

# BIGDATA 210: Introduction to Data Engineering

Autumn 2017

Module 4: Data Processing Using Spark Part II

Jason Kolter

*[jkolter@uw.edu](mailto:jkolter@uw.edu)*

# Week 4 Agenda

- Outstanding VM/Docker Issues
- Assignment 2 Review
- Assignment 3 Discussion
- Programming in Spark Part II
  - Scaling Spark Applications
    - Understanding partitioning and shuffling
    - Persistence and shared variables
  - Programming in Spark
    - Spark SQL/DataFrames/Datasets

# Docker/VM Issues

- Announcement made re: MS Azure Ubuntu kernel bug
  - [https://canvas.uw.edu/courses/1166750/discussion\\_topics/4003691](https://canvas.uw.edu/courses/1166750/discussion_topics/4003691)
- “Waiting for Ambari agent to connect.....”
  - Did you change the Ambari “admin” user pw?
    - ambari-admin-password-reset
    - Change password on Web UI after logged in
  - Run “ambari-admin-password-reset” in Docker container (not VM) or through Web UI
    - Set password back to “4o12t0n”

# Assignment 2 Review

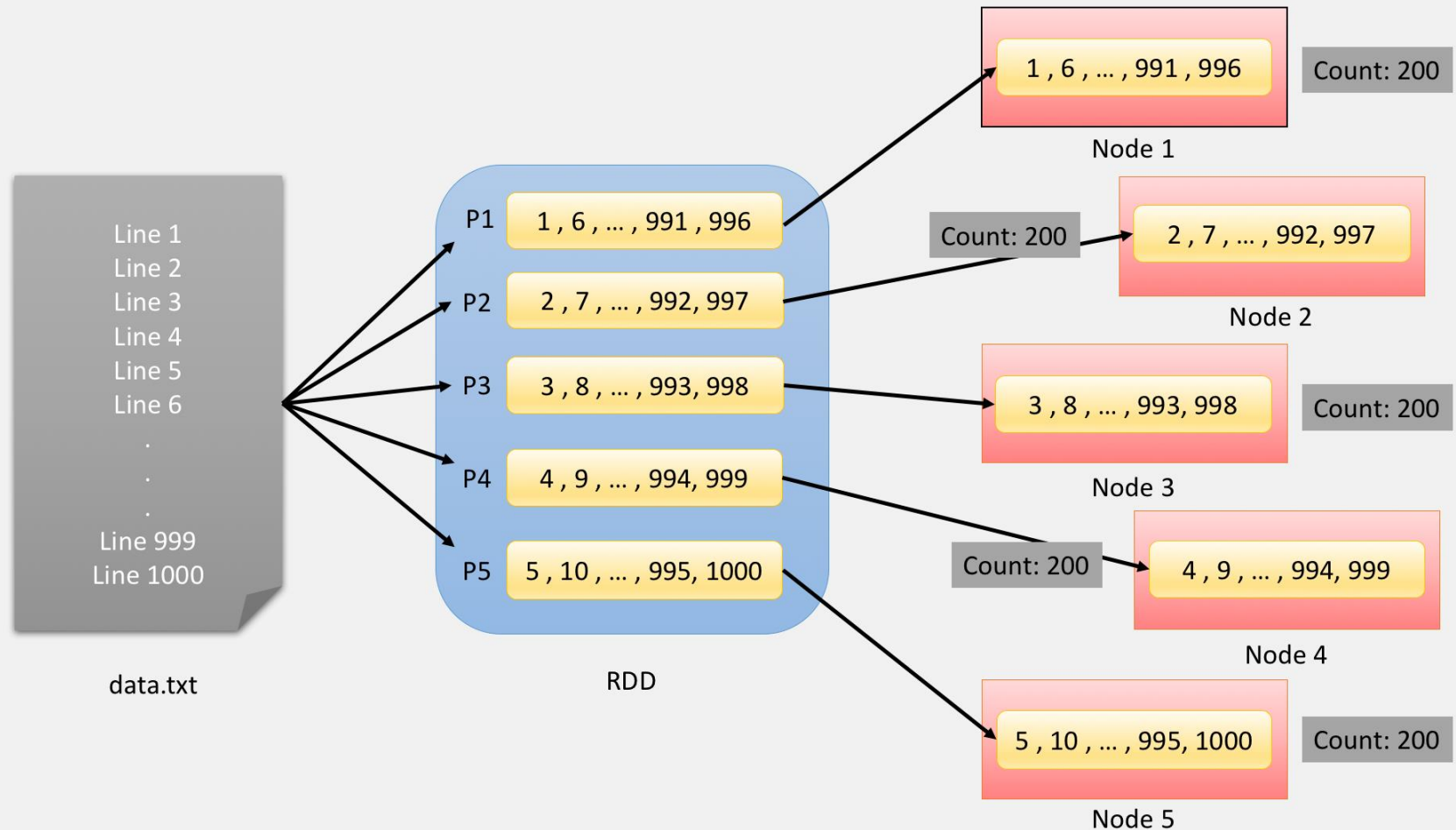
- Homework questions
- Killing YARN applications

# Assignment 3 Discussion

- Use RDD API !!!

