

# HCIA Big Data Spark — In-memory Distributed Computing Engine

**Eng. Hagar Muhammed**

## Lecture 6

# Contents

- 1. Spark Overview**
2. Spark Use Cases
3. Spark Data Structure
4. RDD Operations Type
5. Spark Architecture
6. Structured Streaming Overview

## Spark Overview

- It is a one-stop solution that integrates batch processing, real-time streaming, interactive query, graph programming, and machine learning in memory data processing engine



## Spark Overview

- Apache spark is a Parallel Processing framework that supports in-memory processing to improve performance of big-data analytic applications
- Run programs up to 100x faster than Hadoop MapReduce.
- A Spark job can
- **Load** and
- **Cache** data into memory and
- **Query** it repeatedly

# Contents

1. Spark Overview
- 2. Spark Use Cases**
3. Spark Data Structure
4. RDD Operations Type
5. Spark Architecture
6. Structured Streaming Overview

## Spark Use Cases

- Batch processing can be used for extracting, transforming, and loading (ETL).
- Machine learning can be used to automatically determine whether the comments of online buyers are positive or negative.
- Interactive analysis can be used to query the Hive warehouse.
- Streaming processing can be used for real-time businesses analysis, recommendation systems, and public opinion analysis.

## Highlights of Spark

**Lightweight**

Spark only has  
30,000 lines of core  
code.

**Fast**

Spark can respond to  
small data set  
queries in just  
subseconds.

**Flexible**

Spark offers  
flexibility at  
different levels.

**Smart**

Spark utilizes  
existing big data  
components.

## Spark vs. MapReduce

|                         | <b>Hadoop<br/>(Map Reduce)</b> | <b>Spark</b> |
|-------------------------|--------------------------------|--------------|
| Data Size               | 102.5 TB                       | 102 TB       |
| Time Required (Minutes) | 72                             | 23           |
| Number of Nodes         | 2100                           | 206          |
| Cores                   | 50400                          | 6592         |
| Rate                    | 1.4 TB/min                     | 4.27 TB/min  |
| Rate/node               | 0.67 GB/min                    | 20.7 GB/min  |

# Contents

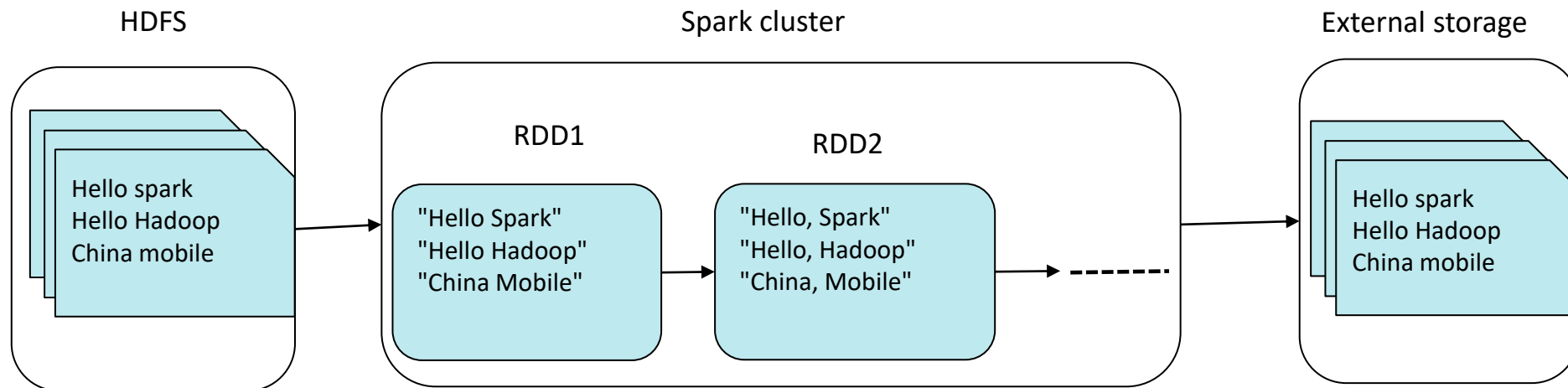
1. Spark Overview
2. Spark Use Cases
- 3. Spark Data Structure**
4. RDD Operations Type
5. Spark Architecture
6. Structured Streaming Overview

