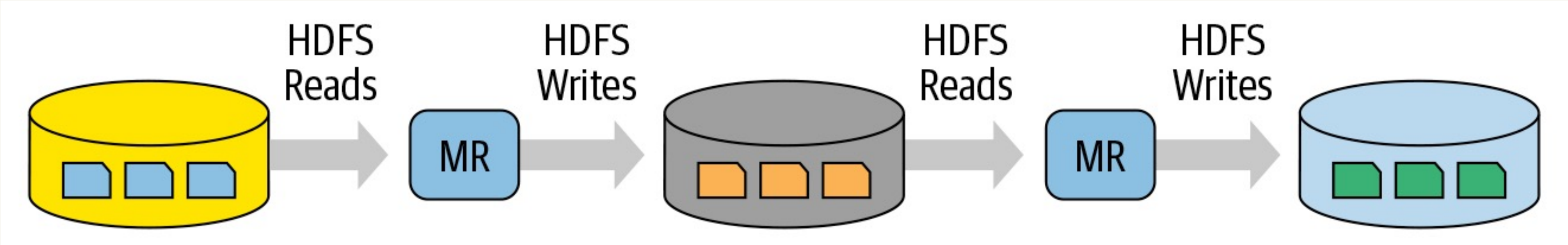
SPARK

Prasad M Deshpande

MR limitations

- Verbose API
- Low level abstraction – led to many layers built on top
  - *Hive, Impala for SQL*
  - *Storm for streaming*
  - *Giraph for graph data*
  - *Mahout for linear algebra, ML*
- Efficiency



Spark – a unified engine

What is Spark?

Fast and **Expressive** Cluster Computing System Compatible with Apache Hadoop

Up to **10×** faster on disk, **100×** in memory

2-5× less code

Efficient

- General execution graphs
- In-memory storage

Usable

- Rich APIs in Java, Scala, Python
- Interactive shell

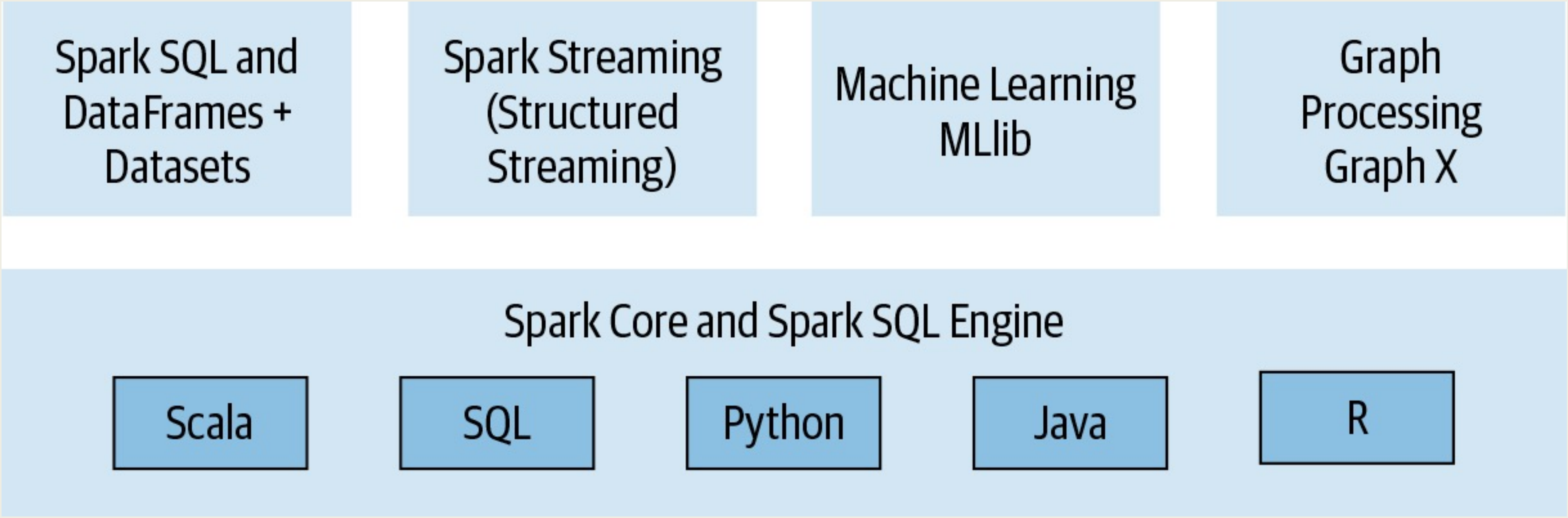
Spark

From McDonough Spark tutorial from Spark Summit 2013

Design goals

- Speed
  - *Hold intermediate data in memory*
  - *DAG scheduler executes task in parallel*
  - *Code generation*
- Ease of use
  - *Simple abstractions RDD, DataFrame, DataSet*
  - *Operations*
- Modularity
  - *Components for various functionalities*
  - *Libraries in different languages*
- Extensibility
  - *Decouples storage and compute*
  - *Can read data from many sources – Hadoop, Cassandra, HBase, etc*