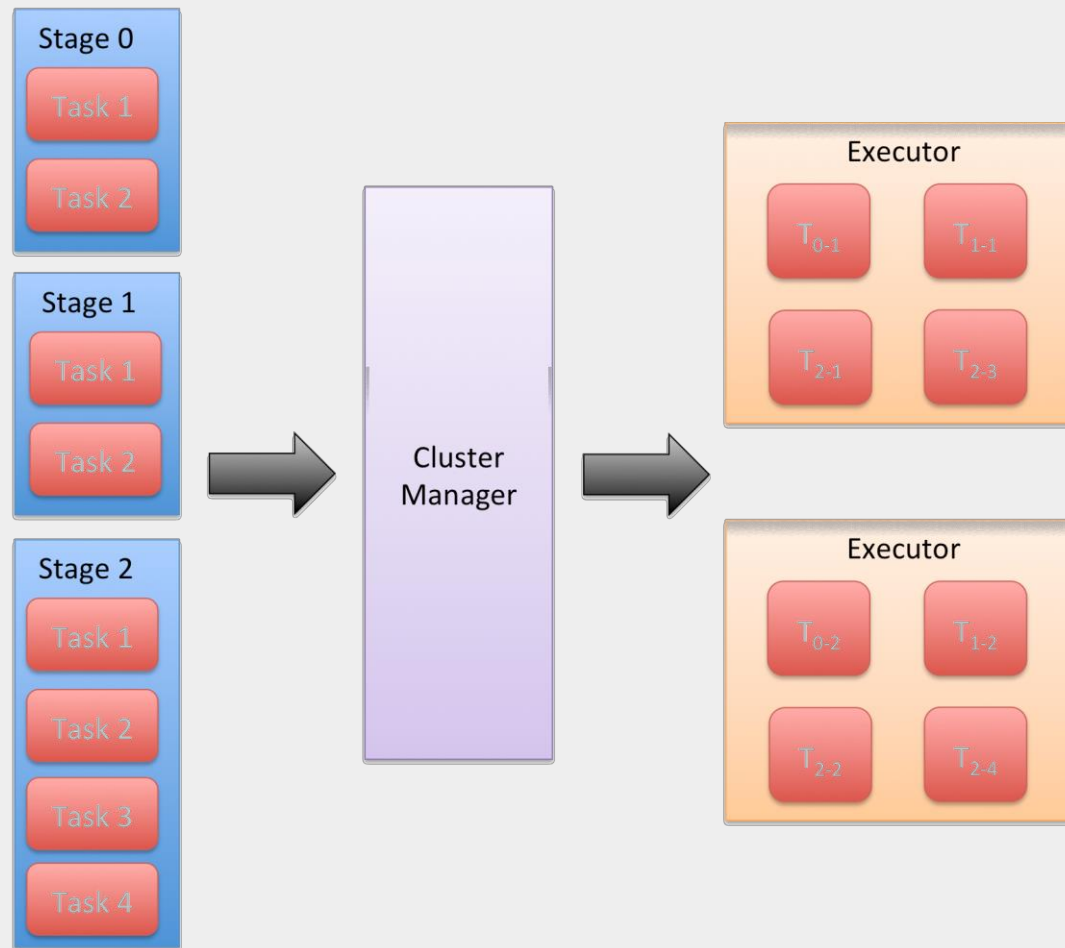
# Partitioning and Shuffling

Distributing an RDD among nodes of a cluster



# Partitioning and Shuffling

## Partitions Map to Tasks



# Partitioning and Shuffling

- Default Number of Partitions
  - Total number of cores in cluster
  - `spark.default.parallelism` config parameter
  - HDFS block size



# Partitioning and Shuffling

- Controlling Partition Size
  - When loading data

```
val defaultFile = sc.textFile("file:///data/war_and_peace.txt")
defaultFile.partitions.size
// 2
```

```
val manualFile = sc.textFile(file:///data/war\_and\_peace.txt, 100)
manualFile.partitions.size
// 100
```

# Partitioning and Shuffling

- Controlling Partition Size
  - Resizing after loading

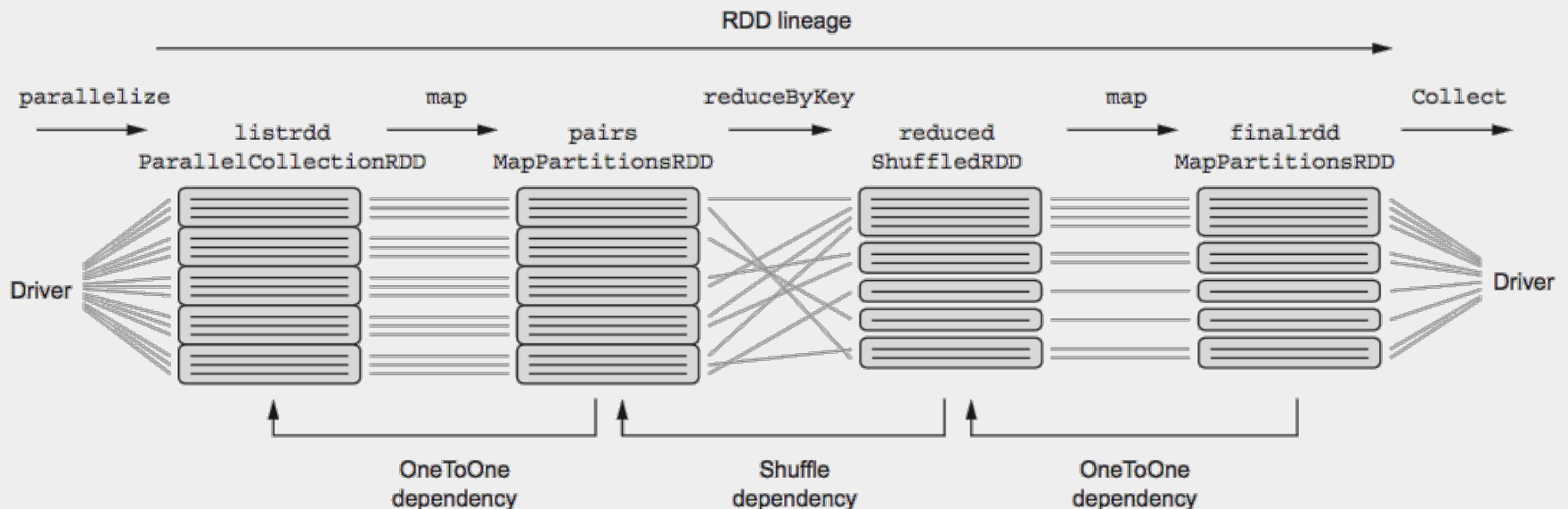
```
val file = sc.textFile("file:///data/war_and_peace.txt")
file.partitions.size
// 2
```

```
val resized = file.repartition(1000)
resized.partitions.size
// 1000
```

```
val coalesced = resized.coalesce(50)
coalesced.partitions.size
// 50
```

