Scalable Data Processing :
By applying MapReduce in Spark

#Big-data #MapReduce #ML

Dayoung Kang
Jaehyeon Kim
Subin Seo

What is Big-data ?

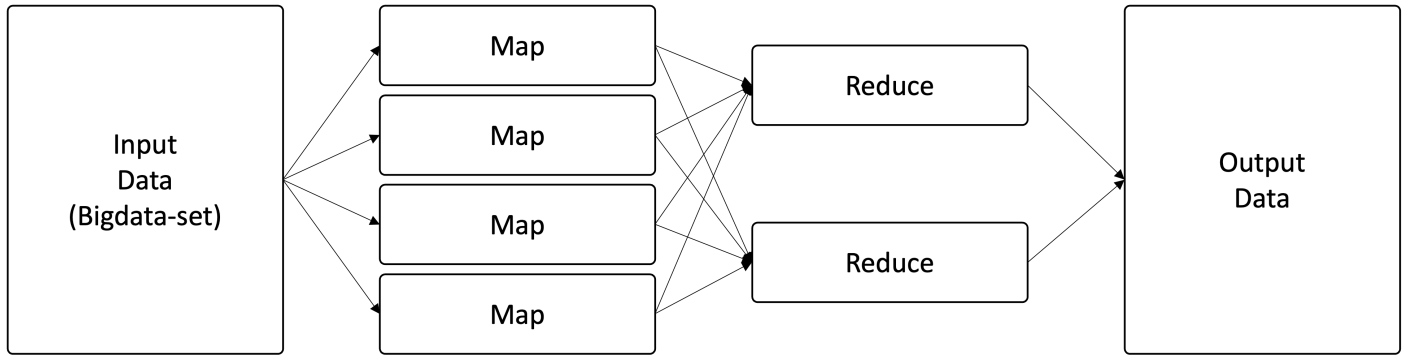
: Any data set contains large volumes of information and complex data is called Big Data (BD). It has 4 characteristics. (4V's) [1]

- **Volume** which is the quantity of data.
- **Velocity** is the speed of the data that during handling and generating.
- **Variety** refers to the range of data types and sources.
- **Veracity** is related to the truth of data which is important for precision in analysis.
- **+ Value** is the importance of the data importance and this is a very significant feature in BD6.

➡ **BD is unlike traditional data, so it requires special processing to manage it.**

How to process Big-data ?

: apply for MapReduce to process Big Data in parallel on multiple node.



Step1. Map

- **Split** input data to number of slices
- Apply specific function to each to generate intermediate results

Step2. Reduce

- **Combine** the intermediate results to make the final result.

About Data

: Customer information to predict who possible Defaulters are for Loans Product

```
train .head(10)
```

	Id	Income	Age	Experience	Married/Single	House_Ownership	Car_Ownership	Profession	CITY	STATE	CURRENT_JOB_YRS	CURRENT_HOUSE_YRS	Risk_Flag
0	1	1303834	23	3	single	rented	no	Mechanical_engineer	Rewa	Madhya_Pradesh	3	13	0
1	2	7574516	40	10	single	rented	no	Software_Developer	Parbhani	Maharashtra	9	13	0
2	3	3991815	66	4	married	rented	no	Technical_writer	Alappuzha	Kerala	4	10	0
3	4	6256451	41	2	single	rented	yes	Software_Developer	Bhubaneswar	Odisha	2	12	1
4	5	5768871	47	11	single	rented	no	Civil_servant	Tiruchirappalli[10]	Tamil_Nadu	3	14	1
5	6	6915937	64	0	single	rented	no	Civil_servant	Jalgaon	Maharashtra	0	12	0
6	7	3954973	58	14	married	rented	no	Librarian	Tiruppur	Tamil_Nadu	8	12	0
7	8	1706172	33	2	single	rented	no	Economist	Jamnagar	Gujarat	2	14	0
8	9	7566849	24	17	single	rented	yes	Flight_attendant	Kota[6]	Rajasthan	11	11	0
9	10	8964846	23	12	single	rented	no	Architect	Karimnagar	Telangana	5	13	0

Independent variables (X)

Dependent variables (Y)

- Train_Data_shape : (252000, 13)
- It has **252,000 samples and 11 features.**
- Independent variables are used to predict of Risk_Flag which is dependent variables.
- Risk_Flag(Y) is binary clas (0 or 1) .

Preview

We try 2 way to Implement MapReduce !

