Week 3 - Introduction to Apache Spark

Why use Apache Spark?

Apache Spark attribute

`Spark is an` **`open-source`** `,` **`in-memory`** `application framework for` **`distributed`** `data processing and` **`iterative`** `analysis on` **`massive`** `data volumes.`

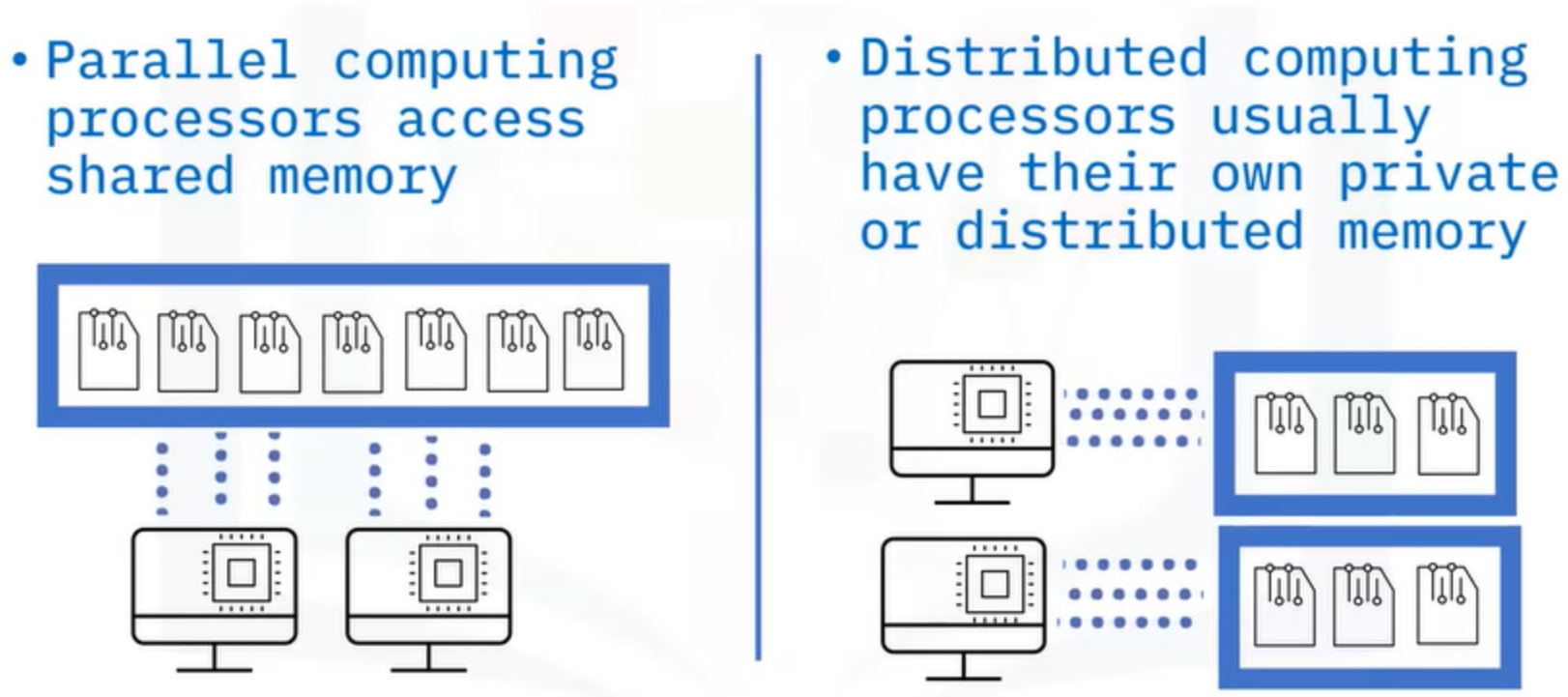
- Entirely open source. Spark is available under the Apache Foundation hence the name Apache Spark.
- “In-memory,” which means that all operations happen within the memory or RAM.
- Distributed data processing uses Spark.
- Spark scales very well with data, which is ideal for massive datasets.
- Spark is primarily written in Scala.
- Spark runs on JVM.

What is Distributed computing?

Distributed computing, as the name suggests, is a group of computers or processors working together behind the scenes.

Paralleling vs Distributed computing

People often use the terms Distributed computing and Parallel Computing interchangeably, as the two computing types have many similarities.



Access memory differently:

- parallel computing shares all the memory
- distributed computing, each processor accesses its own memory.

Distributed computing benefits

- **Offers scalability and modular growth.**
  Distributed systems are inherently scalable as they work across multiple machines and scale horizontally. A user can add additional machines to handle the increasing workload instead of repeatedly updating a single system, with virtually no cap for scalability.
- **Fault tolerance and redundancy**
  Distributed systems require both fault tolerance and redundancy as they use independent nodes that could fail. Distributed computing offers more than fault tolerance. Distributed computing provides redundancy that enables business continuity.

Spark benefits

Supports a computing framework for large-scale data processing and analysis.

Provides parallel distributed data processing capabilities, scalability, and fault-tolerance on commodity hardware.

Provides in-memory processing

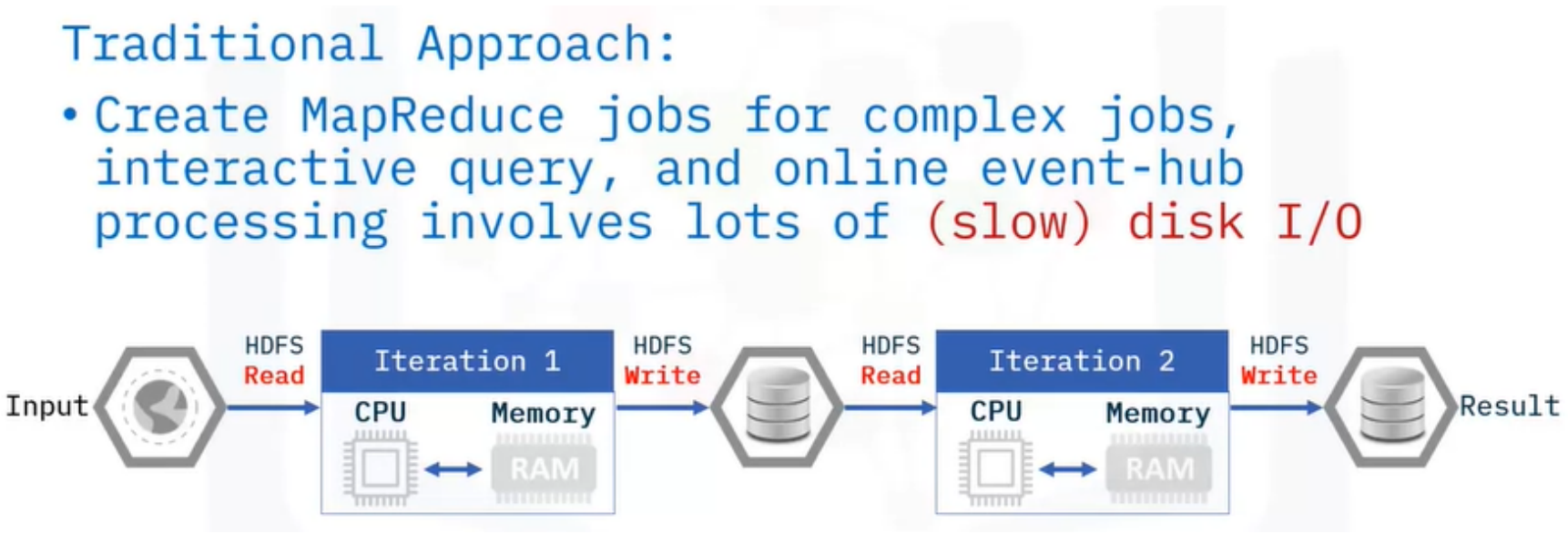
Creates a comprehensive, unified framework to manage big data processing.