Unified stack

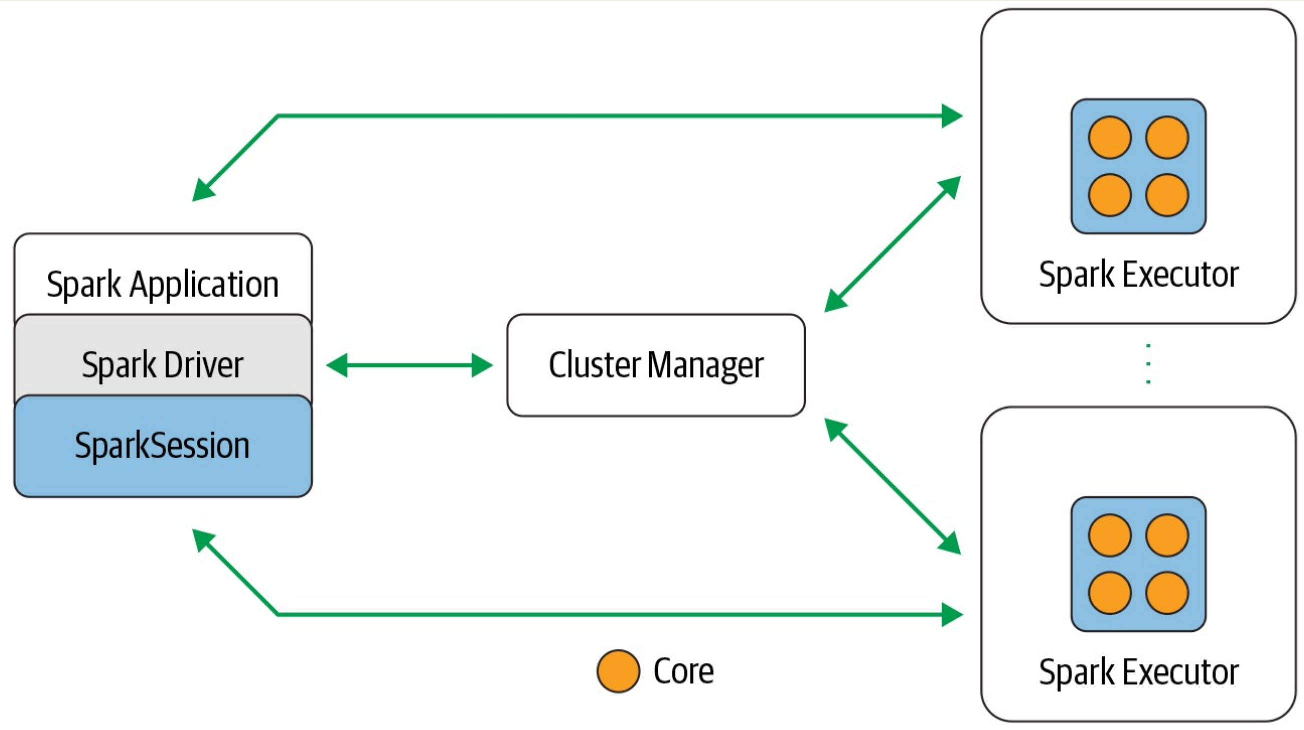


Spark Core

Responsible for:

- ✓ Memory Management and fault recovery
- ✓ Supports/implements key concepts of RDDs and Actions
- ✓ Scheduling, Monitoring, Distributing jobs on cluster [via YARN]

Architecture



Driver program

- The process running the main() function of the application and creating the SparkContext

Executor

- A process launched for an application on a worker node, that runs tasks and keeps data in memory or disk storage across them.