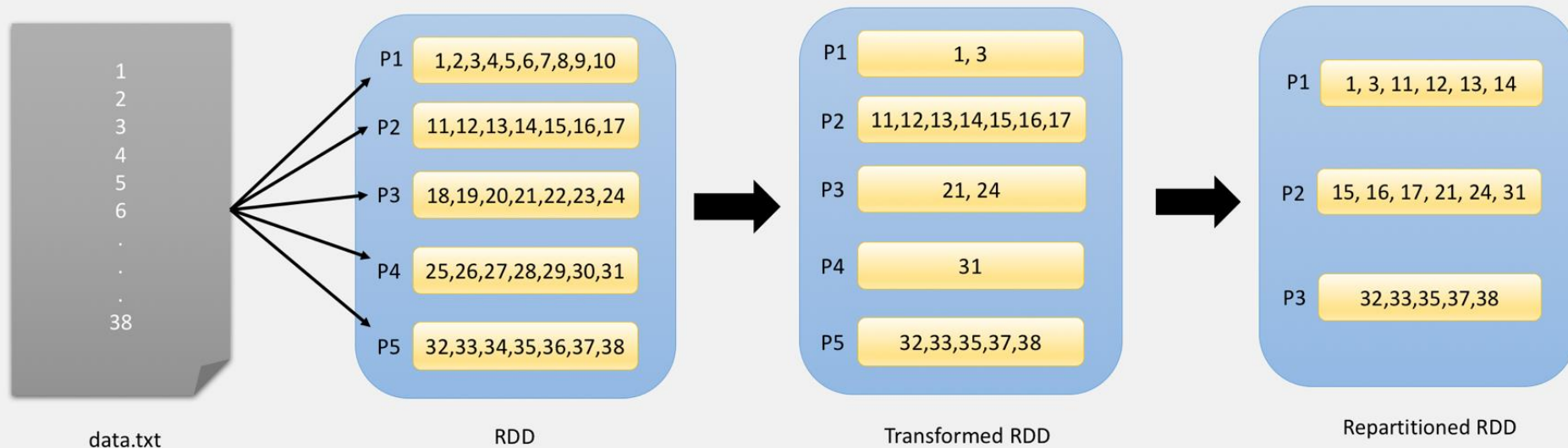
# Partitioning and Shuffling

- Shuffling
  - Narrow vs. Wide Dependencies



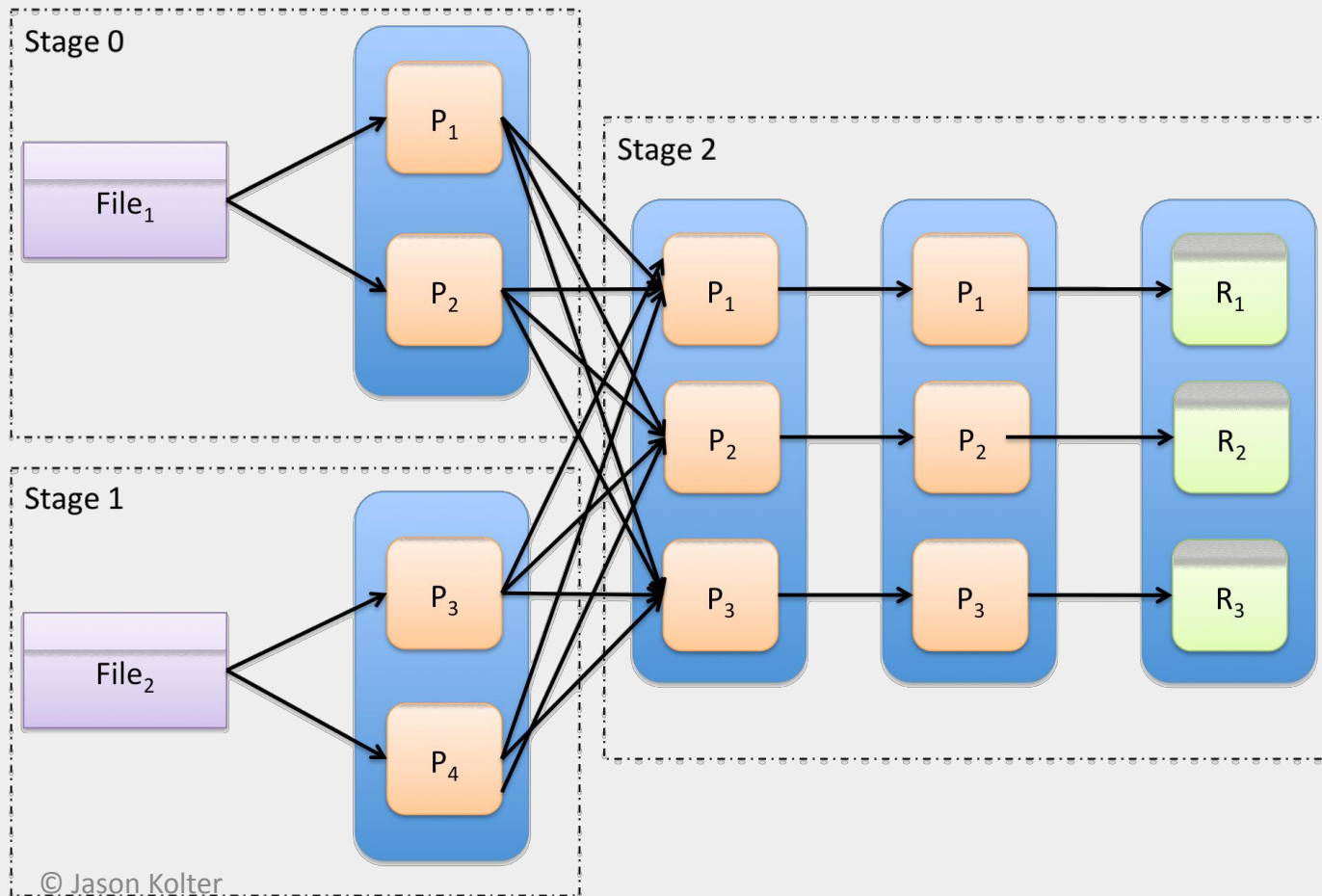
# Partitioning and Shuffling

- Shuffling
  - Force Repartition



# Partitioning and Shuffling

- Shuffling Determines Stage Boundaries



# Partitioning and Shuffling

- Best Practices
  - Know your data



# Partitioning and Shuffling

- Best Practices
  - mapPartitions vs. map

```
val airports = sc.textFile("file:///data/airport_codes.csv")

def expensiveMapFunction (airport:String) {
    val expensiveConnection = new ExpensiveConnection()
    expensiveConnection.enrichAirportData(airport)
}

val enrichedAirportData = airports.map(expensiveMapFunction)
```

```
def betterMapFunction (airportData:Iterator[String]) {
    val expensiveConnection = new ExpensiveConnection()
    airportData.map(expensiveConnection.enrichAirportData(_)).iterator
}

val enrichedAirportData = airports.mapPartitions(betterMapFunction)
```

# Partitioning and Shuffling

- Best Practices
  - foreachPartition vs. foreach

```
val airports = sc.textFile("file:///data/airport_codes.csv")

def expensiveForeachFunction (airport : String) {
    val dbConnection = new DBConnection
    dbConnection.save(airport)
}

airports.foreach(expensiveForeachFunction)
```

```
def betterForeachFunction (airportData:Iterator[String]) {
    val dbConnection = new DBConnection
    airportData.foreach(dbConnection.save(_))
}

airports.foreachPartition(betterForeachFunction)
```



# Partitioning and Shuffling

- Best Practices
  - Custom Partitioner

```
import org.apache.spark.Partitioner

class MyPartitioner(override val numPartitions : Int) extends
org.apache.spark.Partitioner {
  override def getPartition(key: Any): Int = {
    return key.asInstanceOf[Int];
  }

  override def equals(other: Any): Boolean = {
    return true;
  }
}
```

