Advanced HPC-CI Webinar
Scalable Machine Learning - Spark

Mai H. Nguyen, Ph.D.

2025-01

SDSC SAN DIEGO SUPERCOMPUTER CENTER

UC San Diego

Spark Introduction

Spark Topics

- **Spark**
  - History
  - RDDs
  - DataFrames
  - Spark Design Goals
  - Spark API
  - Spark Core & Libraries
- **Spark Demo**
  - Scaling
  - Cluster Analysis

SPARK

- **Computing platform for scalable computing**
  - Designed for big data workloads
  - Built-in parallelism & fault-tolerance on commodity cluster
  - Provides interactive querying, iterative analytics, streaming processing, along with batch processing
  - 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

SPARK

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

RESILIENT DISTRIBUTED DATASETS (RDDs)

- Spark central concept
  - Abstraction of data as distributed collection of objects

- Resilient Distributed Datasets (RDDs)
  - Data abstraction
  - Programming construct for storing and organizing data
  - Spark uses RDDs to distribute data and computations across nodes in cluster

RDD

- 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

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

DataSet

- Lazy evaluation
- Immutable
- 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**

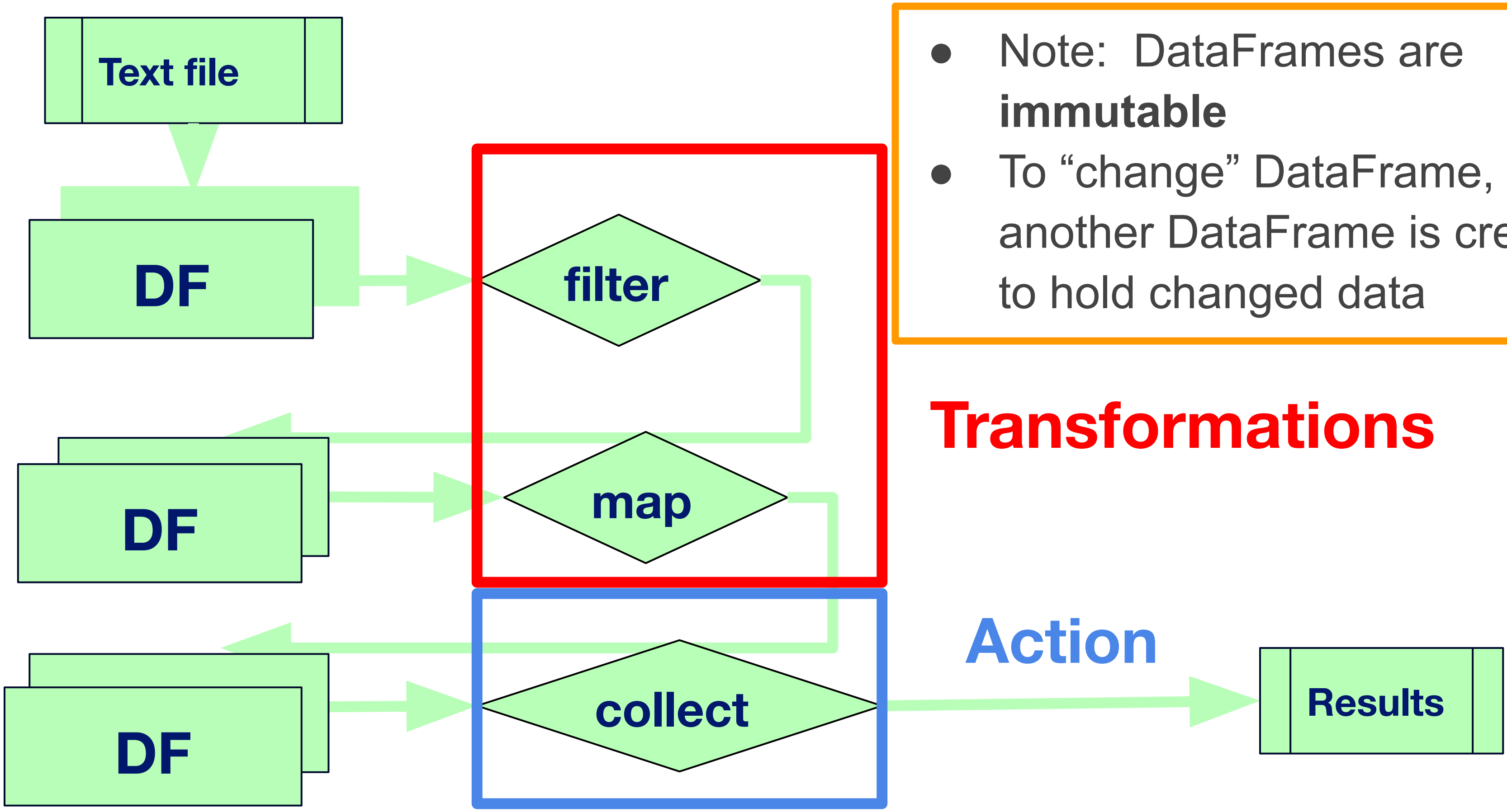
```
from pyspark import SparkSession, SparkConf

conf = SparkConf \
   .setAll \
   ([("spark.app.name","DataFrame Example") \
   ("config.option","config.value")])

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”, 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”))

PROCESSING DATAFRAMES

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

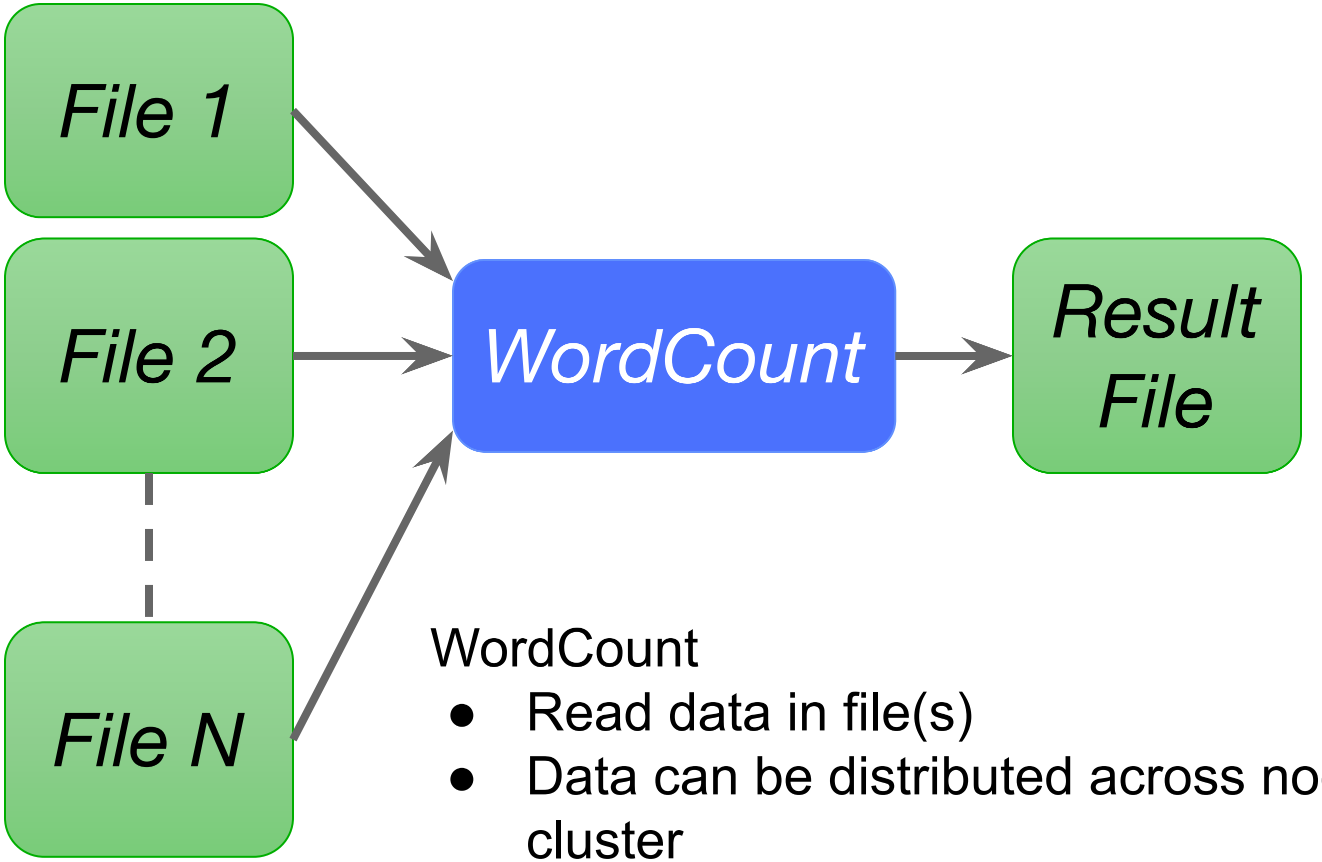


- Note: DataFrames are **immutable**
- To "change" DataFrame, another DataFrame is created to hold changed data

LAZY EVALUATION