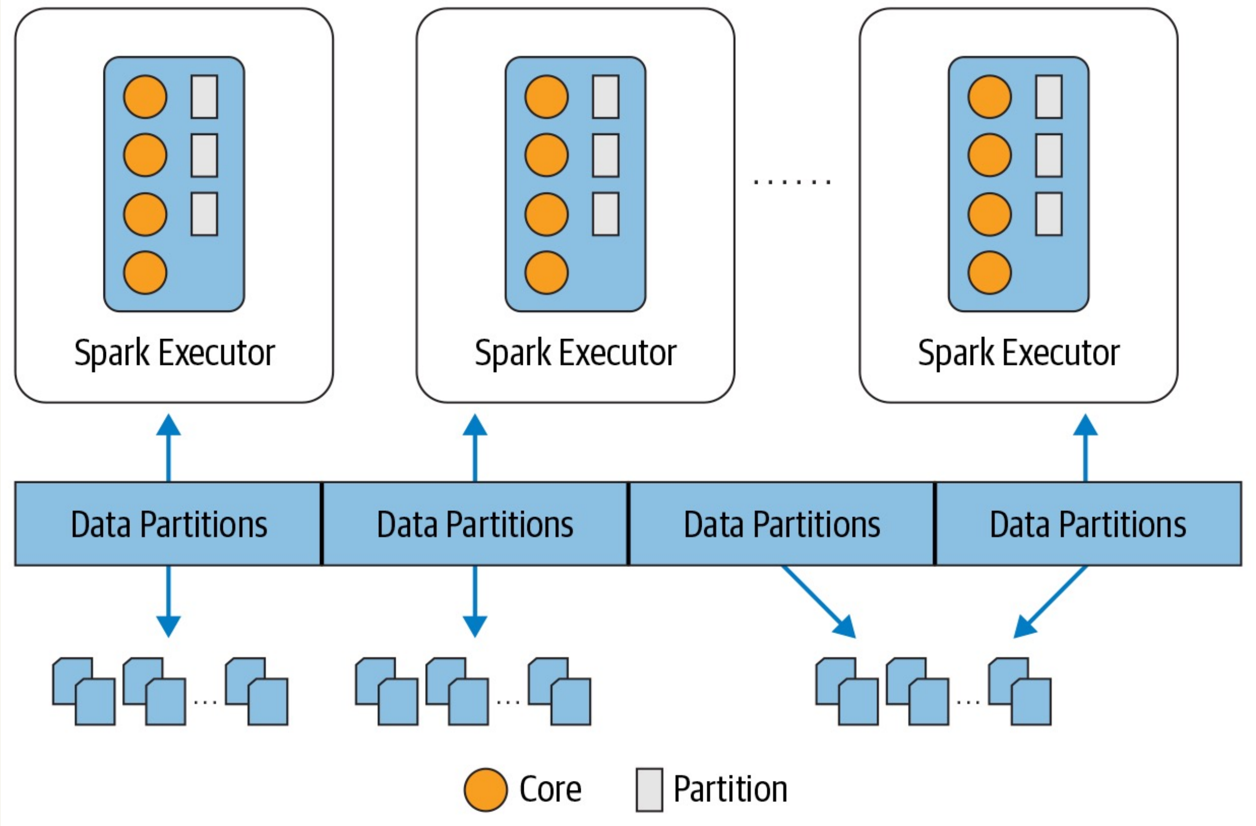
Spark session

```scala
// In Scala
import org.apache.spark.sql.SparkSession

// Build SparkSession
val spark = SparkSession
  .builder
  .appName("LearnSpark")
  .config("spark.sql.shuffle.partitions", 6)
  .getOrCreate()
...
// Use the session to read JSON
val people = spark.read.json("...")
...
// Use the session to issue a SQL query
val resultsDF = spark.sql("SELECT city, pop, state, zip FROM table_name")
```