Enables programming flexibility with easy-to-use Python, Scala, and Java APIs.

Apache Spark vs MapReduce

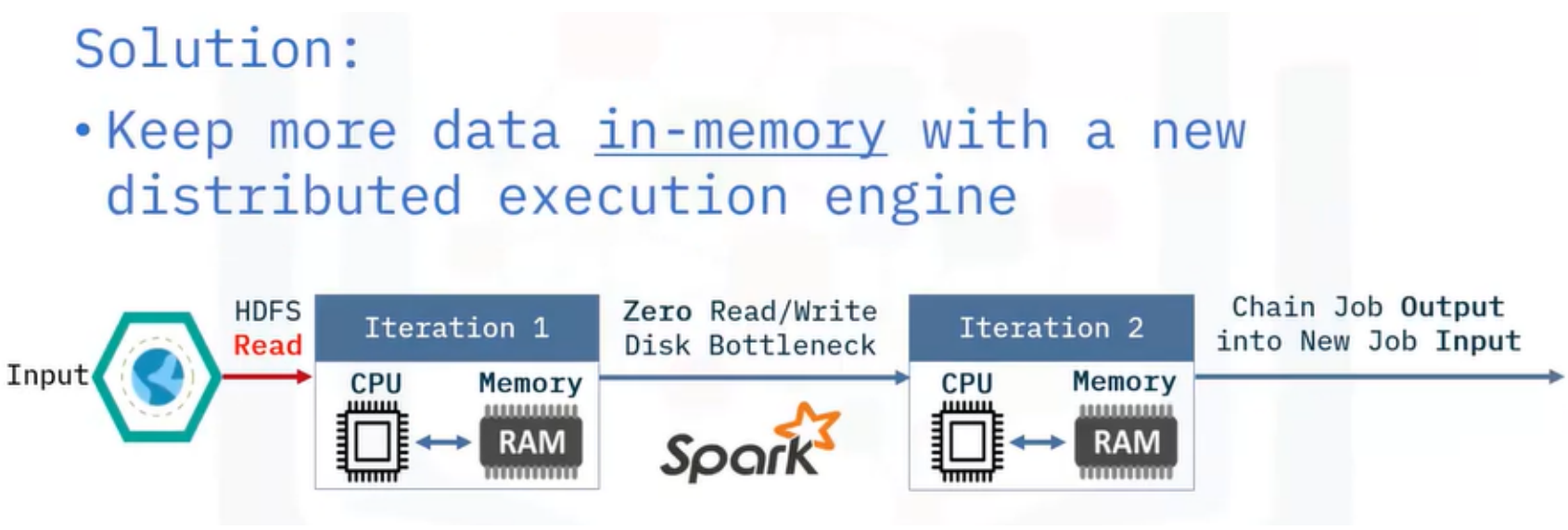
A traditional MapReduce job

- creates iterations that require reads and writes to disk or HDFS. These reads and writes are usually time-consuming and expensive.

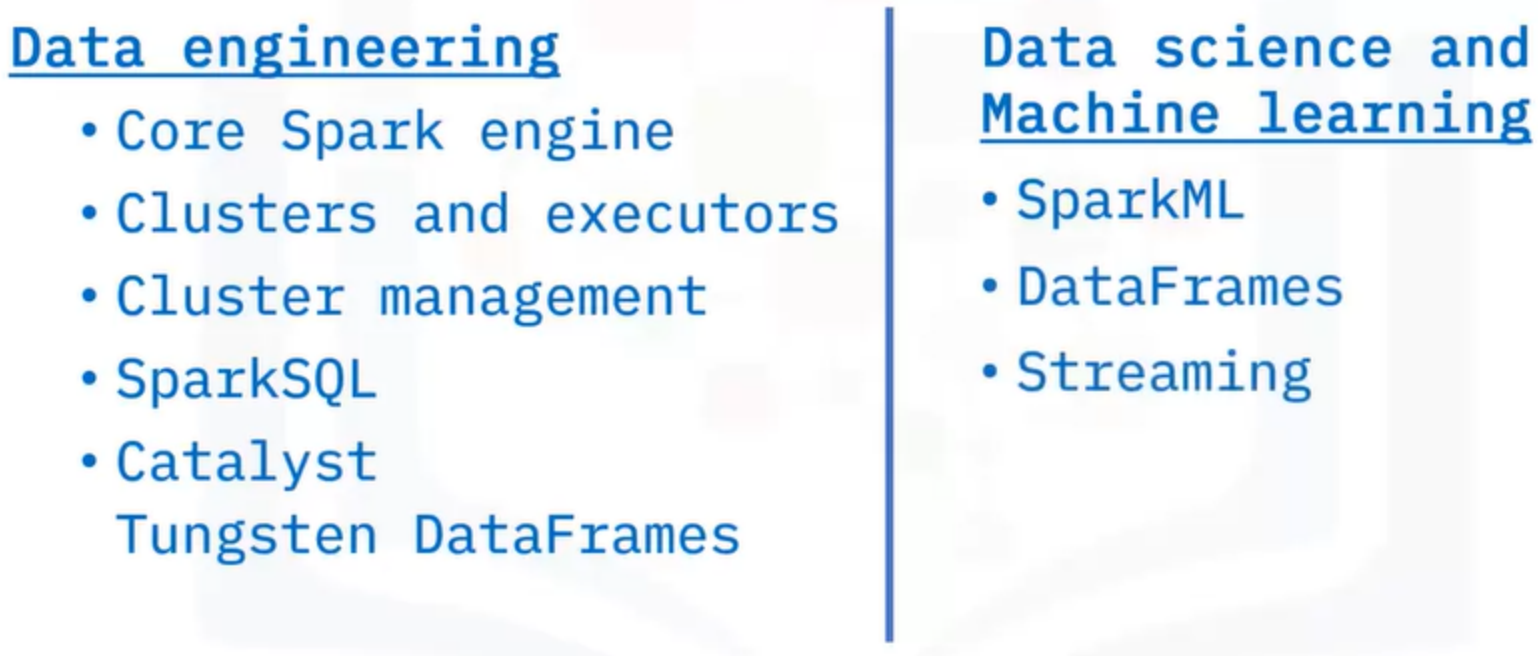


Apache Spark

solves the read/write problems encountered with MapReduce by keeping much of the data required in-memory and avoiding expensive disk I/O, thus reducing the overall time by orders of magnitude.



Spark and Big Data



Data engineers use Spark tools:

- Core Spark engine
- clusters and executors and their management,
- SparkSQL and DataFrames.

Data Science and Machine learning:

- libraries such as SparkML and Streaming.

Conclusion

- Spark is an open-source in-memory application framework for distributed data processing and iterative analysis on massive data volumes.
- Distributed computing is a group of computers or processors working together behind the scenes.
- Both distributed systems and Apache Spark are inherently scalable and fault-tolerant.
- Apache Spark keeps a large portion of the data required in-memory and avoids expensive and time-consuming disk I/O

Functional Programming Basics

What is Functional Programming?

Functional Programming, or FP, is a style of programming that follows the mathematical function format.

Think of an algebra class with the f of x notation. The notation is declarative in nature as opposed to being imperative.

By declarative, we mean that the emphasis of the code or program is on the “what” of the solution as opposed to the “how to” of the solution.

Declarative syntax abstracts out the implementation details and only emphasizes on the final output, restated, “the what.

We use expressions in functional programming, like the expression f of x as mentioned earlier.

Historically, LISt Processing Language, known as LISP, was the first functional programming language, starting in the 1950s.

But today there are many functional programming language options including Scala, Python, R, and Java.