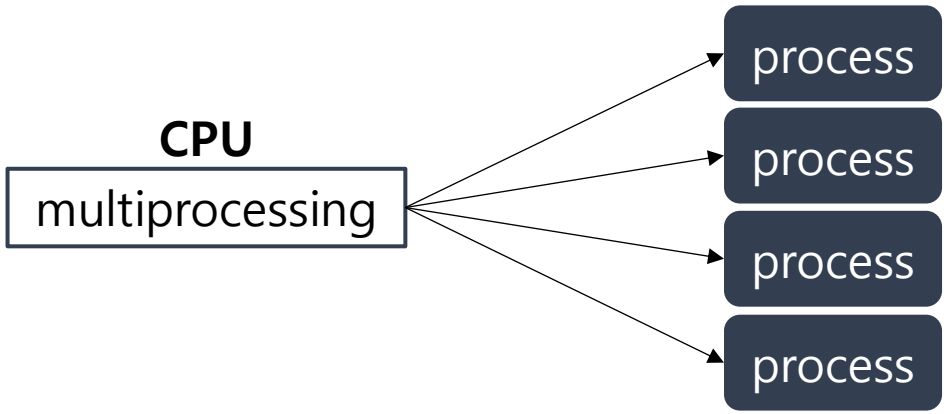
Multiprocessing

VS

PySpark

Using Python Muti processing library

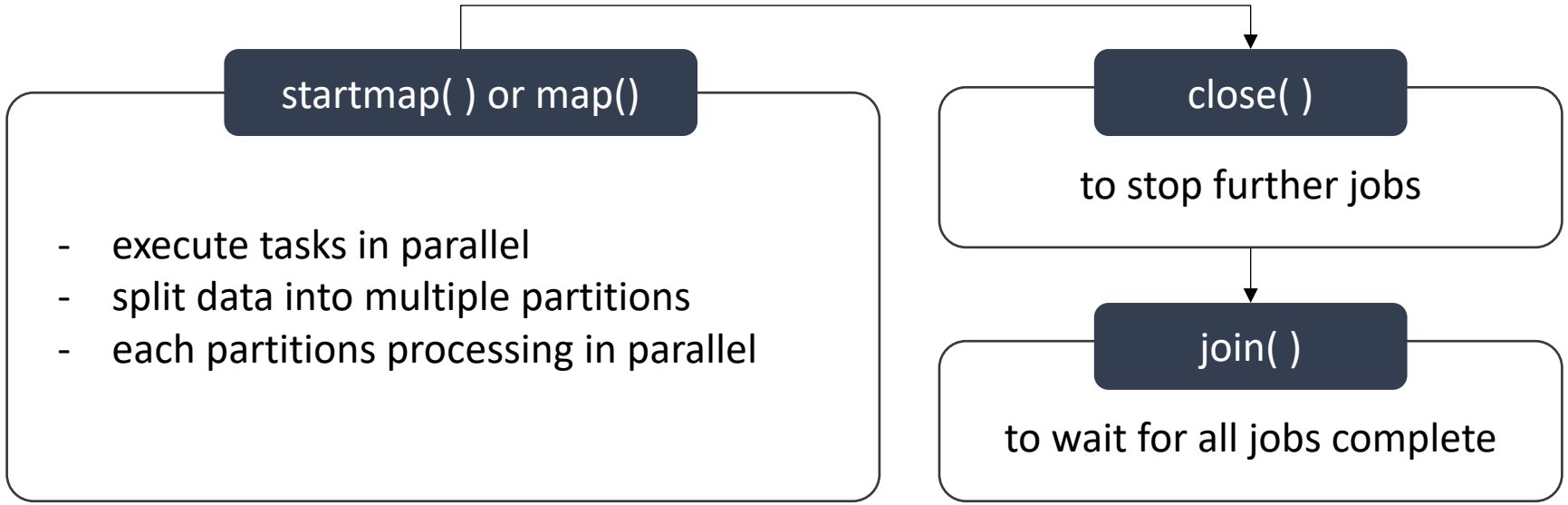
Modules for parallelism in Python: **Multiprocessing**



Pool Object

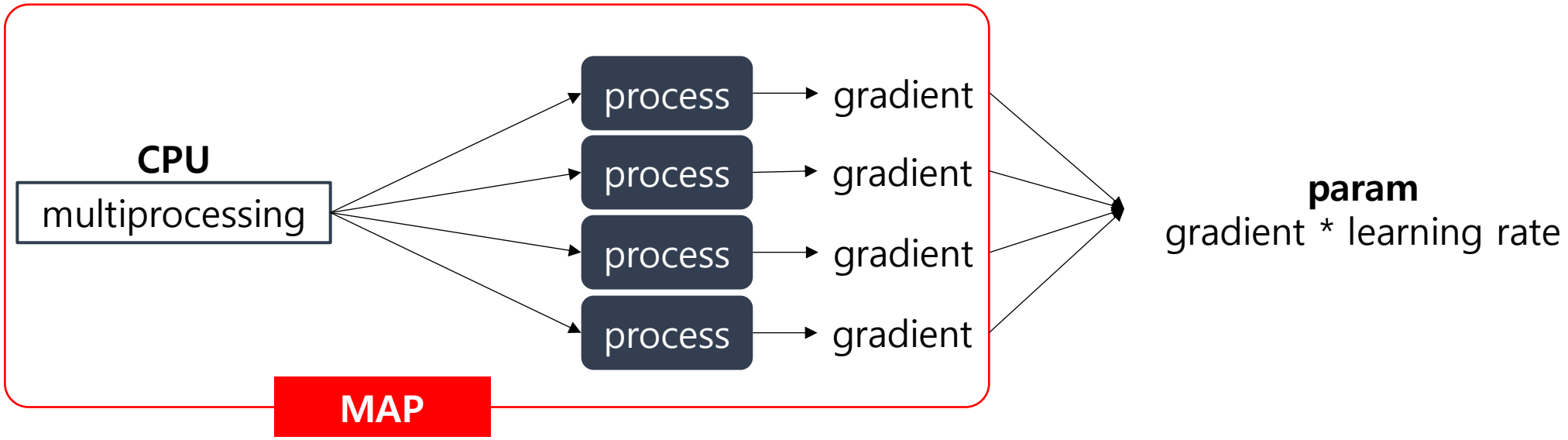
which offers a convenient means of parallelizing the execution of a function across multiple input values, distributing the input data across processes (data parallelism).

Multiprocessing: Pool Object



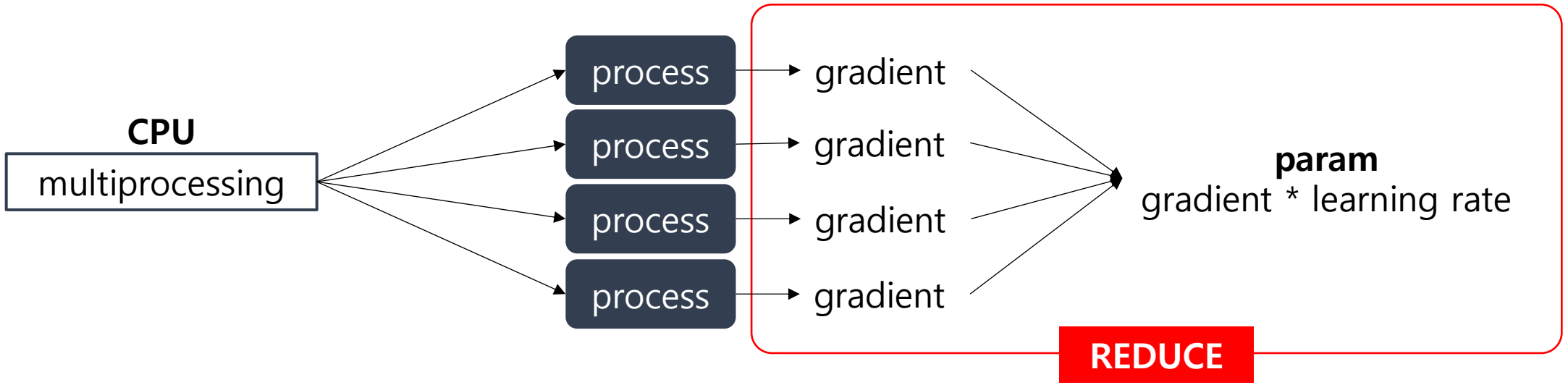
Map

- Goal: Binary Classification with **Linear Regression**
- Calculate gradient by **Ordinary Least Squares (OLS)** for each partition



Reduce

- **Combines** the intermediate results and updates the model parameters
- Sum the **gradients** then multiplied by **learning rate** to return the **updated parameters**