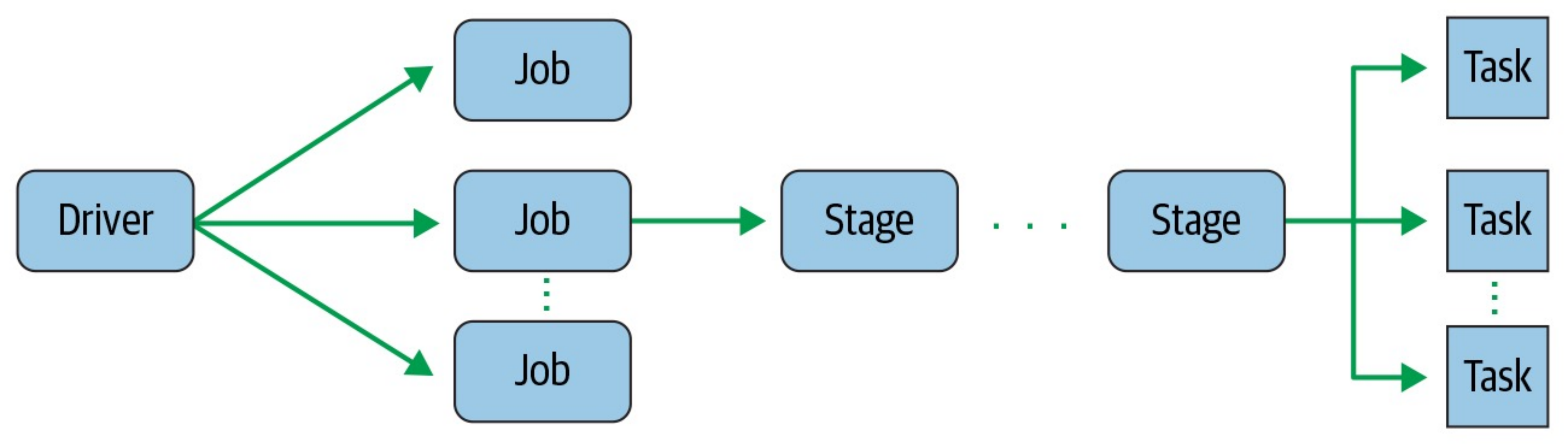
Resilient Distributed Data

- Write programs in terms of transformations on distributed datasets
- RDD – collection of objects spread across the cluster, stored in disk or memory
- Built through parallel transformations
- Automatically rebuilt on failure

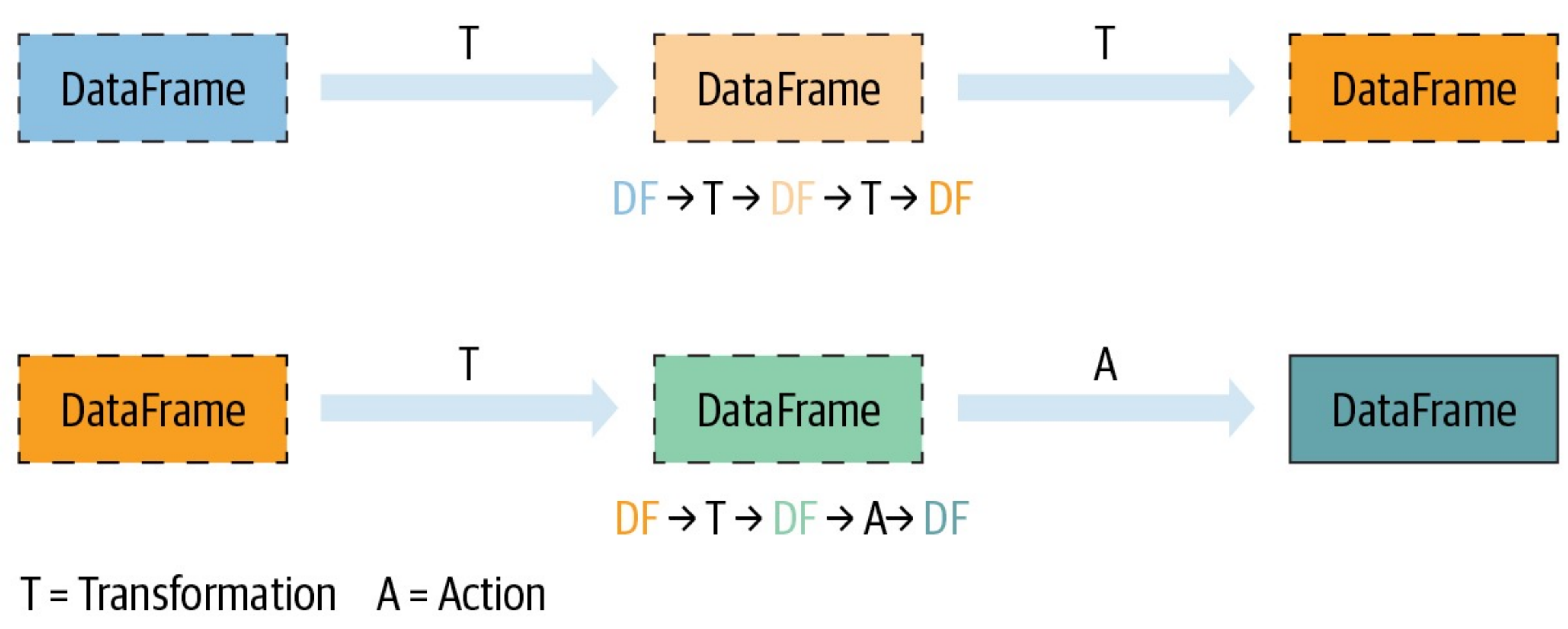
Logical in memory vs Physical distribution



