MAS DSE 230
Scalable Analytics

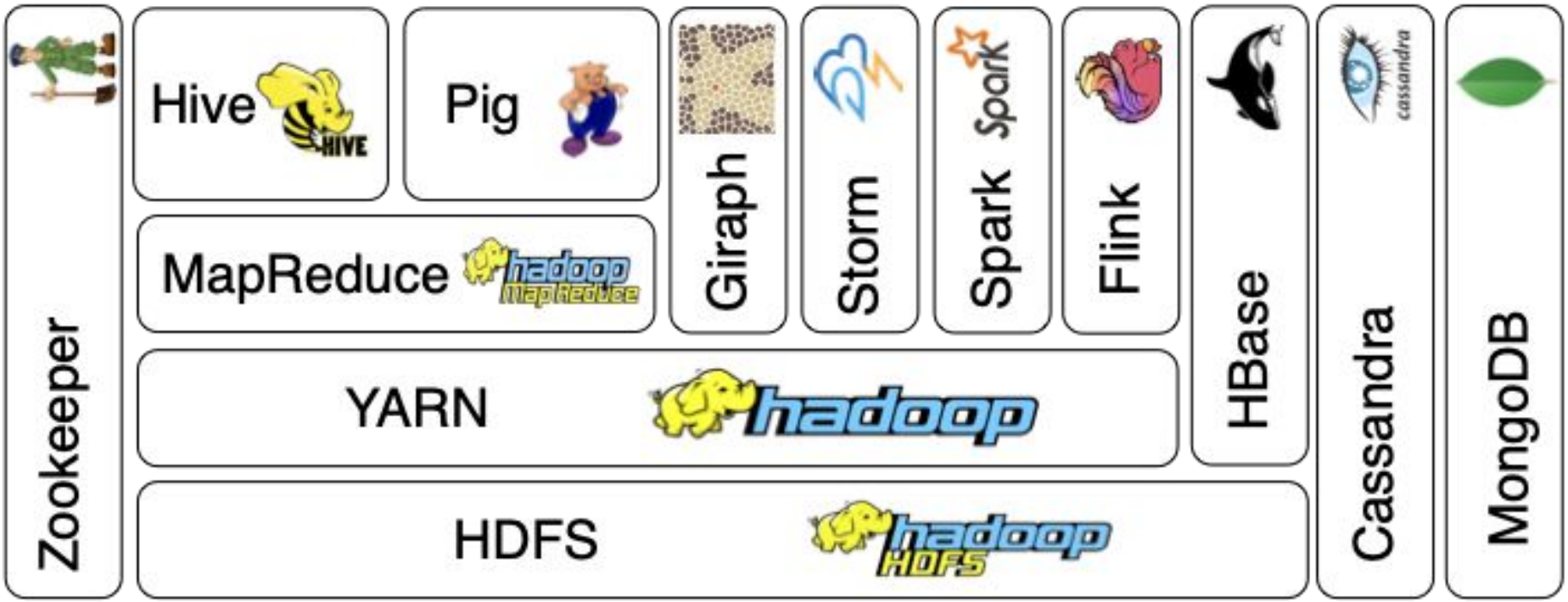
Spark

Mai H. Nguyen

BIG DATA & DISTRIBUTED PROCESSING

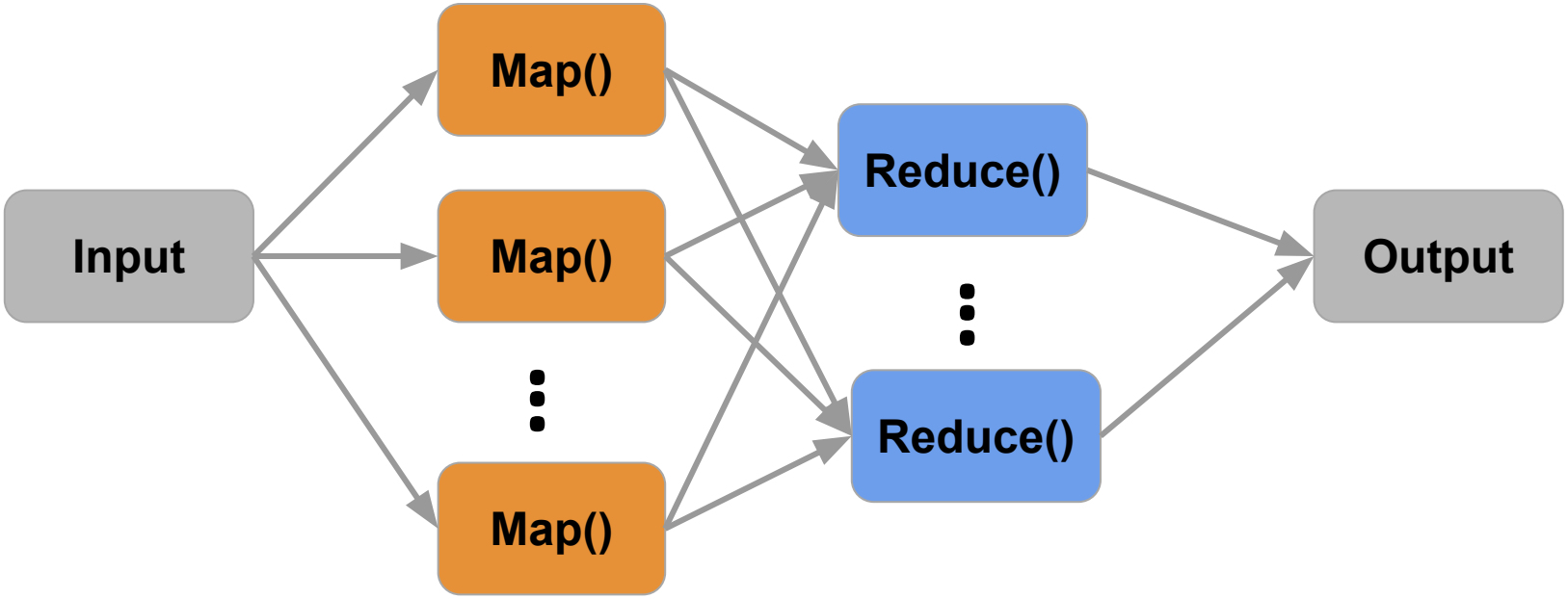
- Big Data Overview
- Scalable Systems
- Hadoop
- Spark
  - History
  - RDDs
  - DataFrames
  - Spark Architecture
  - Spark API
- PySpark Exercise
- Assignment