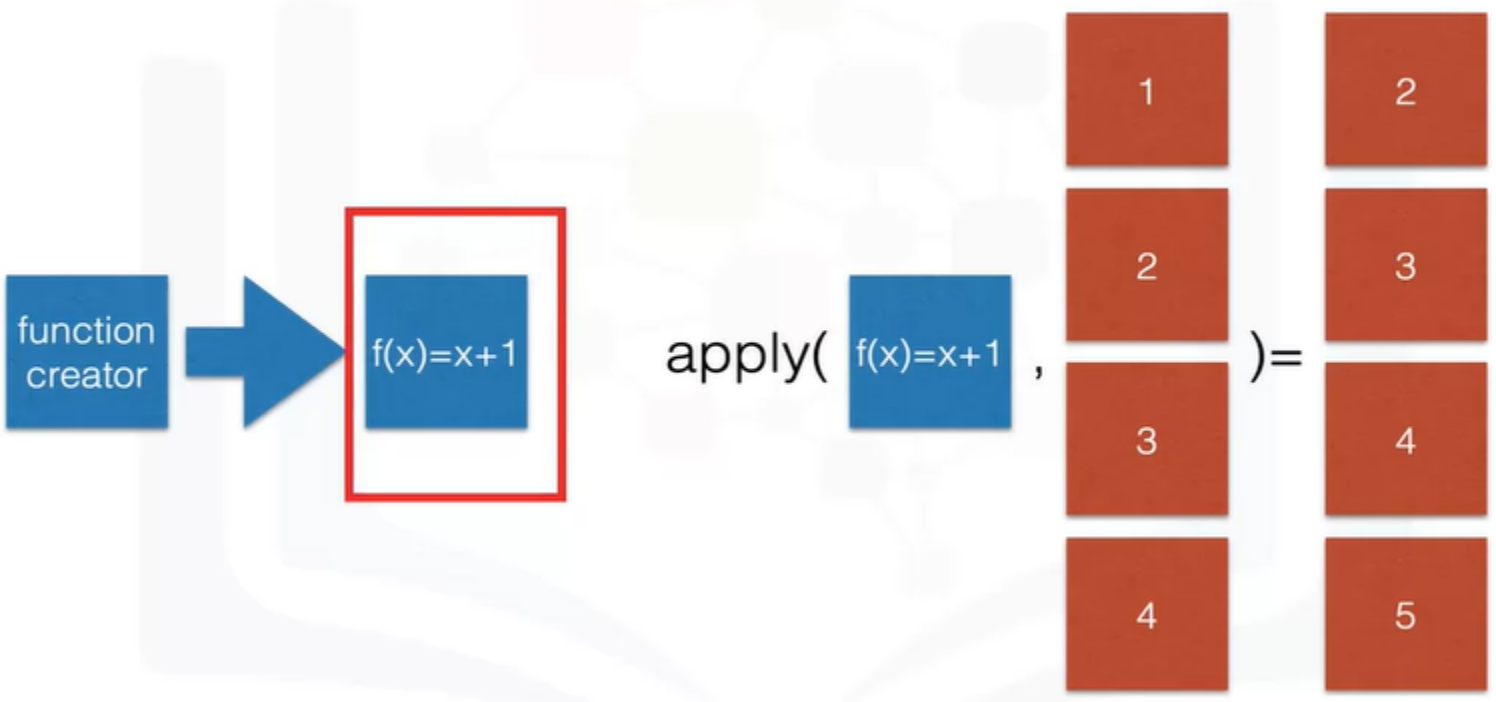
Scala is the most recent representative in this family of programming languages.

Apache Spark is written mainly in Scala, which treats functions as first-class citizens.

Functions in Scala can be passed as arguments to other functions, returned by other functions, and used as variables.

Example

Functional program example

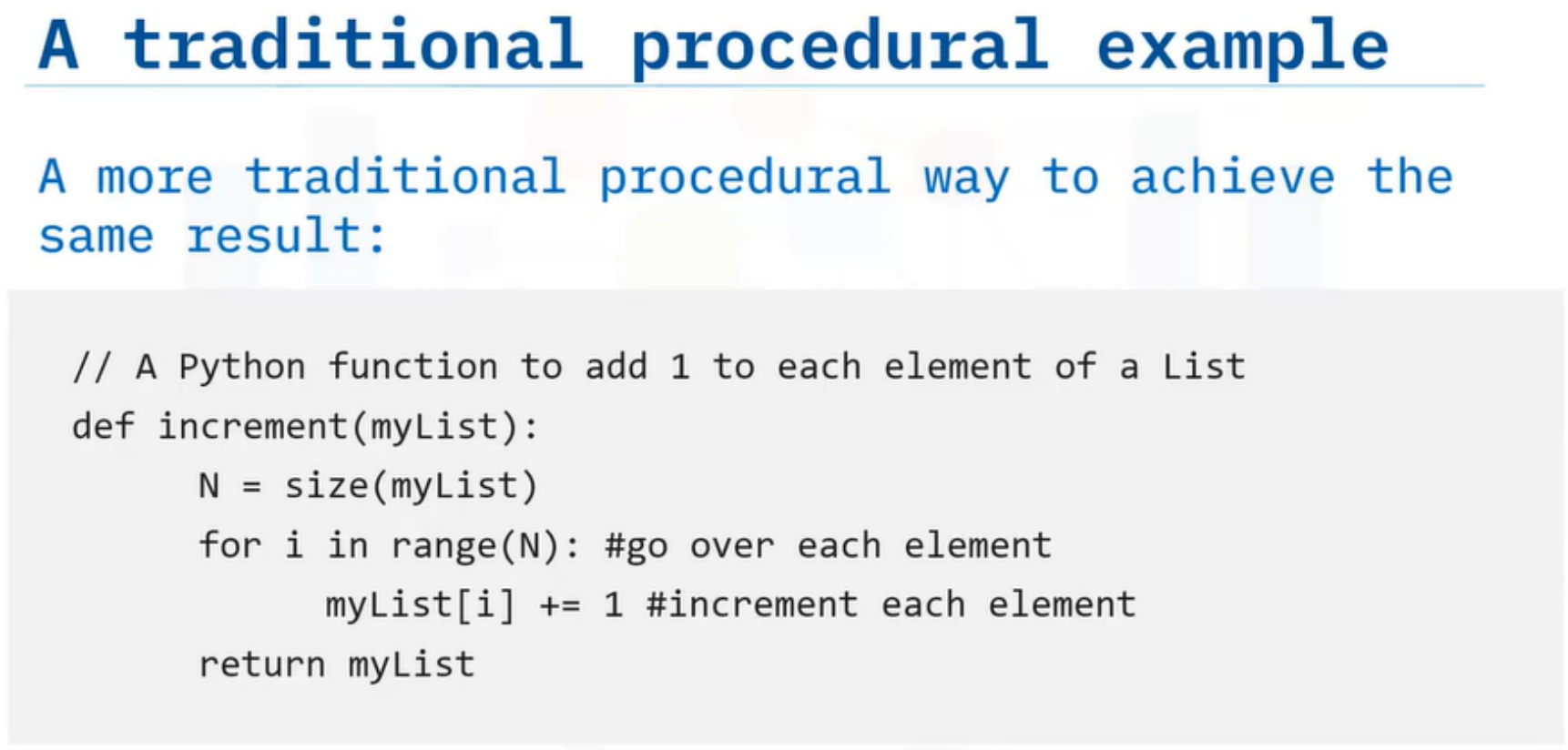


Here is a simple example of a functional program that increments a number by one. We define the function f of x equal to x +1. We can then apply this function to a list of size four as shown in the figure, and the program increments every element in the list by one.

Can directly apply functions to an entire list or array

Traditional procedural example

If you perform the same task in an imperative paradigm using procedural code, you would construct a “for-loop” that iterates over the array and increment each element by 1.