MapReduce with multiprocessing

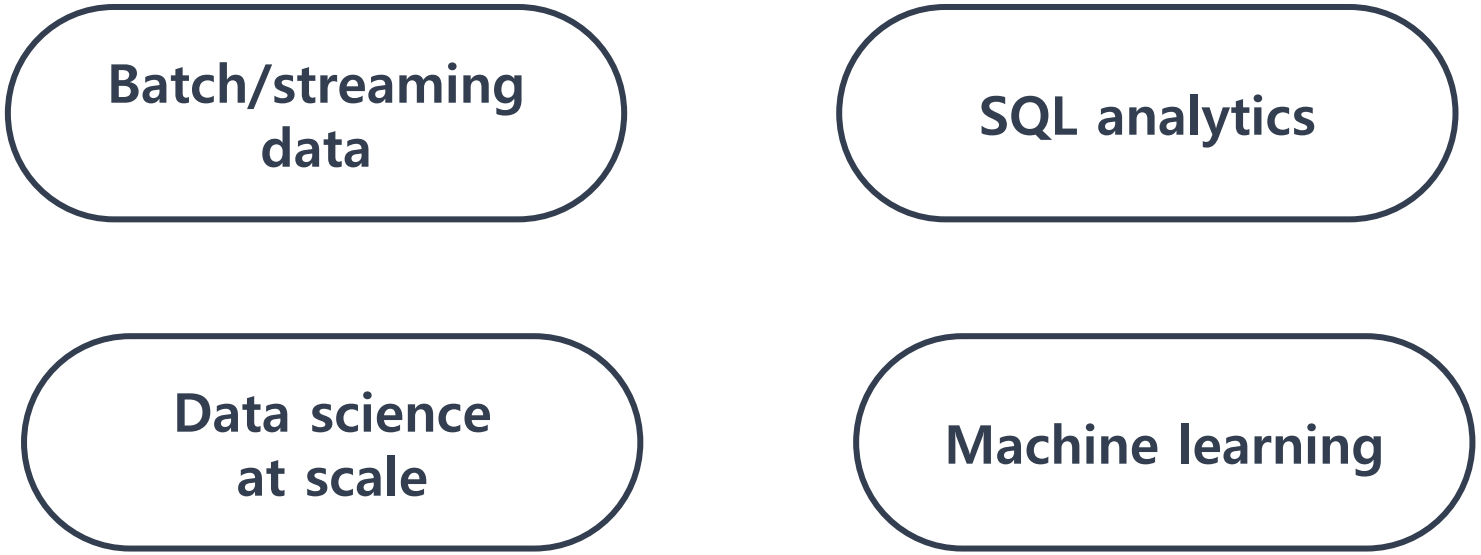
	Pycharm(M1)	JupyterLab(DIONE)
Pool Worker(CPU cores)	8	48
Processing TIme	4min 2.028sec	1min 3.902sec

- Since **multiprocessing** performs parallel processing based on the number of cores in the **CPU**, we were able to get faster results on the **DIONE** server with 48 CPU cores.

Using Pyspark (RDDs)

Spark

Apache Spark™ is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters.

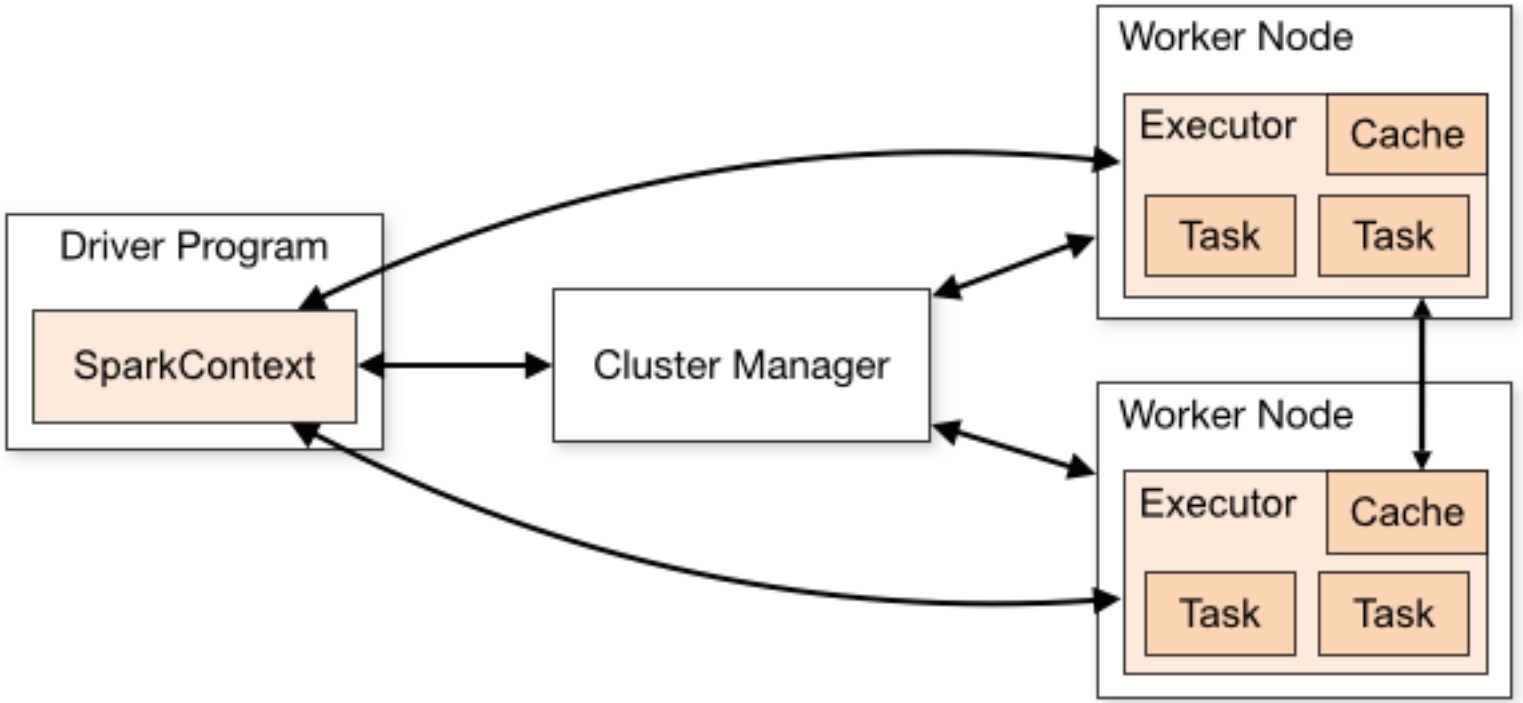


How does **Spark** do **distributed and parallel processing**?

