark SQL allowing parallelization.
- Transformer: Algorithm that transforms one DataFrame into another
- Estimator: Machine Learning algorithm that can be fitted on a DataFrame, returning a model
- Parameter: Uniform structures for Transformers and Estimators
- Pipeline: Chain of Transformers and Estimators

DataFrames

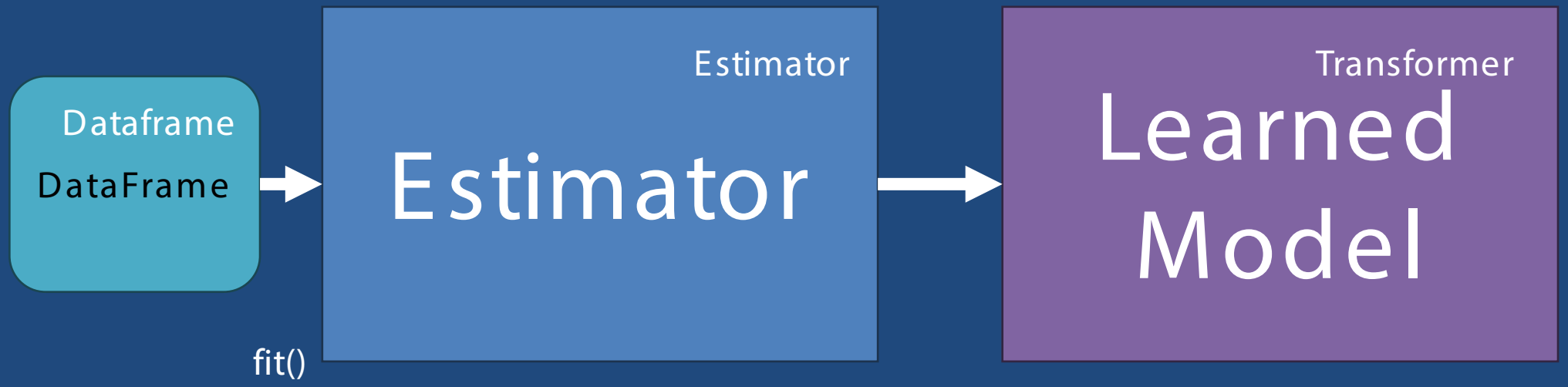
- Conceptually Similar to Tables in relational databases
- Inspired by data frames in Pandas
- Can be created from a file, regular RDD or other sources of data
- Data can be accessed in columns
- Columns has names



Transformers (Engineering)

- Algorithms, which Transform one DataFrame into another
- Transformers
  - Feature transformers
    - Tokenization, normalization, hashing
  - Learned models
    - Results from Estimators