HADOOP ECOSYSTEM



- Scalable platform on commodity clusters
- Fault tolerance via data replication
- Support for variety of workloads

SHORTCOMINGS OF MAP-REDUCE



- Not all computations can be converted to MapReduce
- Involves a lot of disk I/O
- Designed for batch processing
- Very slow for iterative algorithms & interactive jobs
- No support for streaming data
- No interactive shell support

SPARK

- Computing platform for distributed computing
  - Built-in parallelism & fault-tolerance on commodity cluster
  - Extends MapReduce operations
  - Provides interactive querying, iterative analytics, streaming processing
  - Adopts Python Pandas style of function calls
  - Goals: speed, ease of use, generality, unified platform
- History
  - Research project began in 2009 at UC Berkeley's AMPlab
  - Paper published in 2010
  - Contributed to Apache Software Foundation in 2013
  - Commercial version by Databricks
- Compatible with Hadoop!

SPARK

- Goals: **speed**, ease of use, generality, unified platform
- In-memory processing
  - Exploits distributed memory to cache data
  - Intermediate results written to memory instead of disk
- How does Spark manage data in distributed system?

RESILIENT DISTRIBUTED DATASETS (RDDs)

- Spark central concept
  - Abstraction of data as distributed collection of objects
- Data abstraction
  - Resilient Distributed Datasets (RDDs)
  - Programming construct for storing data
  - Spark uses RDDs to distribute data and computations across nodes in cluster

RDDs

- Resilient Distributed **Dataset**
  - Collection of data
    - From files in local filesystem (text, JSON, etc.)
    - From data store (HDFS, RDBMS, NoSQL, etc.)
    - Created from another RDD
- Resilient **Distributed** Dataset
  - Data is divided into partitions
  - Partitions are distributed across nodes in cluster
- **Resilient** Distributed Dataset
  - Provides resilience (e.g., fault tolerance) to failures
  - History of operations performed on each partition is tracked to provide lineage-based fault tolerance
- All provided automatically by Spark engine

SPARK CONTEXT

- Spark Context
  - Entry point to Spark engine
  - Provides way to create RDDs

```
from pyspark import SparkContext, SparkConf

conf = SparkConf()\
   .setAppName("RDD Example") \
   .config("config.option","config.value")
sc = SparkContext(conf=conf)
```

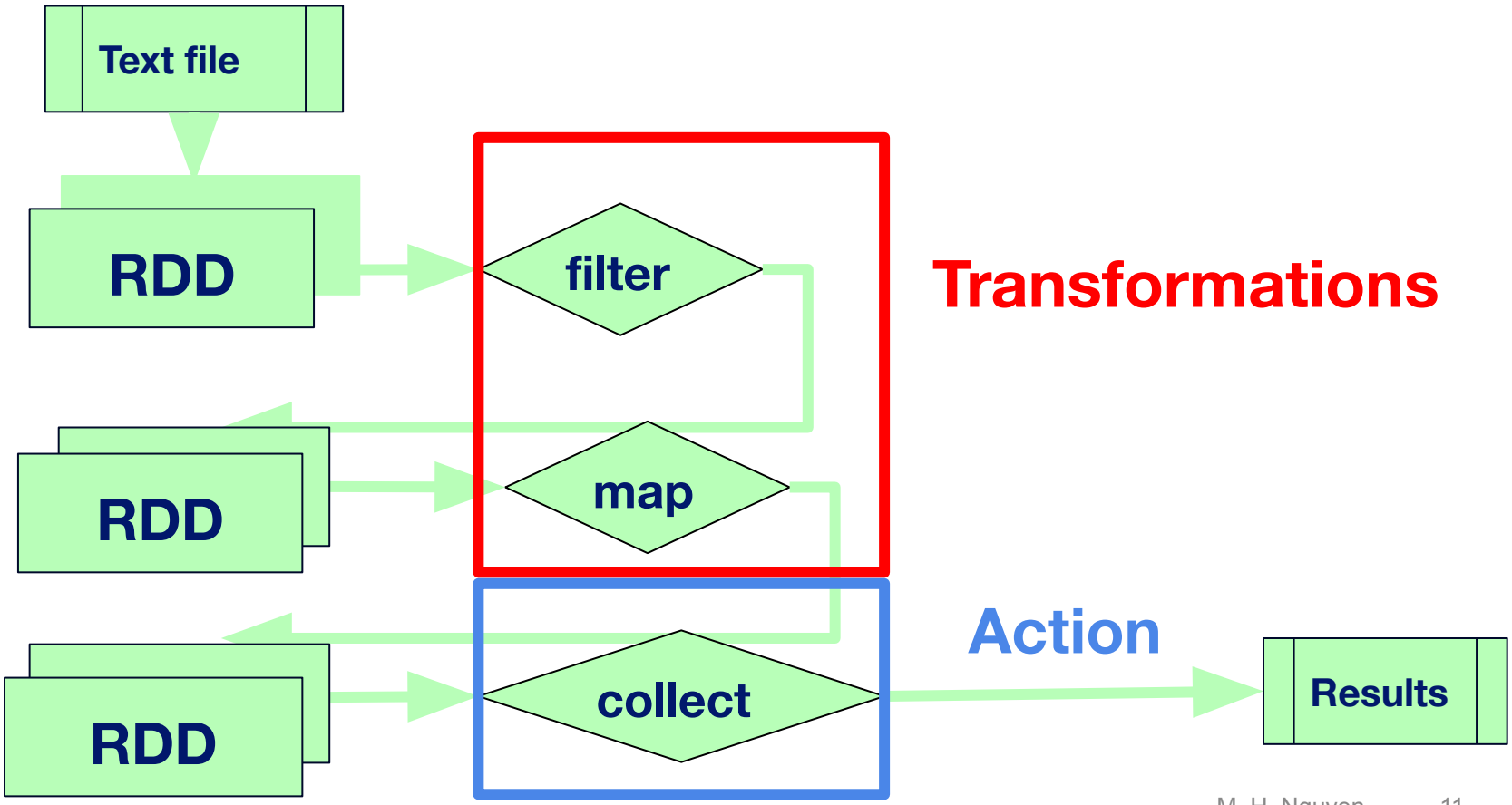
- SparkContext: connection to Spark engine
- SparkConf: configuration parameters for application