Clustering

RDD & DAG

< Clustering >

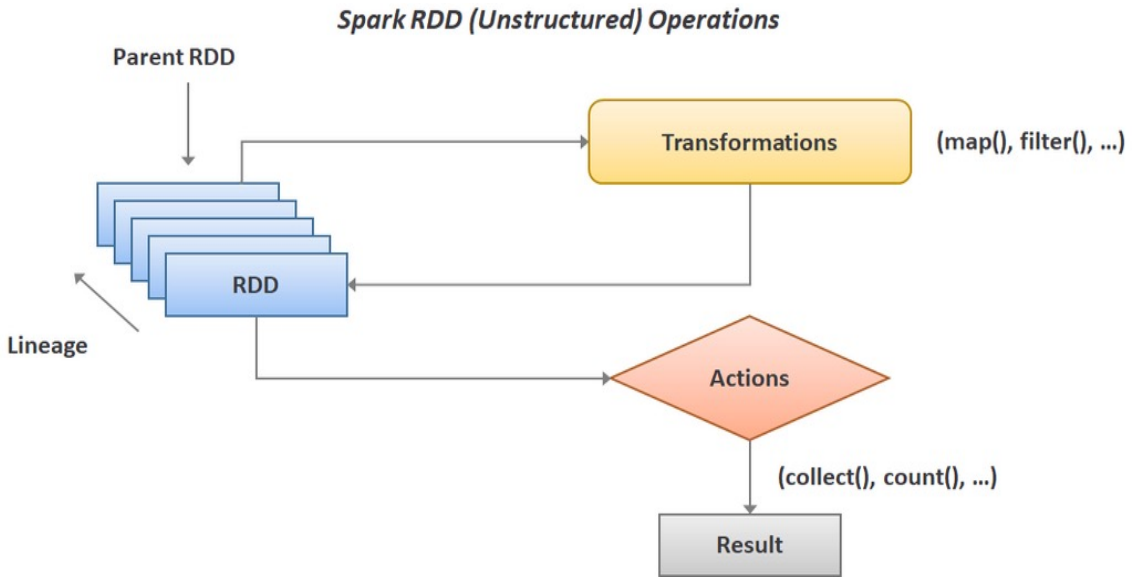


Spark is capable of **in-memory operation** and **distributed processing**, which speeds up.

➡ Because, Cluster Manager allocates Worker nodes to CPUs on its own and performs distributed processing.

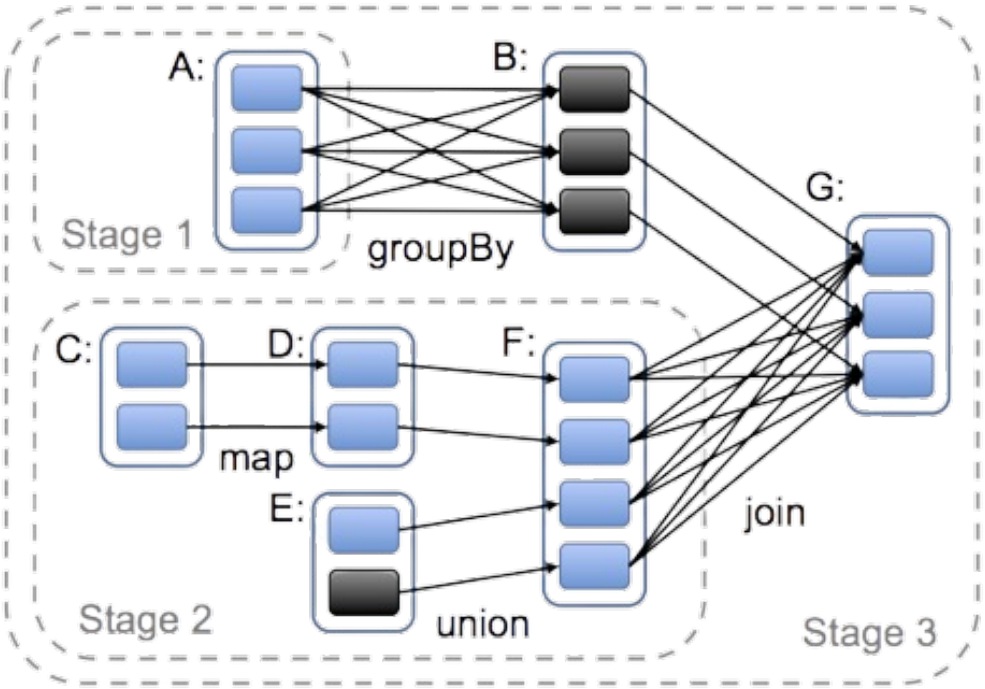
Spark RDD & DAG

< RDD >



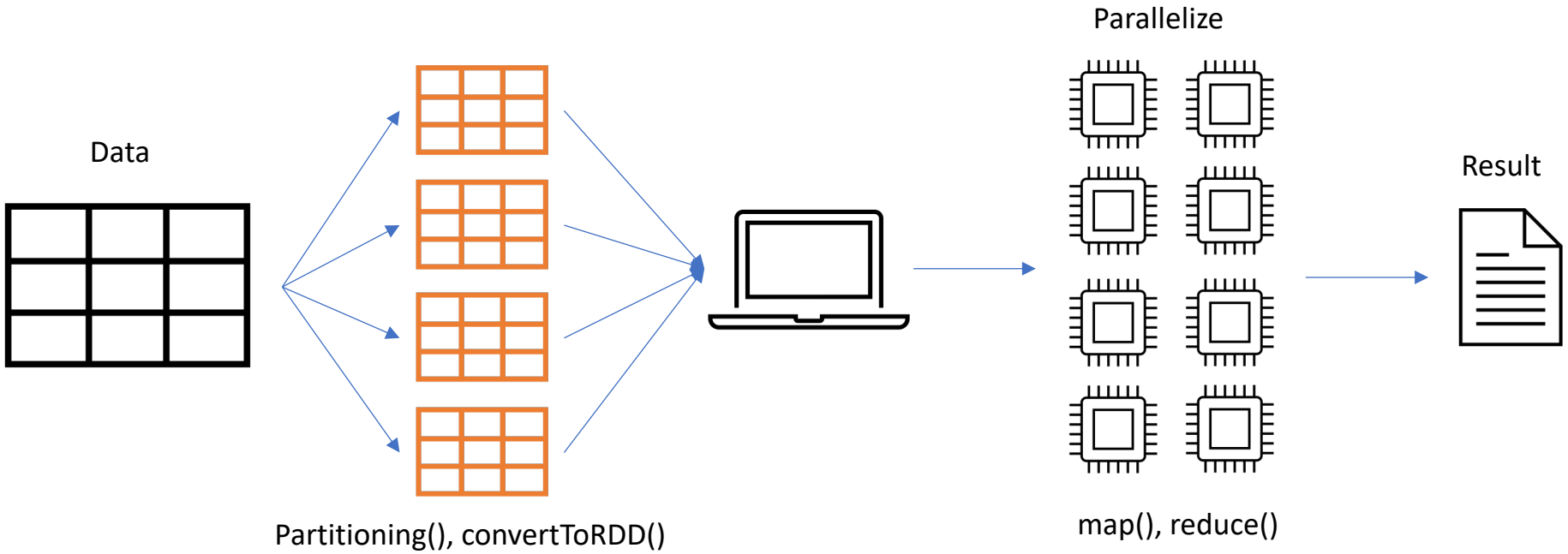
- RDD is Spark's fundamental data abstraction concept, which is a **distributed collection** of data elements divided into multiple partitions.
  ➡ Immutable & Read-only system!