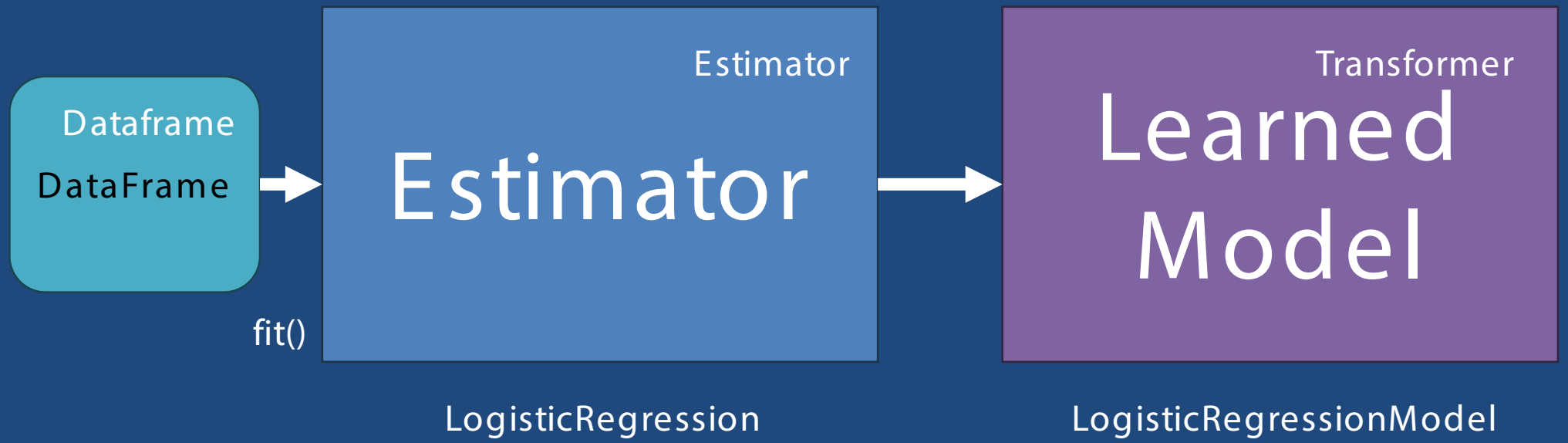
Transformers: Learned Models



Estimator: Learning Algorithm

- Gets fitted on training data
- Have a fit() method that gets a DataFrame as input
- Return a trained model, which is a transformer
- For example, a learning algorithm such as LogisticRegression is an Estimator, and calling fit() trains a LogisticRegressionModel, which is a Model and hence a Transformer

Example of LogisticRegressionModel



Parameters

- There are two main ways to pass parameters to an algorithm:
  - Set parameters for an instance.
  - E.g., if lr is an instance of LogisticRegression, one could call lr.setMaxIter(10) to make lr.fit() use at most 10 iterations.
  - This API resembles the API used in spark.mllib package.
  - Pass a ParamMap to fit() or transform(). Any parameters in the ParamMap will override parameters previously specified via setter methods.

Linear Algebra (Optimization Algorithms)

- Many ML algorithms in MLlib rely on linear algebra operations like matrix multiplications.