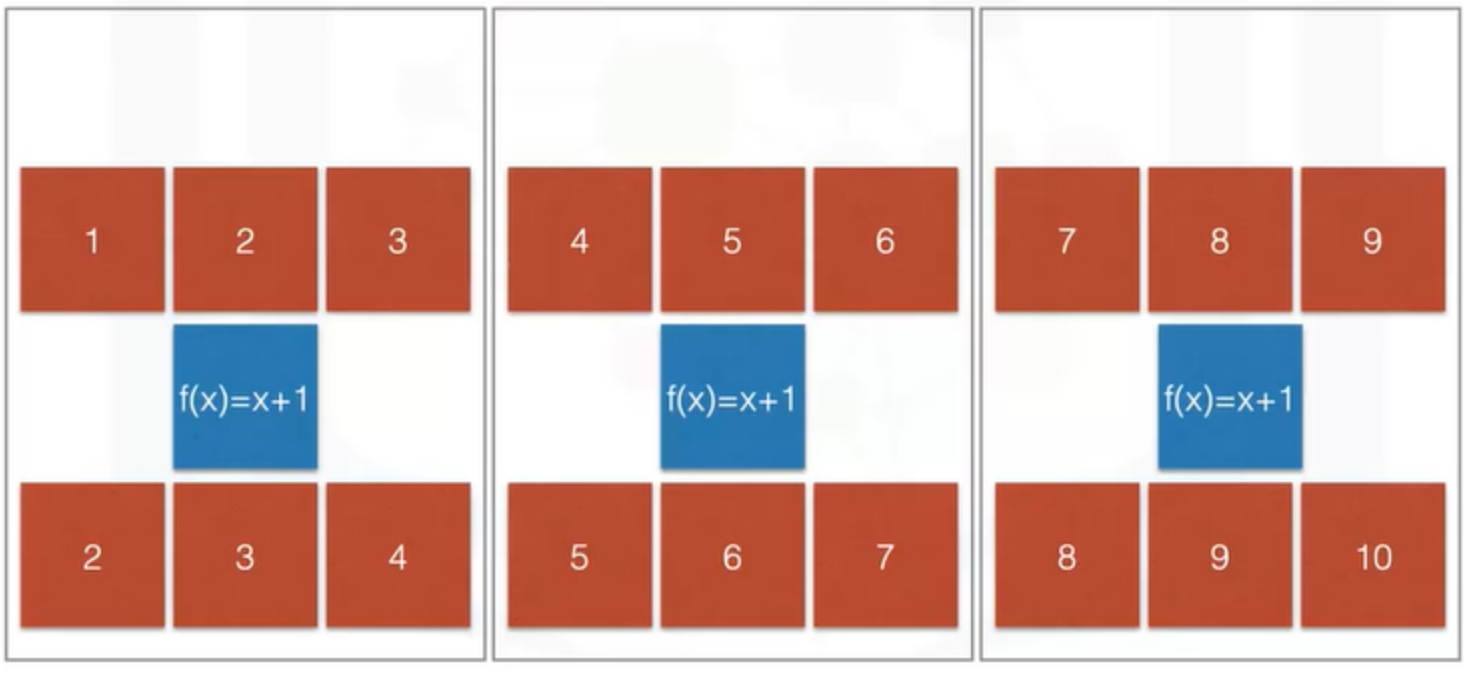


This programming paradigm differs from the preceding example by explicitly listing all the steps to be performed, including the “go over” for each element, incrementing each element by one. This code places emphasis on the ***“how-to.”***

Compare and contrast this traditional programming example to the functional programming example that emphasized on the “what” part of the solution.

Parallelization

Parallelization is one of the main benefits of Functional Programming.



Here is a larger array, from one to nine. You can split the task into three different computing chunks, which can be referred to as nodes, click one, and run those three tasks or nodes in parallel. This method's beauty is that you do not need to change any of the function definitions or the code.

Lambda calculus

Functional programming applies a mathematical concept called lambda calculus

- Lambda calculus functions, or operators, are anonymous functions that enable functional programming

Scala

```
// an example lambda
operator in Scala to add
2 numbers
val add = (x:Int, y:Int)
=> x + y
println(add(1,2))
```

Python

```
// an example lambda
function in Python to add
2 numbers
add = lambda x, y : x + y
print(add(1,2))
```

Lambda calculus basically says that:

every computation can be expressed as an anonymous function which is applied to a data set.

Lambda functions or operators are anonymous functions used to write functional programming code.

Here, we have the code written in functional programming style using lambda functions and operators to add two numbers using two popular languages, Scala as seen on the left, and Python, as seen on the right. Note how both codes are similar in flow, and abstract out the result directly, a hallmark of the declarative paradigm of programming.

Lambda function and Spark

Apache Spark is capable of quickly working through big data, by lambda functions distributing work between worker nodes and parallelized computations.

All Spark programs implemented this way are inherently parallel, so it doesn't matter if you analyze one kilobyte or one petabyte of data.

Conclusion

- Functional programming follows a declarative programming model that emphasizes “What” instead of “how to” and uses expressions.
- Lambda functions or operators are anonymous functions that enable functional programming.

- Spark parallelizes computations using the lambda calculus.
- All functional Spark programs are inherently parallel.

Parallel Programming using Resilient Distributed Datasets

What are RDD?

```
A resilient distributed dataset, also known as an RDD, is Spark's primary data abstraction.
```

A resilient distributed dataset:

- Spark's primary data abstraction
- is a collection of fault tolerant elements
- partitioned across the cluster's nodes
- capable of receiving parallel operations.
- ***immutable***, meaning that these databases cannot be changed once created.

Spark application

Every spark application consists of a driver program

runs the user's main functions

runs multiple parallel operations on a cluster.

RDD supported files

Supported file types:

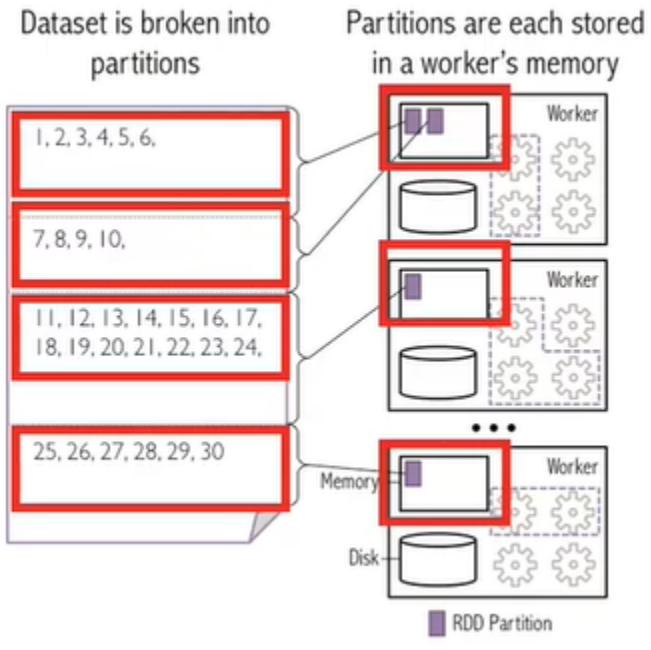
- Text
- Sequence fiels
- Avro
- Parquet
- Hadoop input format file types