Job, Stage, Task



Lazy Transformations, Eager Actions



- Populating of blocks into memory deferred until action is invoked
- action triggers actual evaluation of the RDD

Example

Transformations	Actions
orderBy()	show()
groupBy()	take()
filter()	count()
select()	collect()
join()	save()
map()	
flatmap()	

```
// In Scala
scala> import
org.apache.spark.sql.functions._
scala> val strings =
spark.read.text("../README.md")
scala> val filtered =
strings.filter(col("value").contains("Spark"
))
scala> filtered.count()
res5: Long = 20
```

Parallelizing Data

- How many partitions my RDD is split into?

```
# In Python
print(log_df.rdd.getNumPartitions())
```

- How to enforce “degree of parallelism”?

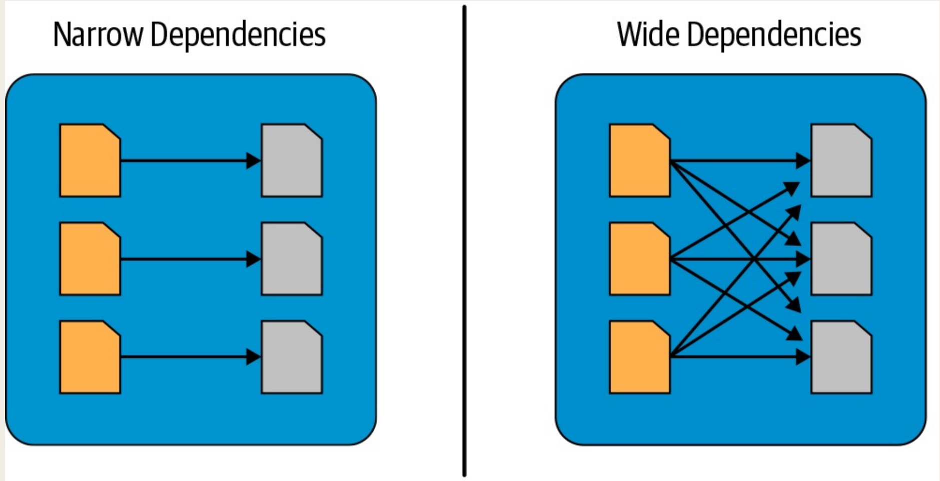
```
# In Python
log_df = spark.read.text("path_to_large_text_file").repartition(8)
print(log_df.rdd.getNumPartitions())
```

repartition vs coalesce

- `repartition` : will shuffle the original partitions and repartition them
- `coalesce` : will just combine original partitions to the new number of partitions.

shuffling could be very costly, but can be used to both increase and decrease number of partitions
coalesce can be used to only to reduce the number of partitions

Narrow vs wide transformations



- Narrow – single output partition can be computed from a single input partition is a narrow transformation
  - *E.g. Filter*
- Wide – data from other partitions is read in, combined, and written to disk
  - *E.g. group-by*
  - *Similar to reduce of MR*

Fault tolerance

- DataFrames are immutable
- Lineage is maintained
- On failure, entire flow is run again

RDD ➔ DataFrames➔ DataSets

RDD

- characteristics
  - *Dependencies*
  - *Partitions (with some locality information)*
  - *Compute function: Partition => Iterator[T]*
- Limitations
  - *Data type is opaque to spark*
  - *Compute function is also opaque*

Dataframe

- Introduced in spark 1.3 release
- Data organized into named columns (tabular format)
- Computations expressed as high-level operations
  - *such as filtering, selecting, counting, aggregating, averaging, and grouping.*
- Spark can optimize the query plan

Example RDD and DF

```python
# In Python

# Create an RDD of tuples (name, age)

dataRDD = sc.parallelize([("Brooke", 20), ("Denny", 31), ("Jules", 30), ("TD", 35), ("Brooke", 25)])

# Use map and reduceByKey transformations with their lambda

# expressions to aggregate and then compute average

agesRDD = (dataRDD

.map(lambda x: (x[0], (x[1], 1)))

.reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))

.map(lambda x: (x[0], x[1][0]/x[1][1])))
```

```python
# Create a DataFrame

data_df = spark.createDataFrame([("Brooke", 20), ("Denny", 31), ("Jules", 30),("TD", 35), ("Brooke", 25)], ["name", "age"])

# Group the same names together, aggregate their ages, and compute an average

avg_df = data_df.groupBy("name").agg(avg("age"))

# Show the results of the final execution

avg_df.show()
```

