Apache Spark Best Practices

Eren Avşaroğulları

Data Science & Engineering Club Meetup Talk

Dublin, 27 April 2019

Agenda

- Architecture
- Data Structures
- Persistency + GC Policy
- Partitioning + Data Skew
- Data Locality
- Job Scheduling
- Ser/De
- Event Sourcing on Transformations
- Checkpointing

Bio

- B.Sc & M.Sc on Electronics & Control Engineering
- Data Engineer @ 

- Working on Data Analytics (Data Transformations & Cleaning)
- Open Source Contributor @ 

  { Apache Spark | Pulsar | Heron }

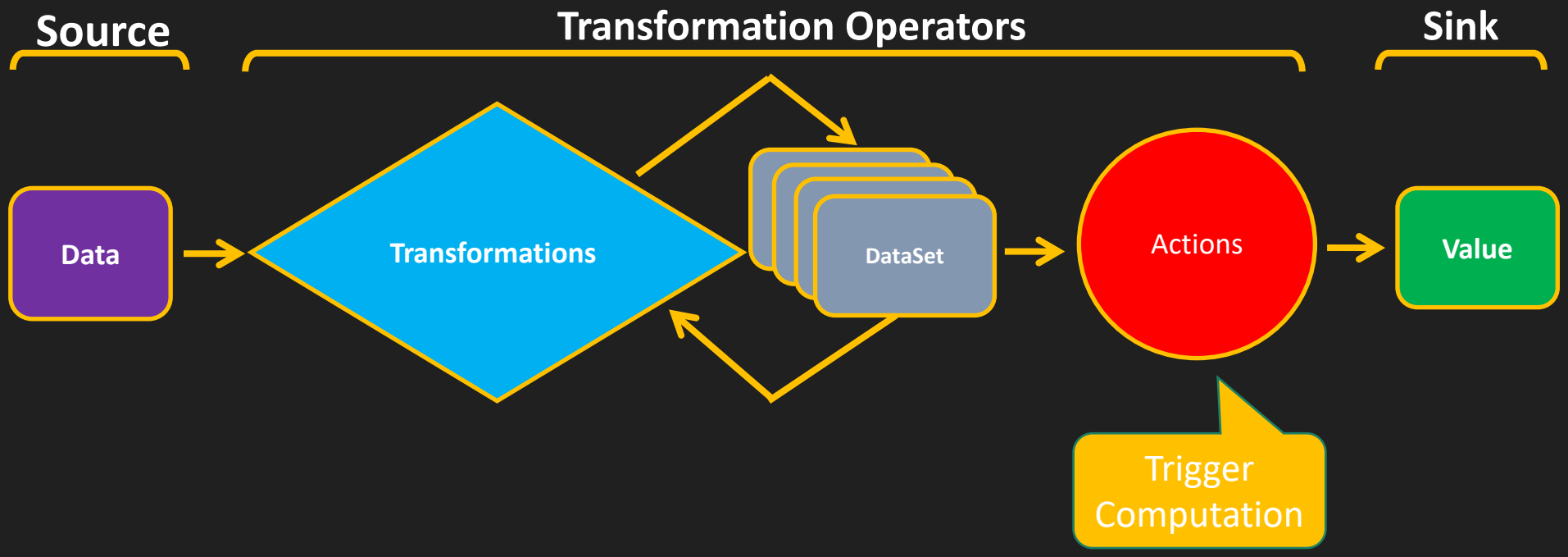

erenavsarogullari


erenavsar@apache.org

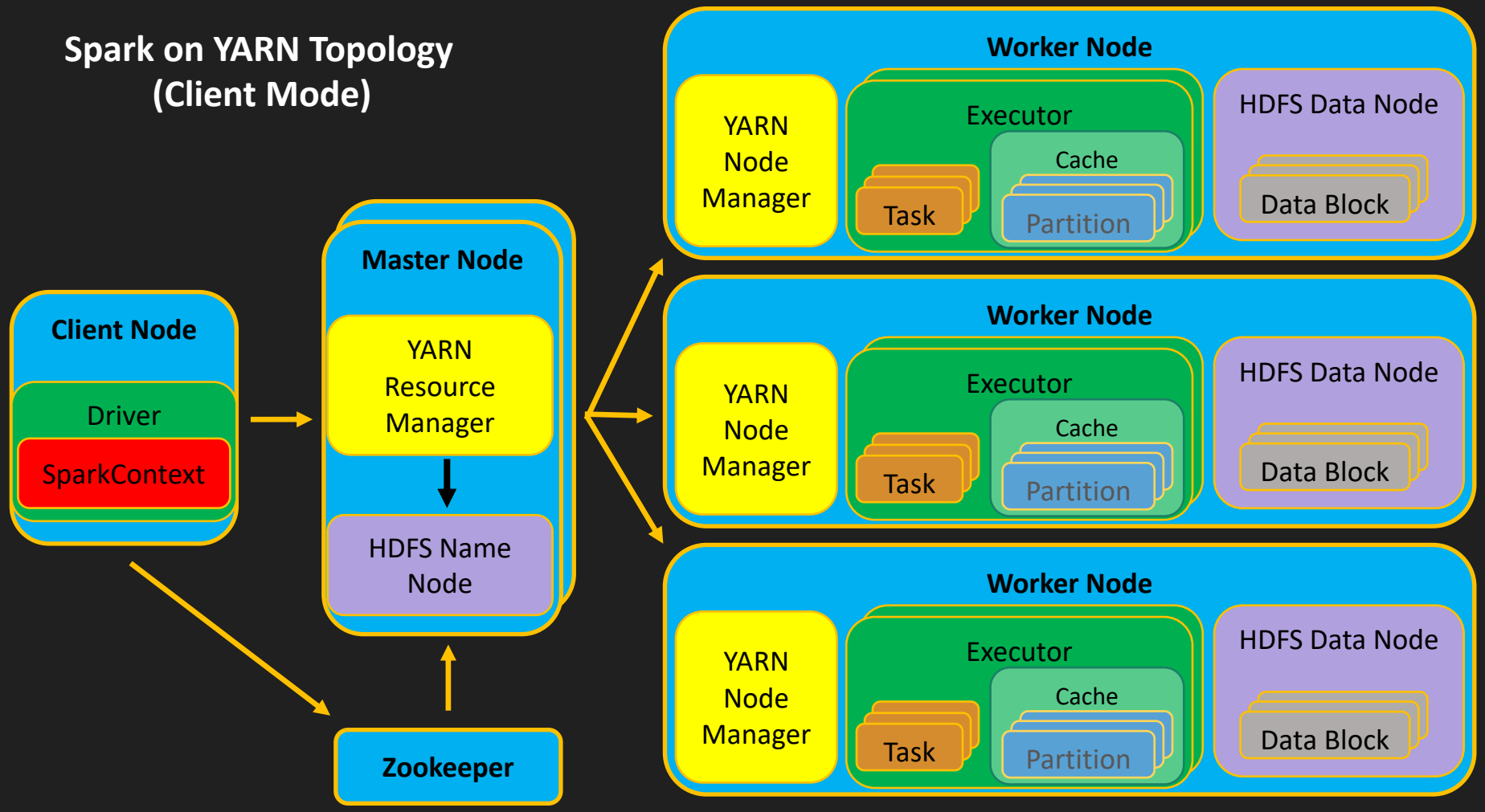
Job Pipeline

Two types of Spark Operations on DataSet:

- **Transformations**: lazy evaluated (not computed immediately)
- **Actions:** triggers the computation and returns value