## RDD — Core Concept of Spark

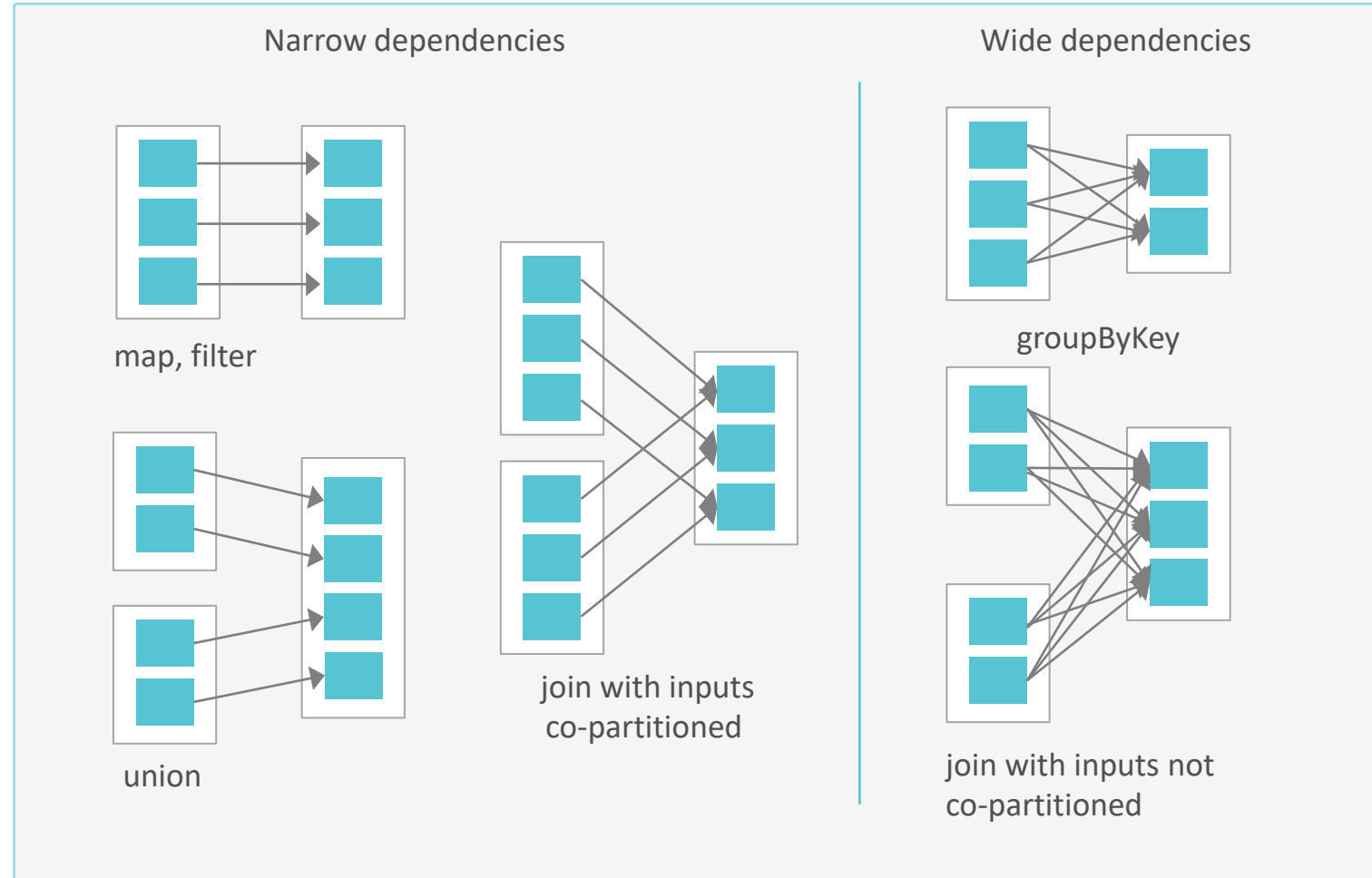
- **Resilient Distributed Datasets:**
- **Resilient:** fault- tolerant with the help of RDD lineage graph (DAG),able to recompute missing or damaged partitions due to node failures.
- **Distributed:** collection of data elements resides on multiple node in cluster.
- **Dataset :** represents records of the data you work with.

## RDD — Core Concept of Spark

- RDDs are elastic, read-only, and partitioned distributed datasets.
- By default, RDDs are stored in the memory and are written to disks when memory is insufficient.
- RDD data is stored in clusters as partitions.
- RDDs have a lineage mechanism, which allows for rapid data recovery when data is lost.



