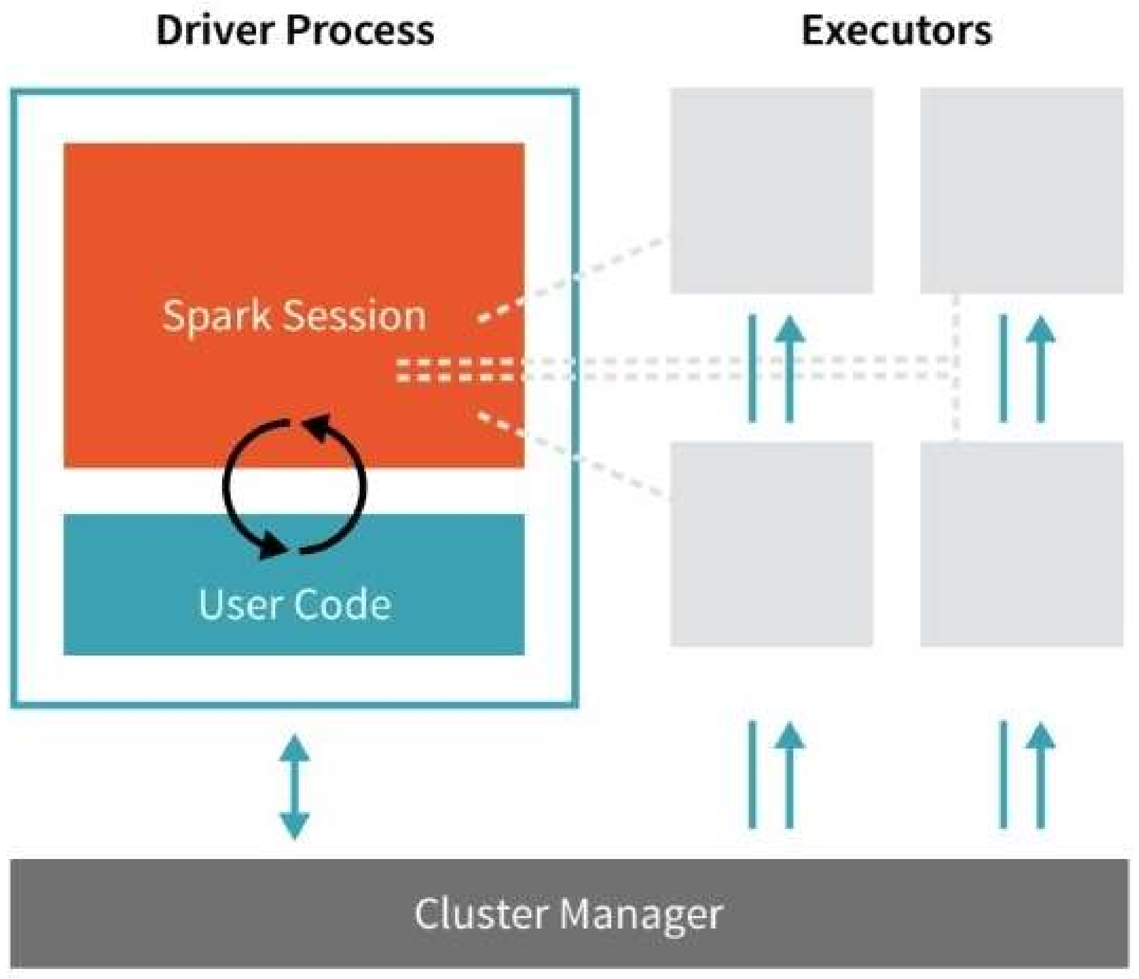
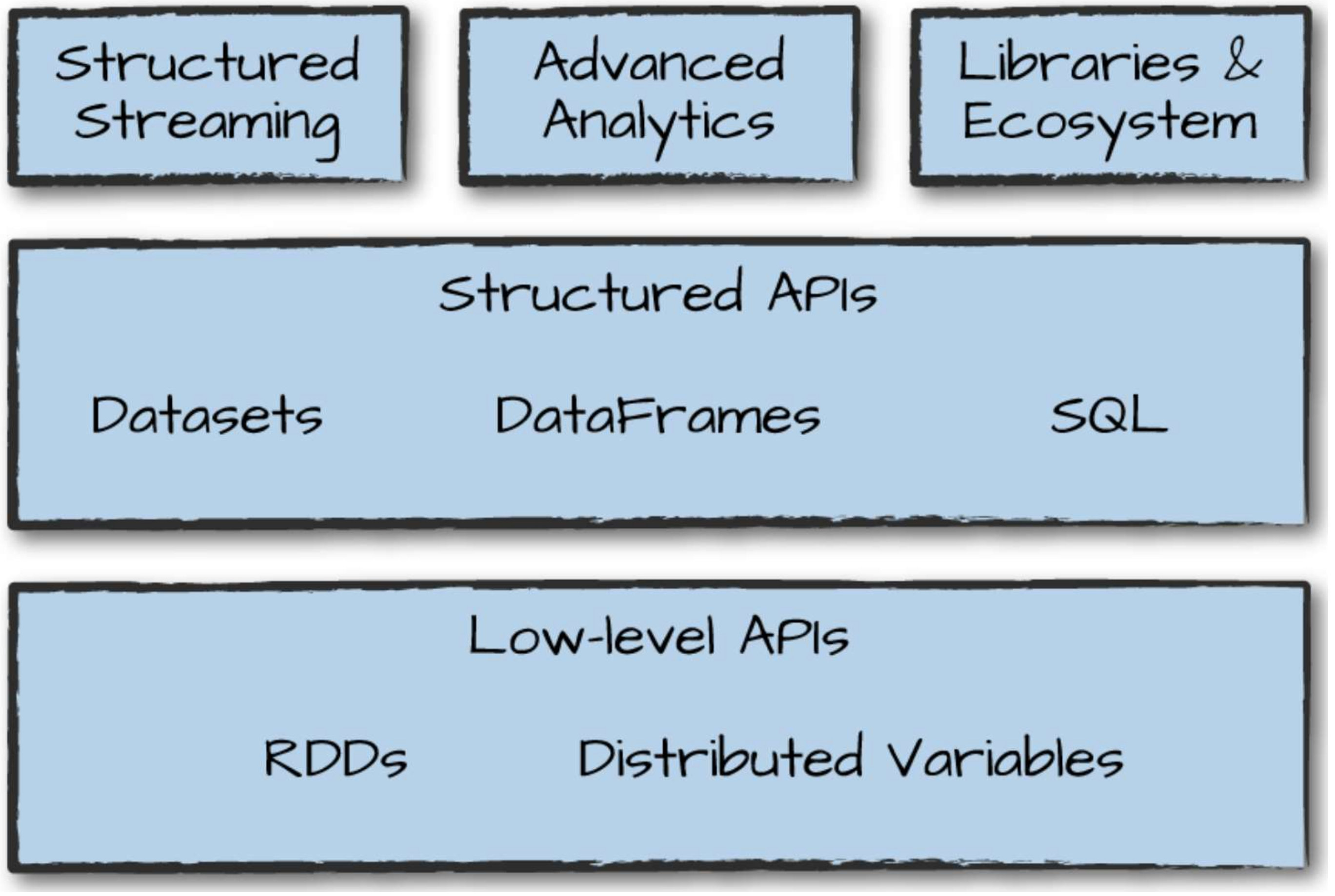
An Introduction to Apache Spark