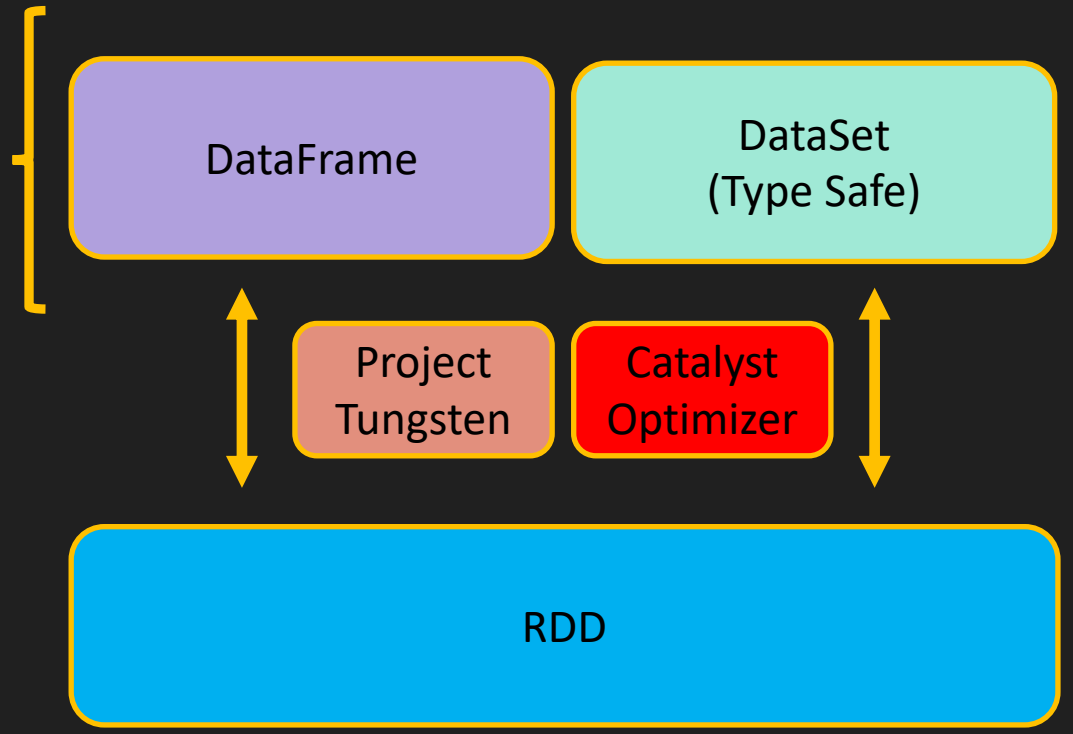


Architecture



1- Data Structures

Tabular Data Structures For **Structured** & **Semi-Structured** Data processing



Takeaway 1
DataFrame or **DataSet APIs** can be preferred against to **RDD** to get more performance efficiency

2- Persistency

```
// Import Packages
import org.apache.spark.sql.Dataset
import spark.implicits._

// Define DataSet Model
case class Employee(id: String, name: String, department: String)

// Create DataSet by loading file
val employeeDS: Dataset[Employee] = spark.read.option("header", true).csv("$FilePath").as[Employee]

// Transform DataSet and Cache
val filteredEmployeeDS = employeeDS.filter(...).map(...).persist()

// Get count of DataSet
filteredEmployeeDS.count()

// Get content of DataSet
filteredEmployeeDS.take(10)
```

Takeaway 2
Caching can be thought before multiple jobs use same DataSet.

Takeaway 3
DataSet needs to be **unpersisted** if it is not used anymore.

Spark uses **LRU** (Least Recently Used) eviction policy

3- Storage Level

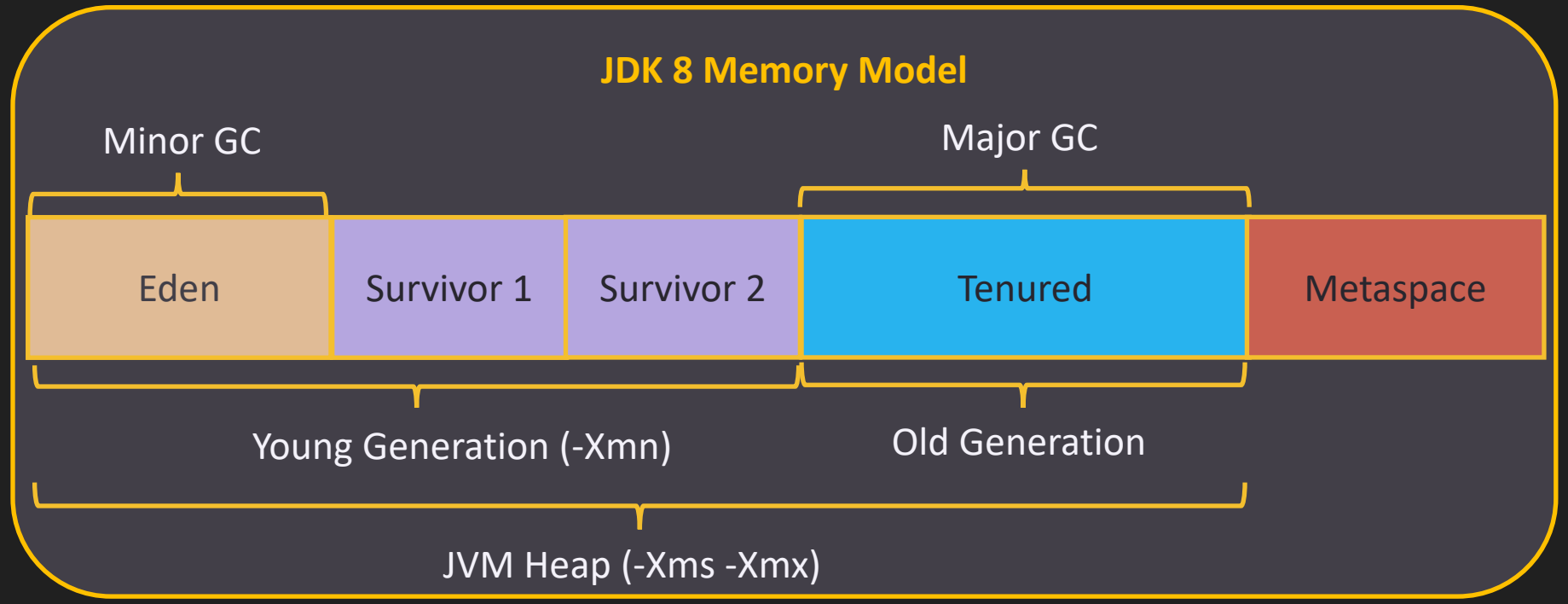
Storage Mode	Description
MEMORY_ONLY	
MEMORY_AND_DISK	
MEMORY_ONLY_SER	
MEMORY_AND_DISK_SER	
DISK_ONLY	
MEMORY_ONLY_2 / MEMORY_AND_DISK_2	
OFF_HEAP	