- Transformations on DataFrames have **lazy evaluation**
  - Transformations 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

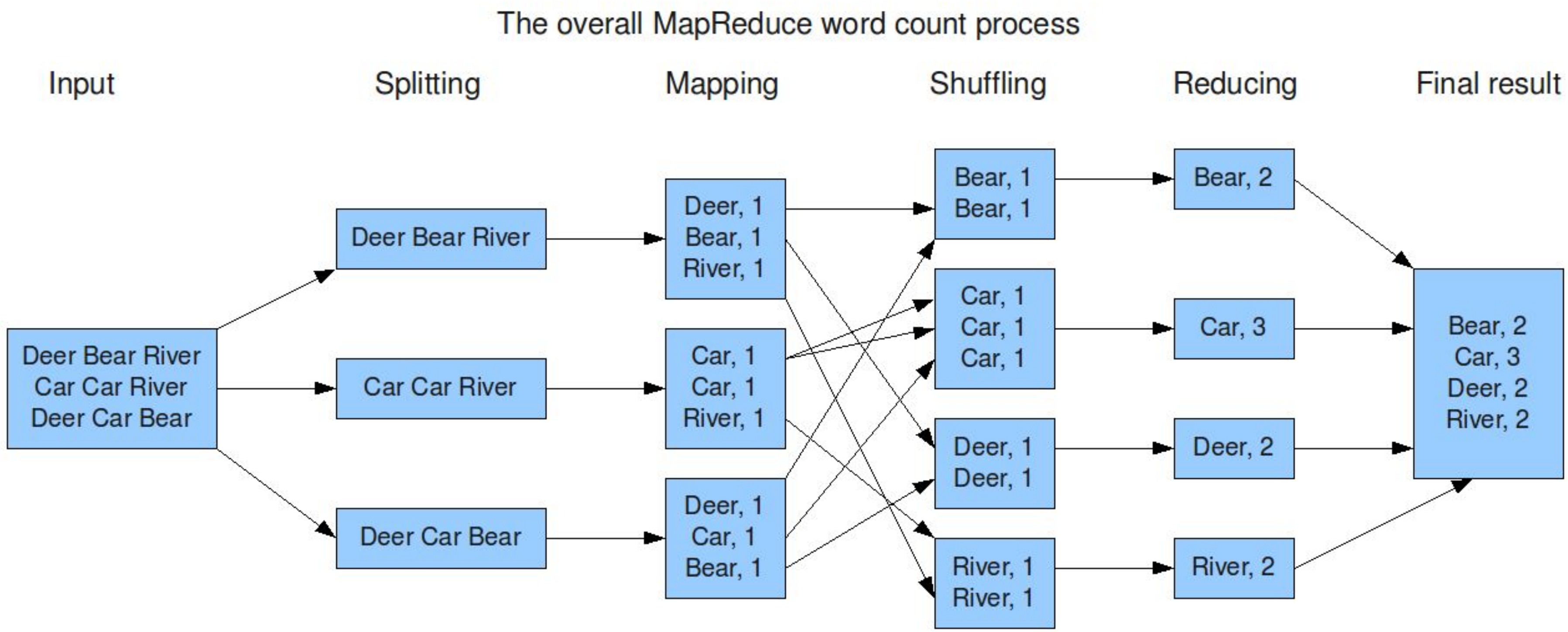
WordCount



WordCount

- Read data in file(s)
- Data can be distributed across nodes in cluster
- Count number of occurrences of each word

WordCount



https://www.todaysoftmag.com/article/1358/hadoop-mapreduce-deep-diving-and-tuning

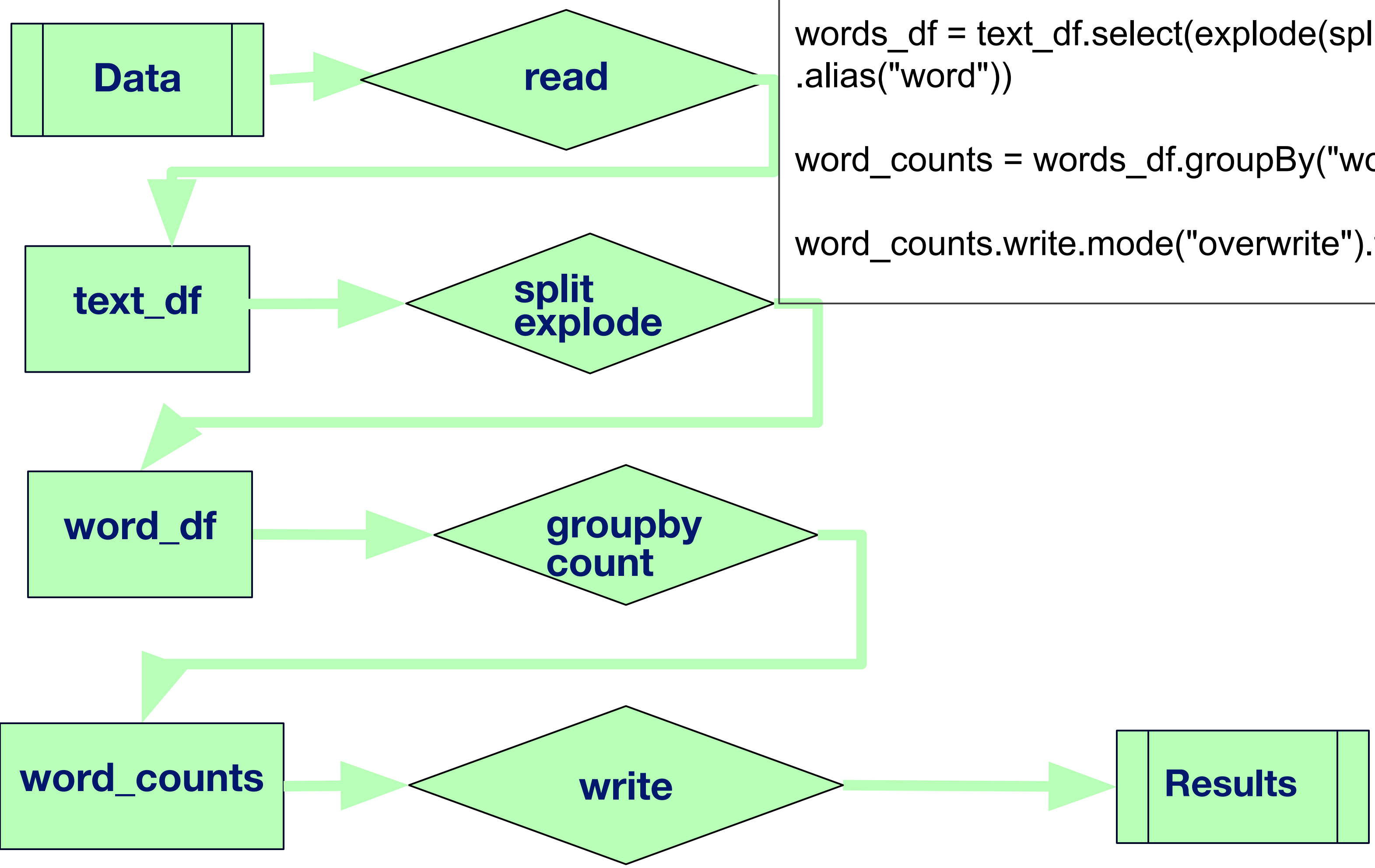
Data is split into partitions

Map generates key-value pairs

Pairs with same key moved to same partition

Reduce sums values for each key

WordCount

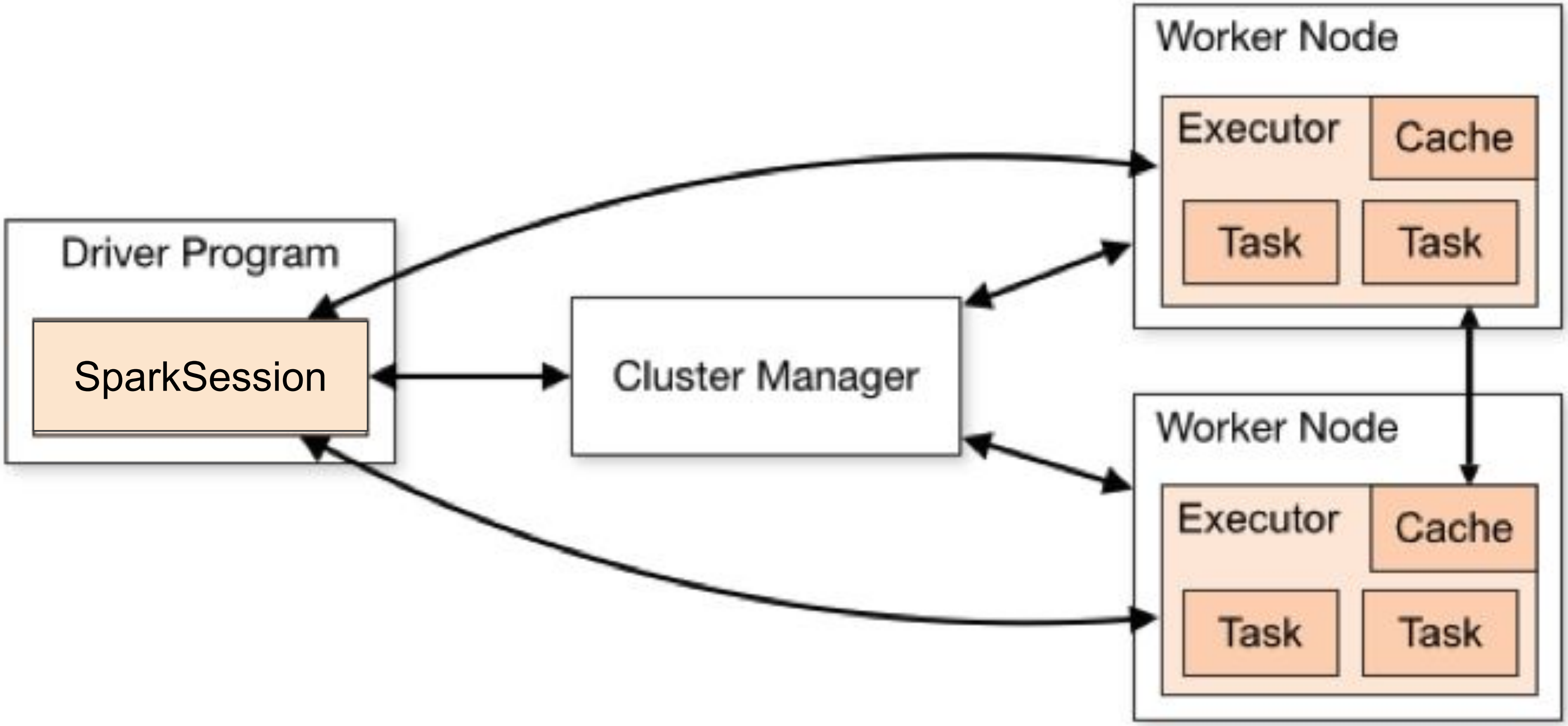