CREATING RDDS

- Read data from files in local filesystem (text, JSON, etc.)
  - lines = sc.textFile(“inputfile.txt”)
- Data read in from data store (HDFS, RDBMS, NoSQL, etc.)
  - lines = sc.textFile(“hdfs://<path>/inputfile.txt”)
- Generate data
  - numbers = sc.parallelize(range(100),3)
    - Divide data into 3 partitions
- Created by transforming another RDD
  - newLines = lines.filter(lambda s: “Spark” in s)
- Note: RDDs are **immutable**
  - To “change” RDD, create another RDD to hold changed data

PROCESSING RDDs

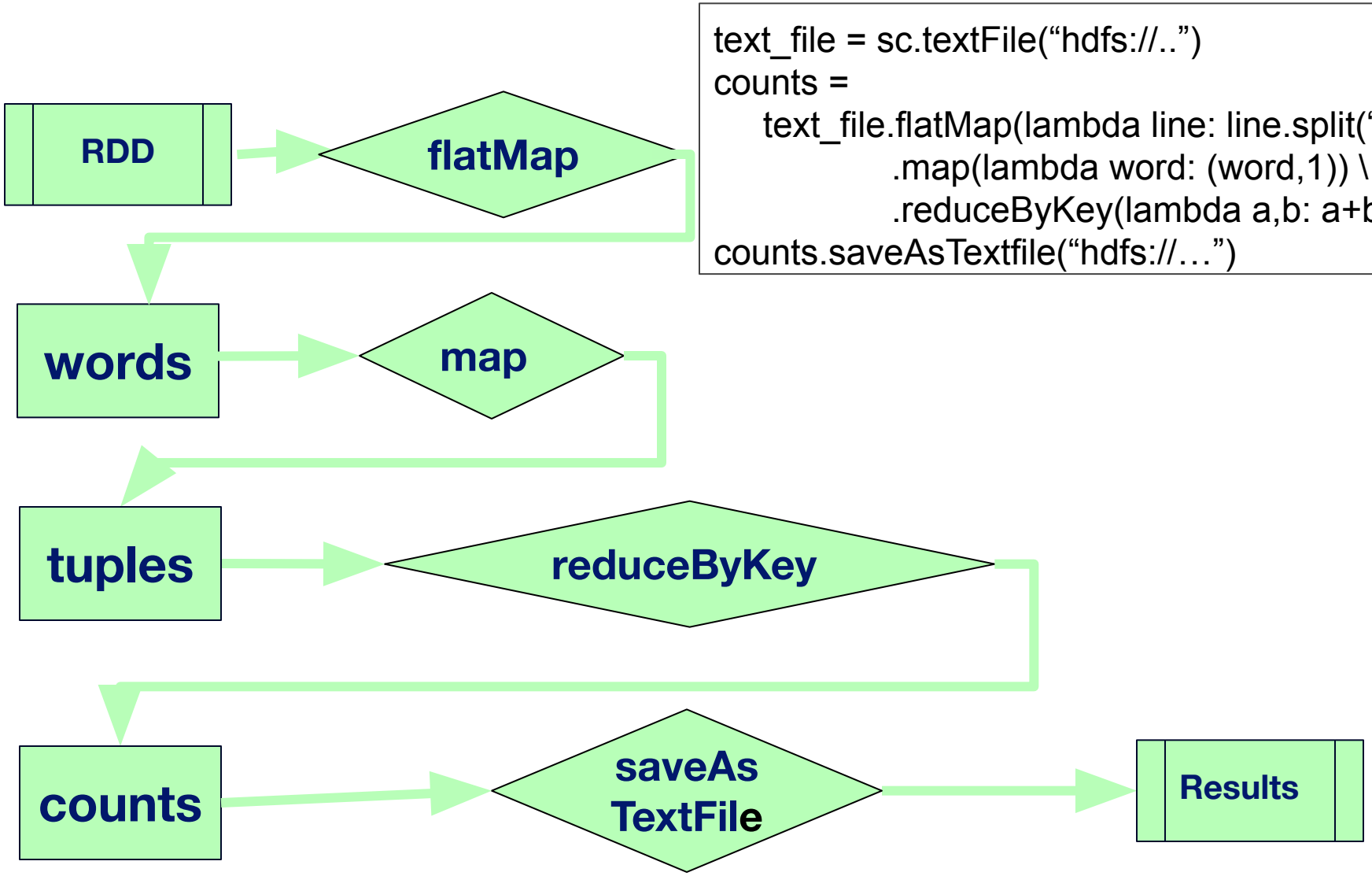
- RDDs can be processed using 2 types of operations
  - **Transformation**: Creates new RDD from existing RDD
  - **Action**: Runs computation(s) on RDD and returns value



LAZY EVALUATION

- Transformations on RDDs have **lazy evaluation**
  - Transformation are not immediately processed
  - Plan of operations is built
- Operations executed when **action** is performed
  - i.e., actions force computation
- Allows for optimizations in generating physical plan
- Example:
  - filtered = strings.filter(strings.value.contains(“Spark”))
    - Nothing is returned
  - filtered.count()
    - ‘filter’ is performed, and count is returned

