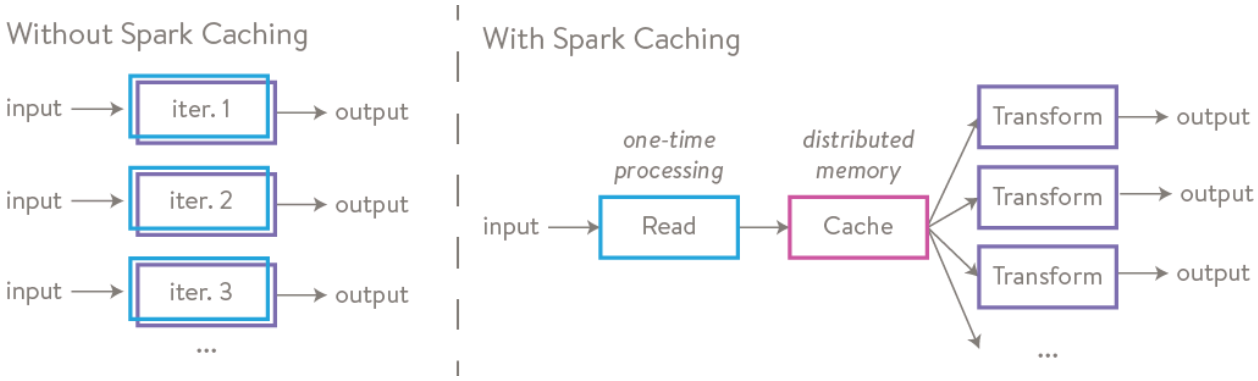
Spark Optimisation

1 - Caching

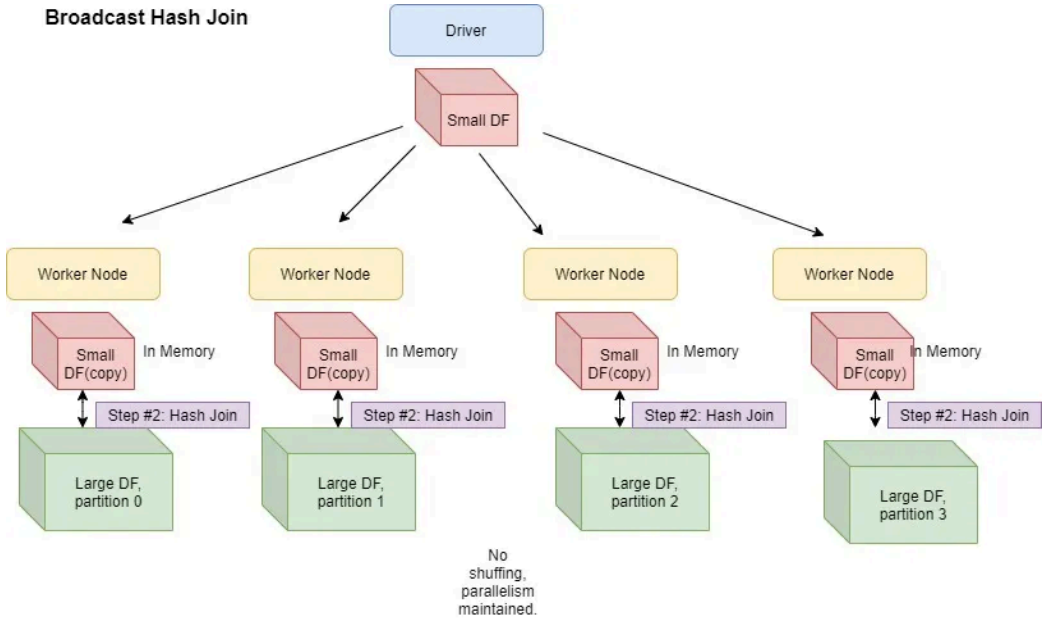



```
dataframe.cache
```

- It can be used with large dataset
- You can edit the dataset

2 - Broadcast table

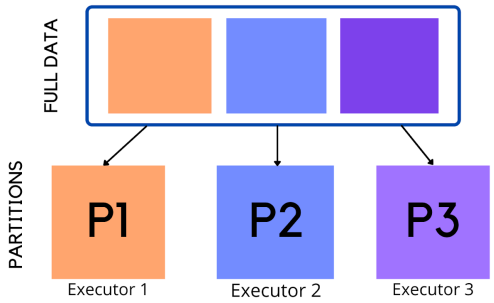
It is suitable with sharing small and read-only dataset across nodes



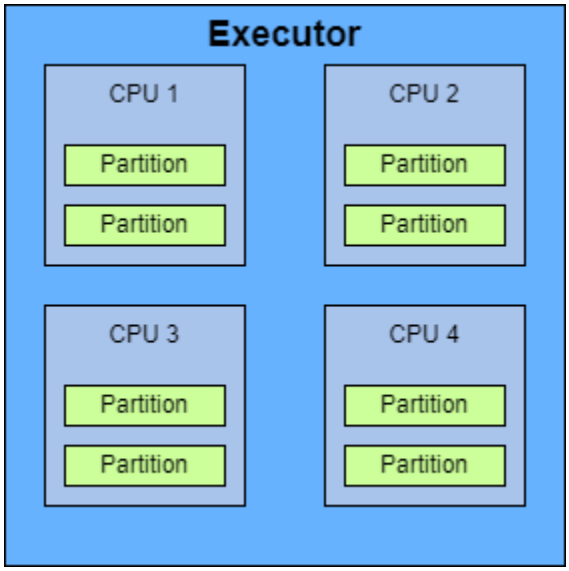
```
val countries = Map(("USA","United States of America"),("IN","India"))