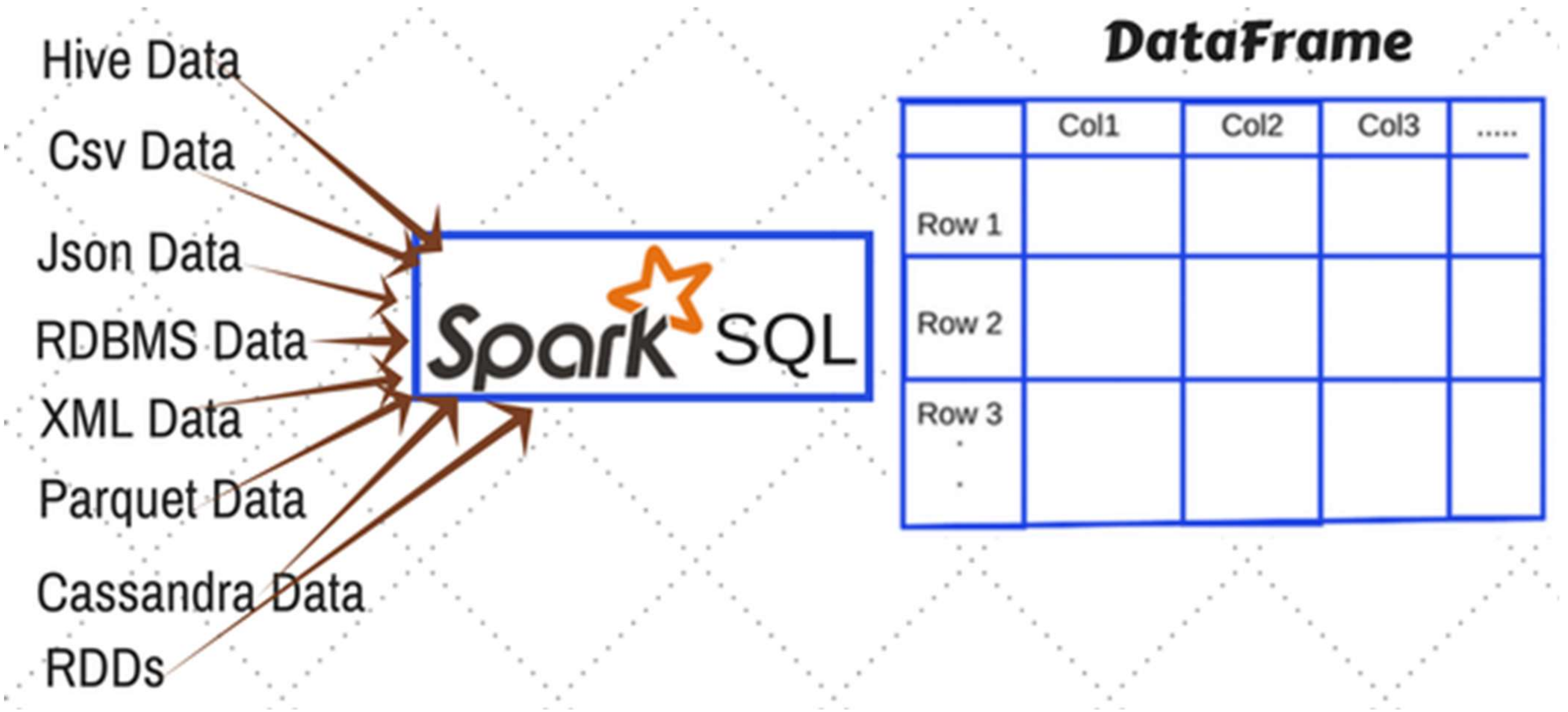
Spark Architecture



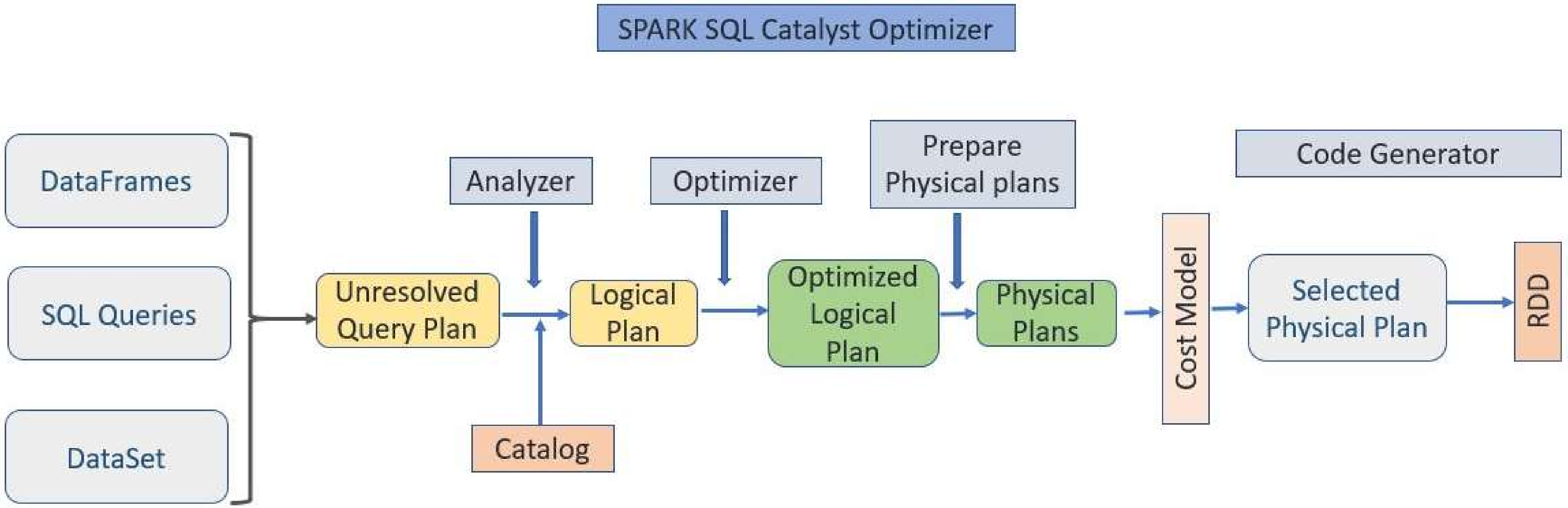
Spark's Language APIs



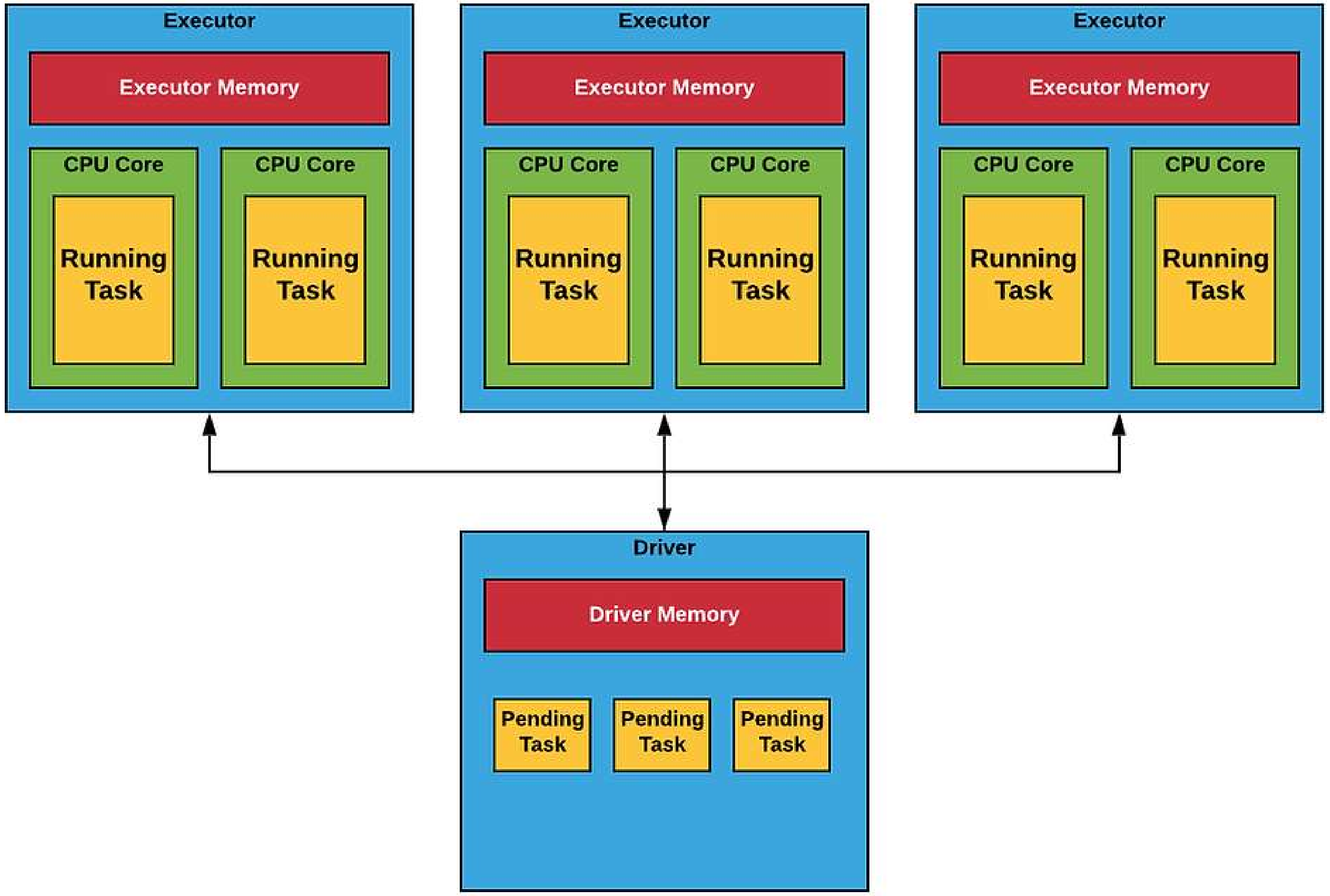
Ways to create DataFrame in Spark



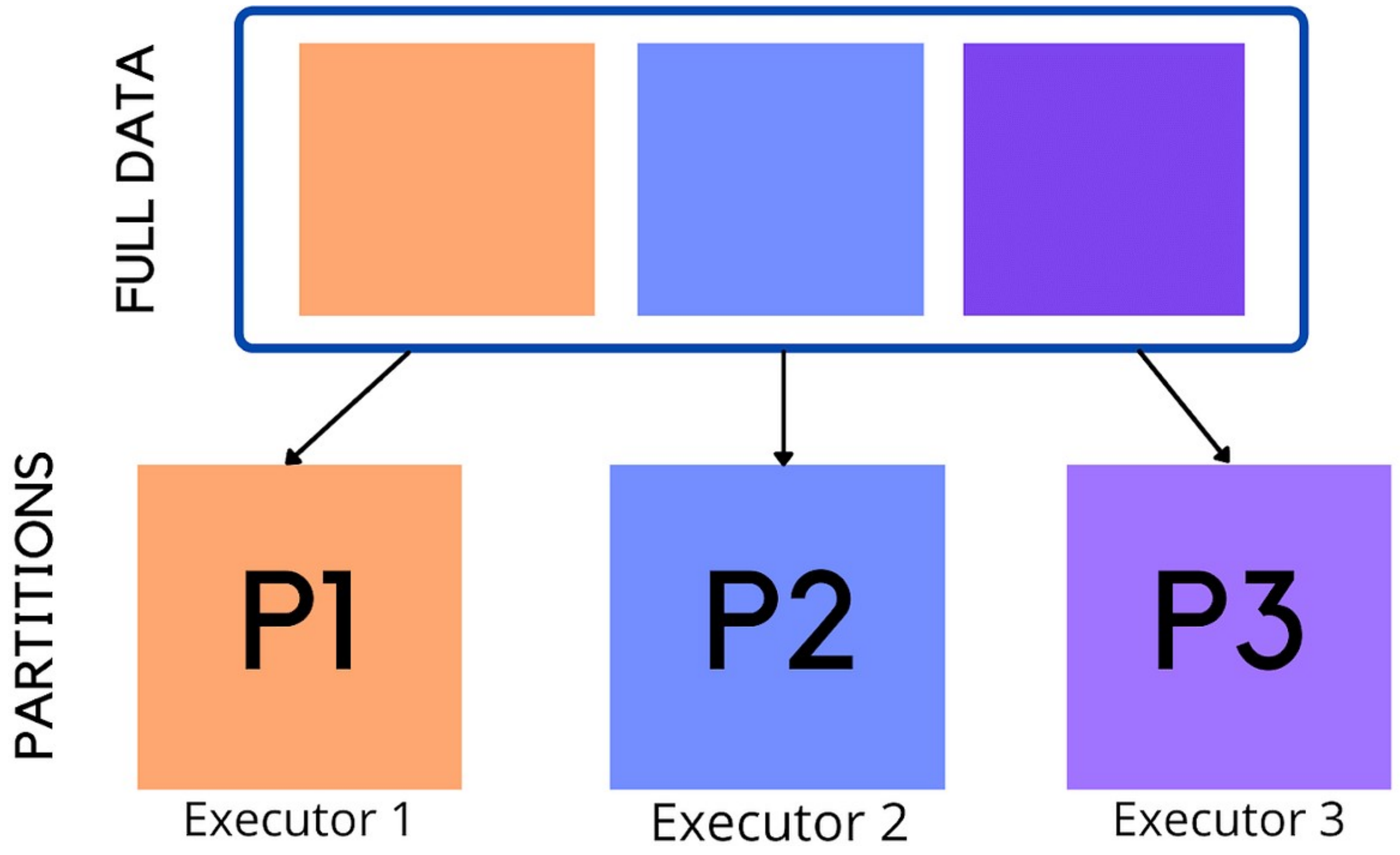
Spark Optimization



Spark Executor / Cores / Memory



Partition and Executor

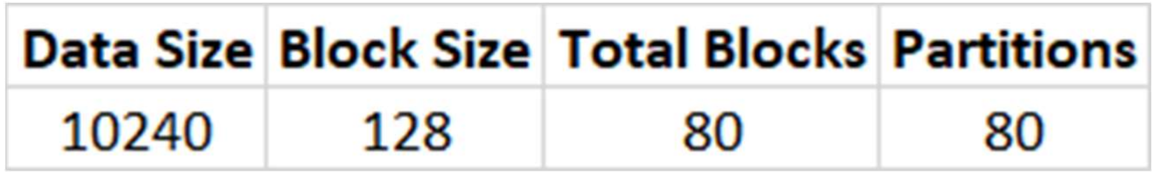


Repartitioning

- The data in the cluster will be split.
- Repartition involves a full shuffle
- Coalesce reduces the number of partitions and avoids a full shuffle