# RDD Dependencies

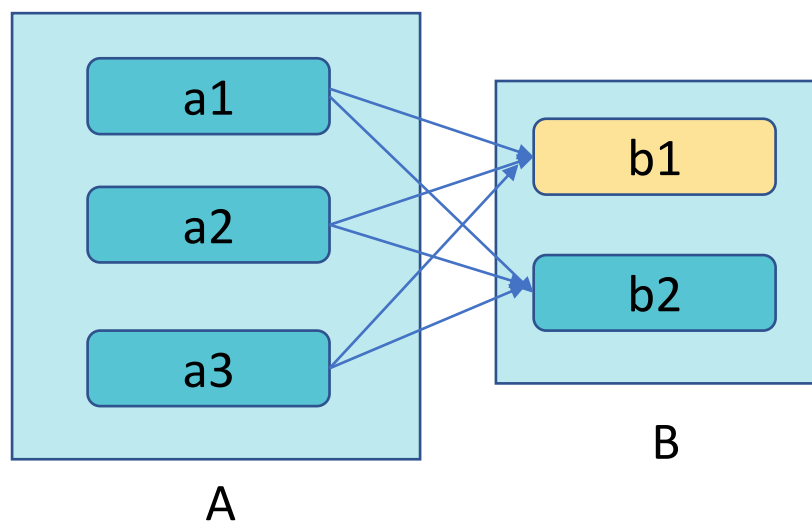


## Differences Between Wide and Narrow Dependencies — Operator

- A narrow dependency indicates that only one child RDD can use each partition of a parent RDD, for example, **map**, **filter**, and **union**.
- A wide dependency indicates that the partitions of multiple child RDDs depend on the partition of the same parent RDD, for example, **groupByKey**, **reduceByKey**, and **sortByKey**.

## Differences Between Wide and Narrow Dependencies — Fault Tolerance

- If a node is faulty:
  - Narrow dependency: Only the parent RDD partition corresponding to the child RDD partition needs to be recalculated.
  - Wide dependency: In extreme cases, all parent RDD partitions need to be recalculated.
- As shown in the following figure, if the b1 partition is lost, a1, a2, and a3 need to be recalculated.



# Contents

1. Spark Overview
2. Spark Use Cases
3. Spark Data Structure
- 4. RDD Operations Type**
5. Spark Architecture
6. Structured Streaming Overview

## RDD Operation Type

**Spark operations can be classified into creation, Transformation, control, and Action.**

- **Creation:** used to create an RDD. An RDD is created through a memory collection and an external storage system or by a transformation operation.
- **Transformation:** An RDD is transformed into a new RDD through certain operations. The transformation operation of the RDD is a lazy operation, which only defines a new RDD but does not execute it immediately.

## RDD Operation Type

- **Control:** RDD persistence is performed. An RDD can be stored in the disk or memory based on different storage policies.
- **Action:** an operation that can trigger Spark running. There are two types of action operations in Spark. One is to output the calculation result, and the other is to save the RDD to an external file system or database.

## Creation Operation

- Currently, there are two types of basic RDD
  - Parallel collection: An existing collection is collected and computed in parallel.
  - External storage: A function is executed on each record in a file. The file system must be the HDFS or any storage system supported by Hadoop.
- The two types of RDD can be operated in the same way to obtain a series of extensions, such as sub-RDD, and form a lineage diagram.

## Control Operation

- Spark can persistently store RDD to the memory or disk file system persistently. The RDD in memory can significantly improve iterative computing and data sharing between computing models. Generally, 60% of the memory of the execution node is used to cache data, and the remaining 40% is used to run tasks. In Spark, **persist** and **cache** operations are used for data persistency. Cache is a special case of **persist()**.

# Transformation Operation — Transformation Operator

| Transformation                 | Description                                                                                                                                                                                                    |
|--------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| map(func)                      | Uses the <b>func</b> method to generate a new RDD for each element in the RDD that invokes <b>map</b> .                                                                                                        |
| filter(func)                   | <b>func</b> is used for each element of an RDD that invokes <b>filter</b> and then an RDD with elements containing <b>func</b> (the value is <b>true</b> ) is returned.                                        |
| reduceByKey(func, [numTasks])  | It is similar to <b>groupByKey</b> . However, the value of each key is calculated based on the provided <b>func</b> to obtain a new value.                                                                     |
| join(otherDataset, [numTasks]) | If the data set is <b>(K, V)</b> and the associated data set is <b>(K, W)</b> , then <b>(K, (V, W))</b> is returned.<br><b>leftOuterJoin</b> , <b>rightOuterJoin</b> , and <b>fullOuterJoin</b> are supported. |