# Persistence and Caching

```
val rdd = sc.textFile("file://data/war_and_peace.txt")

val filtered = rdd.filter(x => x.toUpperCase().contains("WAR"))
filtered.saveAsTextFile("2.3-1")

val mapped = rdd.map(x => x.toUpperCase().replace("WAR", "PEACE"))
mapped.saveAsTextFile("2.3-2")
```

# Persistence and Caching

```
val rdd = sc.textFile("file://data/war_and_peace.txt")

rdd.persist

val filtered = rdd.filter(x => x.toUpperCase().contains("WAR"))
filtered.saveAsTextFile("2.3-3")

val mapped = rdd.map(x => x.toUpperCase().replace("WAR", "PEACE"))
mapped.saveAsTextFile("2.3-4")
```

# Persistence and Caching

- Storage Levels
  - MEMORY\_ONLY (DEFAULT)
    - Store RDD as deserialized Java objects in memory
    - If entire RDD doesn't fit
      - Some partitions will not be cached and will be recomputed
  - MEMORY\_AND\_DISK
    - Store RDD as deserialized Java objects in memory
    - If entire RDD doesn't fit
      - Store partitions that don't fit on disk read as (if) needed
  - DISK\_ONLY
    - “cache” on disk

# Persistence and Caching

- Storage Levels
  - MEMORY\_ONLY\_SER
    - Store RDD as **serialized** Java objects in memory
    - If entire RDD doesn't fit
      - Some partitions will not be cached and will be recomputed
  - MEMORY\_AND\_DISK\_SER
    - Store RDD as **serialized** Java objects in memory
    - If entire RDD doesn't fit
      - Store partitions that don't fit on disk read as (if) needed

# Persistence and Caching

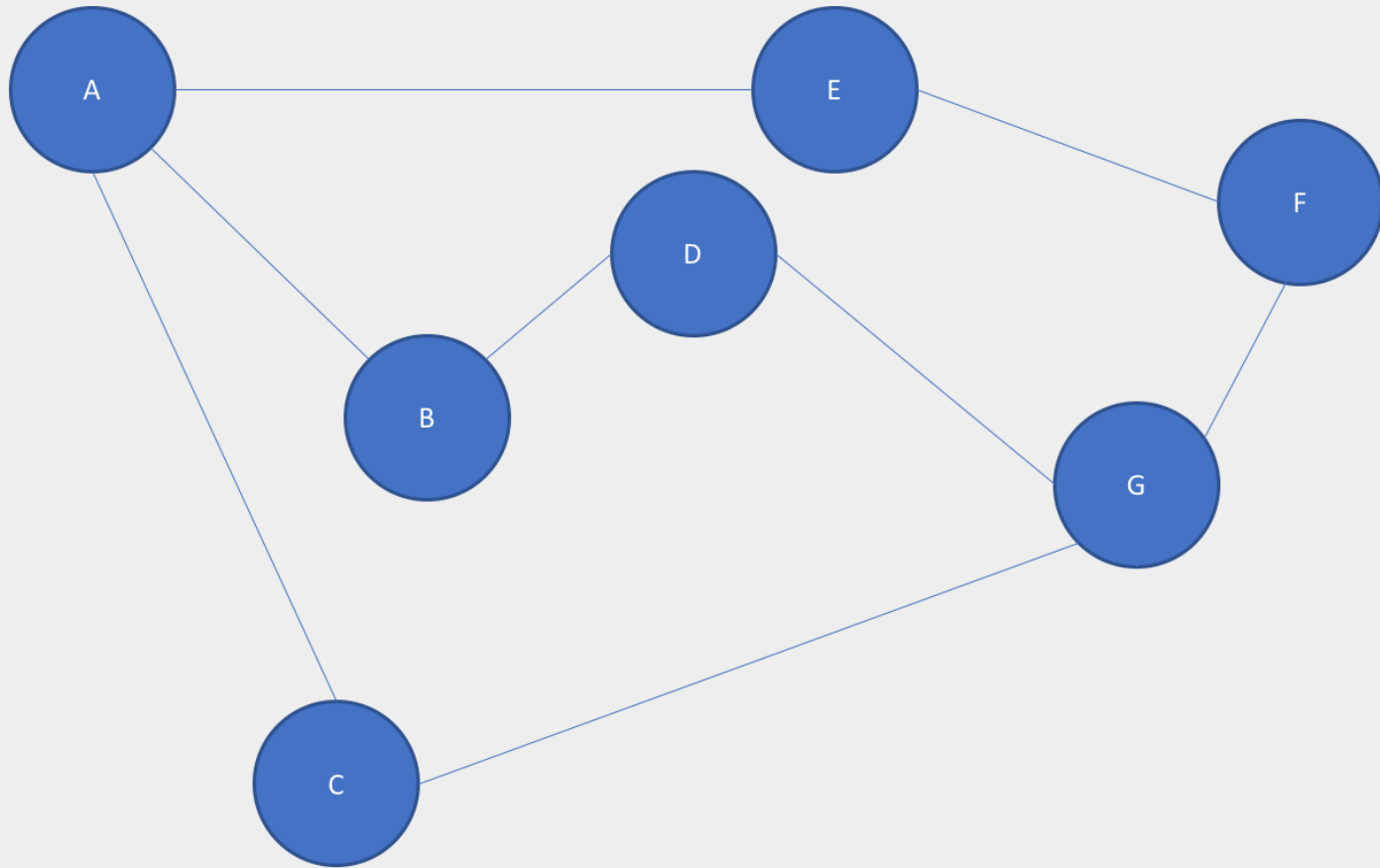
- Storage Levels
  - MEMORY\_ONLY\_2
  - MEMORY\_ONLY\_SER\_2
  - MEMORY\_AND\_DISK\_2
  - MEMORY\_AND\_DISK\_SER\_2
- Same as previous but replicate 2X

# Persistence and Caching

```
val lines = sc.textFile("data/war_and_peace.txt")
val complexResult = lines.filter(x => complexFilterFunction()).map(x =>
  complexMapFunction)
complexResult.persist(StorageLevel.MEMORY_AND_DISK)
```