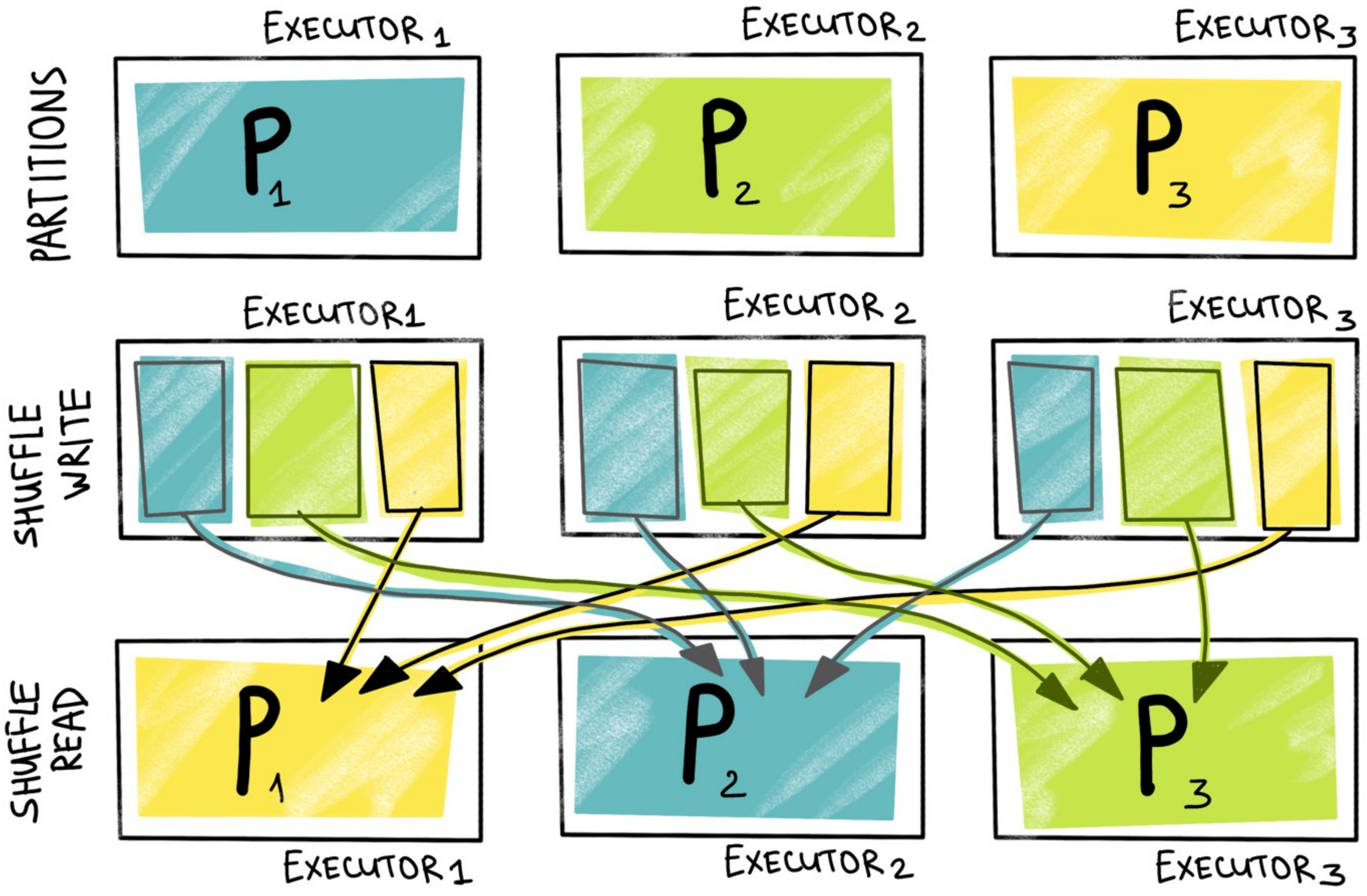
Partitioning on input stage

- When Spark reads a file from HDFS, it creates a single partition for a single input split