```
text_df = spark.read.text("hdfs://...")

words_df = text_df.select(explode(split(col("value"), " "))
.alias("word"))

word_counts = words_df.groupBy("word").count()

word_counts.write.mode("overwrite").text("hdfs://...")
```

SPARK PROGRAM STRUCTURE

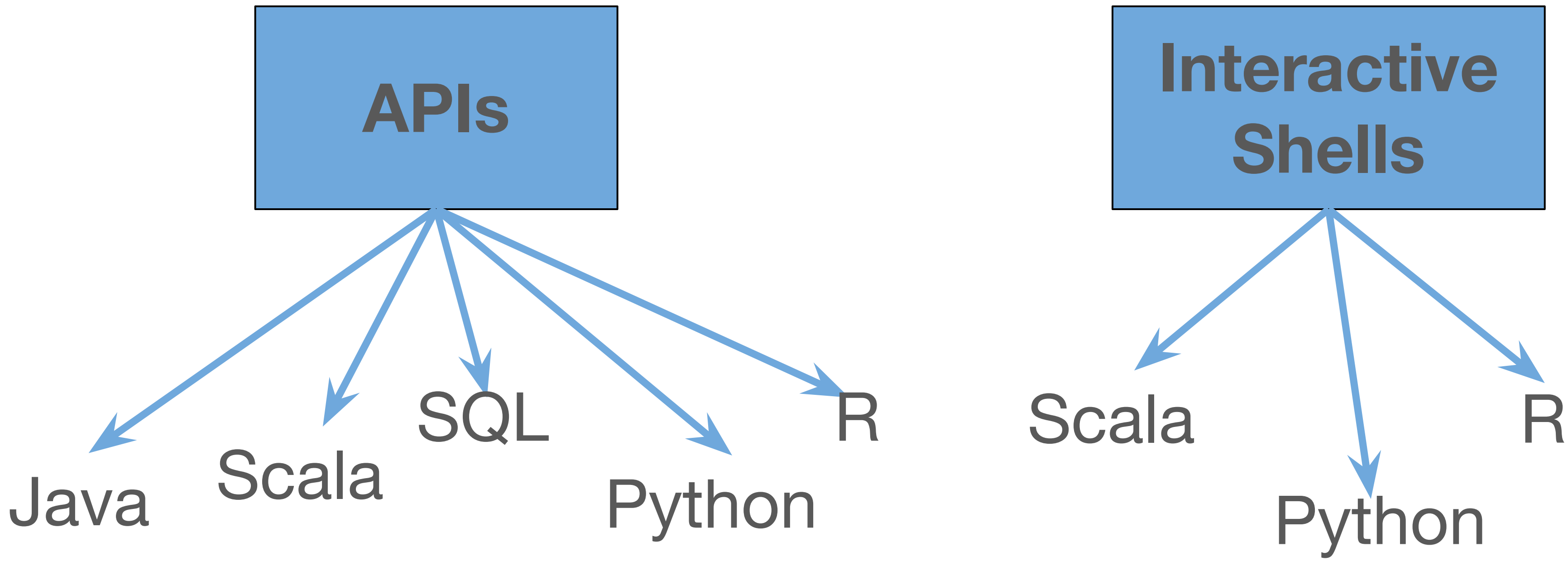
- **Start Spark session**
  - spark = SparkSession.builder.config(conf=conf).getOrCreate()
- **Create distributed dataset**
  - df = spark.read.csv(“data.csv”,header=”True”)
- **Apply transformations**
  - new_df = df.filter(col(“dept”) == “Sales”)
- **Perform actions**
  - df.collect()
- **Stop Spark session**
  - spark.stop()

SPARK ARCHITECTURE



SPARK INTERFACE

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



WORDCOUNT EXAMPLE IN SPARK

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

PySpark

```
text_df = spark.read.text("hdfs://...")
words_df = text_df.select(explode(split(col("value"), " ")).alias("word"))
word_counts = words_df.groupBy("word").count()
word_counts.write.mode("overwrite").text("hdfs://...")
```

SparkR