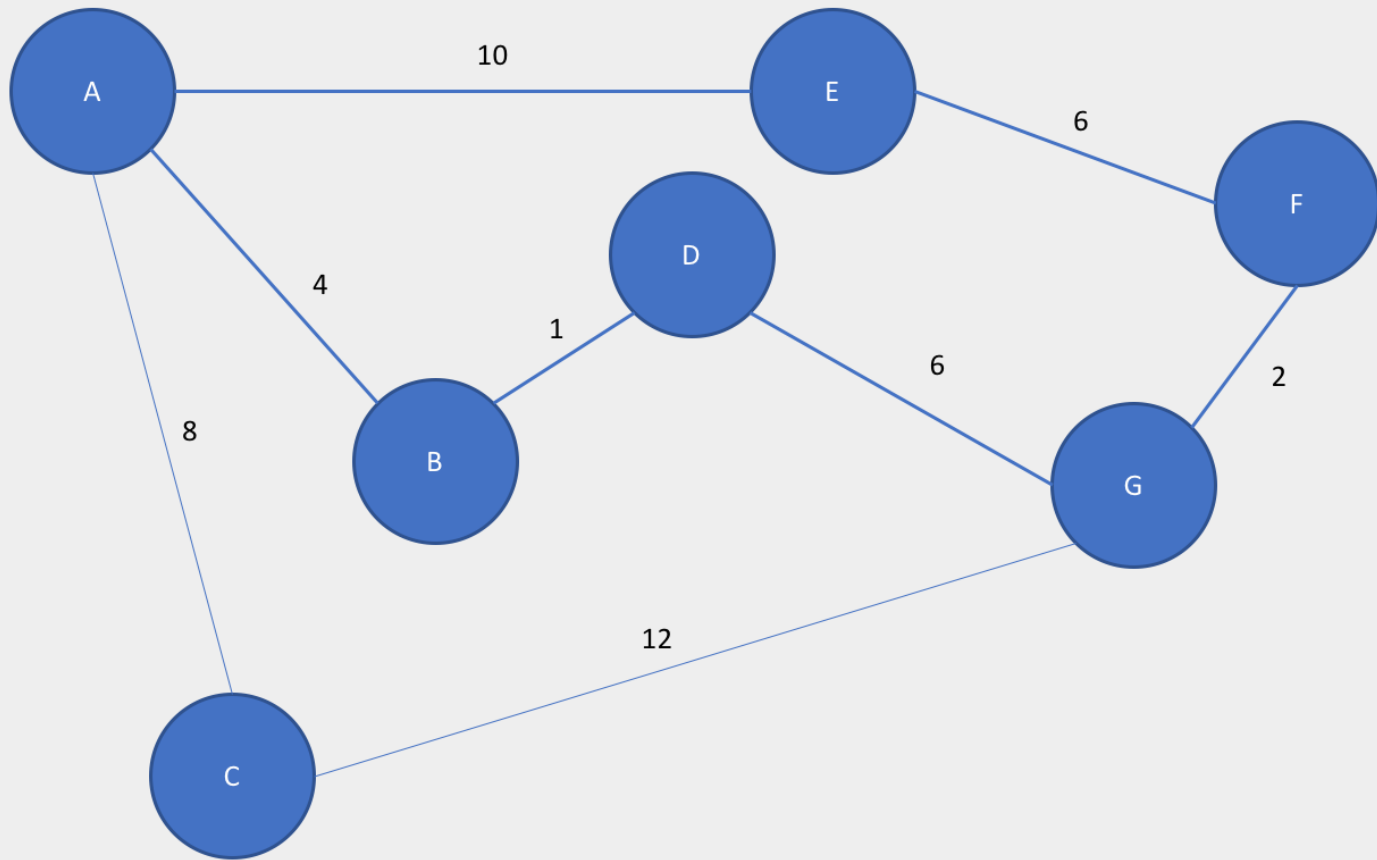
```
complexResult.unpersist
```

# Spark SQL

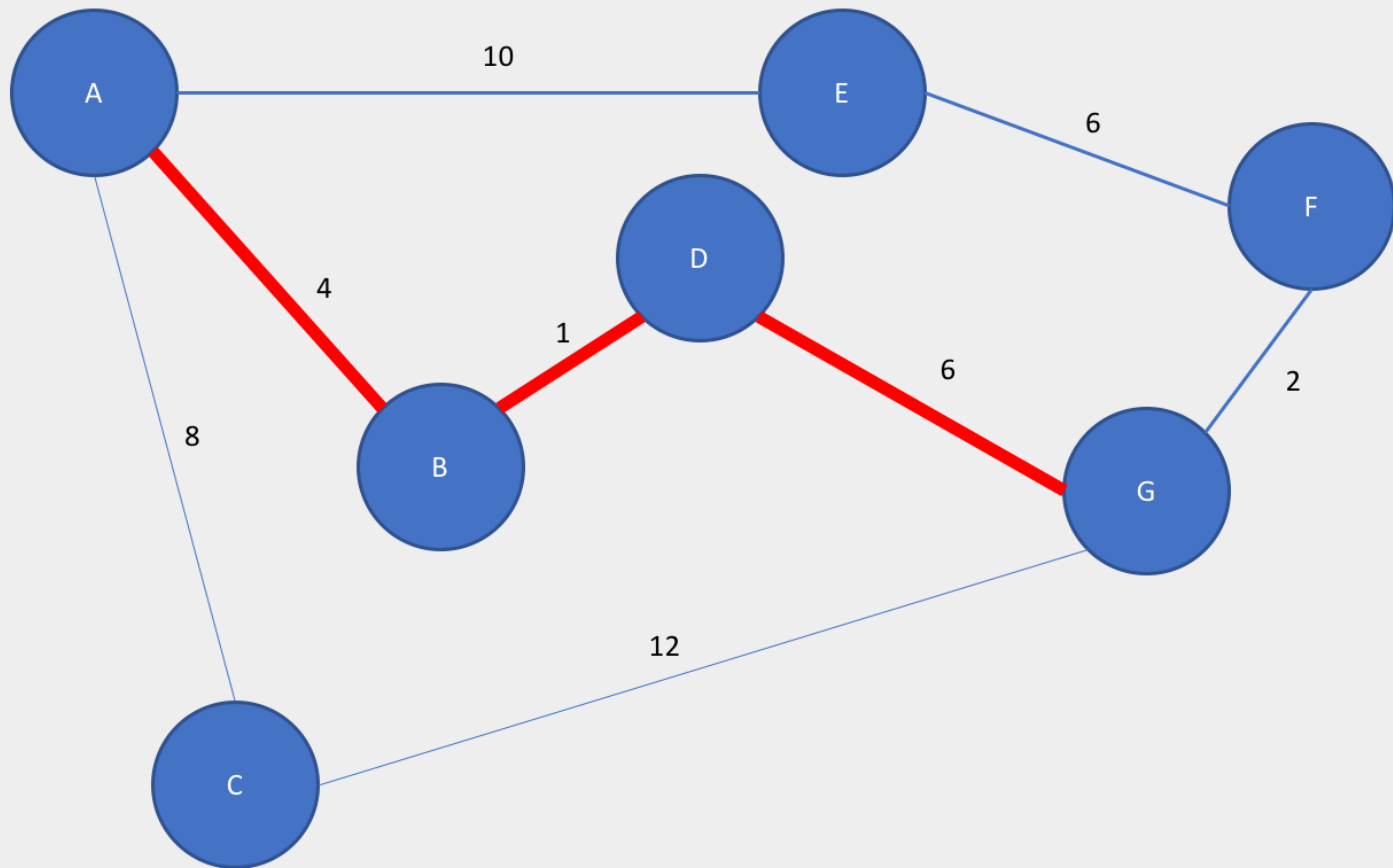




# Spark SQL

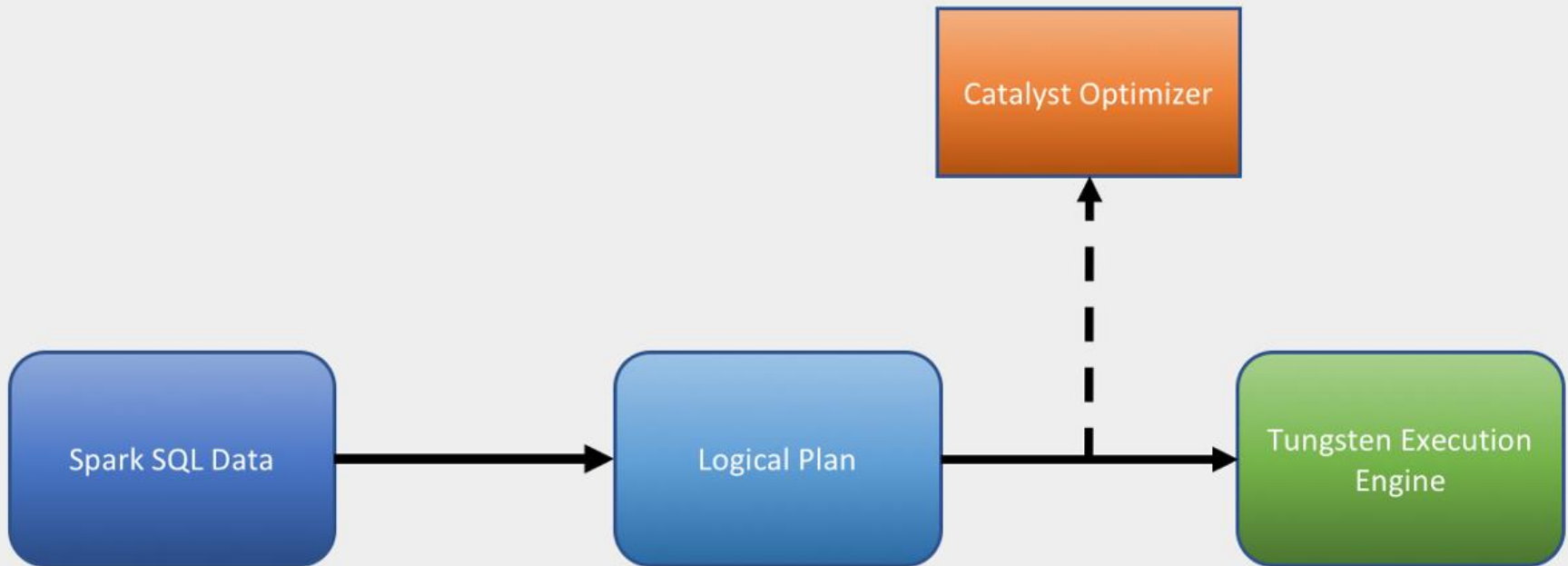


# Spark SQL



# Spark SQL

- Spark SQL Components



# Spark SQL

- DataFrame vs. Dataset

Year	Make	Model
Int	String	String
Int	String	String
Int	String	String

Object (Car)
Object (Car)
Object (Car)

# Spark SQL - Datasets

```
import org.apache.spark.sql.SparkSession

val ss = SparkSession.
    builder().
    master("local").
    appName("Spark in Motion Example").
    config("spark.config.option", "some-value").
    getOrCreate()
```

```
//In notebook or shell
print(spark)
```

```
import ss.implicits._
val ds = spark.createDataset(1 to 500)
ds.show(10)
```

```
+-----+
|value|
+-----+
|  1  |
|  2  |
|  3  |
|  4  |
|  5  |
|  6  |
|  7  |
|  8  |
|  9  |
| 10  |
+-----+
```

# Spark SQL - Datasets

- Creating

```
val textRDD = sc.textFile("file:///data/war_and_peace.txt")
```

```
import spark.implicits._  
val textDS = spark.read.textFile("file:///data/war_and_peace.txt")
```

```
val lowerText = textDS.map(line => line.toLowerCase)  
val words = lowerText.flatMap(line => line.split("\\s+"))  