< DAG>



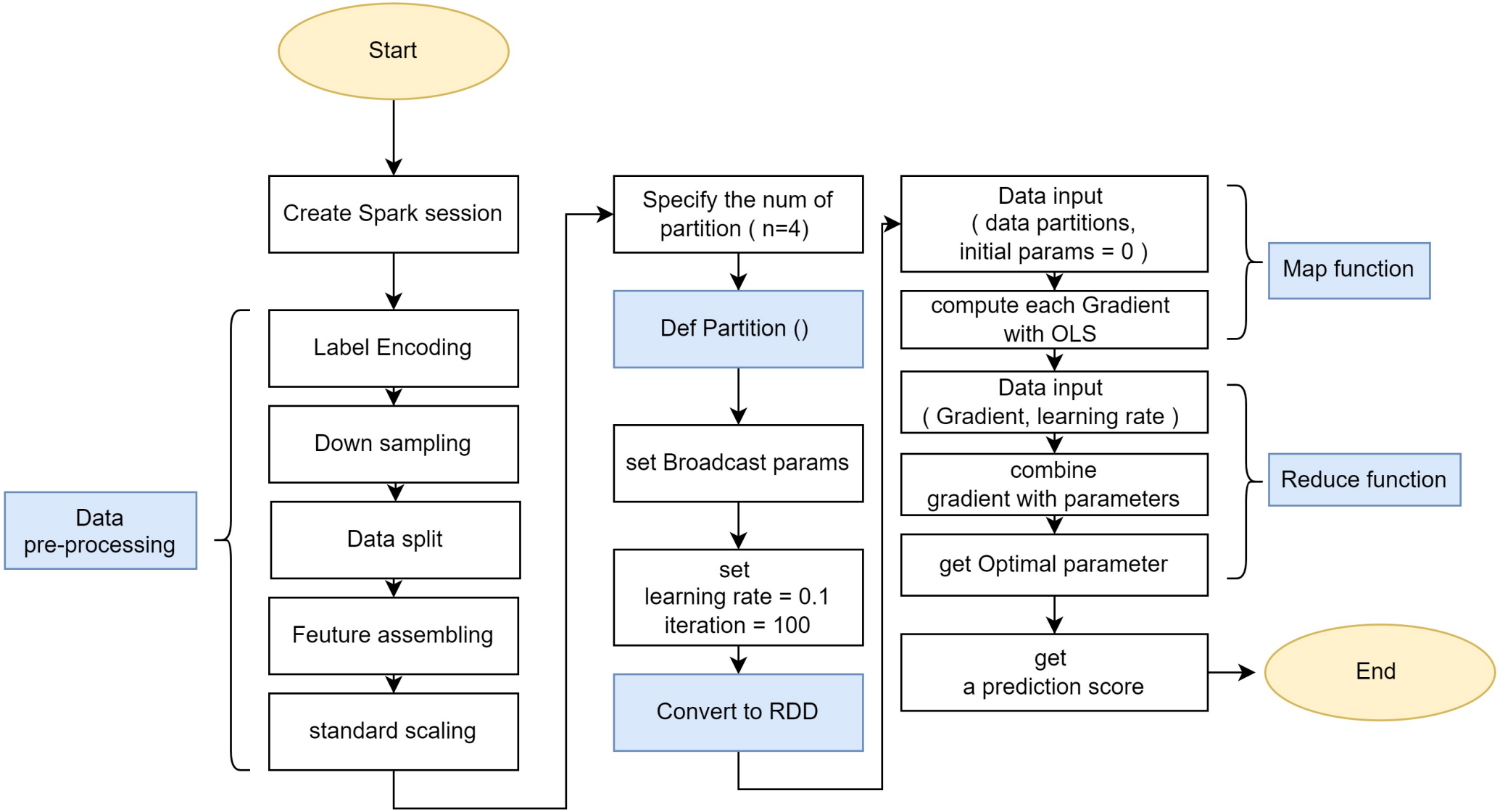
- DAG is a **Directed Acyclic Graph** that expresses dependencies between tasks.
  ➡ All transformation is recorded as a DAG.