```
textDF <- read.text(spark, "hdfs://...")
wordsDF <- selectExpr(textDF, "explode(split(value, ' ')) as word")
wordCounts <- count(groupBy(wordsDF, "word"))
write.df(wordCounts, "hdfs://...", "text", mode = "overwrite")
```

Scala

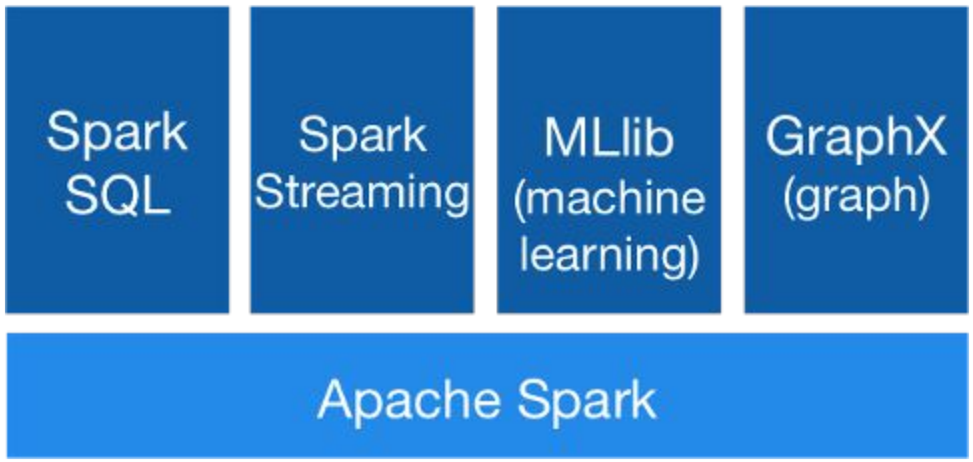
```
val textDF = spark.read.text("hdfs://...")
val wordsDF = textDF.select(explode(split(col("value"), "”)).alias("word"))
val wordCounts = wordsDF.groupBy("word").count()
wordCounts.write.mode("overwrite").text("hdfs://...")
```

SPARK - GENERALITY

- Goals: speed, ease of use, **generality,** unified platform
- Support for several data sources
  - Local file systems, HDFS, RDBMSs, MongoDB, Kafka, AWS S3, etc.
- Can run on various platforms
  - Hadoop, Kubernetes, cloud, standalone
- Support for multiple workloads
  - batch, streaming
  - machine learning, SQL, graph processing

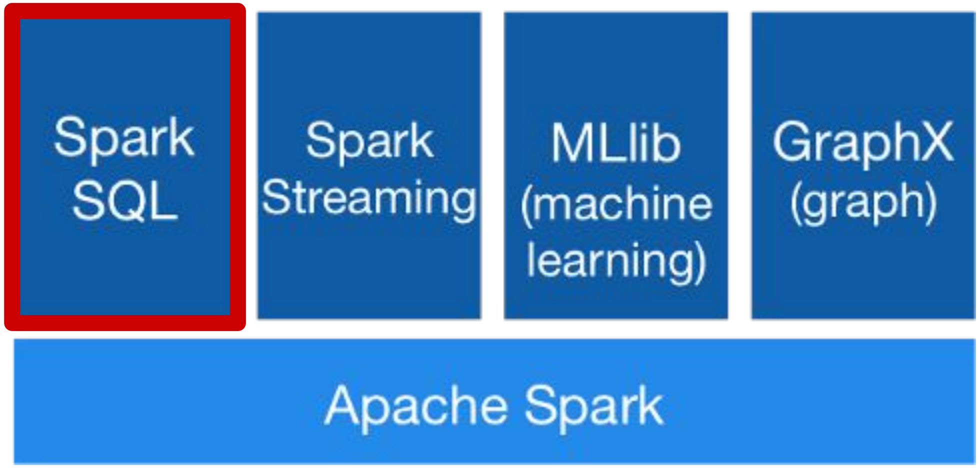
SPARK - UNIFIED PLATFORM

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



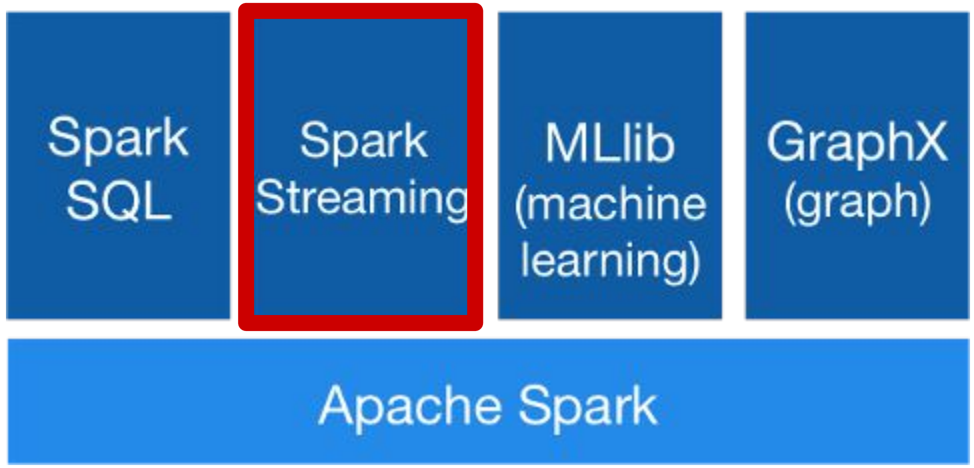
- Provides unified platform for various analytics processing
- **Spark engine** provides core capabilities for scalable processing
- **Spark libraries** provide additional higher-level functionality for diverse workloads

SPARK SQL



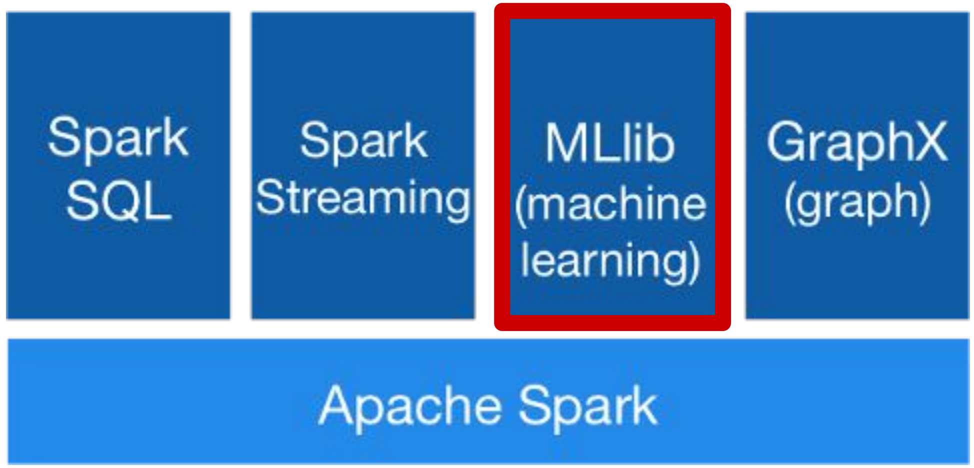
- **Structured Data Processing**
  - Provides support for SQL and query processing
  - Has APIs for SQL, Scala, Java, Python, and R
  - Generated underlying code is identical

SPARK STREAMING



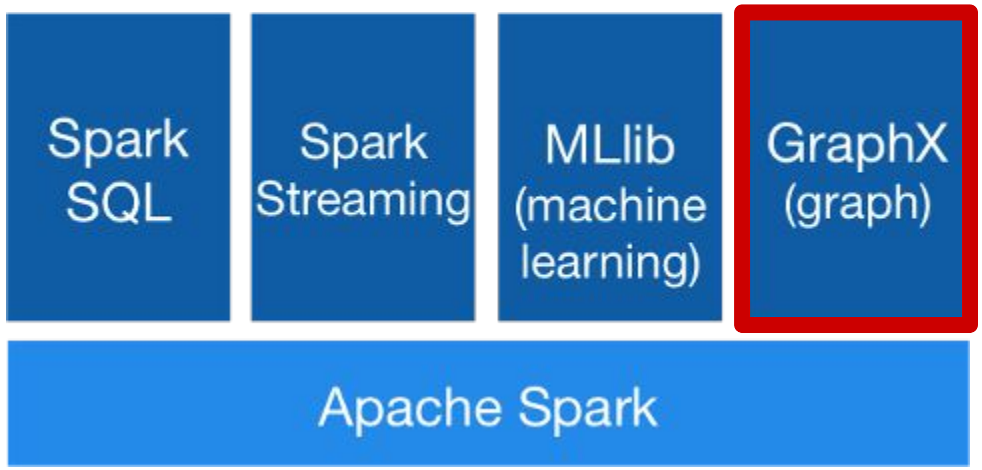
- **Streaming Data Processing**
  - Scalable processing for real-time analytics
  - Structured streaming
    - Data stream is divided into micro-batches of data
    - Same operations for static data can be used
  - Has APIs for Scala, Java, and Python

SPARK MLLIB



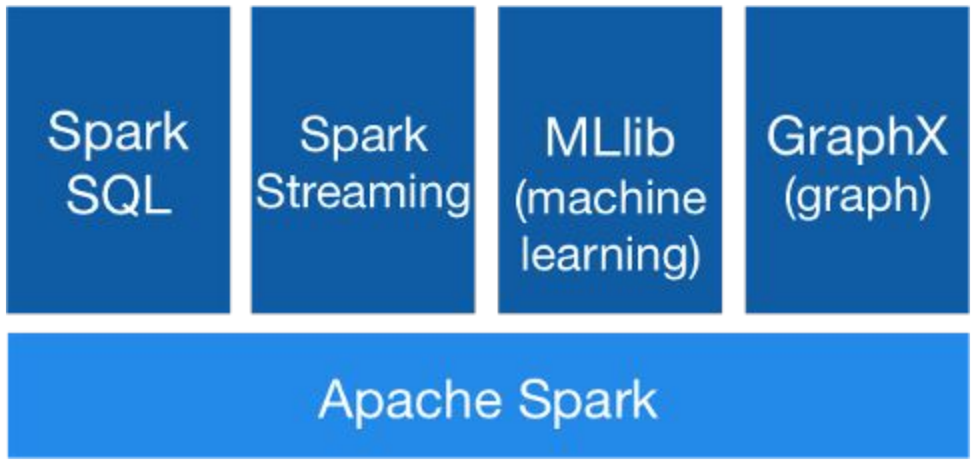
- **Machine Learning**
  - Scalable machine learning library
  - Scalable implementations of machine learning algorithms and utilities