Takeaway 4
MEMORY_ONLY is the **Best Option** for Prod

Takeaway 5
MEMORY_AND_DISK Option can be thought for **limited environments**

Takeaway 6
MEMORY_ONLY_SER provides **Less memory** usage. However, more **CPU** intensive for **Ser/De**

4- Driver/Executor GC Policy



Takeaway 7
Enable GC Stats first to analyze **JVM Pause Times**

Takeaway 8
G1GC can be preferred when **shorter pause time** and less throughput

ParallelGC can be preferred when longer pause time and **more throughput**

```
// Enable GC Logging
-verbose:gc -XX:+PrintGCDetails  -XX:+PrintGCTimeStamps

// Enable G1GC through SparkConf
spark.executor.extraJavaOptions=-XX:+UseG1GC    // Important when caching the data on executors
spark.driver.extraJavaOptions=-XX:+UseG1GC      // Important  when collecting the data to driver
```

5- Partitioning

3 Factors affecting Partition Size & Number:

- External DataSource (HDFS, Cassandra Table)
- CPU Core Numbers
- Properties (Parallelism Level)

Supported Partitioners:

- Hash Partitioner (**default**)

```
Partition Index = key.hashCode() % numberOfPartitions
```

- Range Partitioner
- Custom Partitioner

Partitioning Operations

- **coalesce**
- **repartition**

Takeaway 9

HDFS Default Partition Size: 128MB (HDFS Data Block Size)
Cassandra Input Split Size: 64MB

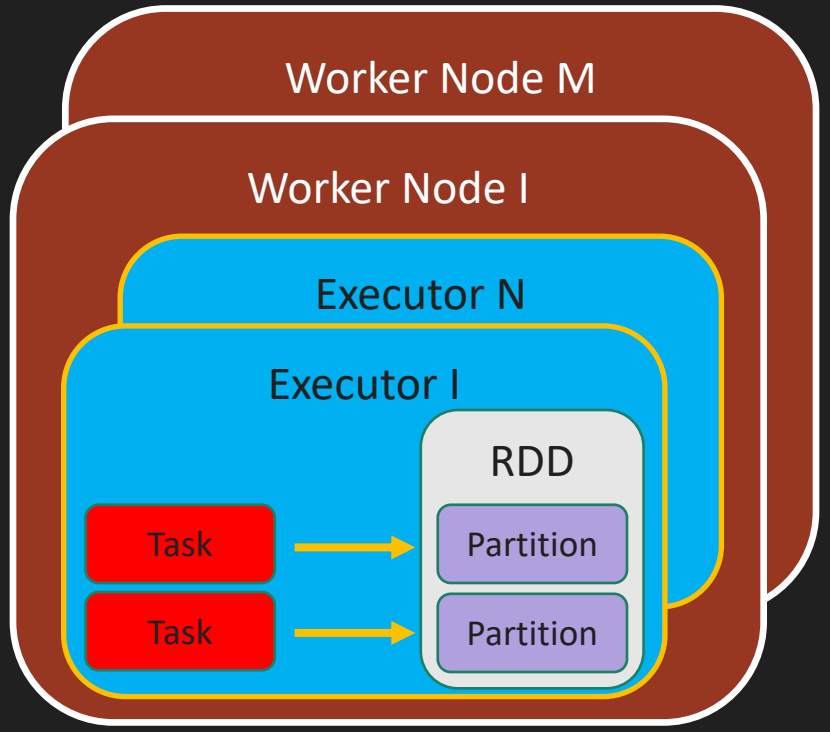
Takeaway 10

Use suitable Partitioner to create well balanced partitions.
Each Partitioner type may not be fit for each dataset.

Takeaway 11

Too many small files: Less efficient with excessive parallelism
Too few large files: Less efficient due to not utilizing all available cores in the cluster.

6- Level of Parallelism



Takeaway 12

Partition Number 1 x 1 Task Number

Max Partitions Number = Total Core Numbers * (2 or 3)

Takeaway 13

spark.default.parallelism affects for **RDD**

spark.sql.shuffle.partitions affects for **DF / DS**

// Default Parallelism

spark.default.parallelism: Default number of partitions in RDDs returned by transformations like **join, reduceByKey** and **parallelize** when not set by user.
Local Mode and **YARN**: All number of CPU cores.