Spark Process

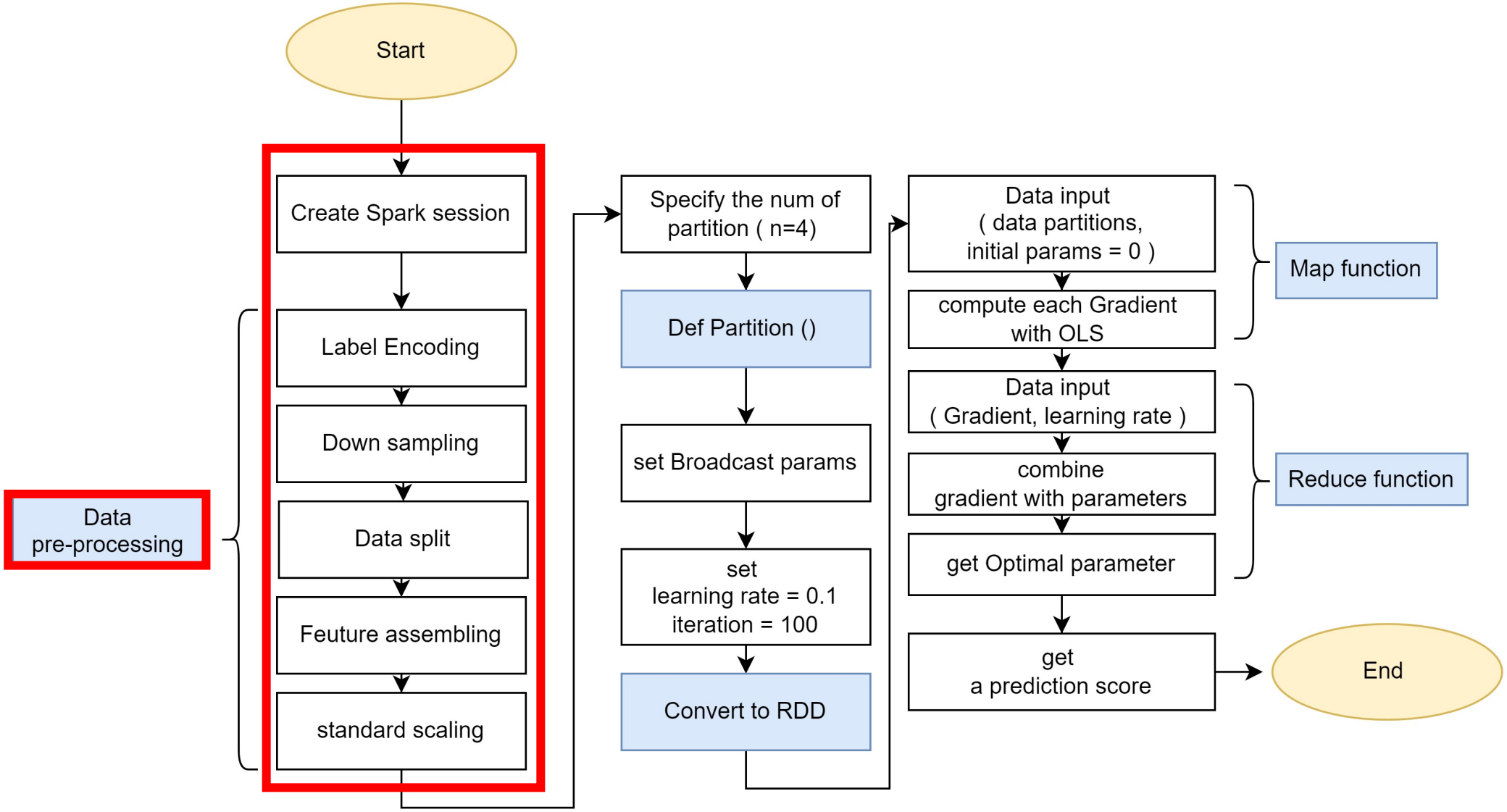


- The data is split into multiple **partitions** (num_partitions = 4).
- **The map function** calculates the gradients for each partition in parallel (worker = 8).
- **The reduce function** combines the gradients and updates the model parameters using a learning rate.
- This process is repeated for a certain number of iterations (num_iterations = 100).

Flowchart


Start
Create Spark session
Label Encoding
Down sampling
Data split
Feuture assembling
standard scaling
Data
pre-processing
Specify the num of
partition (n=4)
Def Partition ()
set Broadcast params
set
learning rate = 0.1
iteration = 100
Convert to RDD
Data input
(data partitions,
initial params = 0)
compute each Gradient
with OLS
Map function
Data input
(Gradient, learning rate)
combine
gradient with parameters
get Optimal parameter
Reduce function
get
a prediction score
End

Current location



Data Pre-Processing

Label Encoding

Categorical data to number

features
- "married_single”
- "profession”
- "house_ownership”
- "car_ownership”
- "city”
- "state”

Down Sampling

Class0
- 221,004 samples

Class1
- 30,996 samples

Each class 30,996 samples
Total 61,992 samples

Data Split

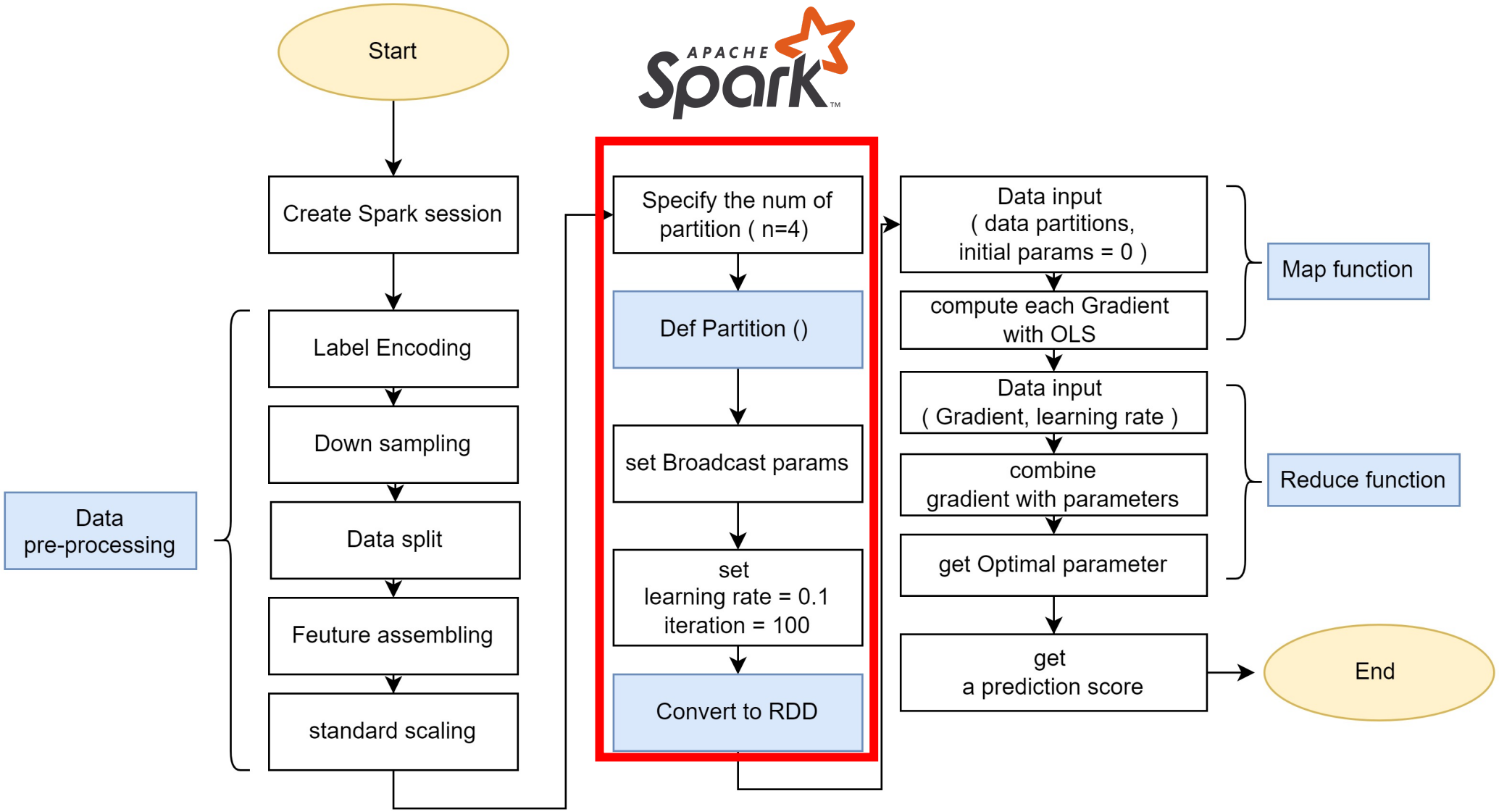
One Dataset file
- **Train** 64%
- **Validation** 16%
- **Test** 20%