Supported file formats

- Local
- Cassandra
- HBase
- HDFS
- Amazon S3
- SQL and NoSQL

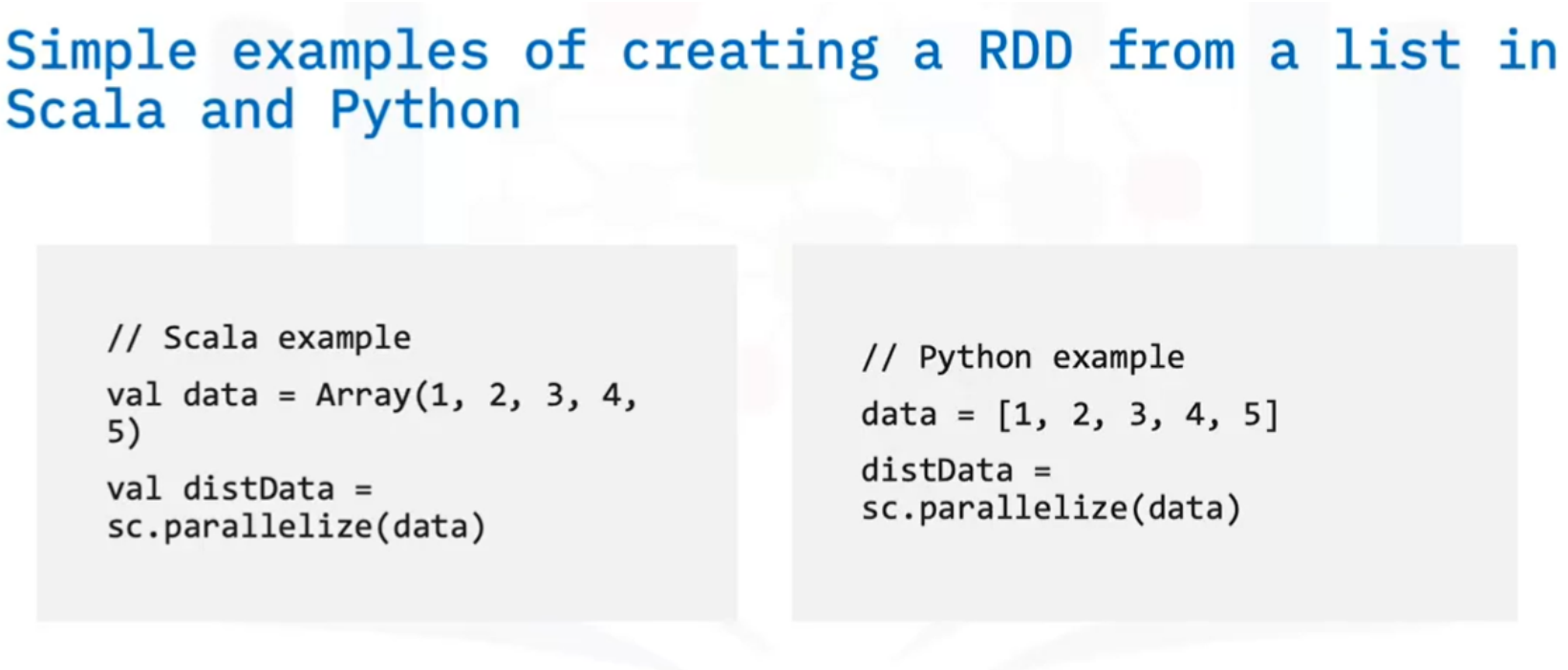
Create an RDD in Spark

Create an RDD using an **external** or **local file** from a Hadoop supported file system such as HDFS, Cassandra, H Base or Amazon S3



Create an RDD from collection

apply the parallelize function to an existing collection in the driver program.



Driver program can be in any of the supported high level APIs, including Python, Java, and Scala.

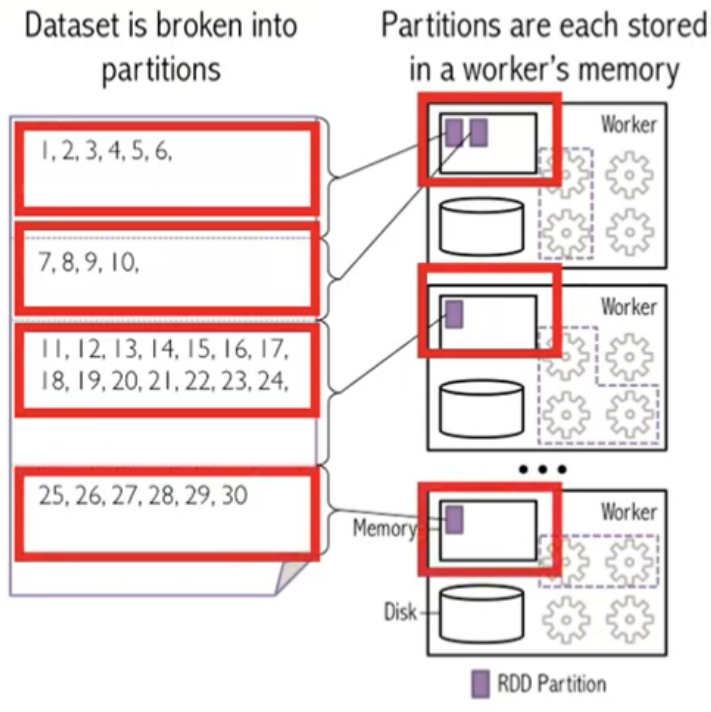
One important parameter for parallel collections is the number of partitions specified to cut the dataset. Spark runs one task for each partition of the cluster. Typically you want two to four partitions for each CPU in your cluster.

Spark tries to automatically set the number of partitions based on your cluster. However, you can also set partitions manually by passing the number as a second parameter to the parallelize function.

Creating an RDD by apply a transformation on an existing RDD

A third method of creating an RDD in Spark is to apply a transformation on an existing RDD to create a new RDD.

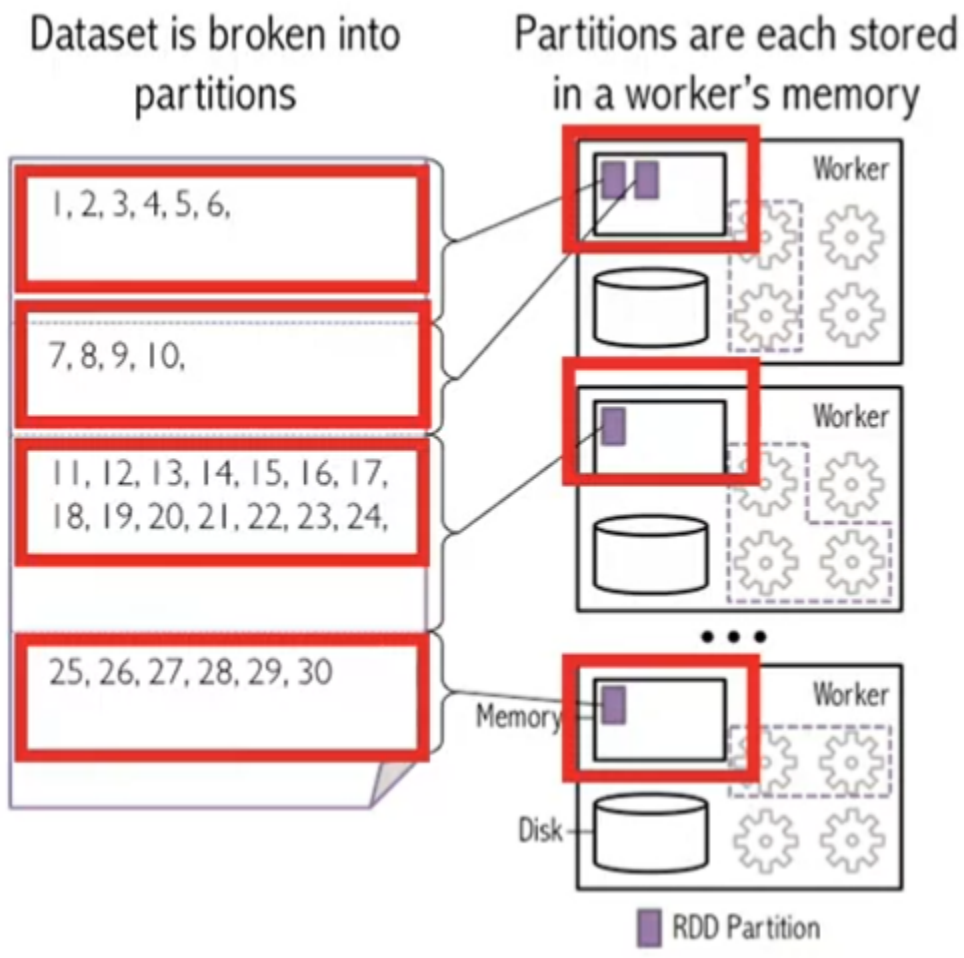
Apply a transformation on an existing RDD to create a new RDD



Parallel programming

- is the simultaneous use of multiple compute resources to solve a computational task
- parses tasks into discrete parts that are solved concurrently using multiple processors.
- The processors access a shared pool of memory, which has in place control and coordination mechanisms.