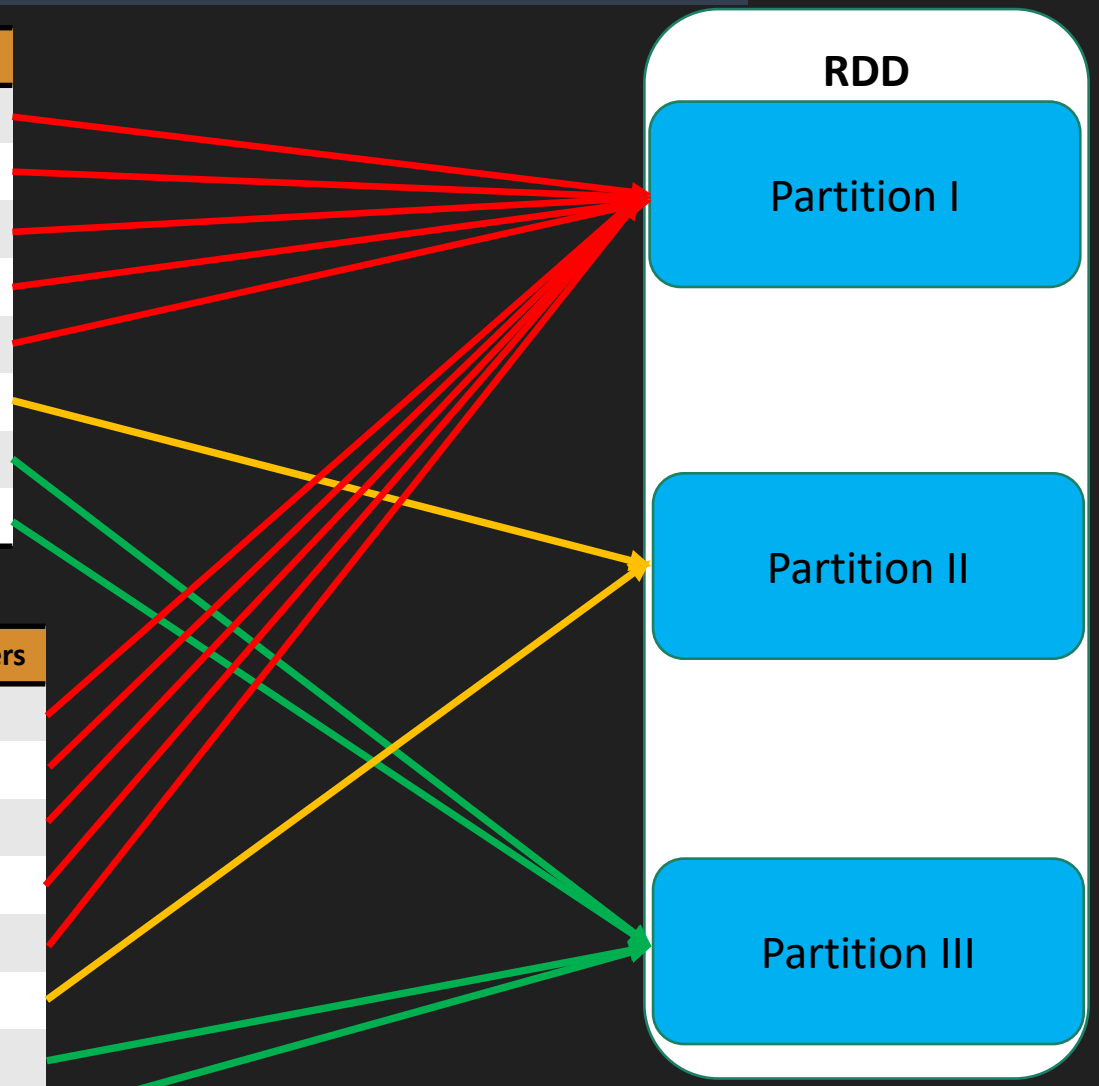
spark.sql.shuffle.partitions: Configures the number of partitions to use when shuffling data for **joins** or **aggregations**. Default is **200**

7- Data Skew I

```
ds1.join(ds2, Seq("Country", "Instrument"))
```

Id	Composer	Country	Instrument	Listeners
1	Paco de Lucia	Spain	Guitar	337K
2	Vincente Amigo	Spain	Guitar	106K
3	Paco Pena	Spain	Guitar	65K
4	Enrique Granados	Spain	Guitar	55K
5	Fernando Sor	Spain	Guitar	63K
6	Johannes Linstead	Canada	Guitar	65K
7	Govi	Germany	Guitar	115K
8	Ottmart Liebert	Germany	Guitar	1K