## Action Operation — Action Operator

| Action               | Description                                                                                                                         |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| reduce(func)         | Aggregates elements in a dataset based on functions.                                                                                |
| collect()            | Used to encapsulate the <b>filter</b> result or a small enough result and return an array.                                          |
| count()              | Collects statistics on the number of elements in an RDD.                                                                            |
| first()              | Obtains the first element of a dataset.                                                                                             |
| take(n)              | Obtains the top elements of a dataset and returns an array.                                                                         |
| saveAsTextFile(path) | This API is used to write the dataset to a text file or HDFS. Spark converts each record into a row and writes the row to the file. |

- Similar to RDD, DataFrame is also an invariable, elastic, and distributed dataset. It records the data structure information, that is, schema, in addition to data. The schema is similar to a two-dimensional table.
- The query plan of DataFrame can be optimized by Spark Catalyst Optimizer. Spark is easy to use, which improves high-quality execution efficiency. Programs written by DataFrame can be converted into efficient forms for execution.

## DataSet

- DataFrame is a special case of DataSet (**DataFrame=Dataset[Row]**). Therefore, you can use the **as** method to convert DataFrame to DataSet. **Row** is a common type where all table structure information is represented by row.
- DataSet, a typed dataset, includes Dataset[Car] and Dataset[Person].

# Differences Between DataFrame, DataSet, and RDD

- Assume that there are two lines of data in an RDD, which are displayed as follows:

```
1, Lucy, 23
```

```
2, Tony, 35
```

- The data in DataFrame is displayed as follows:

| ID:String | Name:String | Age:int |
|-----------|-------------|---------|
| 1         | Lucy        | 23      |
| 2         | Tony        | 35      |

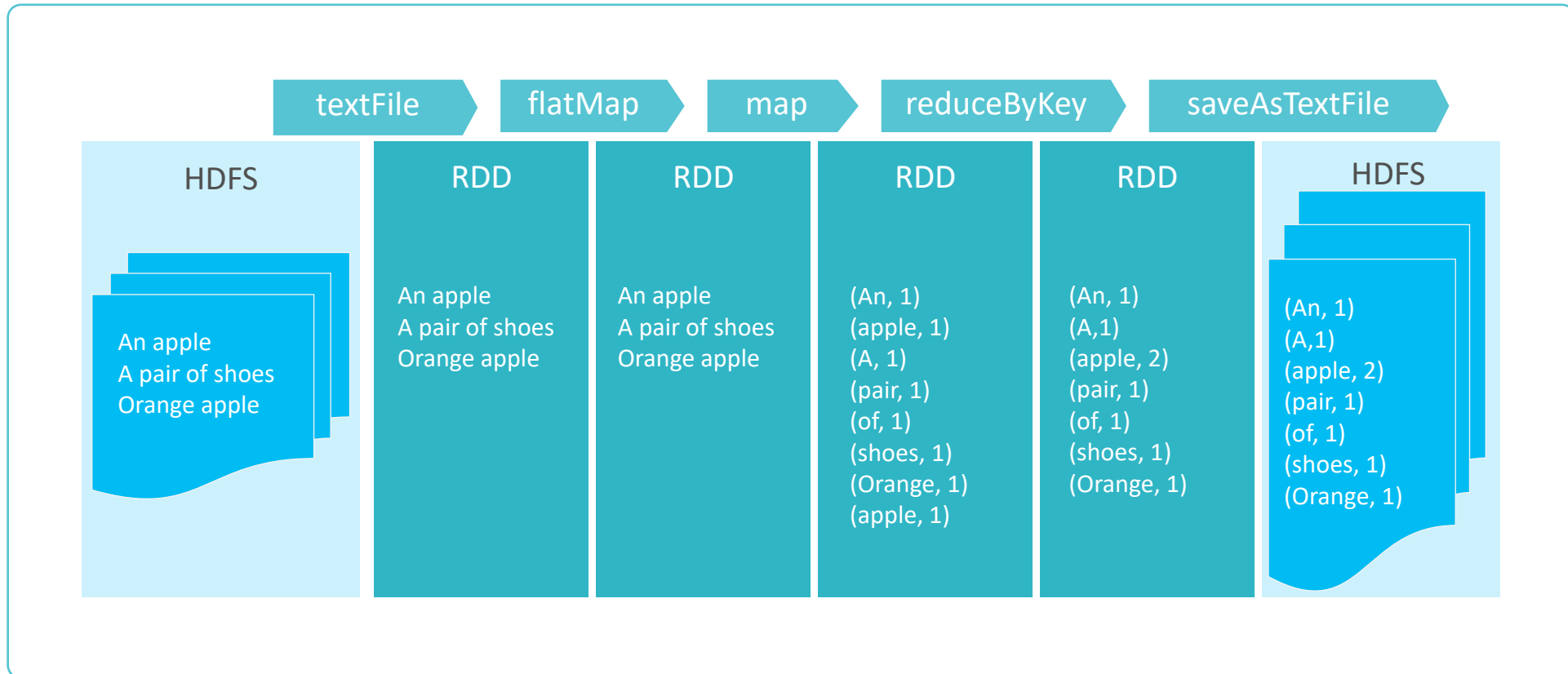
- The data in DataSet is displayed as follows:

```
Value:People[age:bight,id:bigint,name:string]
```

```
People(id=1,name="Lucy",age=23)
```

```
People(id=2,name="Tony",age=35)
```