val groupedWords = words.groupByKey(x => x)  
groupedWords.count.show(10)
```

value	count(1)
07	2
still	611
some	825
cold,	32
those	635
"oh,	174
connected	20
hope	65
seat.	10
spared	2

only showing top 10 rows

# Spark SQL - Datasets

- Using

```
ident,category,name,latitude_deg,longitude_deg,elevation_ft,continent,iso_country,iso_region,municipality,gps_code,iata_code,local_code
00A,heliport,Total Rf Heliport,40.07080078,-74.93360138,11,NA,US,US-PA,Bensalem,00A,,00A
00AK,small_airport,Lowell Field,59.94919968,-151.6959991,450,NA,US,US-AK,Anchor Point,00AK,,00AK
00AL,small_airport,Epps Airpark,34.8647995,-86.77030182,820,NA,US,US-AL,Harvest,00AL,,00AL
00AR,heliport,Newport Hospital & Clinic Heliport,35.6086998,-91.25489807,237,NA,US,US-AR,Newport,00AR,,00AR
```

```
case class Airport (ident:String, category:String, name:String, latitude_deg:Double, longitude_deg:Double,
                    elevation_ft:Integer, continent:String, iso_country:String, iso_region:String, municipality:String,
                    gps_code:String, iata_code:String, local_code:String)

val airports = spark.read.option("inferSchema", "true").option("header","true").csv("data/airport_codes.csv").as[Airport]
airports.show(5)
```