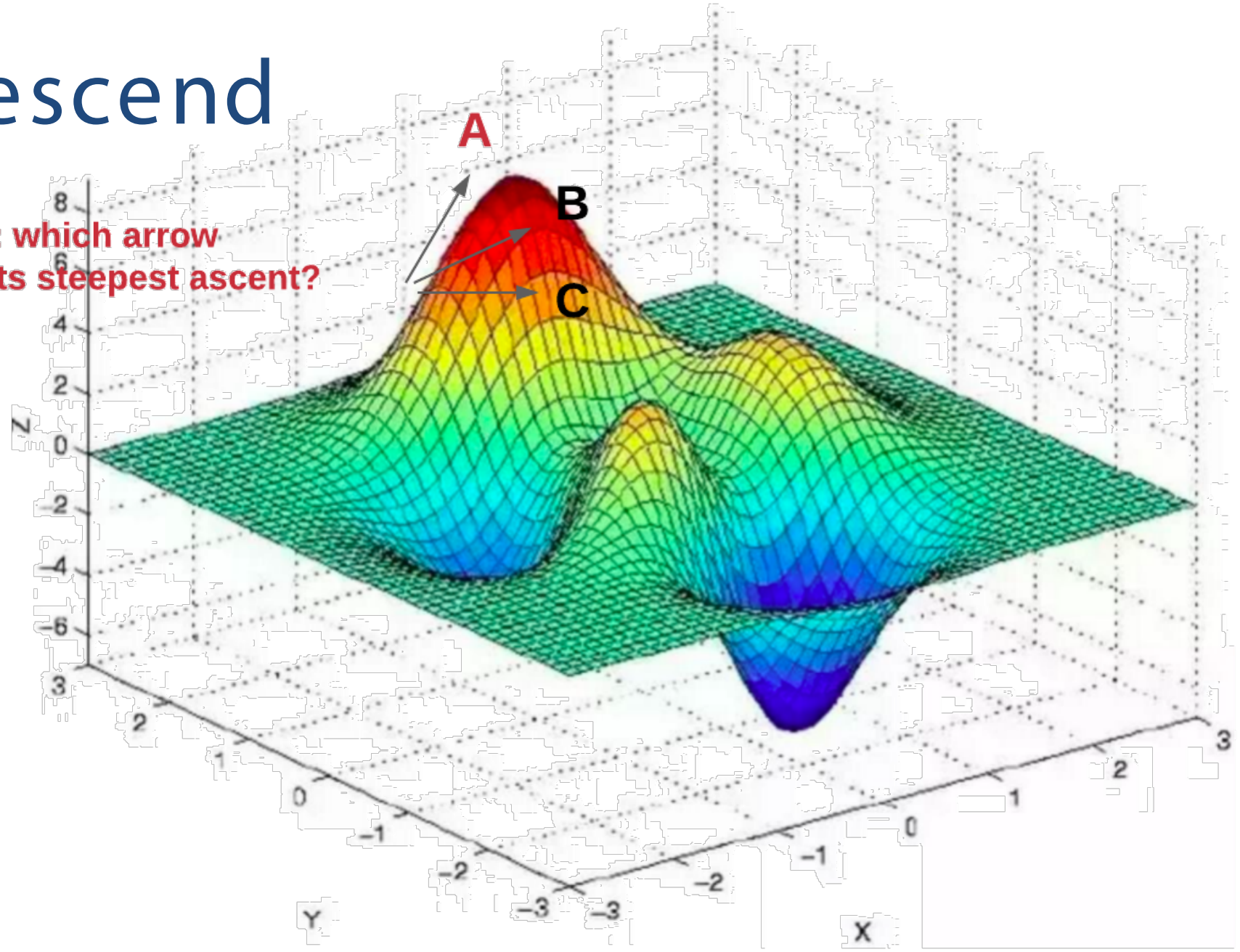
- Example: Gradient Descent-based algorithms (Logistic Regression, Linear Regression)
- Each worker computes gradients for its data partition.
- Gradients are aggregated centrally (using tree aggregation to reduce network overhead).
- The central node updates the model parameters and redistributes them.