Composer	Year	Country	Instrument	Listeners
Paco de Lucia	1947	Spain	Guitar	337K
Vincente Amigo	1967	Spain	Guitar	106K
Paco Pena	1942	Spain	Guitar	65K
Enrique Granados	1867	Spain	Guitar	55K
Fernando Sor	1778	Spain	Guitar	63K
Johannes Linstead	1969	Canada	Guitar	65K
Govi	1949	Germany	Guitar	115K
Ottmart Liebert	1959	Germany	Guitar	1K

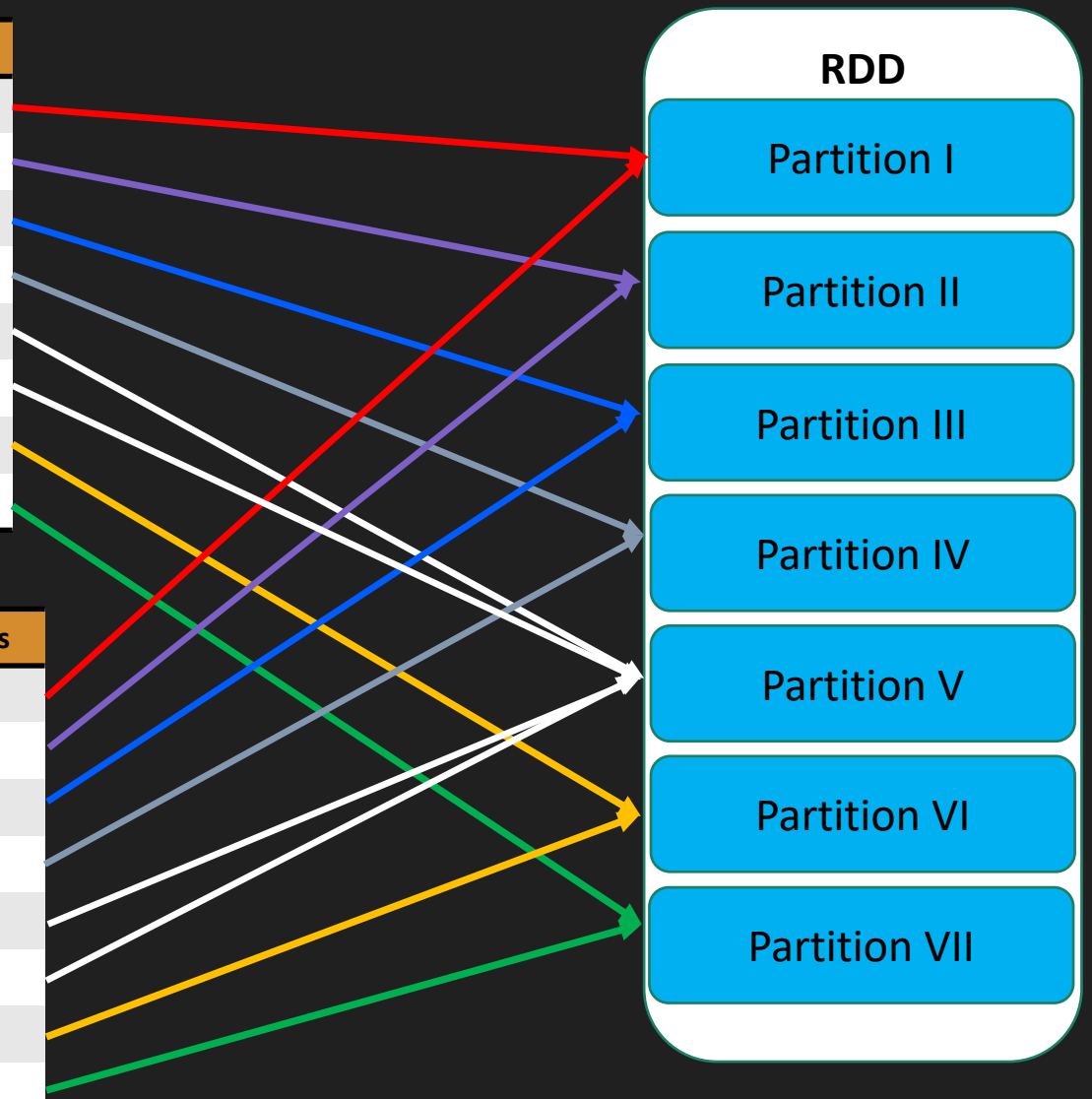


7- Data Skew II

```
ds1.join(ds2, Seq("Country", "Instrument", "Listeners"))
```

Id	Virtuoso	Country	Instrument	Listeners
1	Paco de Lucia	Spain	Guitar	337K
2	Vincente Amigo	Spain	Guitar	106K
3	Fernando Sor	Spain	Guitar	63K
4	Enrique Granados	Spain	Guitar	55K
5	Paco Pena	Spain	Guitar	65K
6	Johannes Linstead	Canada	Guitar	65K
7	Govi	Germany	Guitar	115K
8	Ottmart Liebert	Germany	Guitar	1K