val broadcastStates = spark.sparkContext.broadcast(states)
```

3 - Division large into smaller data sets

It supports processing small data across multiple nodes



4 - Number of partition



- Input Stage Data 100GB
- Target Size = 100MB

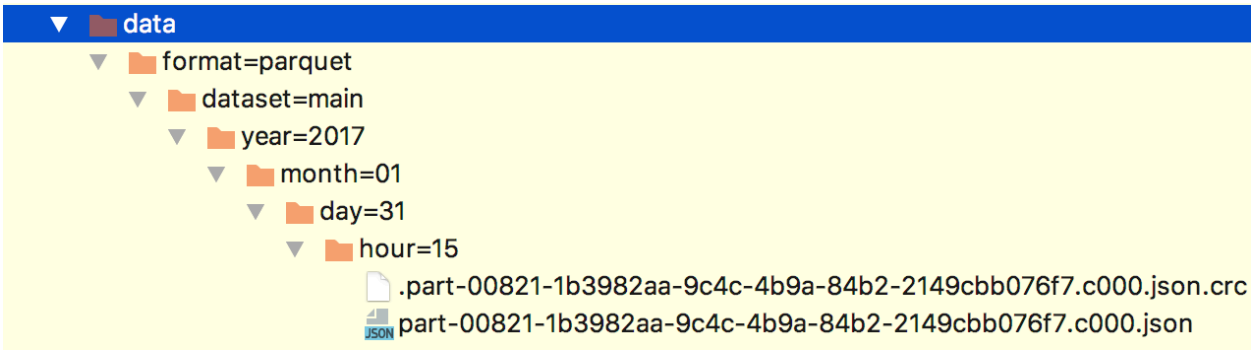
- Cores = 1000
- Optimal Count of Partitions = 100,000 MB / 100 = 1000 partitions

```
Spark.conf.set("spark.sql.shuffle.partitions",1000)
```

Too few partitions You will not utilize all of the cores available in the cluster.