Standard Scaling

Calibrate scale differences

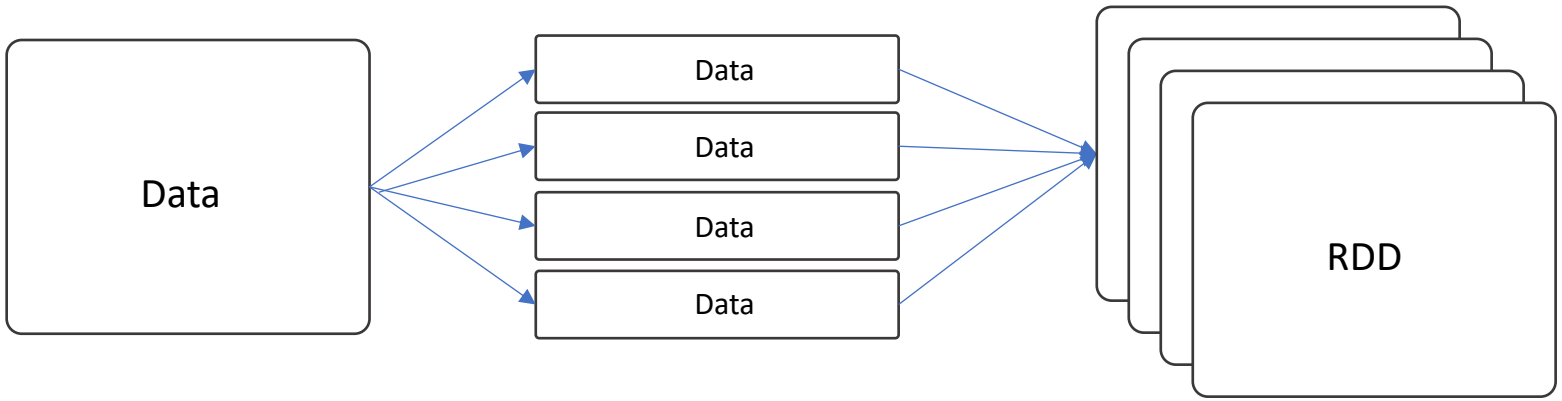
Features ex.
- Income(10,000,000)
- Age(14)

Current location



Function codes

Data Partitioning & Covert to RDD



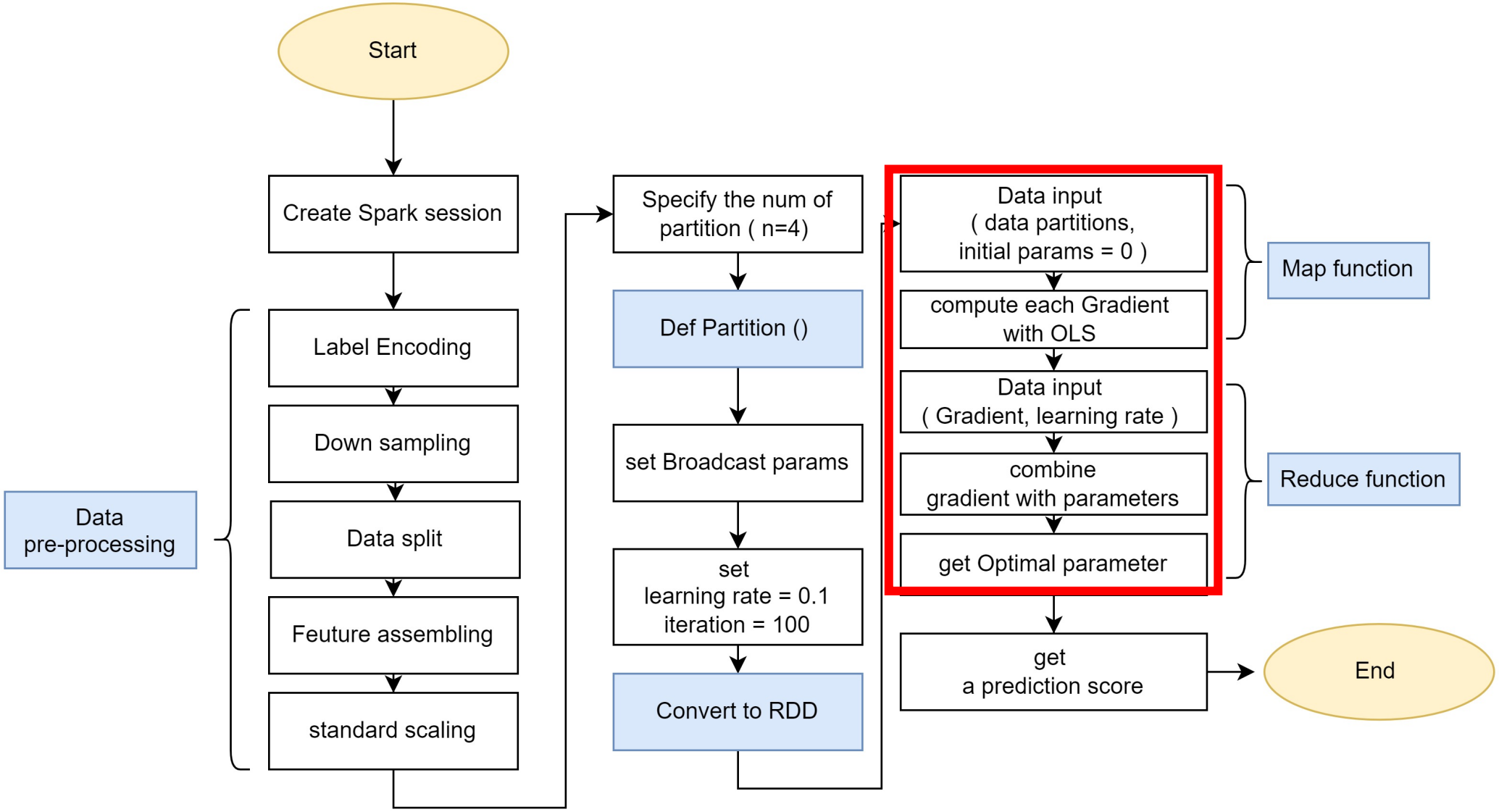

```
def split_data_into_partitions(X, y, num_partitions):
    data_partitions = []
    chunk_size = len(X) // num_partitions

    for i in range(num_partitions):
        start_idx = i * chunk_size
        end_idx = (i + 1) * chunk_size
        X_partition = X[start_idx:end_idx]
        y_partition = y[start_idx:end_idx]
        data_partitions.append((X_partition, y_partition))

    return data_partitions
```