Data Size	Block Size	Total Blocks	Partitions
10240	128	80	80

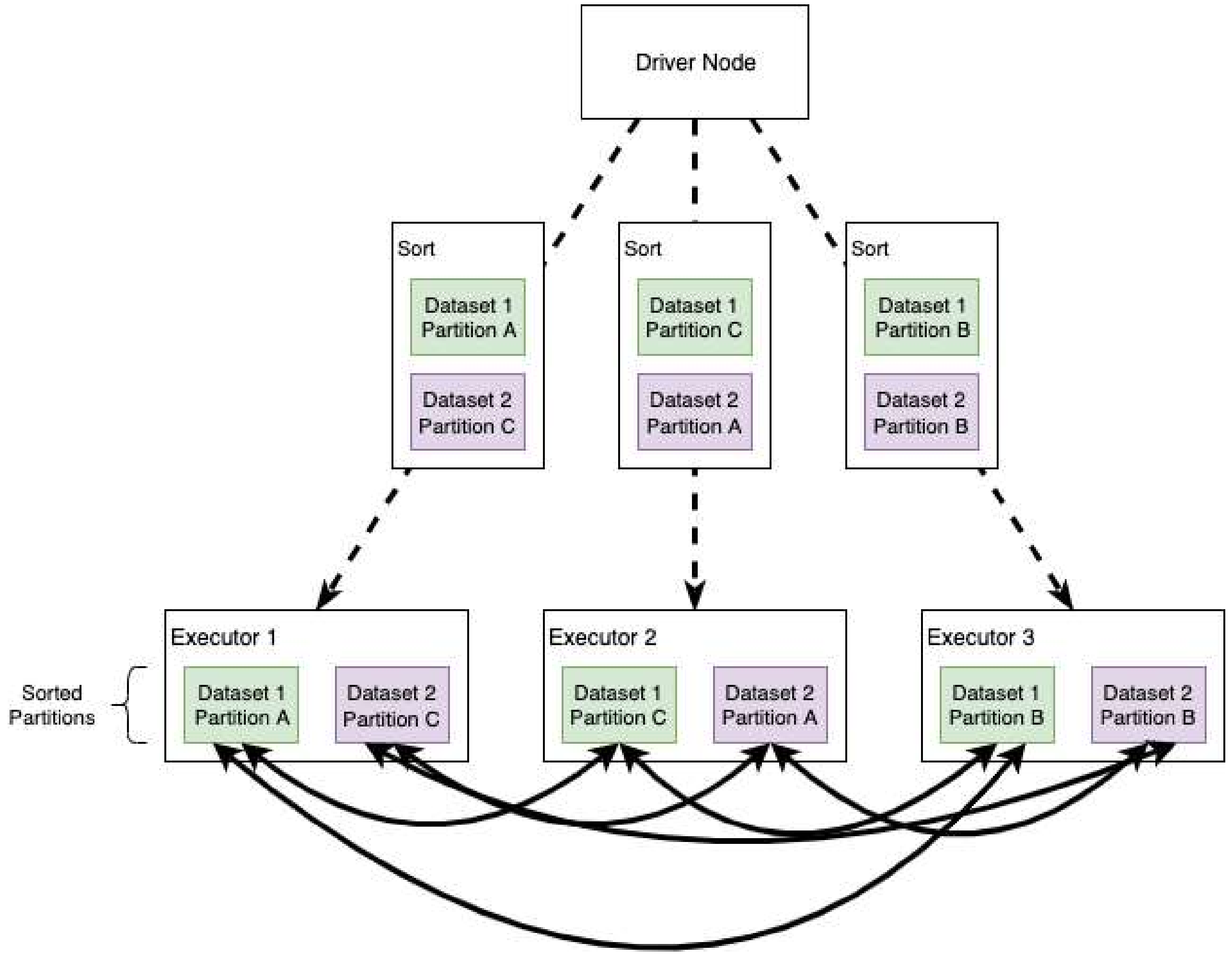
Shuffle partitioning



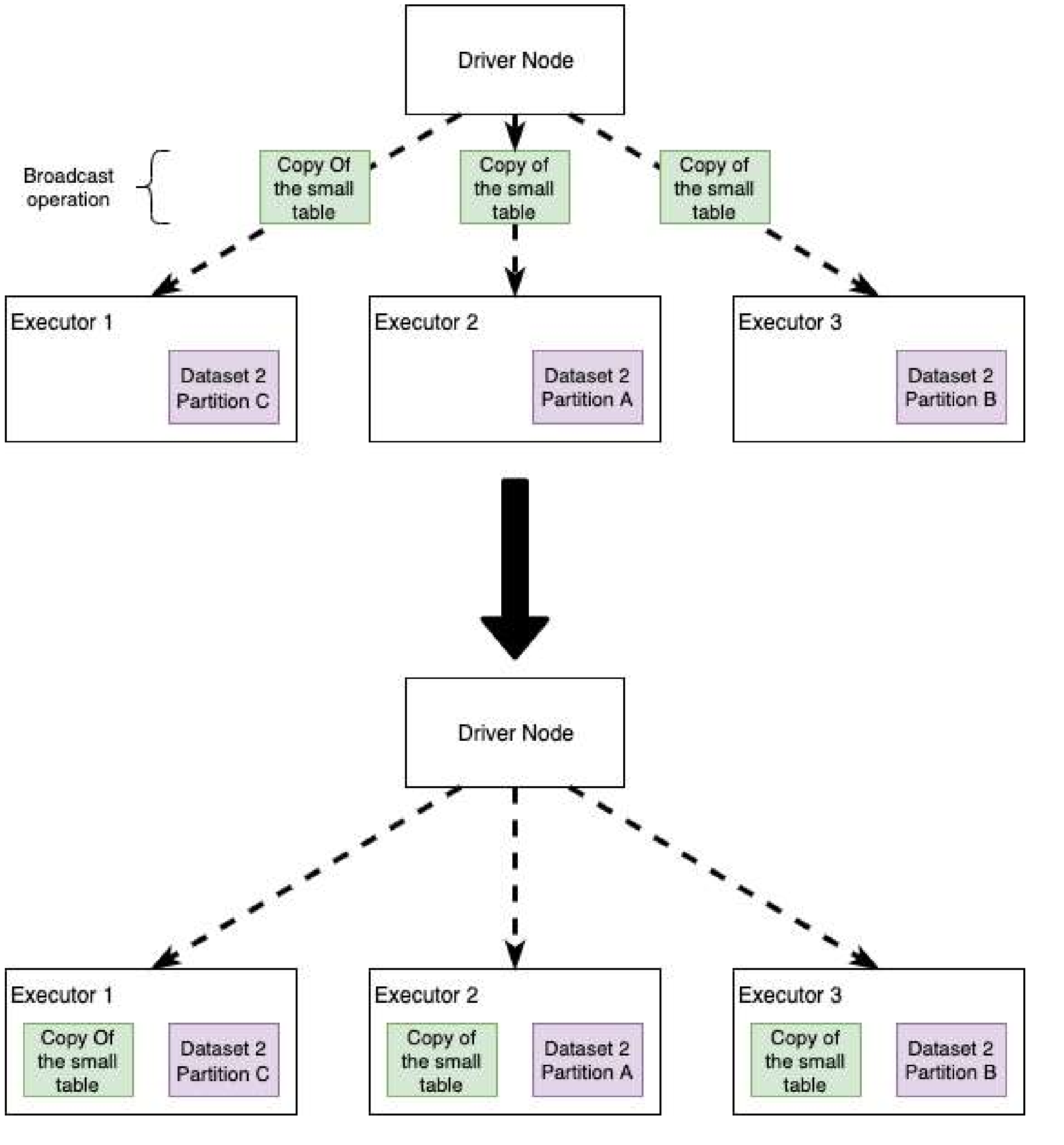
Output partitioning

- Saving the partitioned data on the properly selected condition can significantly speed up the reading and retrieval of data