RDDs and Parallel programming



- Create an RDD by parallelizing an array of objects or by splitting a dataset into partitions.
- Nodes receive the distributed partitions.
- Spark runs one task for each partition of the cluster.
- Based on how RDDs are created, RDDs can inherently be operated on in parallel.

Resilience and Spark

RDD's provide resilience in Spark through ***immutability and caching***.

- Always recoverable as the data is immutable.
- Persisting or caching of a data set in memory across operations.

The cache is fault tolerant and always recoverable. as RDDs are immutable and the Hadoop data sources are also fault tolerant. When you persist in RDD, each node stores the partitions that the node computed in memory and reuses the same partition in other actions on that data set or the subsequent datasets derived from the first RDD. Persistence allows future actions to be much faster, often by more than 10 times.

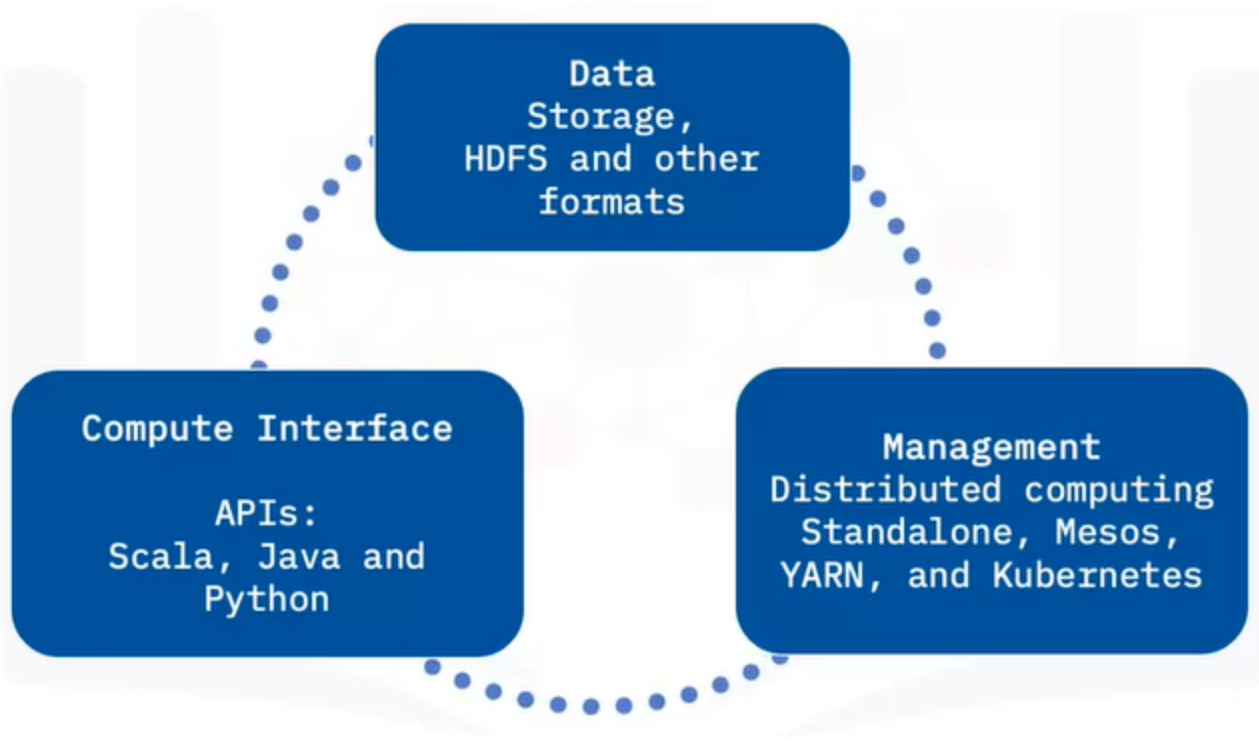
Persisting or caching is used as a key tool for iterative algorithms and fast interactive use.

Conclusion

- Create RDD using an external or local file from a Hadoop supported file, a collection or another RDD.
- RDDs are immutable and always recoverable.
- Parallel programming is the simultaneous use of multiple compute resources to solve a computational task.
- RDDs can persist or cache datasets in memory across operations, which spades iterative operations.

Scale out / Data Parallelism in Apache Spark

Apache Spark components



Apache Spark architecture consists of three main components.

- `Data storage` - Datasets load from data storage into memory. Any Hadoop compatible data source is acceptable.
- `High-level programming APIs` - Spark has APIs in Scala, Python, and Java.
- `Cluster management framework` - which handles the distributed computing aspects of Spark. Spark's cluster management framework can exist as a stand-alone server, Mesos or Yet Another Resource Network, or YARN.
  A cluster management framework is essential for scaling big data.

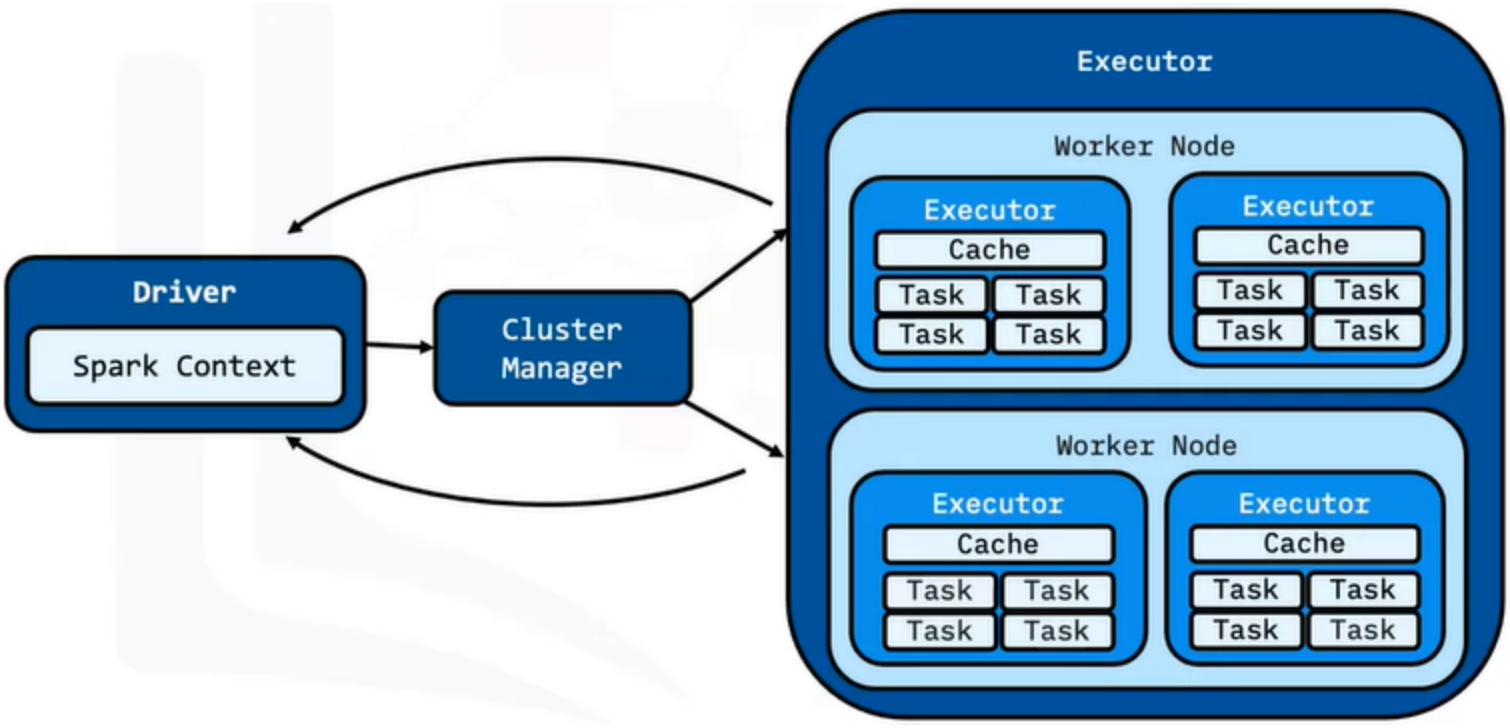
💡 *The data from a Hadoop file system flows into the compute interface or API, which then flows into different nodes to perform distributed/parallel tasks.*

Spark Core

- Is the base engine
- Is fault-tolerant
- Base engine for large-scale parallel and distributed data processing.
- Manages memory
- Task scheduling.
- Spark Core also contains the APIs used to define RDDs and other datatypes.