Gradient Descend

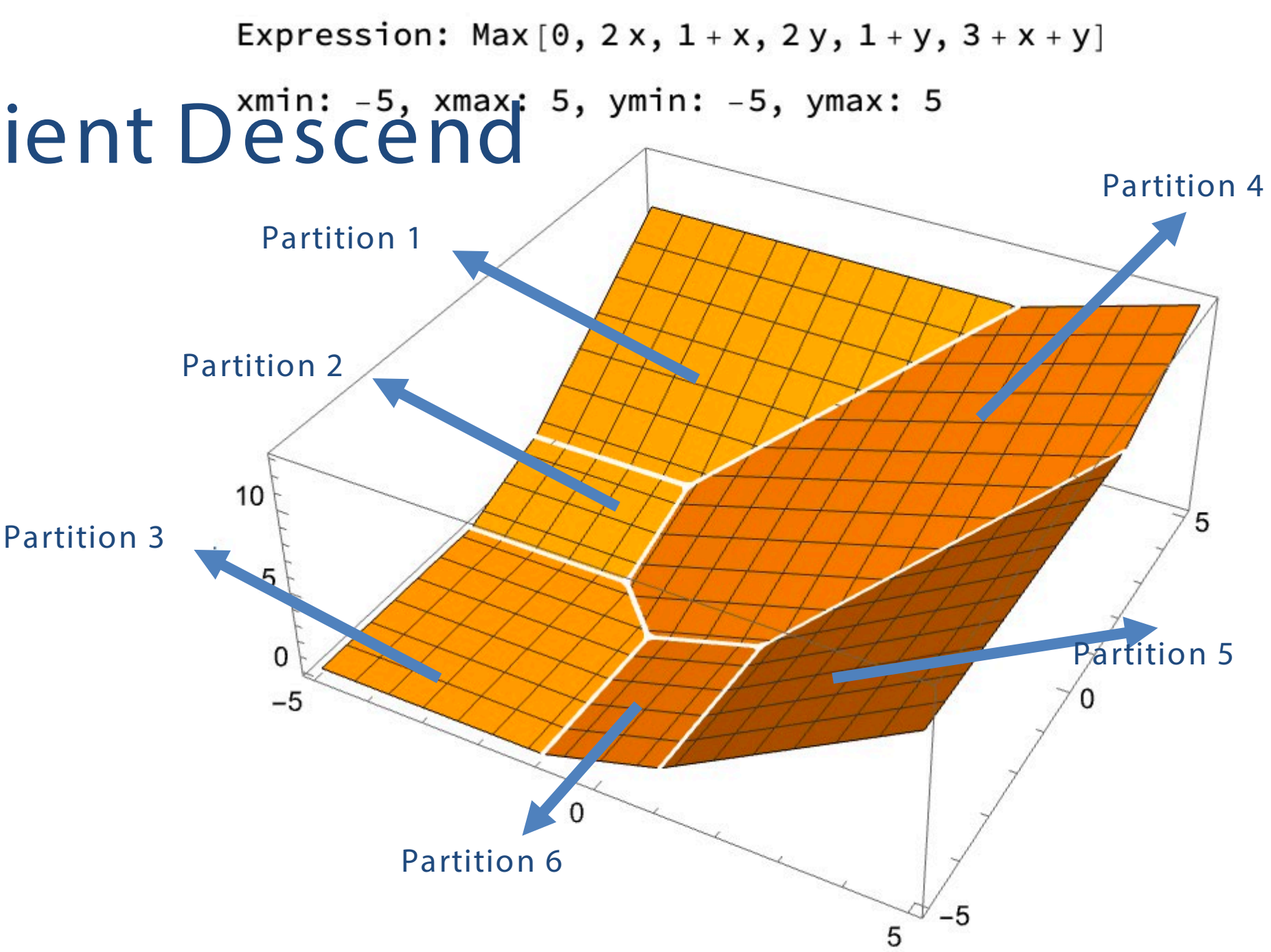
- Gradient Descent is an optimization algorithm used to find the minimum of a function by iteratively adjusting parameters in the direction of the steepest descent (negative gradient). It is commonly used in machine learning and deep learning to minimize loss functions in models like linear regression, logistic regression, and neural networks.