Word Count (RDD)




```
text_file = sc.textFile("hdfs://..")
counts =
    text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word,1)) \
             .reduceByKey(lambda a,b: a+b)
counts.saveAsTextfile("hdfs://…")
```

DATAFRAMES & DATASETS

- Extensions to RDDs
  - Higher-level abstractions
  - Improved performance
  - Better scalability
- Can convert to/from RDDs and use with RDDs

DATAFRAMES & DATASETS

DataFrame

- Lazy evaluation
- Data organized as collection of Rows
- No static type checking
- APIs in Java, Scala, Python, R

DataSet

- Lazy evaluation
- Data organized as collection of Rows
- Provides static type checking
- APIs in Java and Scala

USING DATAFRAMES

- Spark Session
  - Entry point to Spark engine
  - Note that SparkContext is now **SparkSession**

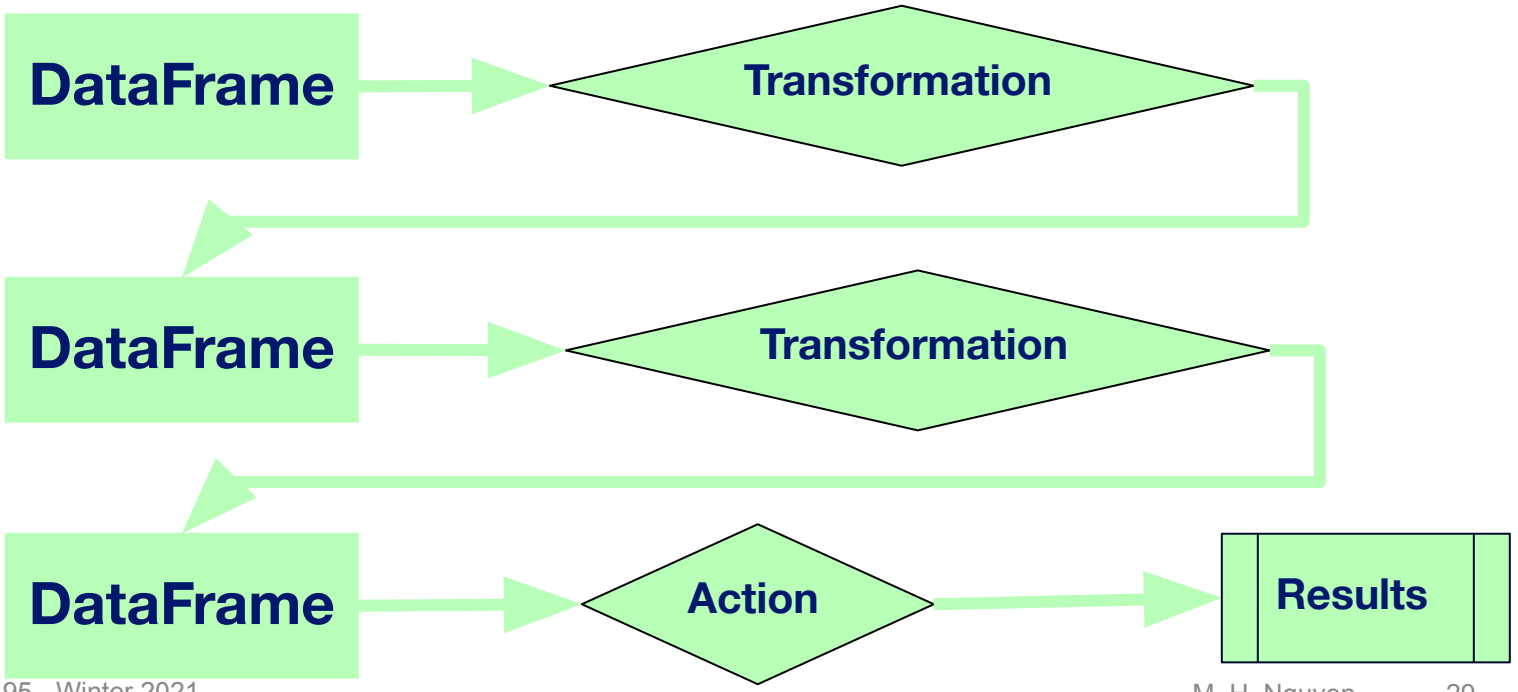
```
from pyspark import SparkSession

conf = SparkSession \
   .builder \
   .appName("DataFrame Example") \
   .config("config.option","config.value")\
   .getOrCreate()

spark =
   SparkSession.builder.config(conf=conf)\
               .getOrCreate()
```

CREATING DATAFRAMES

- Read data from files in local filesystem (text, JSON, etc.)
  - df = spark.read.csv(“data.csv”, inferSchema=”True”, header=”True”)
- Data read in from data store (HDFS, RDBMS, NoSQL, etc.)
  - df = spark.read.csv(“hdfs:///<path>/data.csv”)
- Generate data
  - empl_0 = Row(id=”123”, name=”John”)
  - empl_1 = Row(id=”456”, name=”Mary”)
  - employees = [empl_0, empl_1]
  - df = spark.createDataFrame(employees)
- Created by transforming another DataFrame
  - filter_df = df.filter(col(“name”)==”Mary”))

DATAFRAME TRANSFORMATIONS & ACTIONS

- Similar to RDDs, DataFrames can be processed using transformations and actions
- Transformations on DataFrames also have lazy evaluation
- Operations executed when action is performed