Virtuoso	Year	Country	Instrument	Listeners
Paco de Lucia	1947	Spain	Guitar	337K
Vincente Amigo	1967	Spain	Guitar	106K
Fernando Sor	1942	Spain	Guitar	63K
Enrique Granados	1867	Spain	Guitar	55K
Paco Pena	1778	Spain	Guitar	65K
Johannes Linstead	1969	Canada	Guitar	65K
Govi	1949	Germany	Guitar	115K
Ottmart Liebert	1959	Germany	Guitar	1K



Takeaway 14
Adding more Well Balanced Column or **Salting** can help for well balanced partitioning

8- Data Locality

Data Locality Level	Description
PROCESS_LOCAL	Data is in the same JVM where code is run.
NODE_LOCAL	Data is in the same Node (HDFS Data Node || different executor memory) where executor running the code.
RACK_LOCAL	Data is on the different node in the same Rack
ANY	Data is elsewhere in the network, not in the same Rack.
NO_PREF	No data locality preference

Takeaway 15
Spark runs the code close the data as much as possible

Takeaway 16
PROCESS_LOCAL is the best option. Needs to check when monitoring the job

9- Job Scheduling

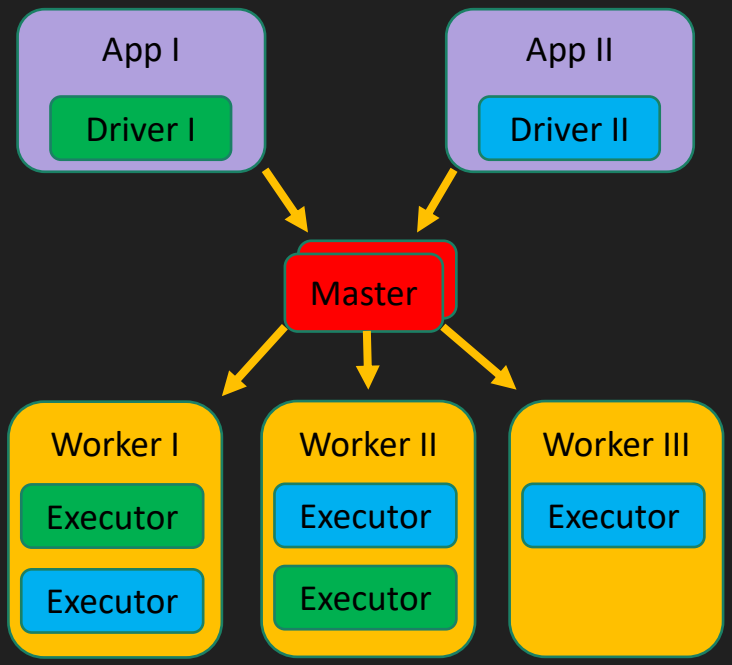
Single Application

- **FIFO (default)**
- **FAIR**

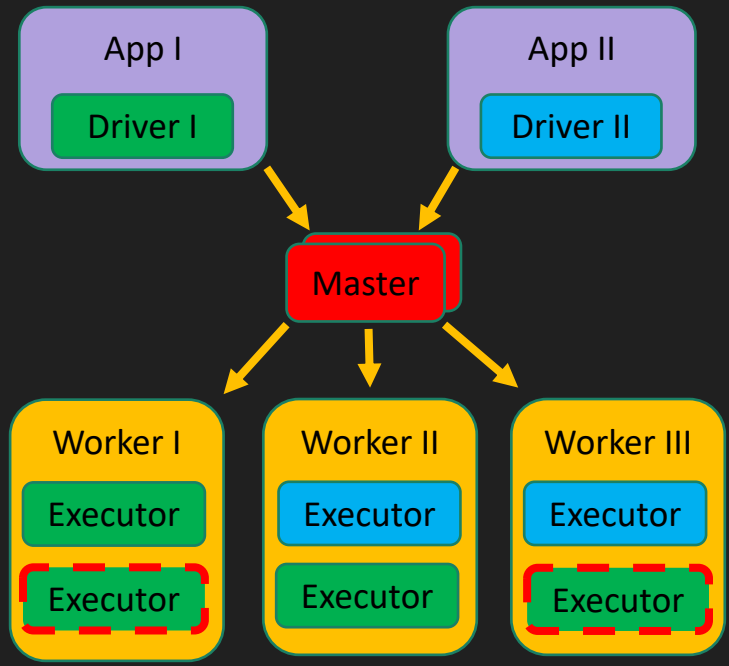
Takeaway 17
FAIR Pool is useful for parallel job submission

Across Application

Static Allocation



Dynamic Allocation



Takeaway 18
Dynamic Allocation can be useful for auto scaling at the runtime in the light of increased/decreased traffic

10- Serialization

Data Structure	Description
RDD	JDK Serialization as default. **Kryo** is suggested. Kryo is not as default because requiring custom registration.
DataFrame / DataSet	Tungsten serialization format is used.