Gradient Descend

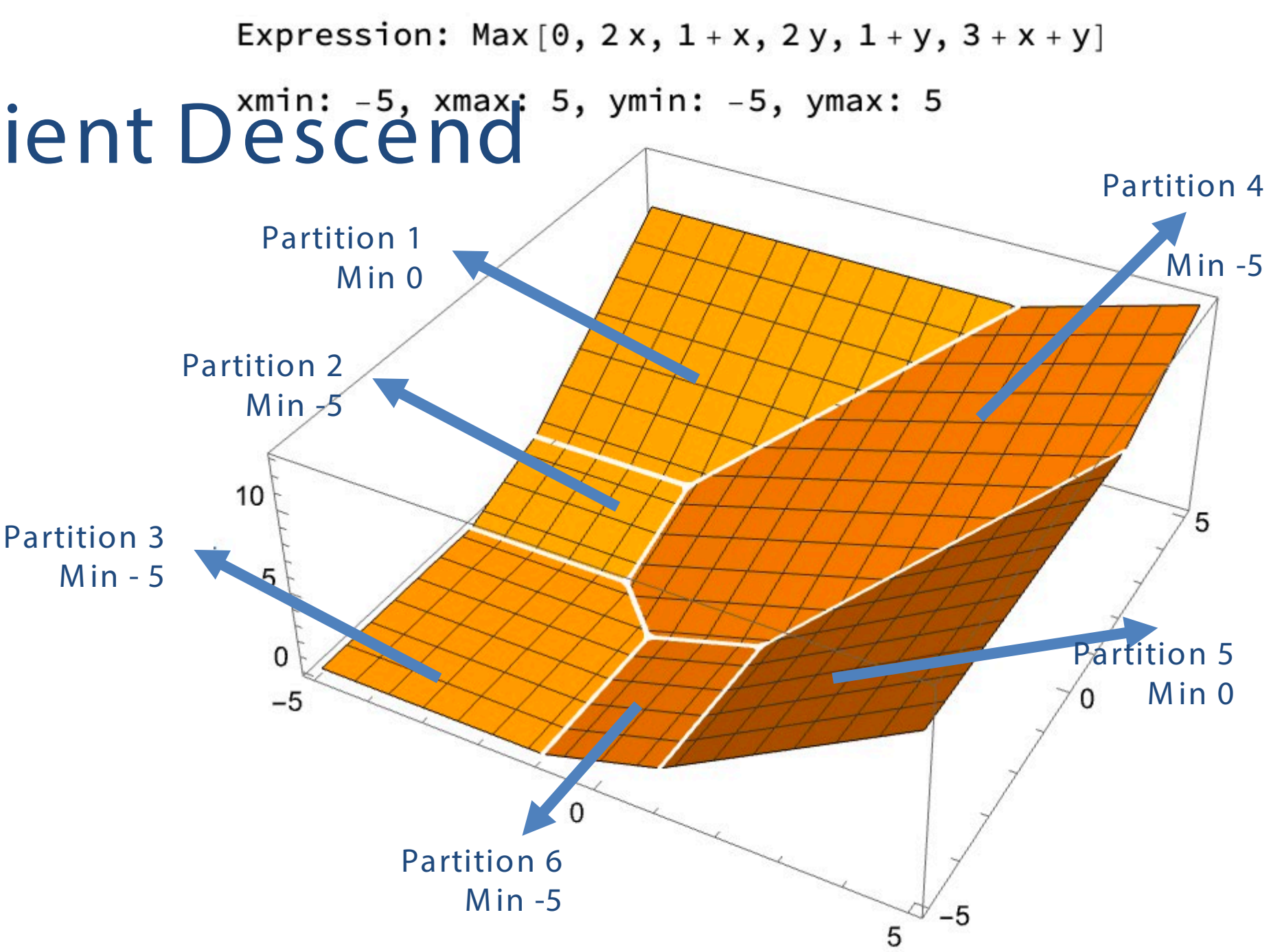
Gradient: which arrow represents steepest ascent?