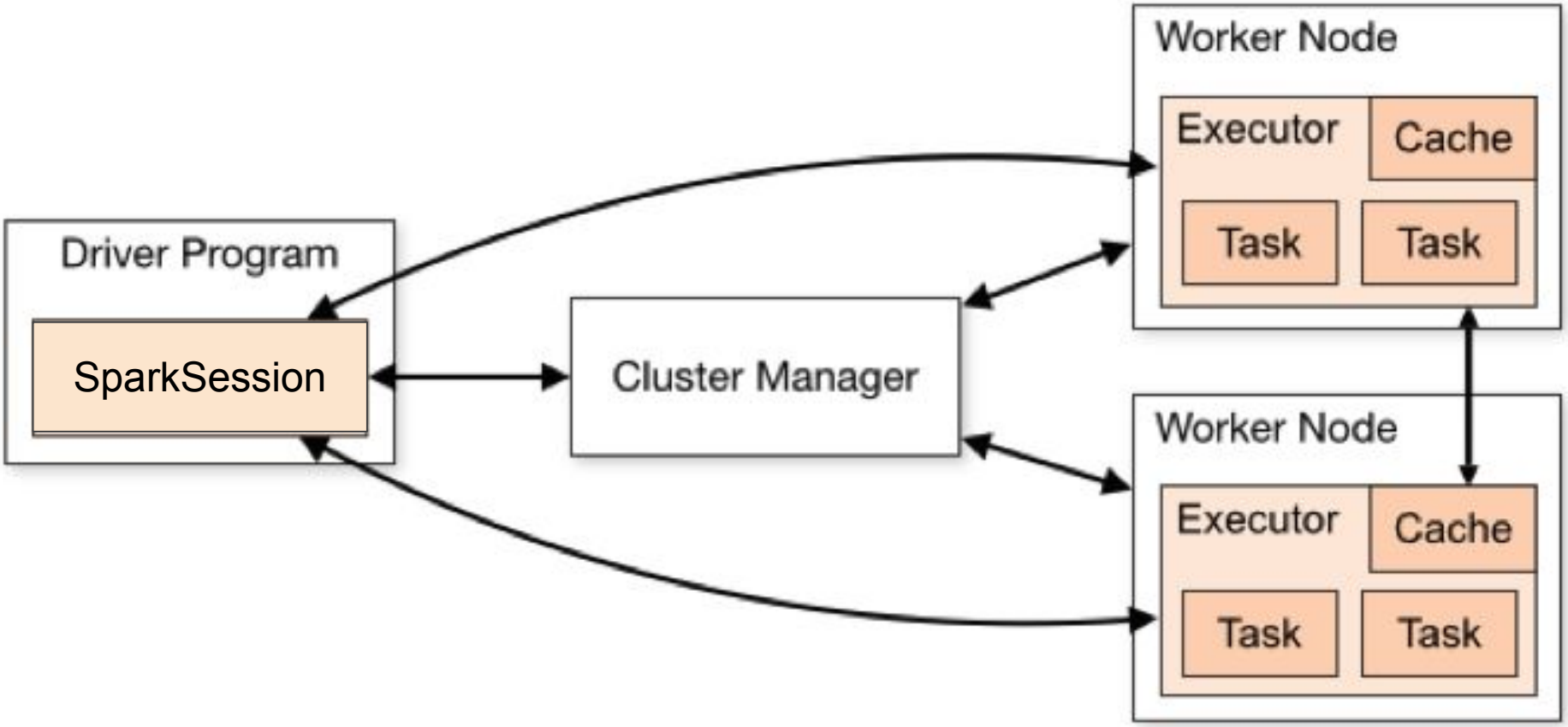
DATA PERSISTENCE

- Persist data through caching
  - Data is stored in memory to avoid re-computing
- Can specify different storage levels
  - In memory, on disk, serialized in memory, etc.
- Examples
  - df.cache() – MEMORY_ONLY
  - df.persist(MEMORY_ONLY_SER) – Serialized in memory
  - df.unpersist() – Remove from cache