Takeaway 19
Kryo can be preferred against to JDK IO Serialization when using RDD.

When is used?

- **Shuffle**: Data transferring between Worker Nodes
- **Disk I/O:** Data is being written to disk
- **Memory:** Data is being stored to Heap in serialized form.

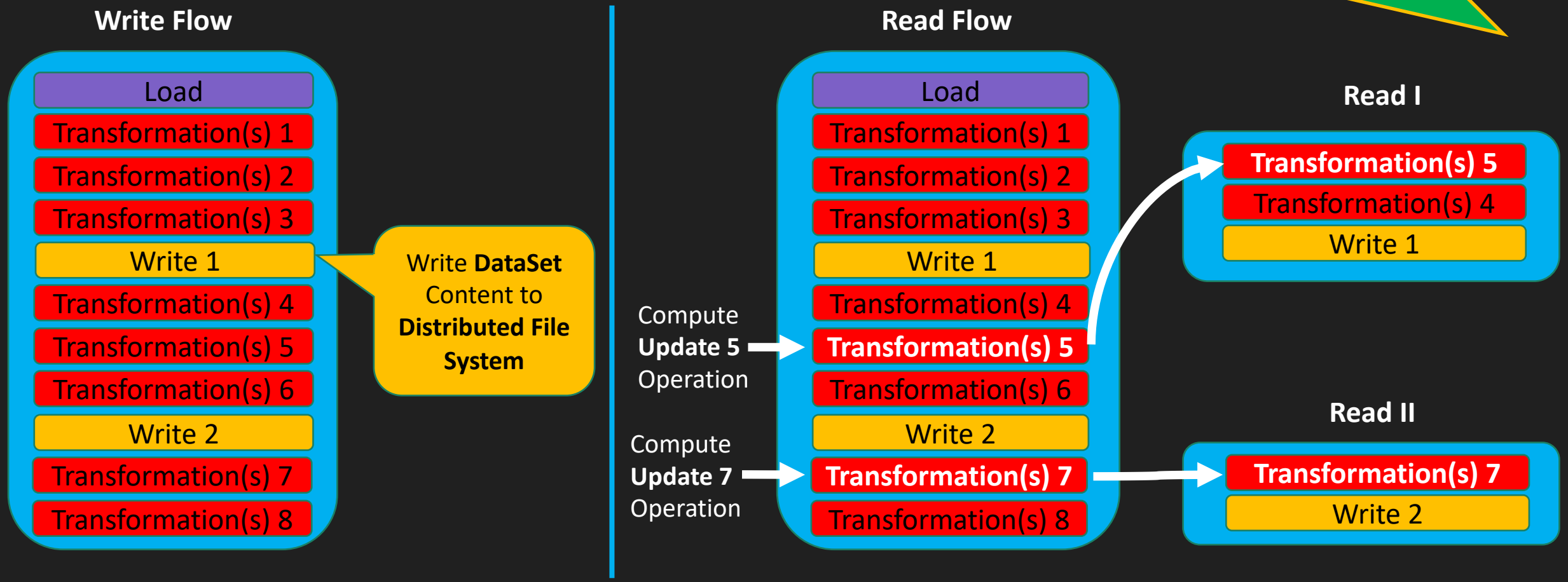
11- Event Sourcing on Transformations

Microservices Data Management Patterns

- **CQRS (*Command Query Responsibility Segregation*)**
- **Event Sourcing**

Takeaway 20

RDDs are **immutable**. **Event Sourcing** is useful pattern at serving layer to **keep RDD audit history** and **computing historical data.**



12- Checkpointing

Checkpointing is a technique for fault tolerance and performance efficiency in computing systems.

- No need re-computation of complex Spark jobs requiring **shuffle**
  (e.g: Complex Join, GroupBy, Sort requiring shuffle)
- Help for re-computation of required DataSet fastly.
  (e.g: Single and Batch Transformations)
- Fault Tolerance (System can re-continue from failure point)

References

- https://spark.apache.org/docs/latest/
- https://jaceklaskowski.gitbooks.io/mastering-apache-spark/
- https://stackoverflow.com/questions/36215672/spark-yarn-architecture
- https://microservices.io/patterns/data/event-sourcing.html
- https://coxautomotivedatasolutions.github.io/datadriven/spark/data%20skew/joins/data_skew/

Q & A

?

Thanks