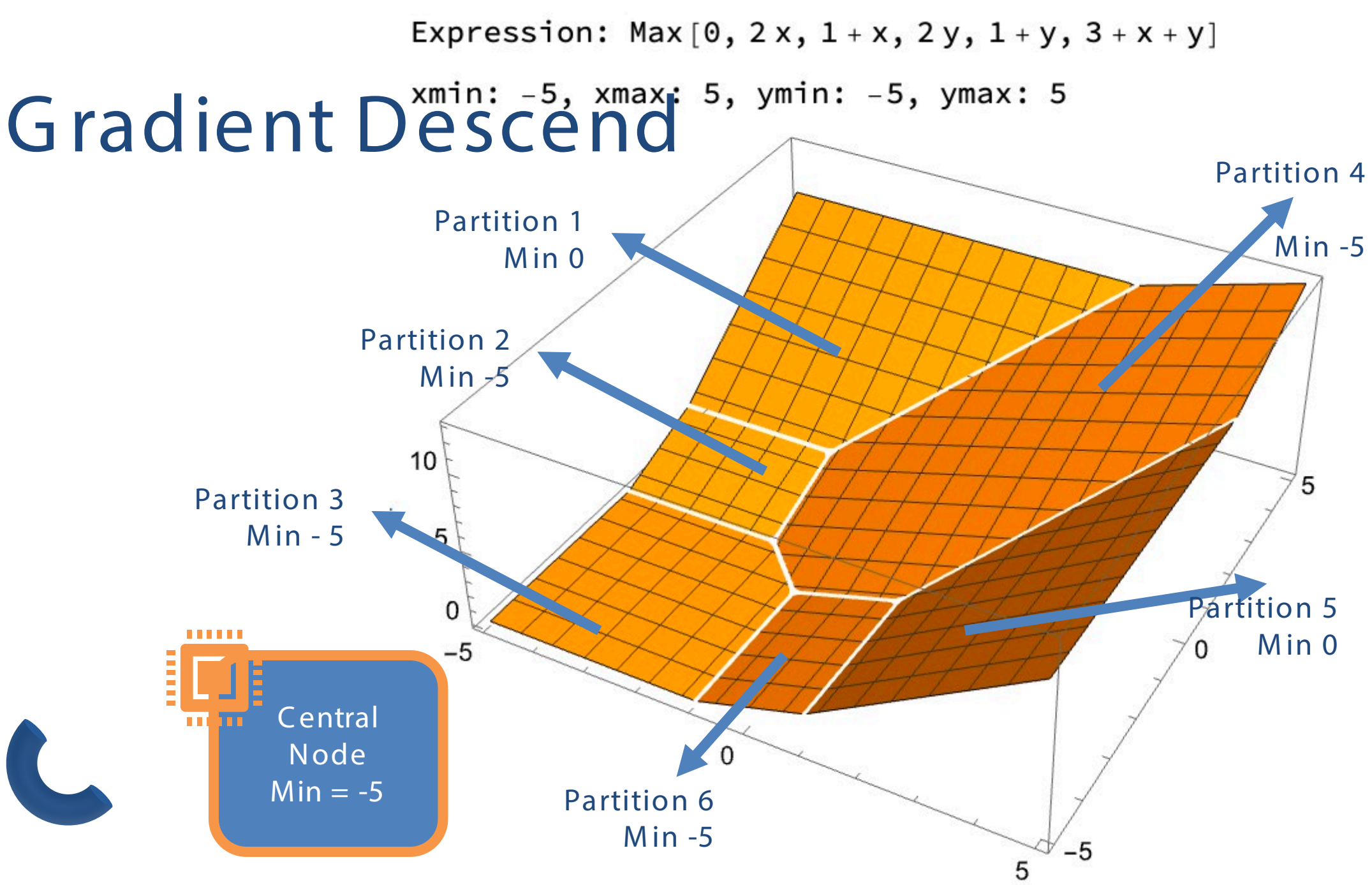
Gradient Descend



Gradient Descend

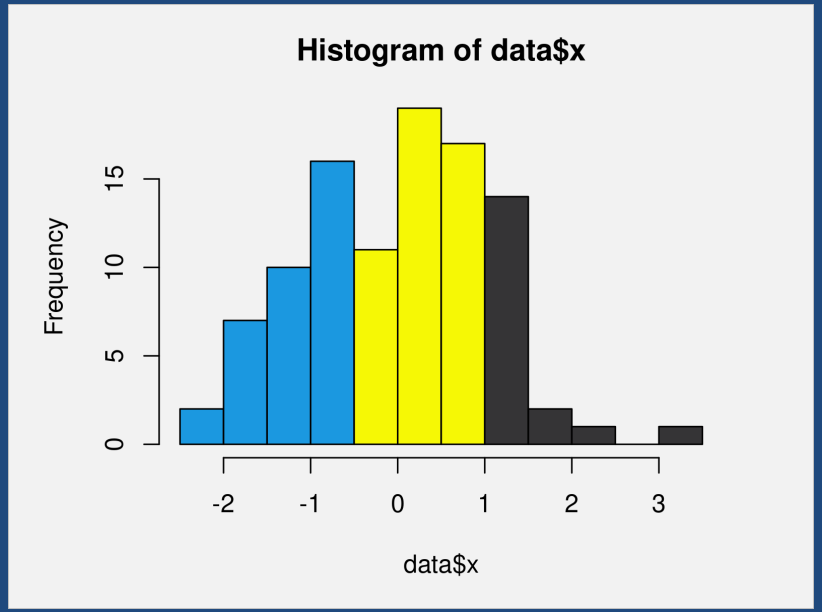


Gradient Descend



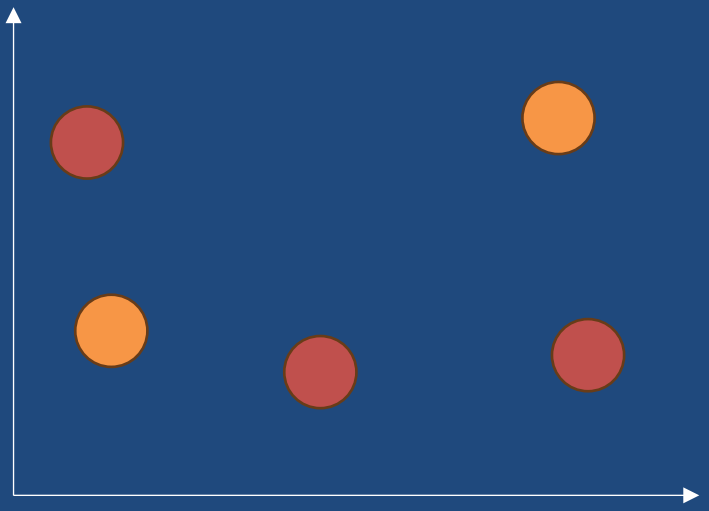
Decision Trees / Random Forest

- Random Forests, in their standard form, do not inherently rely on histograms for distribution. However, in distributed implementations, such as those in Apache Spark MLlib or XGBoost, histograms can play a crucial role in optimizing performance.



- The dataset is distributed across nodes.
- Each node computes histograms of feature values in parallel.