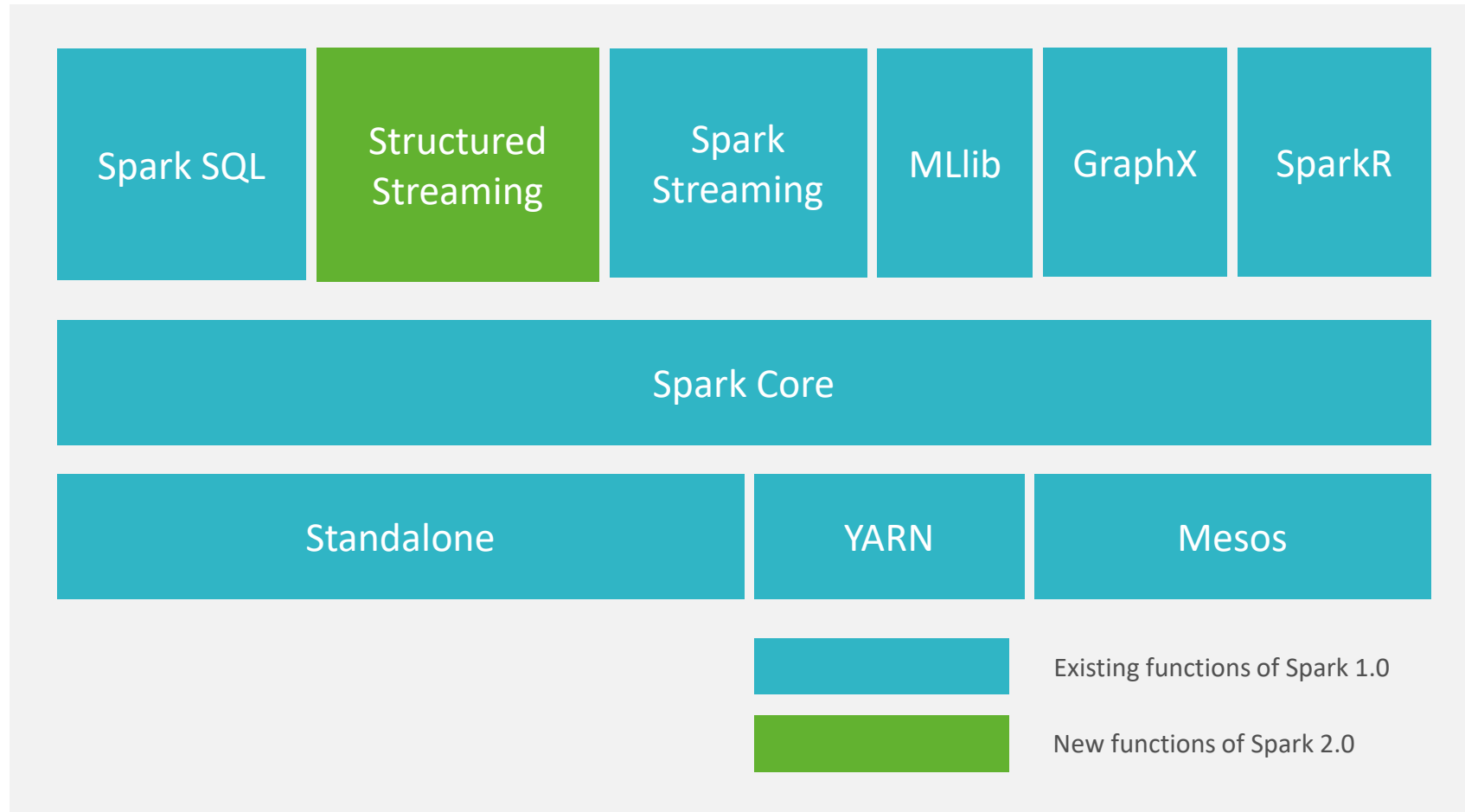
## Typical Case — WordCount



# Contents

1. Spark Overview
2. Spark Use Cases
3. Spark Data Structure
4. RDD Operations Type
- 5. Spark Architecture**
6. Structured Streaming Overview