- Spark Core also parallelizes a distributed collection of elements across the cluster.

Scaling Big data in Spark



The Spark Application Architecture.

The Spark Application consists of the `driver program` and `the executor program`.

Executor programs run on `worker nodes`.

Spark can start additional executor processes on a worker if there is enough memory and cores available. Similarly, executors can also take multiple cores for multithreaded calculations.

Spark distributes RDDs among executors.

Communication occurs among the driver and the executors.

- The driver contains the Spark jobs that the application needs to run and splits the jobs into tasks submitted to the executors.
- The driver receives the task results when the executors complete the tasks.

> 💡 If Apache Spark were a large organization or company, the driver code would be the executive management of that company that makes decisions about allocating work, obtaining capital, and more. The junior employees are the executors who do the jobs assigned to them with the resources provided. The worker nodes correspond to the physical office space that the employees occupy.

You can add additional worker nodes to scale big data processing incrementally.

Conclusion

- Apache Spark architecture consists of three main pieces components data, compute input, and management.
- The fault-tolerant Spark Core base engine performs large-scale parallel and distributed data processing, manages memory, schedules tasks, and houses APIs that define RDDs.
- The Spark driver program communicates with the cluster and then distributes RDDs among worker nodes.

Dataframes and SparkSQL

SparkSQL

- Spark SQL is a Spark module for structured data processing.
- Interact with Spark SQL using SQL queries and the DataFrame API
- Spark SQL supports Java, Scala, Python, and R APIs.

- Spark SQL uses the same execution engine to compute the result independently of which API or language you are using for the computation.

Spark SQL query using Python

```
results = spark.sql(
  "SELECT * FROM people")
names = results.map(lambda p:
p.name)
```

Spark SQL benefits

- Includes a cost-based optimizer, columnar storage, and code generation to perform optimizations
- provide Spark with additional information about the structure of both the data and the computation in process.

DataFrame

A DataFrame is a collection of data organized into named columns.

DataFrames are conceptually equivalent to a table in a relational database and similar to a data frame in R/Python, but with richer optimizations.

DataFrames are built on top of the Spark SQL RDD API.

DataFrames use RDDs to perform relational queries.

Python code snippet to read from a JSON file and create a simple DataFrame.

```
df = spark.read.json("people.json")
df.show()
df.printSchema()

# Register the DataFrame as a SQL temporary view
df.createTempView("people")
```

The last step of registering the DataFrame runs SQL queries on this data using Spark SQL

.

Input JSON file

```
{"name":"Michael"}
{"name":"Andy",
"age":30}
{"name":"Justin",
"age":19}
```

Created DataFrame

```
+----+-------+
| age| name  |
+----+-------+
|null|Michael|
| 30|  Andy  |
| 19| Justin |
+----+-------+
```

DataFrame benefits

- DataFrames are highly scalable—from a few kilobytes on a single laptop to petabytes on a large cluster of machines.
- DataFrames support a wide array of data formats and storage systems.
- DataFrames provide optimization and code generation through a Catalyst optimizer.
- DataFrames are developer-friendly, offering integration with most big data tooling via Spark and APIs for Python, Java, Scala, and R.

DataFrame + Spark SQL

SQL Query

```
spark.sql("SELECT name FROM
people").show()
```

DataFrame Python API

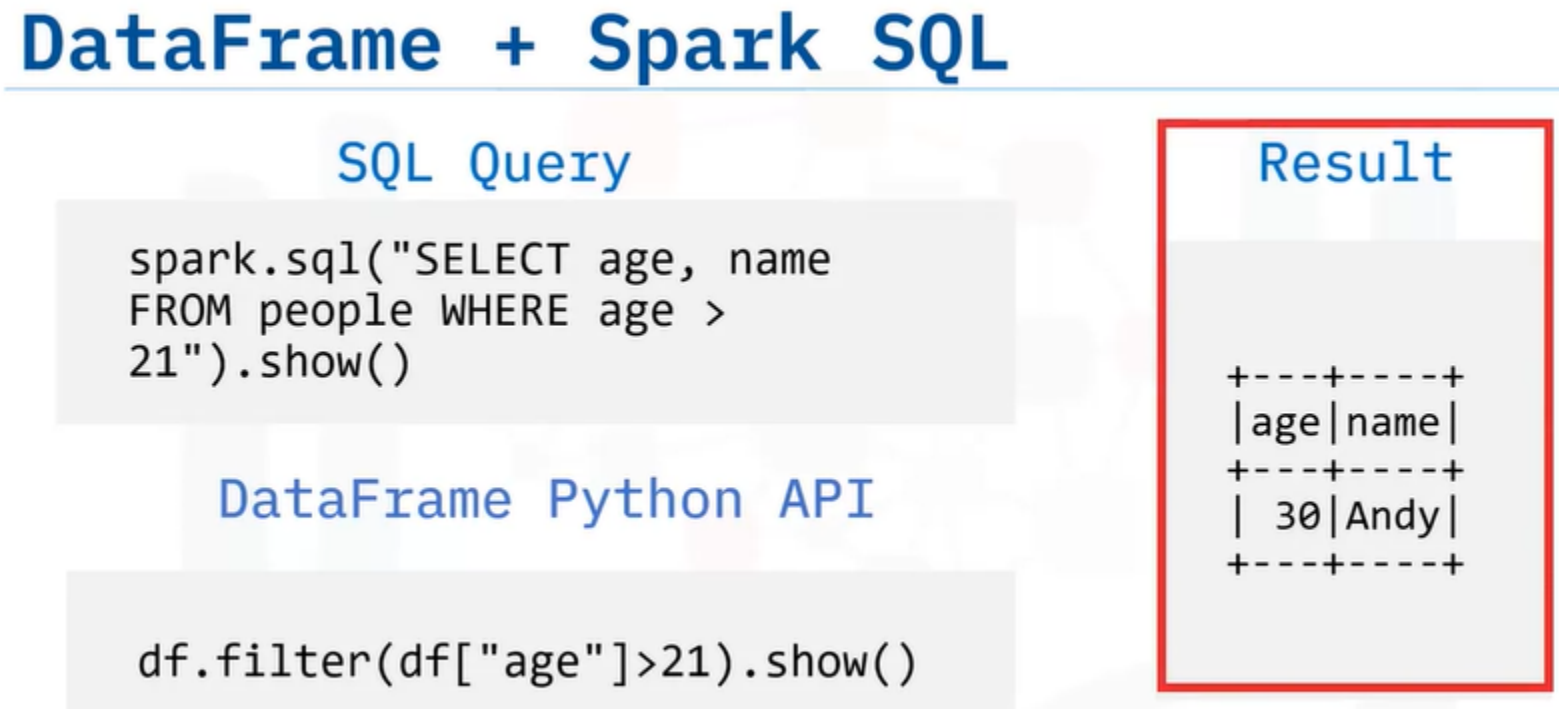
```
df.select("name").show()
df.select(df["name"]).show()
```

Result

```
+-------+
| name  |
+-------+
|Michael|
| Andy  |
| Justin|
+-------+
```

These code snippets show how to run the same SQL query and the result of those queries using Spark SQL.