- Histograms are merged to determine the best split.
- Binning reduces the number of split candidates, making it easier to handle large datasets.

Clustering K-Means

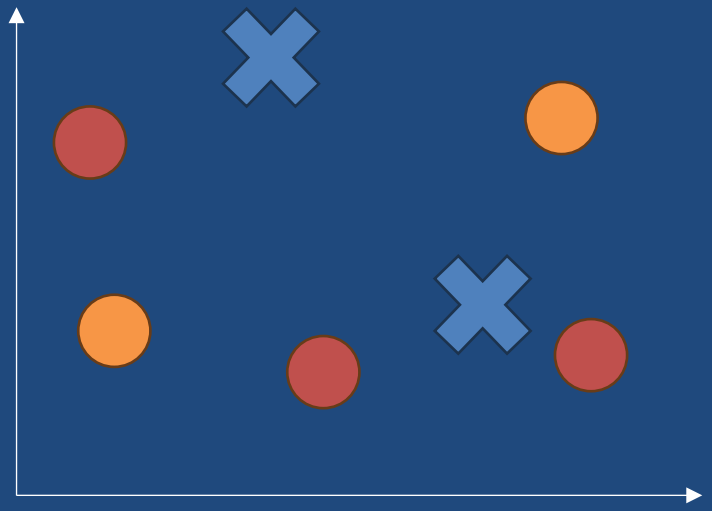


- In **K-Means**, initial centroids are broadcast to all nodes.
- Each node assigns points to the nearest centroid (in parallel).
- Partial cluster statistics (e.g., sum of points, count) are computed per partition.
- These partial results are aggregated using **reduce operations**.
- New centroids are computed and broadcast again for the next iteration.