  - Has APIs for Scala, Java, Python, and R

SPARK GRAPHX / GRAPHFRAMES



- **Graph Computation**
  - Scalable graph processing
  - Special structures for storing vertex and edge information & operations for manipulating graphs
  - GraphX (RDD-based) & GraphFrames (DF-based)
  - Has APIs in Scala, Java, Python (GraphFrames)

SPARK



Unified engine for large-scale data analytics

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

Spark Resources

- PySpark SQL Basics Cheat Sheet
  - PDF
- Spark Main Page
  - https://spark.apache.org/
- Spark Overview
  - https://spark.apache.org/docs/latest/index.html
- Spark Examples
  - https://spark.apache.org/examples.html
- Spark SQL, DataFrames and DataSets Programming Guide
  - https://spark.apache.org/docs/latest/sql-programming-guide.html
- Spark MLlib Programming Guide
  - https://spark.apache.org/docs/latest/ml-guide.html
- PySpark API Documentation
  - https://spark.apache.org/docs/latest/api/python/index.html

Spark Demo

Mai H. Nguyen, Ph.D.

Server Setup for PySpark - Command Line

- **Login to Expanse**
  - Open terminal window on local machine
  - ssh login.expanse.sdsc.edu -l <account>
- **In terminal window**
  - export PATH="/cm/shared/apps/sdsc/galyleo:${PATH}"
  - jupyter-shared-spark
    - Alias for: galyleo launch --account ${HPC_ACCOUNT} --reservation ${HPC_RESERVATION_CPU} --partition shared --cpus 4 --memory 16 --time-limit 02:00:00 --env-modules singularitypro --sif /cm/shared/apps/containers/singularity/spark/spark-latest.sif --bind /expanse,/scratch,/cm --quiet
- **To check queue**
  - squeue -u $USER

PySpark Scaling Hands-On

- **Data**
  - BookReviews_5M.txt
    - Source : https://jmcauley.ucsd.edu/data/amazon/
- **Notebook**
  - pyspark_demo.ipynb
- **To do**
  - Change number of cores: 1, 2, 4
  - Note difference in execution times
  - Run each configuration 3 times

SPARK SESSION

Use * to use all available cores, or integer value to specify number of cores to use