ident	category	name	latitude_deg	longitude_deg	elevation_ft	continent	iso_country	iso_region	municipality	gps_code	iata_code	local_code
00A	heliport	Total Rf Heliport	40.07080078	-74.93360138	11	NA	US	US-PA	Bensalem	00A	null	00A
00AK	small_airport	Lowell Field	59.94919968	-151.6959991	450	NA	US	US-AK	Anchor Point	00AK	null	00AK
00AL	small_airport	Epps Airpark	34.8647995	-86.77030182	820	NA	US	US-AL	Harvest	00AL	null	00AL
00AR	heliport	Newport Hospital ...	35.6086998	-91.25489807	237	NA	US	US-AR	Newport	00AR	null	00AR
00AZ	small_airport	Cordes Airport	34.30559921	-112.1650009	3810	NA	US	US-AZ	Cordes	00AZ	null	00AZ

only showing top 5 rows

# Spark SQL - Datasets

```
val airportsCA = airports.filter(_.iso_region == "US-CA")
airportsCA.show(5)
```

ident	category	name	latitude_deg	longitude_deg	elevation_ft	continent	iso_country	iso_region	municipality	gps_code	iata_code	local_code
00CA	small_airport	Goldstone /Gts/ A...	35.3504982	-116.8880005	3038	NA	US	US-CA	Barstow	00CA	null	00CA
01CA	heliport	Lugo Substation H...	34.36824059	-117.3700587	3733	NA	US	US-CA	Hesperia	01CA	null	01CA
01CL	small_airport	Swansboro Country...	38.79990005	-120.7340012	2594	NA	US	US-CA	Placerville	01CL	null	01CL
01CN	heliport	Los Angeles Count...	34.03779984	-118.1539993	300	NA	US	US-CA	Los Angeles	01CN	null	01CN
02CA	heliport	Swepi Beta Platfo...	33.58250046	-118.1289978	122	NA	US	US-CA	Huntington Beach	02CA	null	02CA

only showing top 5 rows

```
val heliportsCA = airports.filter(_.iso_region == "US-CA").filter(_.category == "heliport").filter(_.elevation_ft > 1000)
heliportsCA.show(5)
```

ident	category	name	latitude_deg	longitude_deg	elevation_ft	continent	iso_country	iso_region	municipality	gps_code	iata_code	local_code
01CA	heliport	Lugo Substation H...	34.36824059	-117.3700587	3733	NA	US	US-CA	Hesperia	01CA	null	01CA
06CA	heliport	Sce Solar I Heliport	34.87080002	-116.8339996	1942	NA	US	US-CA	Yermo	06CA	null	06CA
08CA	heliport	Pg & E Co. Placer...	38.69490051	-120.8270035	1810	NA	US	US-CA	Diamond Springs	08CA	null	08CA
0CL9	heliport	Sce San Jacinto V...	33.74110031	-117.151001	1482	NA	US	US-CA	Romoland	0CL9	null	0CL9
10CA	heliport	William E Poole H...	34.602252	-117.172451	3120	NA	US	US-CA	Apple Valley	10CA	null	10CA

only showing top 5 rows

```
airports.CA.persist
```



# Spark SQL - DataFrames

Year	Make	Model
Int	String	String
Int	String	String
Int	String	String

Year   Make   Model
Year   Make   Model
Year   Make   Model

# Spark SQL - DataFrames

```
case class Airport (ident:String, category:String, name:String, latitude_deg:Double, longitude_deg:Double, elevation_ft:Integer, continent:String, iso_country:String, iso_region:String, municipality:String, gps_code:String, iata_code:String, local_code:String)
```

```
val airportsDF = spark.read.option("inferSchema", "true").option("header", "true").csv("data/airport_codes.csv").as[Airport].toDF
airportsDF.show(5)
```

ident	category	name	latitude_deg	longitude_deg	elevation_ft	continent	iso_country	iso_region	municipality	gps_code	iata_code	local_code
00A	heliport	Total Rf Heliport	40.07080078	-74.93360138	11	NA	US	US-PA	Bensalem	00A	null	00A
00AK	small_airport	Lowell Field	59.94919968	-151.6959991	450	NA	US	US-AK	Anchor Point	00AK	null	00AK
00AL	small_airport	Epps Airpark	34.8647995	-86.77030182	820	NA	US	US-AL	Harvest	00AL	null	00AL
00AR	heliport	Newport Hospital ...	35.6086998	-91.25489807	237	NA	US	US-AR	Newport	00AR	null	00AR
00AZ	small_airport	Cordes Airport	34.30559921	-112.1650009	3810	NA	US	US-AZ	Cordes	00AZ	null	00AZ

only showing top 5 rows

```
val airportsDF = spark.read.option("inferSchema", "true").option("header", "true").csv("data/airport_codes.csv")
airportsDF.show(5)
```

ident	category	name	latitude_deg	longitude_deg	elevation_ft	continent	iso_country	iso_region	municipality	gps_code	iata_code	local_code
00A	heliport	Total Rf Heliport	40.07080078	-74.93360138	11	NA	US	US-PA	Bensalem	00A	null	00A
00AK	small_airport	Lowell Field	59.94919968	-151.6959991	450	NA	US	US-AK	Anchor Point	00AK	null	00AK
00AL	small_airport	Epps Airpark	34.8647995	-86.77030182	820	NA	US	US-AL	Harvest	00AL	null	00AL
00AR	heliport	Newport Hospital ...	35.6086998	-91.25489807	237	NA	US	US-AR	Newport	00AR	null	00AR
00AZ	small_airport	Cordes Airport	34.30559921	-112.1650009	3810	NA	US	US-AZ	Cordes	00AZ	null	00AZ

only showing top 5 rows

# Spark SQL - DataFrames

```
val airportsCA = airportsDF.filter($"iso_region" === "US-CA")
airportsCA.show(5)
```

ident	category	name	latitude_deg	longitude_deg	elevation_ft	continent	iso_country	iso_region	municipality	gps_code	iata_code	local_code
00CA	small_airport	Goldstone /Gts/ A...	35.3504982	-116.8880005	3038	NA	US	US-CA	Barstow	00CA	null	00CA
01CA	heliport	Lugo Substation H...	34.36824059	-117.3700587	3733	NA	US	US-CA	Hesperia	01CA	null	01CA
01CL	small_airport	Swansboro Country...	38.79990005	-120.7340012	2594	NA	US	US-CA	Placerville	01CL	null	01CL
01CN	heliport	Los Angeles Count...	34.03779984	-118.1539993	300	NA	US	US-CA	Los Angeles	01CN	null	01CN
02CA	heliport	Swepi Beta Platfo...	33.58250046	-118.1289978	122	NA	US	US-CA	Huntington Beach	02CA	null	02CA

only showing top 5 rows

```
val heliportsCA = airports.filter($"iso_region" === "US-CA").filter($"category" === "heliport").filter($"elevation_ft" > 1000)
heliportsCA.show(5)
```