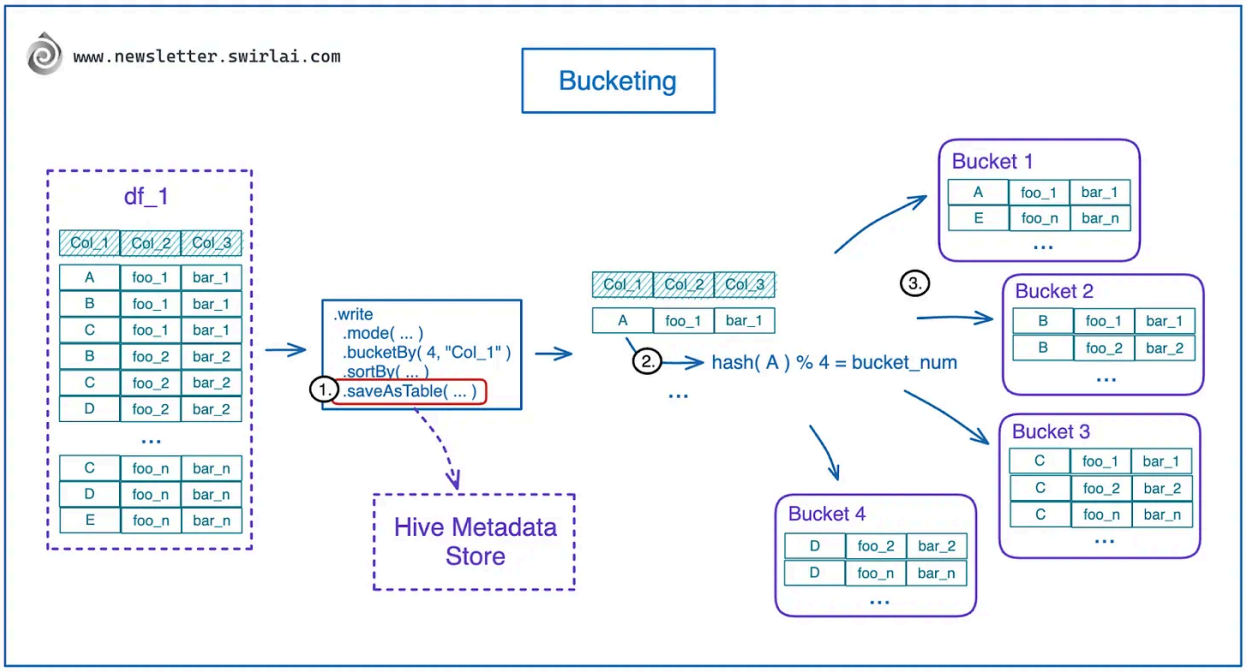
Too many partitions There will be excessive overhead in managing many small tasks

5 - Avoid using UDF
6 - Save data with a partition




```
resultDf.write()
    .mode(SaveMode.Overwrite)
    .partitionBy("year", "month", "day")
    .parquet(outputPath)
```

7 - Bucketing



Bucketing is a technique that puts the same column fields together. It would increase performance for join and grouping operations.

```
resultDf.write()
      .option("path",outputPath)
      .mode(SaveMode.Overwrite)
      .bucketBy(numberOfOutputBuckets, "group_key", "counter_type", "counter_key")
   .saveAsTable(outputTableName);
```

8 - Spark API selection

- RDD is used for low level operation with less optimization
- DataFrame is best choice in most cases due to its catalyst optimizer and low garbage collection (GC) overhead.
- Dataset is highly type safe and use encoders. It uses Tungsten for serialization in binary format

9 - Serialisation

- Kryo serializer is in compact binary format and offers processing 10x faster than Java serializer.

```
conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
```

10 - File Format Selection

- Spark supports many formats, such as CSV, JSON, XML, PARQUET, ORC, AVRO, etc.
- Spark jobs can be optimized by choosing the parquet file with snappy compression which gives the high performance and best analysis.
- Parquet file is native to Spark which carries the metadata along with its footer