```
import pyspark
from pyspark.sql import SparkSession,SparkConf

conf = SparkConf().setAll([
         ('spark.master',    'local[*]'),
         ('spark.app.name', 'PySpark Demo')])

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

Configuration parameters for Spark session

Get existing Spark session or create new one

GETTING EXECUTION TIMES

- In notebook, execution time is printed out in cell before Spark session is stopped (next to last cell)
- Need to restart the kernel and run all cells without stopping to get accurate execution time:
  - Run -> Restart Kernel and Run All Cells
- Find mean and standard deviation of execution times over 3 runs for
  - 1 core, 2 cores, and 4 cores

```
import pyspark
from pyspark.sql import SparkSession

conf = pyspark.SparkConf().setAll([
         ('spark.master',   'local[2]'),
         ('spark.app.name', 'PySpark Demo)])
spark = SparkSession.builder.config(conf=conf).getOrCreate()
```

Specify number of cores.
'*' uses all available cores

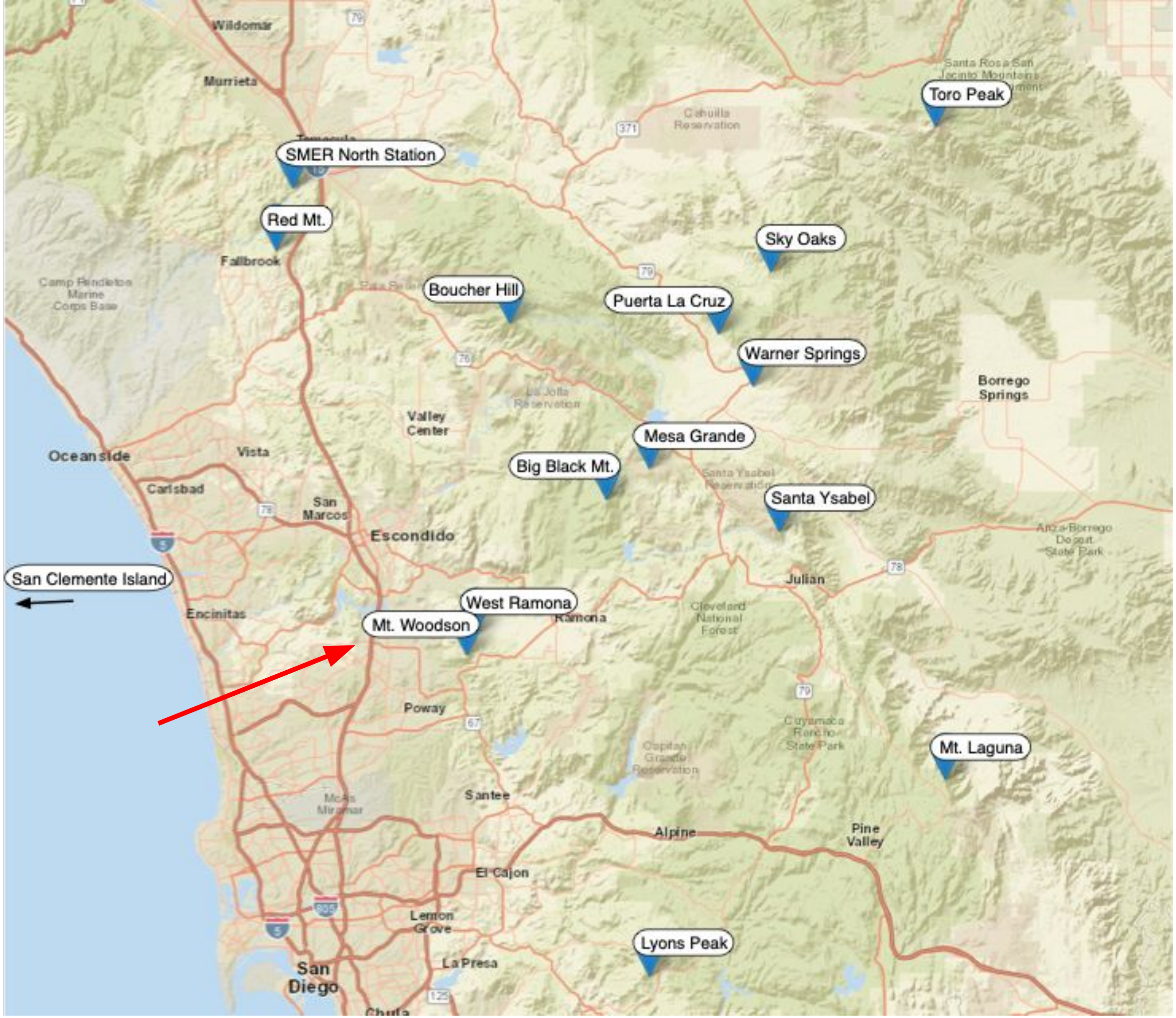
PySpark Cluster Analysis Hands-On

- **Data**
  - Weather station measurements
- **Task**
  - Perform cluster analysis to identify different weather patterns
- **Approach**
  - Spark k-means
- **Notebooks**
  - pyspark-clustering.ipynb # Starter notebook
  - pyspark-clustering-wOutput.ipynb # Has cell outputs
  - pyspark-clustering-soln.ipynb # Solution

Dataset Description

- **Measurements from weather station on Mt. Woodson, San Diego**
- **Air temperature, humidity, wind speed, wind direction, etc.**
- **Three years of data: Sep. 2011 - Sep. 2014**
  - minute_weather.csv: measurement every minute
- **Source**
  - http://hpwren.ucsd.edu

Map of HPWREN Weather Stations



Clustering Hands-On Overview

- **Setup**
  - Start Spark
  - Load modules
- **Load data**
  - Specify schema
  - Read in data from "minute_weather.csv"
- **Explore data**
  - Look at schema, number of rows, summary statistics
- **Prepare data**
  - Drop nulls
  - Create feature vector
- **Perform k-means cluster analysis**
  - Use elbow plot to determine k
  - Build k-means model
- **Evaluate clusters**
  - Plot cluster profiles
- **Stop Spark session**

Resources

- **Spark**
  - https://spark.apache.org/
- **PySpark API**
  - https://spark.apache.org/docs/latest/api/python/index.html
- **Spark DataFrame**
  - https://spark.apache.org/docs/latest/sql-programming-guide.html
- **MLlib**
  - https://spark.apache.org/mllib/
- **User's Guide**
  - https://spark.apache.org/docs/latest/api/python/user_guide/pandas_on_spark/index.html