# Spark Architecture



## Spark Architecture

**Spark Core:** It is similar to the distributed memory computing framework of MapReduce. The most striking feature of Spark Core is that it directly inputs the intermediate computing results into the memory to improve computing performance. In addition to the resource management framework in standalone mode, it also supports the resource management system of Yarn and Mesos.

## Spark Architecture

**Spark SQL:** It is a Spark component for processing structured data. As a part of the big data framework of Apache Spark, it is mainly used to process structured data and perform SQL-like data queries. With Spark SQL, users can perform ETL operations on data in different formats and on different data sources (such as HDFS and databases), completing specific query operations.

**Spark Streaming:** It is the stream processing engine for mini batch processing. After stream data is fragmented, the computing engine of Spark Core is used to process the data.

- **Differences**

- The execution engine of Spark SQL is Spark Core, and the default execution engine of Hive is MapReduce.
- Spark SQL executes 10 to 100 times faster than Hive.
- Spark SQL does not support buckets, but Hive does.

# Spark SQL vs. Hive

- **Relationships**

- Spark SQL depends on Hive's metadata.
- Spark SQL is compatible with most Hive syntax and functions.
- Spark SQL can use the custom functions of Hive.

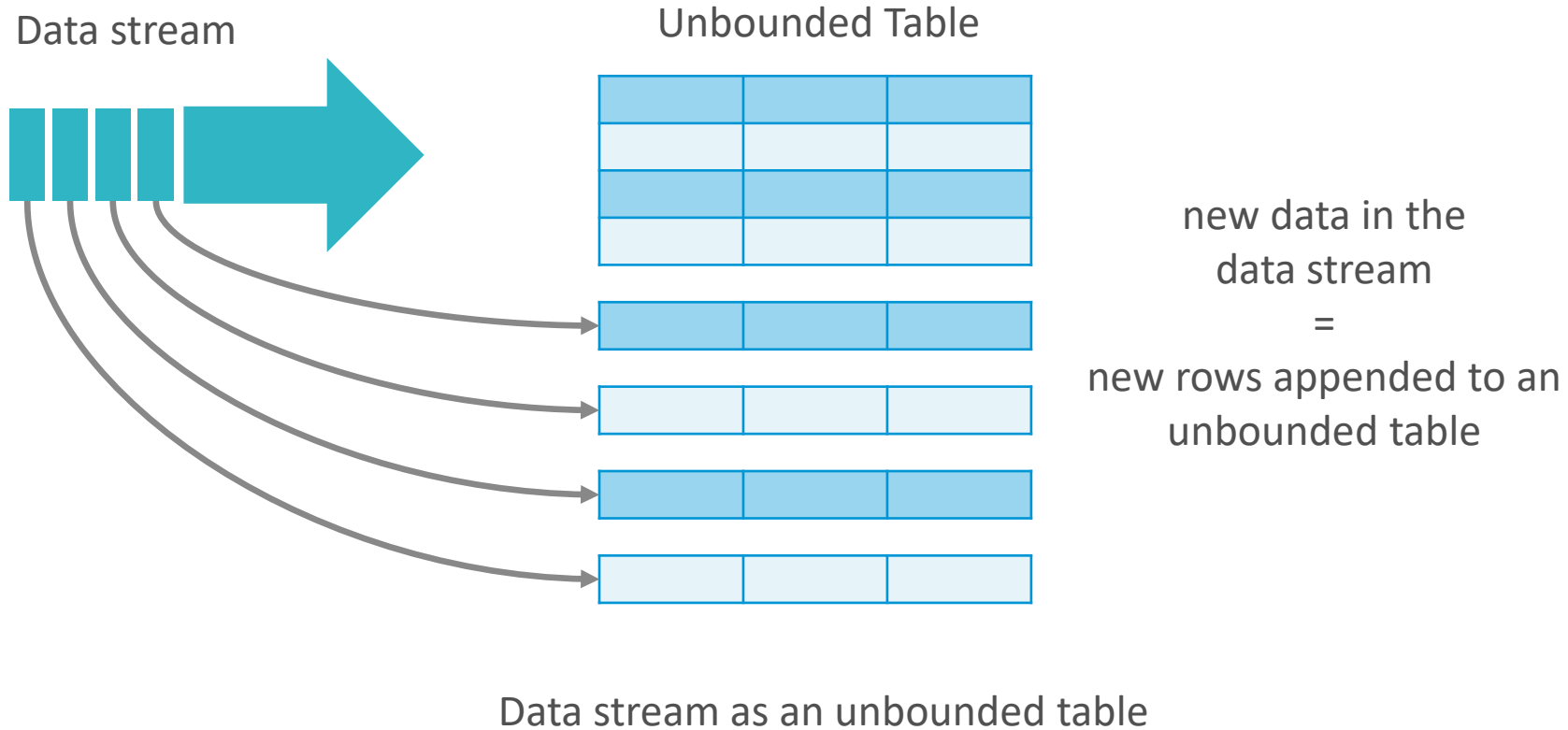
# Contents

1. Spark Overview
2. Spark Use Cases
3. Spark Data Structure
4. RDD Operations Type
5. Spark Architecture
- 6. Structured Streaming Overview**