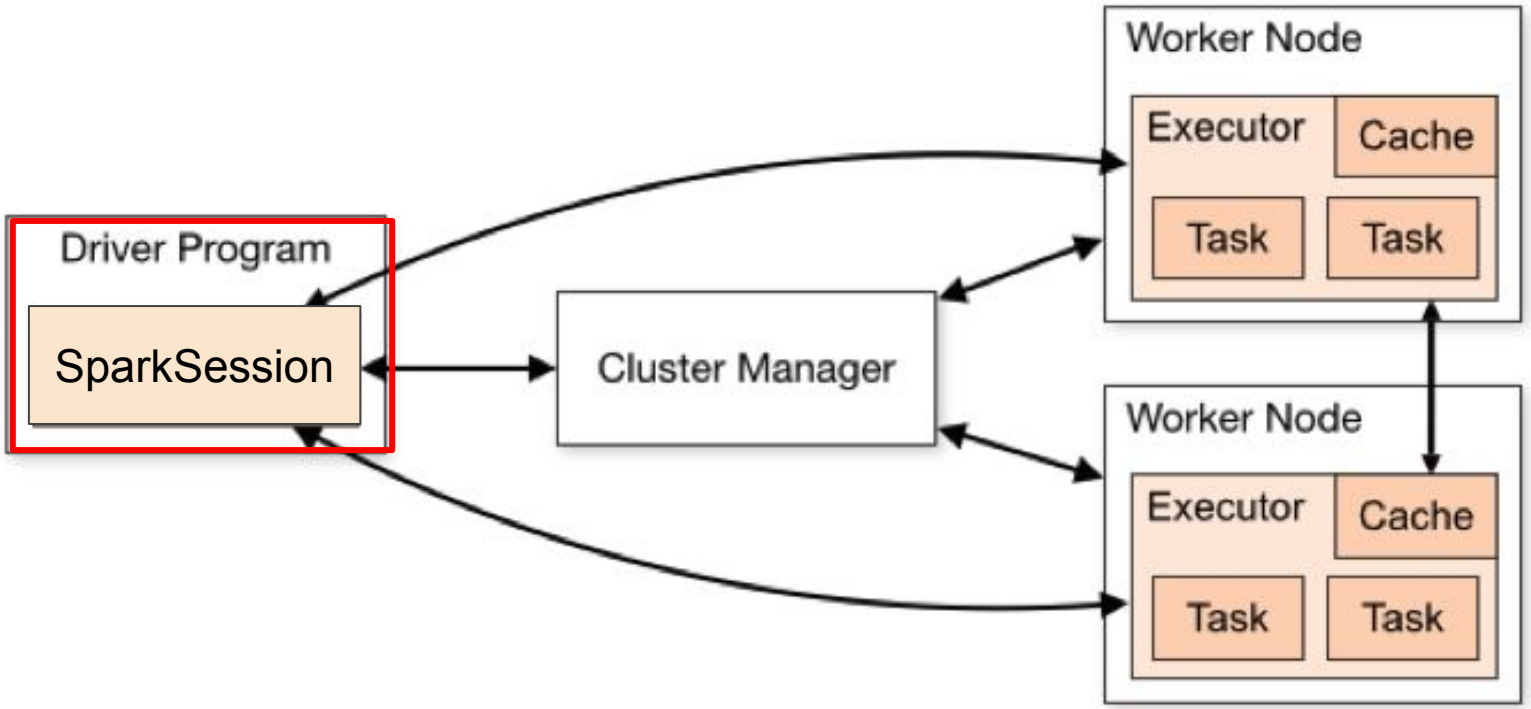
SPARK PROGRAM STRUCTURE

- Start Spark session
- Create distributed dataset
- Apply transformations
- Perform actions
- Stop Spark session
  - spark.stop()