ident	category	name	latitude_deg	longitude_deg	elevation_ft	continent	iso_country	iso_region	municipality	gps_code	iata_code	local_code
01CA	heliport	Lugo Substation H...	34.36824059	-117.3700587	3733	NA	US	US-CA	Hesperia	01CA	null	01CA
06CA	heliport	Sce Solar I Heliport	34.87080002	-116.8339996	1942	NA	US	US-CA	Yermo	06CA	null	06CA
08CA	heliport	Pg & E Co. Placer...	38.69490051	-120.8270035	1810	NA	US	US-CA	Diamond Springs	08CA	null	08CA
0CL9	heliport	Sce San Jacinto V...	33.74110031	-117.151001	1482	NA	US	US-CA	Romoland	0CL9	null	0CL9
10CA	heliport	William E Poole H...	34.602252	-117.172451	3120	NA	US	US-CA	Apple Valley	10CA	null	10CA

only showing top 5 rows

# Spark SQL - DataFrames

```
val highAirportsCA = airports.filter($"iso_region" === "US-CA").groupBy($"iso_region").avg("elevation_ft")
highAirportsCA.show
```

```
+-----+-----+
|iso_region| avg(elevation_ft)|
+-----+-----+
|      US-CA|1047.5383858267717|
+-----+-----+
```

```
val heliportsCAProjection = airports.select($"ident", $"name").filter($"iso_region" === "US-CA").
    filter($"category" === "heliport").filter($"elevation_ft" > 1000)
heliportsCAProjection.show(5)
```

```
+-----+-----+
|ident|          name|
+-----+-----+
| 01CA|Lugo Substation H...|
| 06CA|Sce Solar I Heliport|
| 08CA|Pg & E Co. Placer...|
| 0CL9|Sce San Jacinto V...|
| 10CA|William E Poole H...|
+-----+-----+
only showing top 5 rows
```

# Spark SQL - DataFrames

```
val otherAirportData = spark.read.json("data/airports.json1")
otherAirportData.printSchema
```

```
root
|-- carriers: string (nullable = true)
|-- city: string (nullable = true)
|-- code: string (nullable = true)
|-- country: string (nullable = true)
|-- direct_flights: string (nullable = true)
|-- elev: string (nullable = true)
|-- email: string (nullable = true)
|-- icao: string (nullable = true)
|-- lat: string (nullable = true)
|-- lon: string (nullable = true)
|-- name: string (nullable = true)
|-- phone: string (nullable = true)
|-- runway_length: string (nullable = true)
-- state: string (nullable = true)
-- types: string (nullable = true)
-- tz: string (nullable = true)
-- url: string (nullable = true)
-- woeid: string (nullable = true)
```

```
val joinedAirportData = airports.join(otherAirportData, airports("name") === otherAirportData("name")).select($"ident",airports("name"),$"tz")
joinedAirportData.show(5)
```

ident	name	tz
075N	Churchill Airport	America/Winnipeg
0NE5	Newman Airport	Australia/Perth
0R2	Lincoln Municipal...	America/Chicago
12NK	Westport Airport	Pacific/Auckland
14S	Westport Airport	Pacific/Auckland

only showing top 5 rows

# Spark SQL - DataFrames

```
airportsDF.createOrReplaceTempView("airports")
otherAirportData.createOrReplaceTempView("other_airports")
```

```
spark.sql("SELECT * FROM airports WHERE iso_region = 'US-CA']").show(3)
```

ident	category	name	latitude_deg	longitude_deg	elevation_ft	continent	iso_country	iso_region	municipality	gps_code	iata_code	local_code
00CA	small_airport	Goldstone /Gts/ A...	35.3504982	-116.8880005	3038	NA	US	US-CA	Barstow	00CA	null	00CA
01CA	heliport	Lugo Substation H...	34.36824059	-117.3700587	3733	NA	US	US-CA	Hesperia	01CA	null	01CA
01CL	small_airport	Swansboro Country...	38.79990005	-120.7340012	2594	NA	US	US-CA	Placerville	01CL	null	01CL

only showing top 3 rows

```
spark.sql("SELECT a.ident, a.name, o.tz FROM airports a, other_airports o WHERE a.name = o.name").show(5)
```

ident	name	tz
07SN	Churchill Airport	America/Winnipeg
0NE5	Newman Airport	Australia/Perth
0R2	Lincoln Municipal...	America/Chicago
12NK	Westport Airport	Pacific/Auckland
14S	Westport Airport	Pacific/Auckland

only showing top 5 rows

```
spark.sql("SELECT AVG(elevation_ft) FROM airports WHERE iso_region = 'US-CA']").show
```

avg(elevation_ft)
1047.5383858267717

# Spark SQL - Data Sources

- CSV
  - Can use CSV parser with any delimiter
  - Set option “sep” to delimiter char

```
val suppliersDF = spark.read.option("sep", "|").csv("/data/supplier/suppliers.csv")
```

# Spark SQL - Data Sources

- JSON
  - Default wants to read JSONL format
    - <http://jsonlines.org>
  - As of latest Spark can read multi-line
    - Set option “multiline” to “true”

```
val rows = spark.read.option("multiLine", true).json("/data/rows.json")
```



# Spark SQL - Data Sources

- Parquet

- Spark SQL **HIGHLY**  
optimized for Parquet

# Spark SQL - Data Sources

- JDBC
  - Can read/write directly to DB
  - Meant for small transactional updates not bulk queries
  - Need to include driver on the classpath

```
bin/spark-shell --driver-class-path postgresql-9.4.1207.jar --jars postgresql-9.4.1207.jar
```

```
import java.util.Properties
val connectionProperties = new Properties()
connectionProperties.put("user", "username")
connectionProperties.put("password", "password")
val jdbcDF = spark.read.jdbc("jdbc:postgresql:dbserver", "schema.tablename", connectionProperties)
```

# Spark SQL - Data Sources

- Hive
  - Can access tables/file in Hive metastore
  - Already configured in HDP
  - Tables registered like using raw SQL

```
spark.sql("select id, date, price from home_data limit 10").show
```

```
+-----+-----+-----+
|      id|      date|  price|
+-----+-----+-----+
|7129300520|20141013T000000| 221900|
|6414100192|20141209T000000| 538000|
|5631500400|20150225T000000| 180000|
|2487200875|20141209T000000| 604000|
|1954400510|20150218T000000| 510000|
|7237550310|20140512T000000|1225000|
|1321400060|20140627T000000| 257500|
|2008000270|20150115T000000| 291850|
|2414600126|20150415T000000| 229500|
|3793500160|20150312T000000| 323000|
+-----+-----+-----+
```

```
val hiveDF = spark.sql("select * from home_data")
```