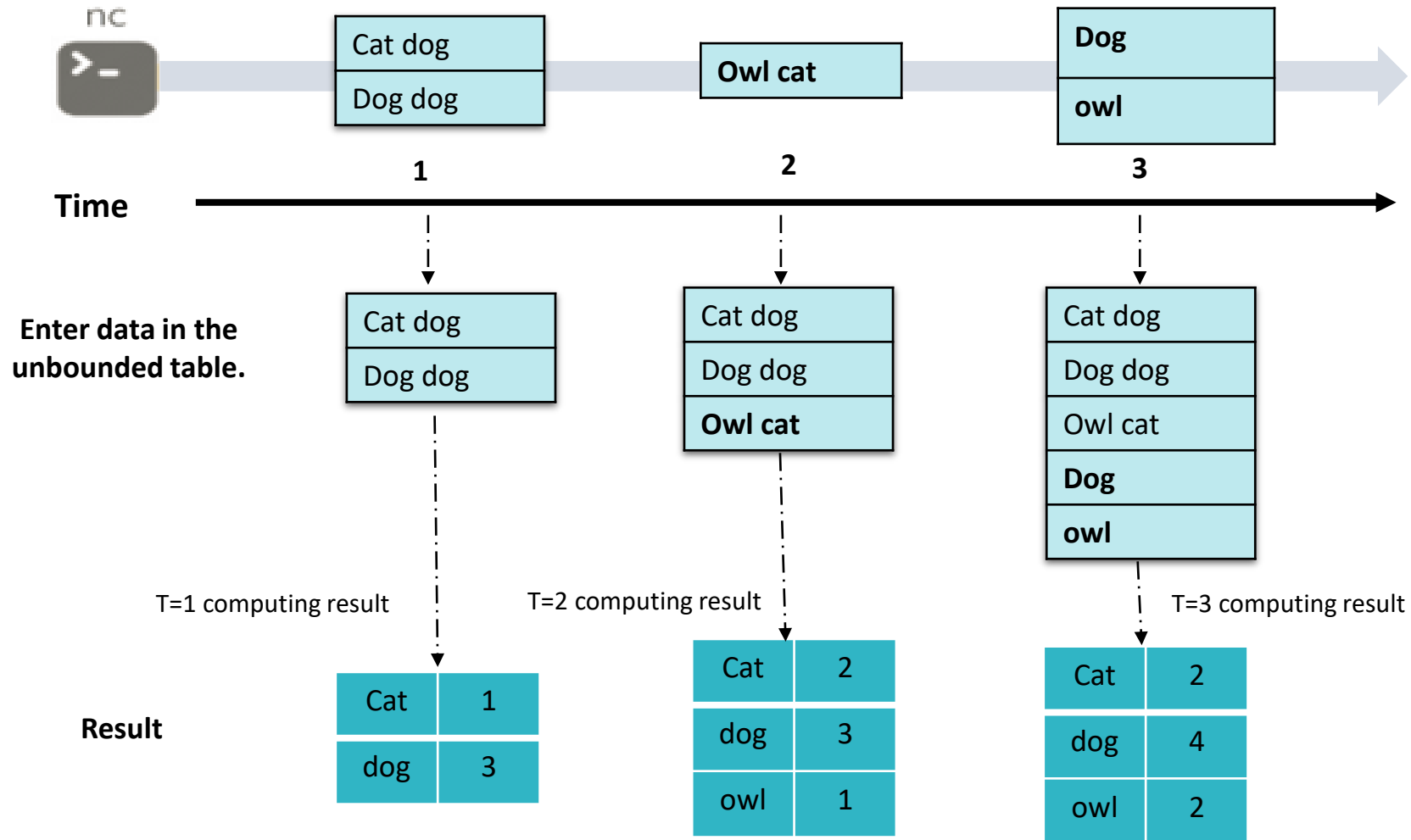
## Structured Streaming Overview

- Structured Streaming is a stream processing engine built on the Spark SQL engine. You can compile a streaming computing process like using static RDD data. Spark SQL will take care of running it incrementally and continuously, and then updates the final result as streaming data continues to arrive.

# Structured Streaming Overview

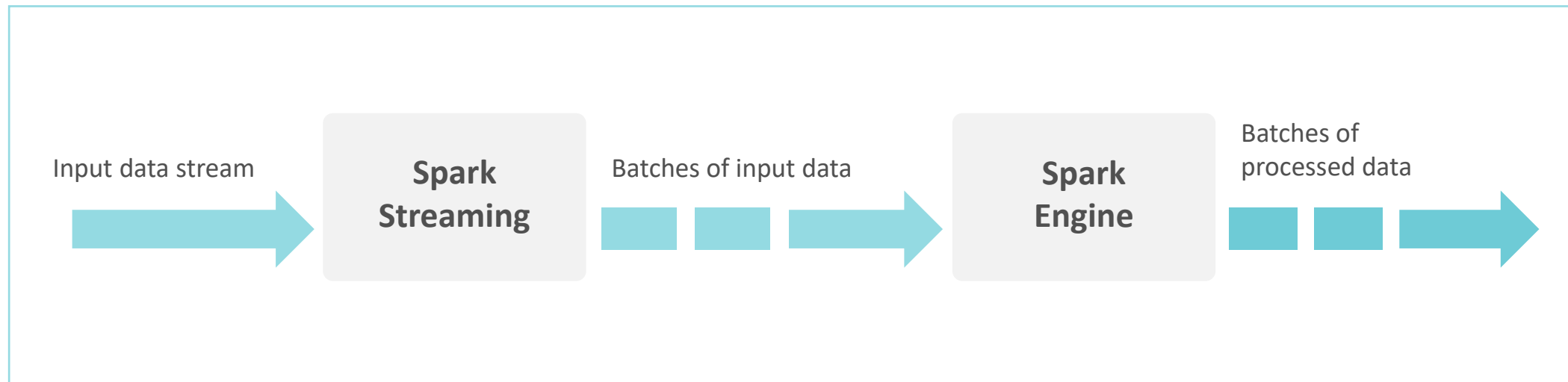


# Sample Programming Model of Structured Streaming



## Spark Streaming Overview

- The basic principle of Spark Streaming is to split real-time input data streams by time slice (in seconds), and then use the Spark engine to process data of each time slice in a way similar to batch processing.



## Spark Streaming vs. Storm

| Parameter                 | Storm                                                         | Spark Streaming                                                                 |
|---------------------------|---------------------------------------------------------------|---------------------------------------------------------------------------------|
| Real-time computing model | Real time: A piece of data is processed once it is requested. | Quasi real time: Data within a time range is collected and processed as an RDD. |
| Real-time computing delay | In milliseconds                                               | In seconds                                                                      |
| Throughput                | Low                                                           | High                                                                            |
| Transaction mechanism     | Supported and perfect                                         | Supported but not perfect                                                       |
| Fault tolerance           | High for ZooKeeper                                            | Normal for Checkpoint                                                           |

# Summary

Apache Spark is an open-source, distributed data processing framework designed for high-speed, large-scale data analysis. It offers in-memory computing, enabling rapid data processing and manipulation. Spark's components include Spark Core for task scheduling and fault tolerance, Spark SQL for structured data querying, Spark Streaming for real-time data processing, Spark MLlib for scalable machine learning, and Spark GraphX for graph computation.

# Summary

It supports various data sources, integrates with multiple programming languages, and can be used with popular cluster managers like YARN and Mesos. Spark's versatility and performance make it a favored choice for diverse data analytics tasks.

Thank You  
For Your Attention