Node Red

Node Orange

Clustering K-Means

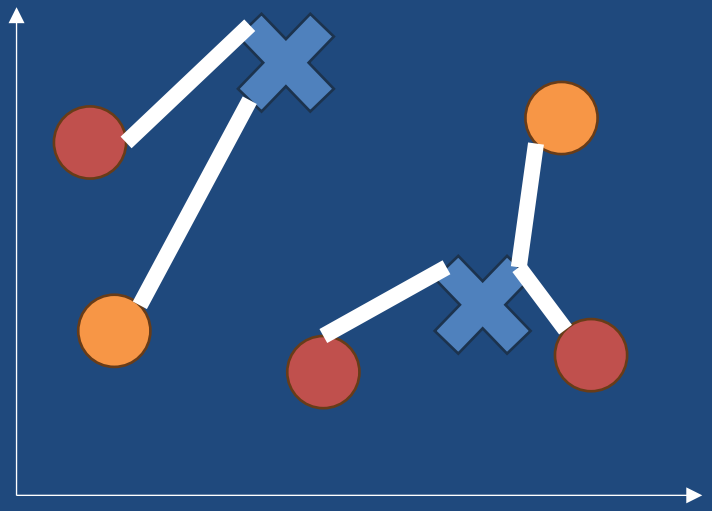


- In **K-Means**, initial centroids are broadcast to all nodes.
- Each node assigns points to the nearest centroid (in parallel).
- Partial cluster statistics (e.g., sum of points, count) are computed per partition.
- These partial results are aggregated using **reduce operations**.
- New centroids are computed and broadcast again for the next iteration.

Node Red

Node Orange

Clustering K-Means



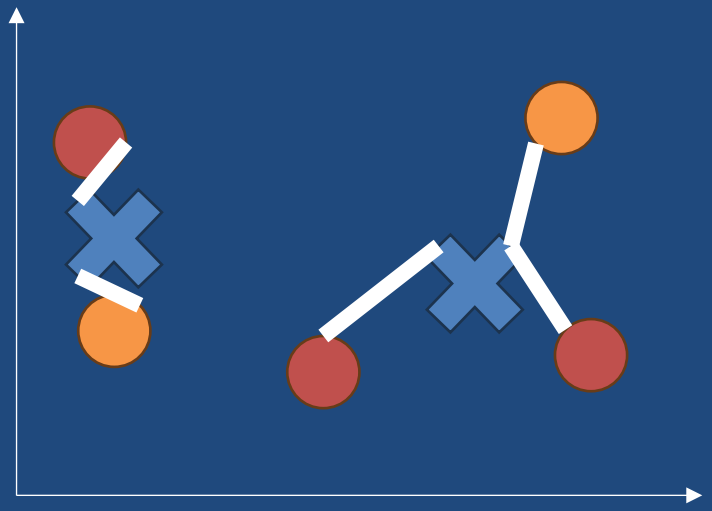
- In **K-Means**, initial centroids are broadcast to all nodes.
- Each node assigns points to the nearest centroid (in parallel).
- Partial cluster statistics (e.g., sum of points, count) are computed per partition.
- These partial results are aggregated using **reduce operations**.
- New centroids are computed and broadcast again for the next iteration.

Node Red

Node Orange

Clustering K-Means

Central Node



- In **K-Means**, initial centroids are broadcast to all nodes.
- Each node assigns points to the nearest centroid (in parallel).
- Partial cluster statistics (e.g., sum of points, count) are computed per partition.
- These partial results are aggregated using **reduce operations**.
- New centroids are computed and broadcast again for the next iteration.

Node Red

Node Orange

Next Lectures…

- We will continue coding in MLLib
- We will talk about Machine Learning Pipelines
- Then we will talk about Dataframes and RDDs in depth
- Some part of the later lectures will support the project
- Visualization, preprocessing and other topics
- Don’t forget to Form Groups!!!