Dataset

- extension of DataFrame API which provides type-safe [compile time], object-oriented programming interface
- Maps the rows in DF into user defined class
- One can seamlessly move between DataFrame or Dataset and RDDs by simple API method calls like .rdd or .toDF or .as[T]

```
case class DeviceIoTData
(battery_level: Long, c02_level:
Long,cca2: String, cca3: String,
cn: String, device_id: Long,
device_name: String, humidity:
Long, ip: String, latitude:
Double, lcd: String, longitude:
Double, scale:String, temp: Long,
timestamp: Long)

val ds =
spark.read.json("/databricks-
datasets/learning-spark-v2/iot-
devices/iot_devices.json").as[Devi
ceIoTData]

val filterTempDS = ds.filter({d =>
{d.temp > 30 && d.humidity > 70})
```

Dataframe vs dataset

- If you want strict compile-time type safety and don’t mind creating multiple case classes for a specific Dataset[T], use Datasets.
- If your processing dictates relational transformations similar to SQL-like queries, use DataFrames.
- If you want to take advantage of and benefit from Tungsten’s efficient serialization with Encoders, , use Datasets.
- If you want unification, code optimization, and simplification of APIs across Spark components, use DataFrames.

When are errors caught?

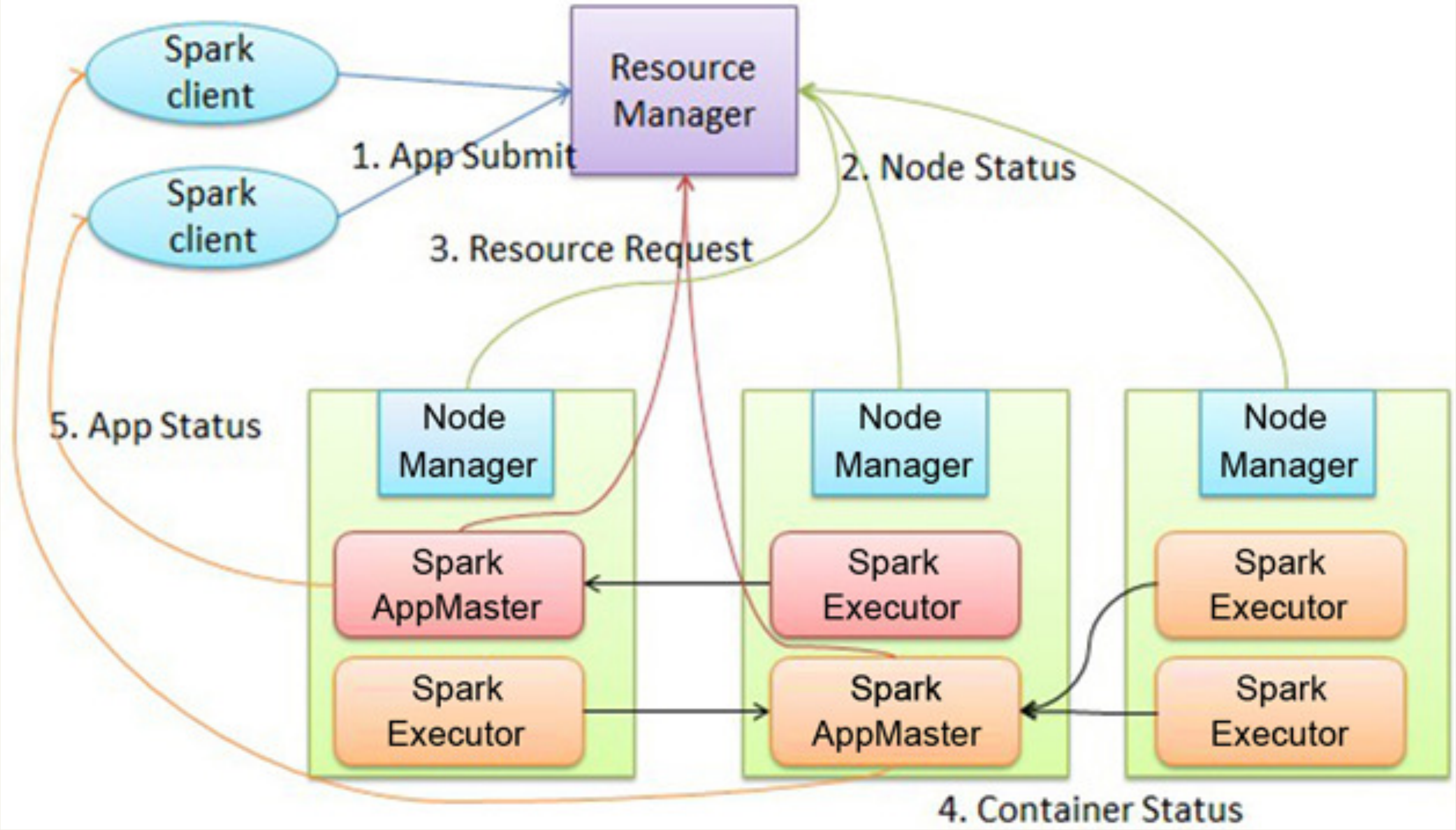
Structured APIs In Spark

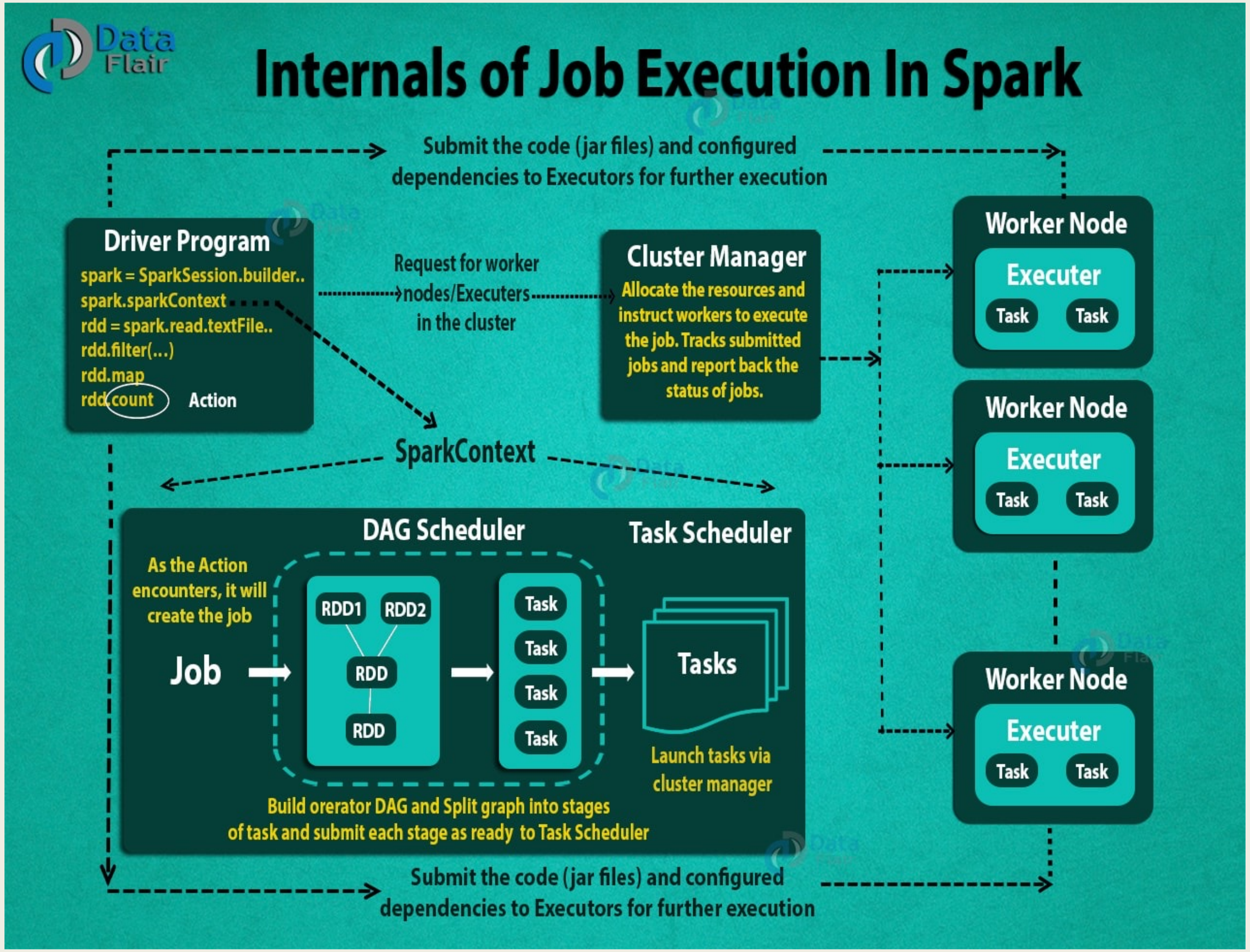
	SQL	DataFrames	Datasets
Syntax Errors	Runtime	Compile Time	Compile Time
Analysis Errors	Runtime	Runtime	Compile Time