Similarly, here are two different ways to query the entries in the DataFrame to locate people who are above the age of 21.



Conclusion

- SparkSQL is a Spark module for structured data processing.
- Spark SQL provides a programming abstraction called DataFrames and can also act as a distributed SQL query engine.
- DataFrames are conceptually equivalent to a table in a relational database or a data frame in R/Python, but with `richer optimizations.`

Labs

Getting Started with Spark using Python: https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-BD0225EN-SkillsNetwork/labs/SparkIntro.ipynb

---

Summary & Highlights

- Spark is an open source in-memory application framework for distributed data processing and iterative analysis on massive data volumes. Both distributed systems and Apache Spark are inherently scalable and fault tolerant. Apache Spark solves the problems encountered with MapReduce by keeping a substantial portion of the data required in-memory, avoiding expensive and time-consuming disk I/O.
- Functional programming follows a declarative programming model that emphasizes “what” instead of “how to” and uses expressions.
- Lambda functions or operators are anonymous functions that enable functional programming. Spark parallelizes computations using the lambda calculus and all functional Spark programs are inherently parallel.
- Resilient distributed datasets, or RDDs, are Spark’s primary data abstraction consisting of a fault-tolerant collection of elements partitioned across the nodes of the cluster, capable of accepting parallel operations.You can create an RDD using an external or local Hadoop-supported file, from a collection, or from another RDD. RDDs are immutable and always recoverable, providing resilience in Apache Spark RDDs can persist or cache datasets in memory across operations, which speeds iterative operations in Spark.
- Apache Spark architecture consists of components data, compute input, and management. The fault-tolerant Spark Core base engine performs large-scale Big Data worthy parallel and distributed data processing jobs, manages memory, schedules tasks, and houses APIs that define RDDs.
- Spark SQL provides a programming abstraction called DataFrames and can also act as a distributed SQL query engine. Spark DataFrames are conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations.

Quiz

Practice Quiz: Introduction to Apache Spark

Question 1

The benefits of working with Apache Spark are _____?

- **It is open-source and keeps a large portion of the data required for processing in-memory to reduce time and expense.**
- It uses parallel computing so that each processor can access its own memory, improving fault tolerance.
- It solves the read/write problems encountered with MapReduce by creating iterations that write to HDFS faster.
- It is primarily written in R and runs on Apache.

Apache Spark is an open-source, in-memory application framework for distributed data processing.

Question 2

What are the features or characteristics of a functional programming language such as Scala? Select all that apply.

- ☐ You must always use “for-loops” to iterate over arrays.
- ☐ It follows an imperative programming model.
- ☑ ~~It follows a declarative programming model.~~

Functional programming languages including Scala, Python, R and Java follow the mathematical function format (for example f of x) and are declarative rather than imperative

- ☑ ~~The emphasis of the code is on the “what” rather than the “how to” of the solution.~~

A functional programming language like Scala uses declarative syntax to abstract out the implementation details (the “how to”) and focuses on the final output (the “what”).

- ☑ ~~It treats functions as first-class citizens, for example functions can be passed as arguments to other functions.~~

Functions in Scala can be passed as arguments to other functions, returned by other functions and used as variables.

Question 3

Which of the following statements are true of Resilient Distributed Datasets (RDDs)? Select all that apply.

- ☑ ~~RDDs are persistent and speed up interactions because they can reuse the same partition in other actions on a dataset.~~

When you persist an RDD, each node stores the partitions that the node computed in memory and reuses (or caches) the same partition in other actions on that dataset or the subsequent datasets derived from the first RDD.

- ☐ An RDD is capable of accepting parallel instructions but is not immutable.
- ☐ You can create an RDD using an external or local file from any MongoDB-supported file system.
- ☑ ~~RDDs enable Apache Spark to reconstruct transformations.~~

You can create an RDD in Spark by applying a transformation on an existing RDD to create a new RDD.

- ☑ ~~An RDD is a distributed collection of elements parallelized across the cluster.~~

An RDD is a distributed collection of elements parallelized across the cluster.

Question 4

How is the Spark Application Architecture configured to scale big data? Select all that apply.

- ☑ ~~Apache Spark architecture consists of three main pieces: components data, compute input, and management.~~

Apache Spark architecture consists of three main pieces: components data, compute input, and management.

- ☑ ~~Spark Application consists of the driver program and the executor program.~~

Spark Application consists of the driver program and the executor program.

- ☑ ~~Executor programs run on worker nodes and you can add additional worker nodes to scale big data processing incrementally.~~

Executor programs run on worker nodes and you can add additional worker nodes to scale big data processing incrementally.

- [ ] The Spark Driver Program performs large-scale parallel and distributed data processing and houses APIs that define Restricted Distributed Datasets (RDDs).
- [ ] Spark can start additional executor processes on worker nodes at any time.

Question 5

Which SQL query options would display the names column from this DataFrame example? Select all that apply.

Name	Age
Michael	35
Claire	50
Ram	26
Zara	72

- [ ] spark.sql(“show”). name()
- [x] ~~df.select(df["name"]).show()~~

This example uses the DataFrame Python API to perform the task of displaying the Name column from the DataFrame.

- [x] ~~spark.sql("SELECT name FROM people").show()~~

This is a more traditional SQL query but will correctly display the Name column from the DataFrame.

- [x] ~~df.select("name").show()~~

This example uses the DataFrame Python API to perform the task of displaying the Name column from the DataFrame.

- [ ] df.select("name-age").show()

Graded Quiz: Introduction to Apache Spark

Question 1

Apache Spark uses distributed data processing and iterative analysis. Parallel and Distributed computing are similar. What is the difference between these computing types?

- [x] ~~Parallel computing utilizes shared memory~~

Correct, parallel computing shares all the memory.

- [x] ~~Distributed computing utilizes each processors own memory~~

Correct, each processor uses their own or distributed memory.

- [ ] Both computing types are exactly the same
- [ ] Parallel computing works best for small data sets

Question 2

Which of the following statements define functional programming?

- [ ] Shares functionality of Java Script
- [ ] Imperative in nature
- [x] ~~Emphasizes "What" instead of "How-to"~~

Correct, the emphasis of the code or program is in the "what" of the solution.

- [x] ~~Follows mathematical function format~~

Correct, FP follows the mathematical function format, like the algebra class with the f of x notation.

Question 3

The three Apache Spark components are data storage, compute interface, and cluster management framework. Which order does the data flow through these components?

- **Data flows from Hadoop file system, into compute interface and then into different nodes for distributed**

- Data flows from API into different nodes for parallel tasks, and then into a Hadoop file system
- Data flows from a Hadoop file system into different nodes for distributed task, but then flows to the APIs
- Data flows from compute interface to various nodes for distributed tasks, then goes to the Hadoop file system

Correct, the data from a Hadoop file system flows into the compute interface or API, which then flows into different nodes to perform distributed/parallel tasks.

Question 4

DataFrames are comparable to tables in relational databases or a data frame in which programming languages?

- RDDs
- **R/Python**
- NoSQL
- Java

Correct, DataFrames are conceptually equivalent to a data frame in R/Python.

Question 5

There are three ways to create an RDD in Spark. Which method involves Hadoop supported file systems like HDFS, Cassandra, or HBase?

- **Create an external or local file.**
- Apply transformation on existing RDDs.
- Code a functional program.
- Apply the parallelize function to existing collection.

Correct, create an external of local file from a Hadoop-supported file system like HDFS, Casandra, or HBase.