SPARK ARCHITECTURE

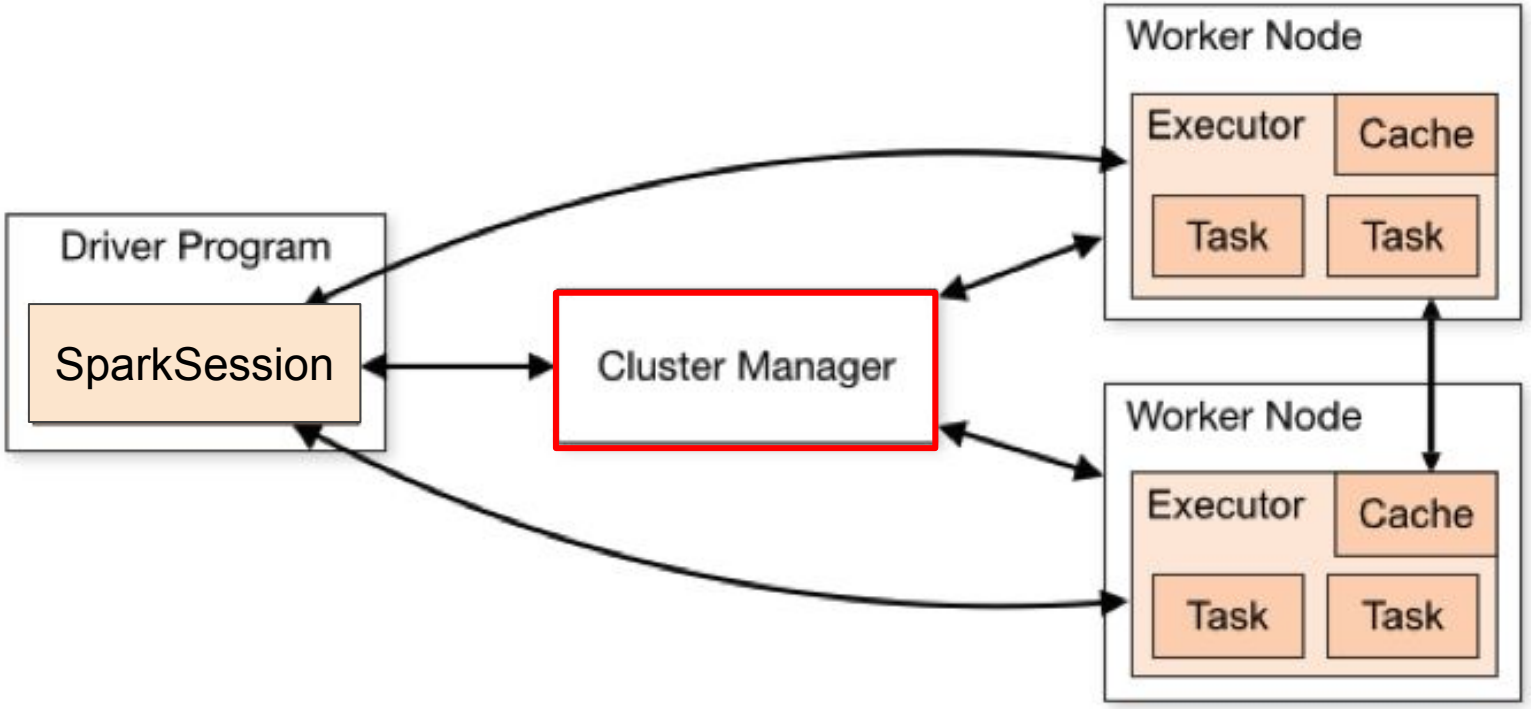


SPARK ARCHITECTURE



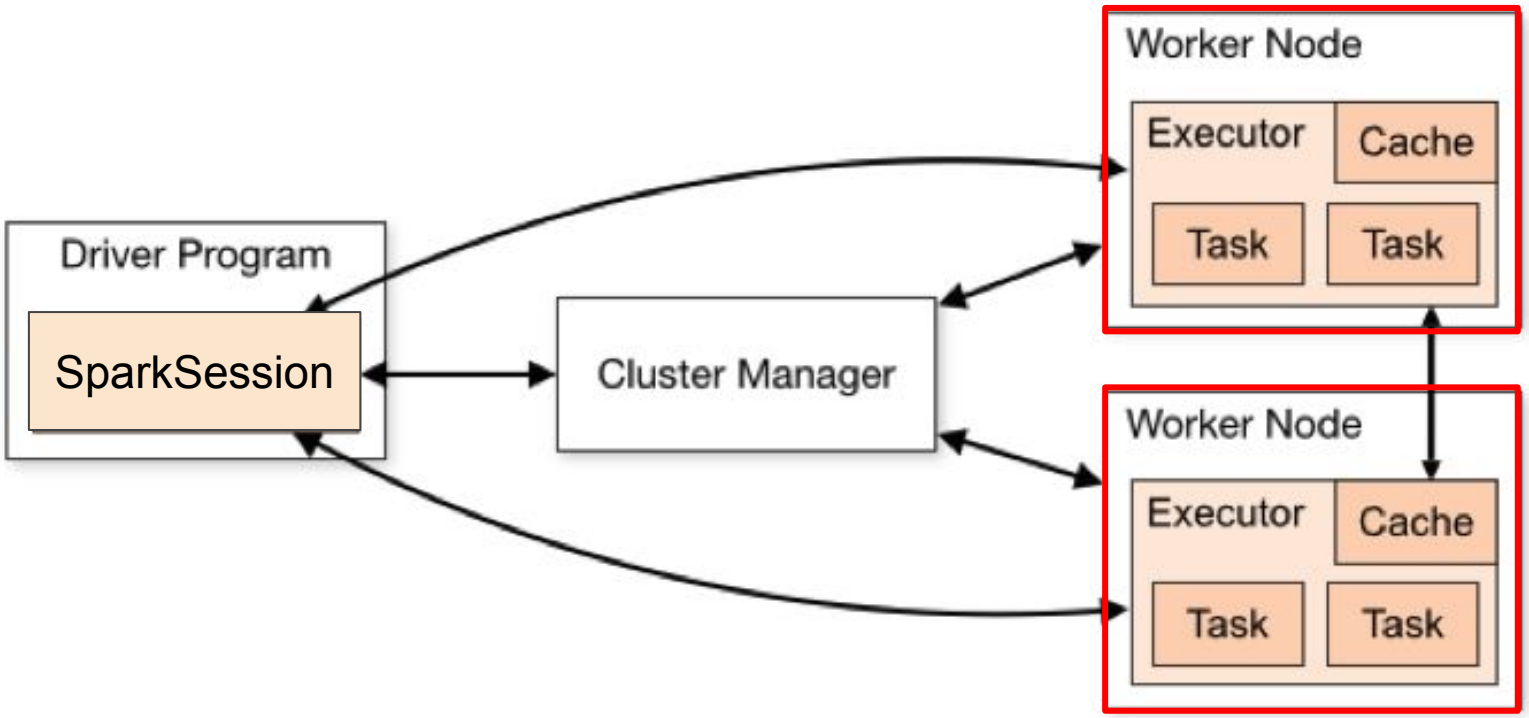
- Client submits Spark application
- Driver launches main() in client application
- Driver creates SparkSession object
- Driver distributes and schedules work across workers

SPARK ARCHITECTURE



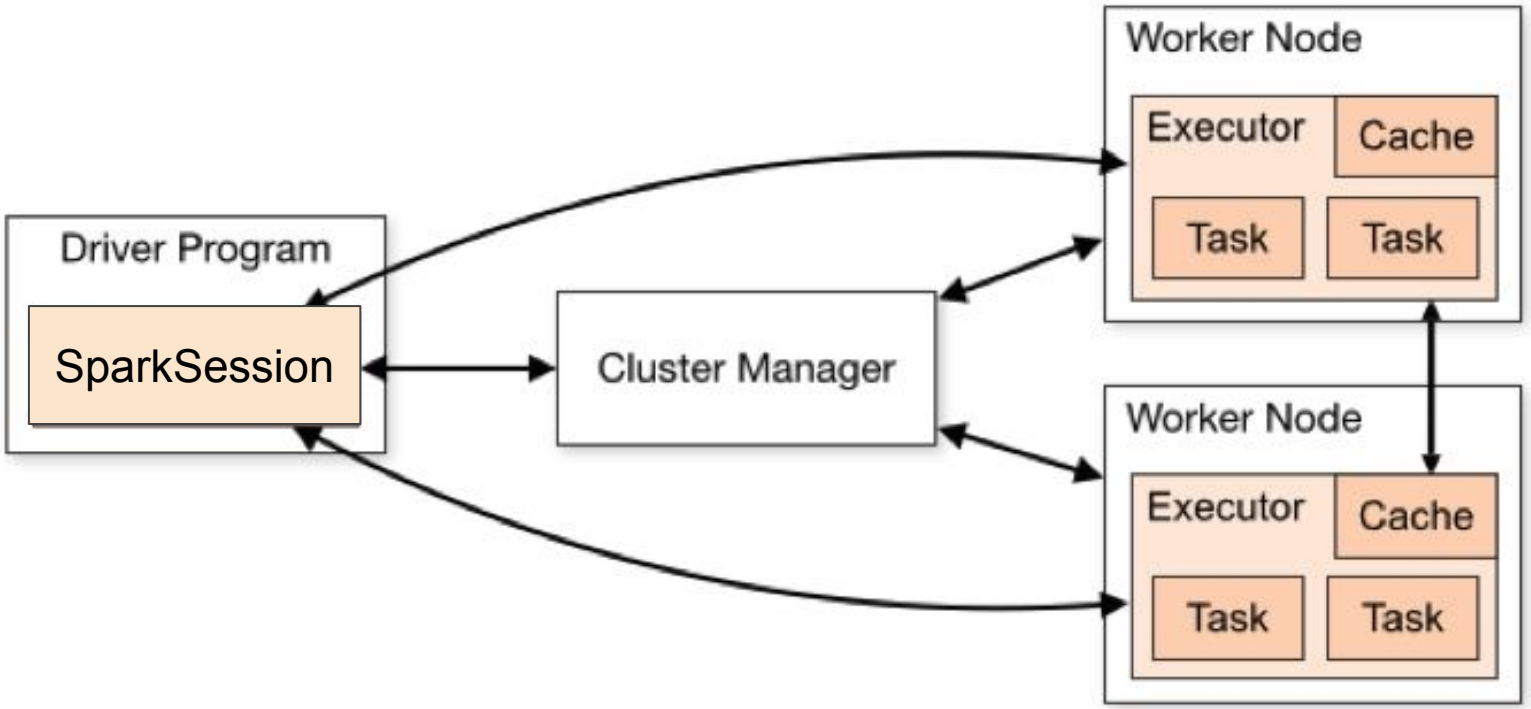
- Driver requests resources from cluster manager to launch executors
- Cluster manager allocates resources to run application
- Cluster managers in Spark
  - Standalone, Hadoop YARN, Apache Mesos, Kubernetes