Glimpse into Spark ML

```
val lr = new LogisticRegression().setMaxIter(10).setRegParam(0.001)

val pipeline = new Pipeline().setStages(Array(tokenizer, hashingTF, lr))

// Fit the pipeline to training documents.
val model = pipeline.fit(training)

 // Now we can optionally save the fitted pipeline to disk
model.write.overwrite().save("/tmp/spark-logistic-regression-model")

// Make predictions on test documents.
model.transform(test)
```

Relating back to YARN





https://data-flair.training/blogs/how-apache-spark-works/

Client vs Cluster mode

- Client mode – the driver runs in the client process on local machine, and the application master is only used for requesting resources from YARN.

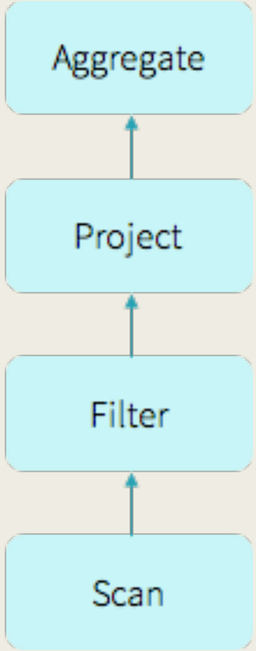
- Cluster mode – the Spark driver runs inside an application master process which is managed by YARN on the cluster, and the client can go away after initiating the application.

```
$ ./bin/spark-submit --class path.to.your.Class --master yarn --deploy-mode cluster [options] <app jar> [app options]
```

```
$ ./bin/spark-shell --master yarn --deploy-mode client
```

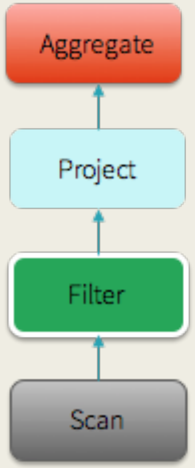
Code generation re-visted

```
select count(*) from store_sales
where ss_item_sk = 1000
```

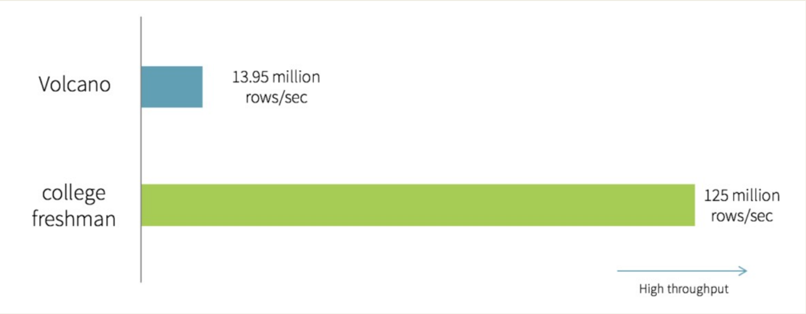


Generic vs hand-written code

```
class Filter(child: Operator, predicate:
(Row => Boolean)) extends Operator {
  def next(): Row = {
    var current = child.next()
    while (current != null &&
!predicate(current)) {
      current = child.next()
    }
    return current
  }
}
```




```
long count = 0;
for (ss_item_sk in store_sales) {
  if (ss_item_sk == 1000) {
    count += 1;
  }
}
```



https://databricks.com/blog/2016/05/23/apache-spark-as-a-compiler-joining-a-billion-rows-per-second-on-a-laptop.html

Summary – SPARK vs MR

- Ease of use
- Developer productivity
- Speed
- Ecosystem: Spark R, Spark MLLib, PySpark, SparkSQL
- Lot of apache projects also moving to support/leverage Spark

Reading material

- Learning Spark, Jules S. Damji, Brooke Wenig, Tathagata Das, and Denny Lee (https://pages.databricks.com/rs/094-YMS-629/images/LearningSpark2.0.pdf)
  - *Chapter 1*
  - *Chapter 2 – Step 3, Transformations, Actions and Lazy Evaluation*
  - *Chapter 3 – Spark: What's Underneath an RDD?, Structuring Spark, Dataset API – creating datasets, Dataframes vs Datasets*