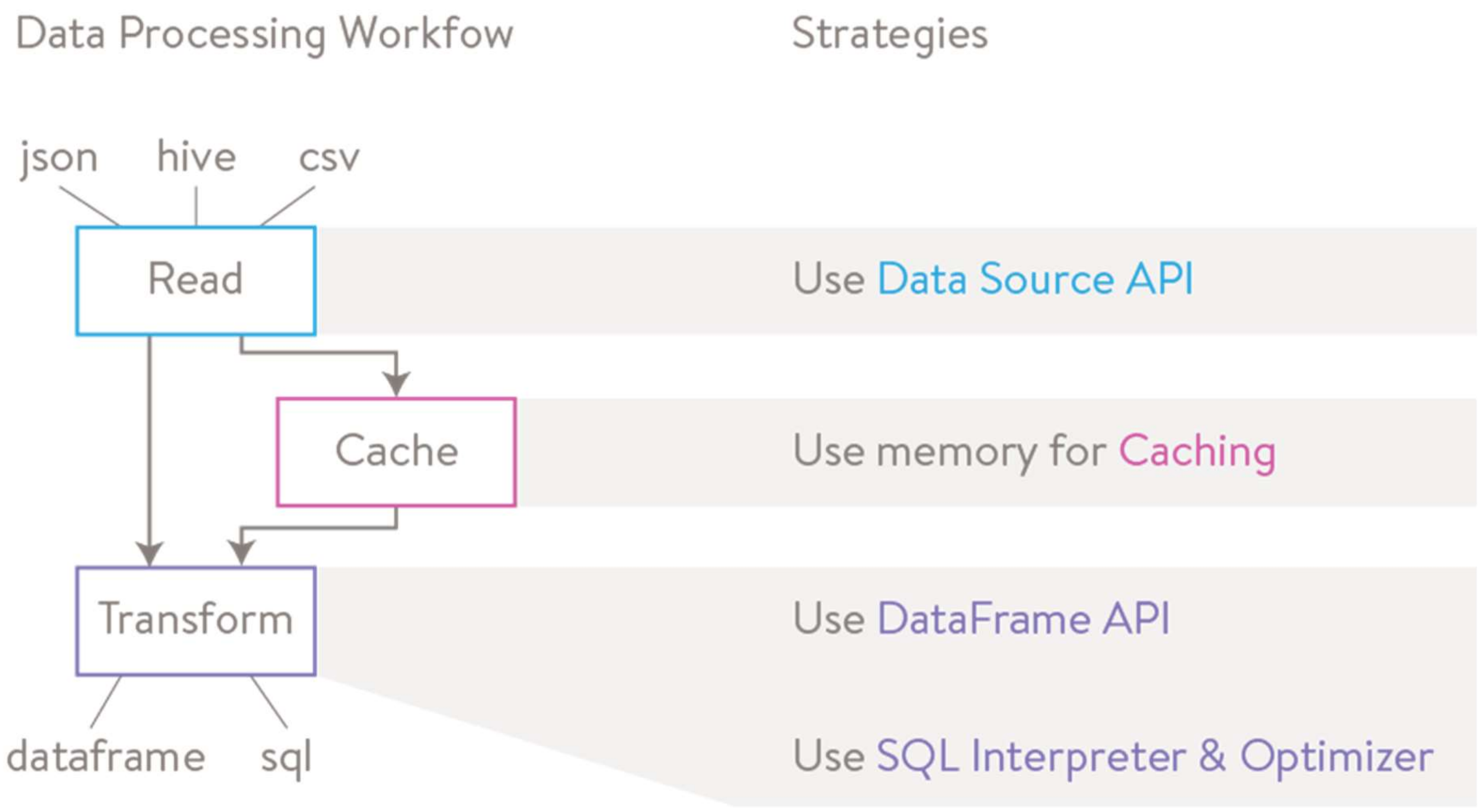
Sort Merge Join



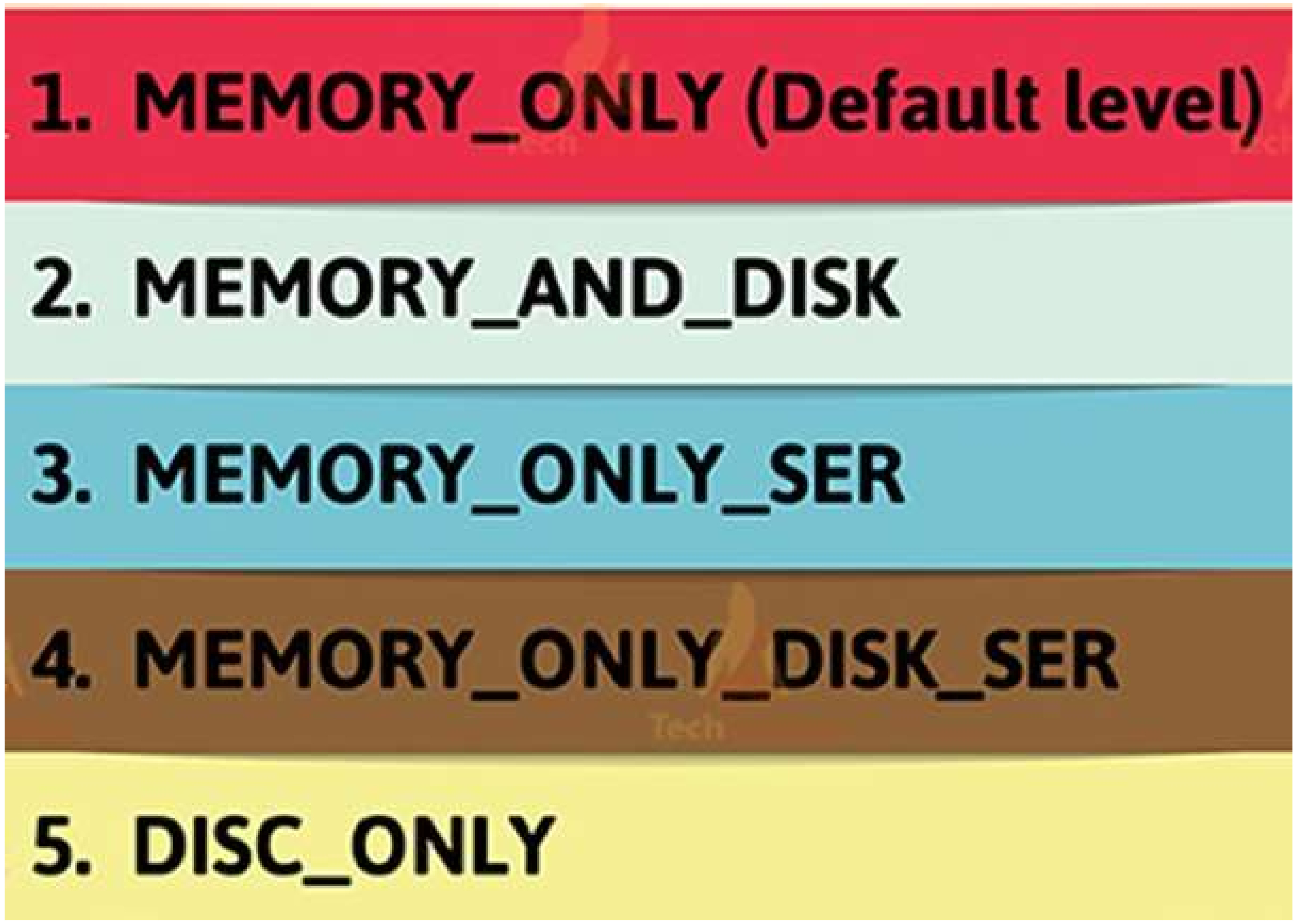
Broadcast Join



Caching

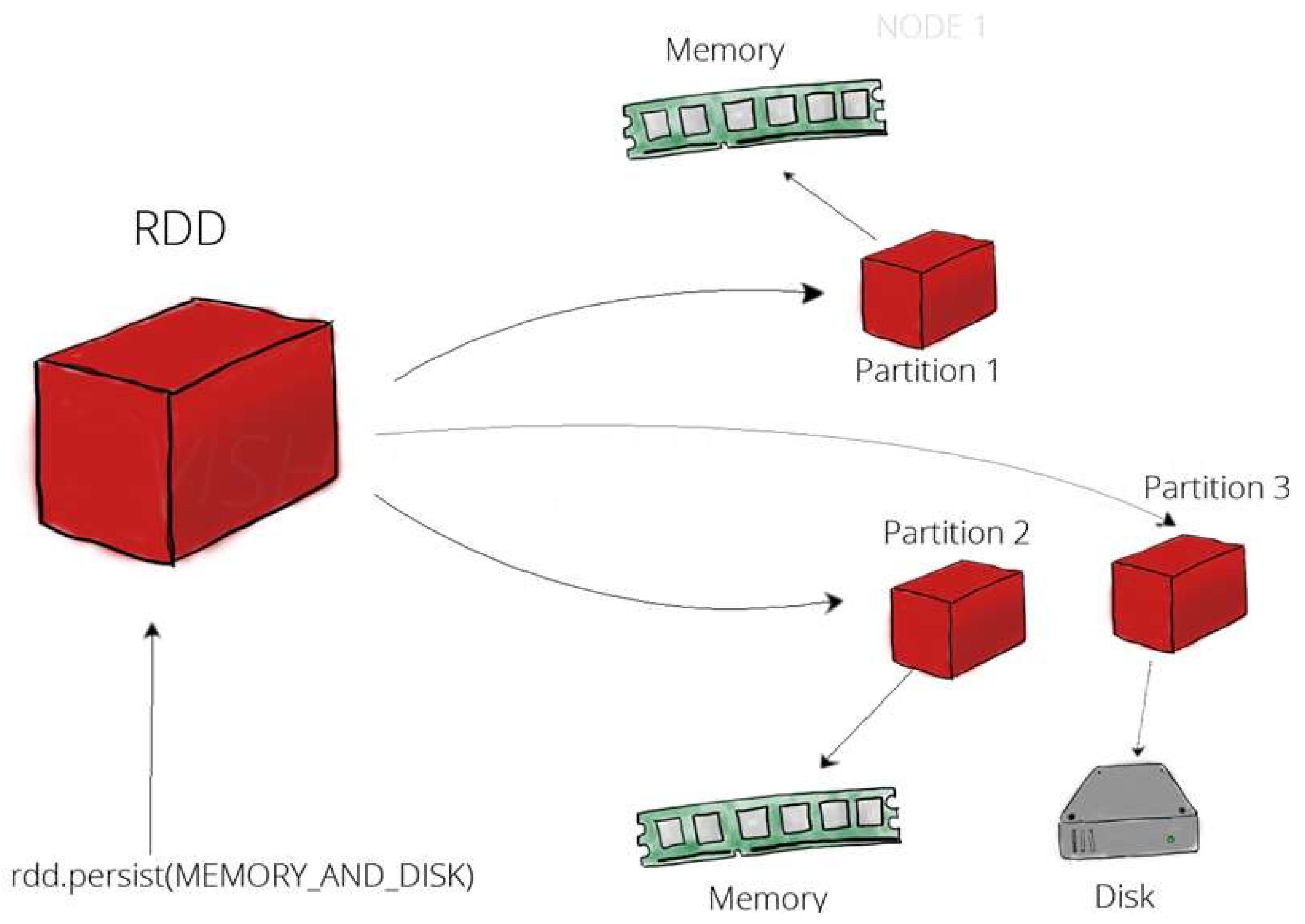

Data Processing Workfow
Strategies
json
hive
csv
Read
Use Data Source API
Cache
Use memory for Caching
Transform
Use DataFrame API
dataframe
sql
Use SQL Interpreter & Optimizer

Caching Storage Levels



1. MEMORY_ONLY (Default level)
2. MEMORY_AND_DISK
3. MEMORY_ONLY_SER
4. MEMORY_ONLY_DISK_SER
5. DISC_ONLY

MEMORY_AND_DISK



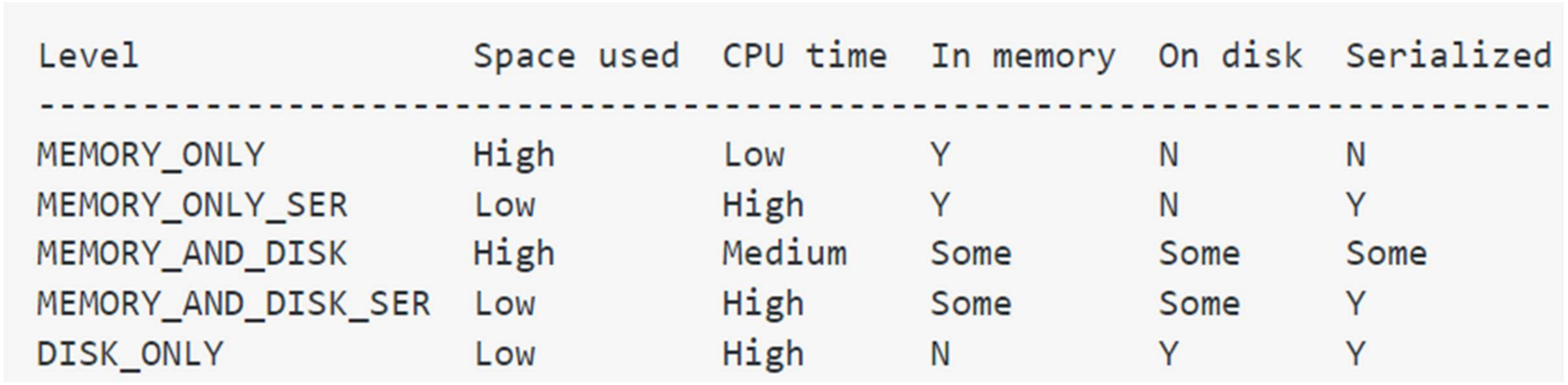
MEMORY_ONLY_SER

- Store RDD as serialized Java objects
- More space-efficient

MEMORY_AND_DISK_SER

- Spill partitions that don't fit in memory to disk

Caching Storage Levels



```
Level                Space used  CPU time  In memory  On disk  Serialized
-------------------------------------------------------------------------
MEMORY_ONLY          High        Low       Y          N        N
MEMORY_ONLY_SER      Low         High      Y          N        Y
MEMORY_AND_DISK      High        Medium    Some       Some     Some
MEMORY_AND_DISK_SER  Low         High      Some       Some     Y
DISK_ONLY            Low         High      N          Y        Y
```

Unpersist RDD

- Spark drop out the old data partition in the LRU (least recently used) fashion.
  - LRU is an algorithm which ensures the least frequently used data
- Can also remove the cache manually using:
  - .unpersist()

Which Storage Level to Choose?

- If data fit comfortably with the default storage level (MEMORY_ONLY), leave them that way.
- If not, try using MEMORY_ONLY_SER
- Use the replicated storage levels if you want fast fault recovery

Thanks