SPARK ARCHITECTURE



- Driver sends work to executors
- Each executor processes work assigned to it, and communicates with driver regarding resources & results

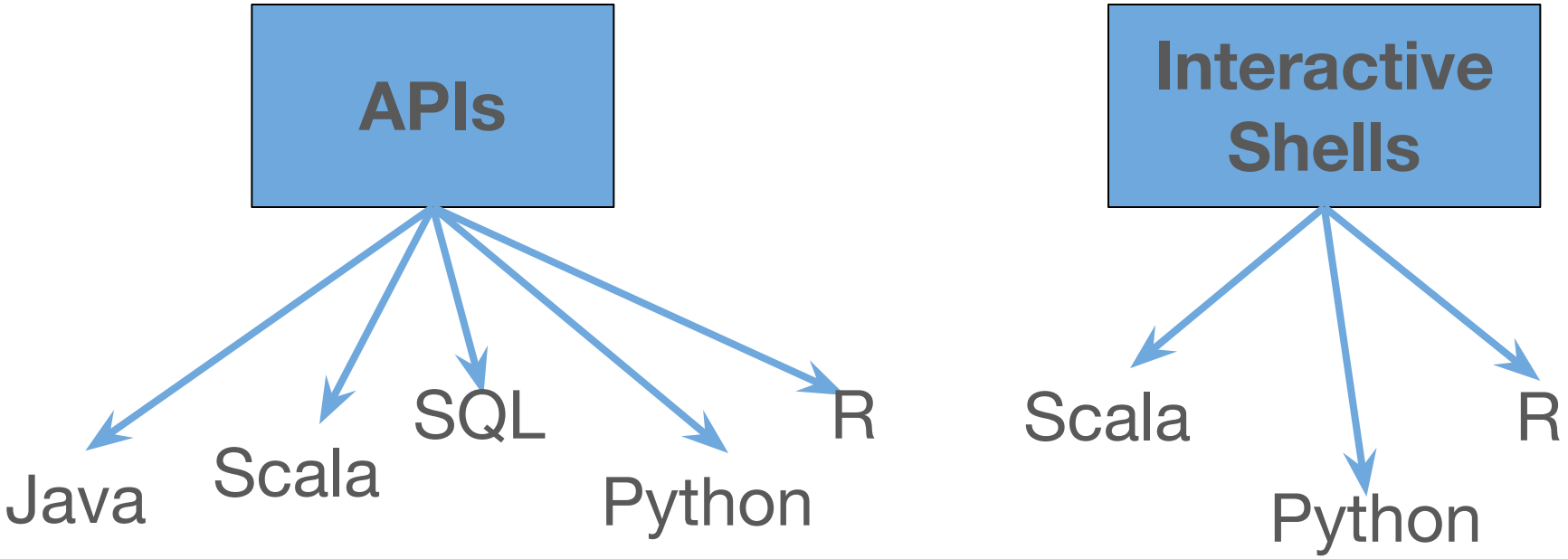
SPARK ARCHITECTURE



- Driver returns results to client
- Driver terminates executors, release resources via cluster manager, and stops SparkSession

SPARK INTERFACE

Goals: speed, **ease of use**, generality, unified platform



RDD WORDCOUNT EXAMPLE IN SPARK

Spark RDD API available in Python, Scala, Java, and R

```python
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

```scala
val textFile = sc.textFile("hdfs://...")
val counts = textFile.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

```java
JavaRDD<String> textFile = sc.textFile("hdfs://...");
JavaPairRDD<String, Integer> counts = textFile
    .flatMap(s -> Arrays.asList(s.split(" ")).iterator())
    .mapToPair(word -> new Tuple2<>(word, 1))
    .reduceByKey((a, b) -> a + b);
counts.saveAsTextFile("hdfs://...");
```

BIG DATA & DISTRIBUTED PROCESSING

- Big Data Overview
- Scalable Systems
- Hadoop
- Spark
  - History
  - RDDs
  - DataFrames
  - Spark Architecture
  - Spark API
- PySpark Exercise
- Assignment

QUIZ

QUIZ

What is the Spark session?

A. Entry point to the Spark engine
B. The first object to be created in a Spark program
C. Object that is used to create RDDs and DataFrames and access Spark functionality
D. A & C
E. A, B & C

QUIZ

Which of the following is/are true about RDDs in Spark?

A. RDD is a distributed collection of data elements
B. RDD partitions can be read from and written to via the SparkSession object
C. RDD provides fault tolerance by tracking lineage information
D. A & C
E. A, B, & C

QUIZ

Spark's Dataset API is available in which of the following programming languages?

A. Scala
B. C++
C. Java
D. None of the above
E. A & C

QUIZ

Which of the following statements is true about lazy evaluation in Spark?

A. Lazy evaluation allows for an efficient physical plan of operations to be created
B. Lazy evaluation allows for transformations to be executed after actions
C. Lazy evaluation is performed on RDDs but not DataFrames
D. Lazy evaluation creates intermediate results for transformations

QUIZ

What is the relationship between RDDs, DataFrames, and DataSets?

A. RDDs are higher-level extensions to DataFrames and DataSets
B. DataFrames and DataSets are built on top of RDDs
C. DataFrames provide static type checking, and DataSets do not
D. RDDs are lazily evaluated, while DataFrames and DataSets are not
E. RDDs cannot be mixed with DataFrames and DataSets

BIG DATA & DISTRIBUTED PROCESSING

- Big Data Overview
- Scalable Systems
- Hadoop
- Spark
- PySpark Exercise
- Assignment