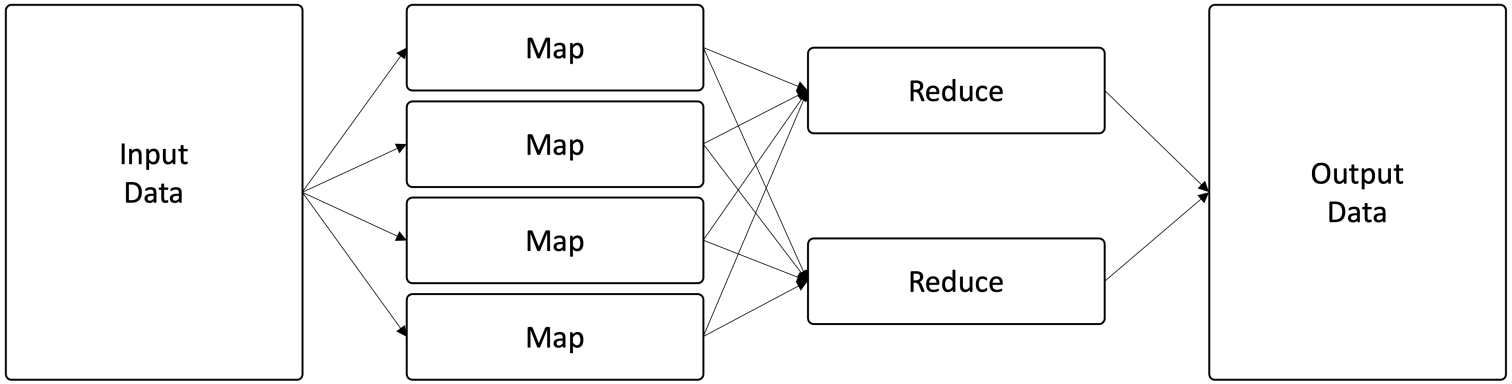
```
Rdd = spark.sparkContext.parallelize(data_partitions)
```

Current location



Function codes

MapReduce
functional programming



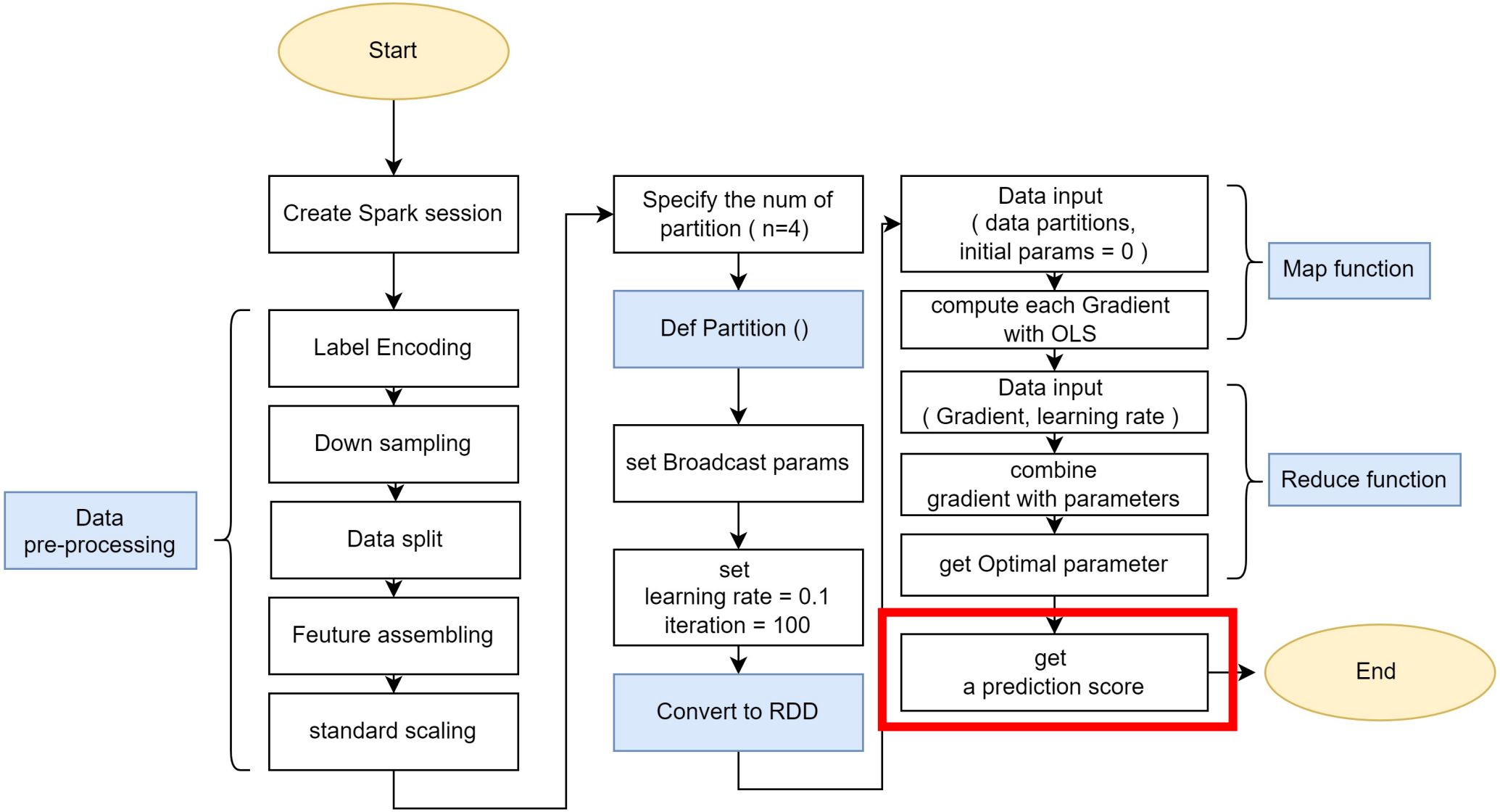

```
def map_function(data_partition, params):
    X, y = data_partition
    gradients = np.dot(X.T, np.dot(X, params) - y)

    return gradients
```

```
def reduce_function(intermediate_results, learning_rate):
    total_gradients = np.sum(intermediate_results, axis=0)
    updated_params = learning_rate * total_gradients

    return updated_params
```

Current location



Spark Result

Spark	
Elapsed Time	0.0 min 41.307 sec
Val-ACC	50.193
Test-ACC	50.191

- Scalable data can be distributed and processed in parallel through spark.
- And this method is effective for processing big data.

Discussion

- In this project,
  Our data set can be storage and processed with in a local storage, which has about 60,000 samples.

=> Therefore, we are going to test the model of this paper with a lot bigger data sets.
(Use large amounts of data that are not even stored on the local storage)

Reference

[1] Hiba Basim Alwan and Ku Ruhana Ku-Mahamud 2020 *IOP Conf. Ser.: Mater. Sci. Eng.* **769** 012007
[2] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: cluster computing with working sets. In Proceedings of the 2nd USENIX conference on Hot topics in cloud computing (HotCloud'10). USENIX Association, USA, 10.
[3] Data Reference : https://www.kaggle.com/datasets/subhamjain/loan-prediction-based-on-customer-behavior?resource=download&select=Sample+Prediction+Dataset.csv
[4] Image Reference : https://subscription.packtpub.com/book/data/9781785889622/5/ch05lvl1sec39/linear-classification