МегаФон

# Hadoop.
# Apache Spark.

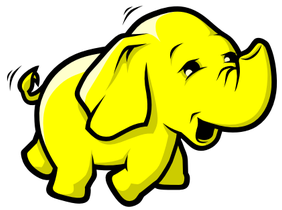
# СУБД или Hadoop?

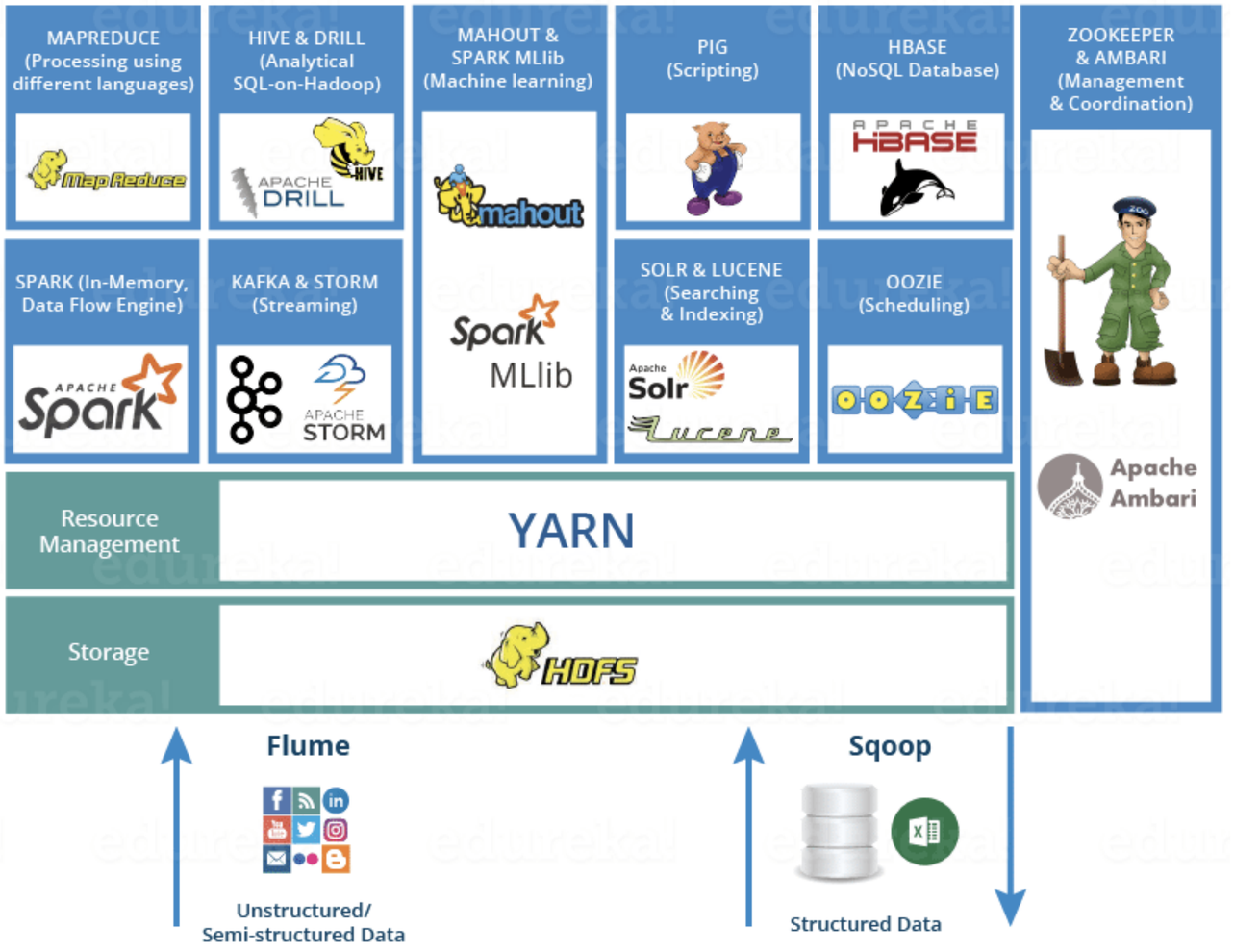
# История Hadoop

# Что такое Hadoop и Apache Spark?

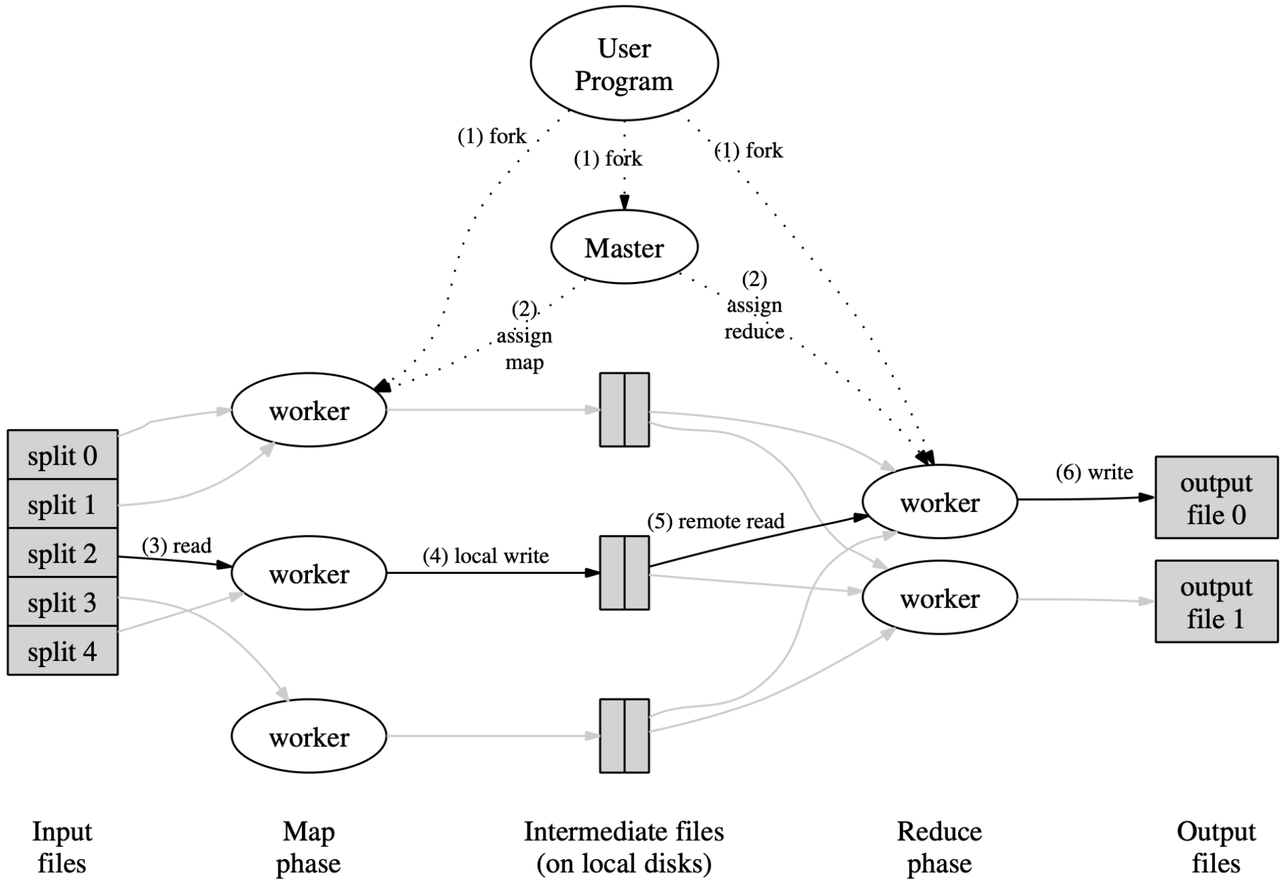
**Hadoop** можно рассматривать как экосистему, включающую в себя набор программ для параллельной обработки больших данных. Разработан в рамках вычислительной парадигмы **MapReduce**, согласно которой приложение разделяется на большое количество одинаковых элементарных заданий, выполнимых на узлах кластера и естественным образом сводимых в конечный результат.

# Hadoop Ecosystem

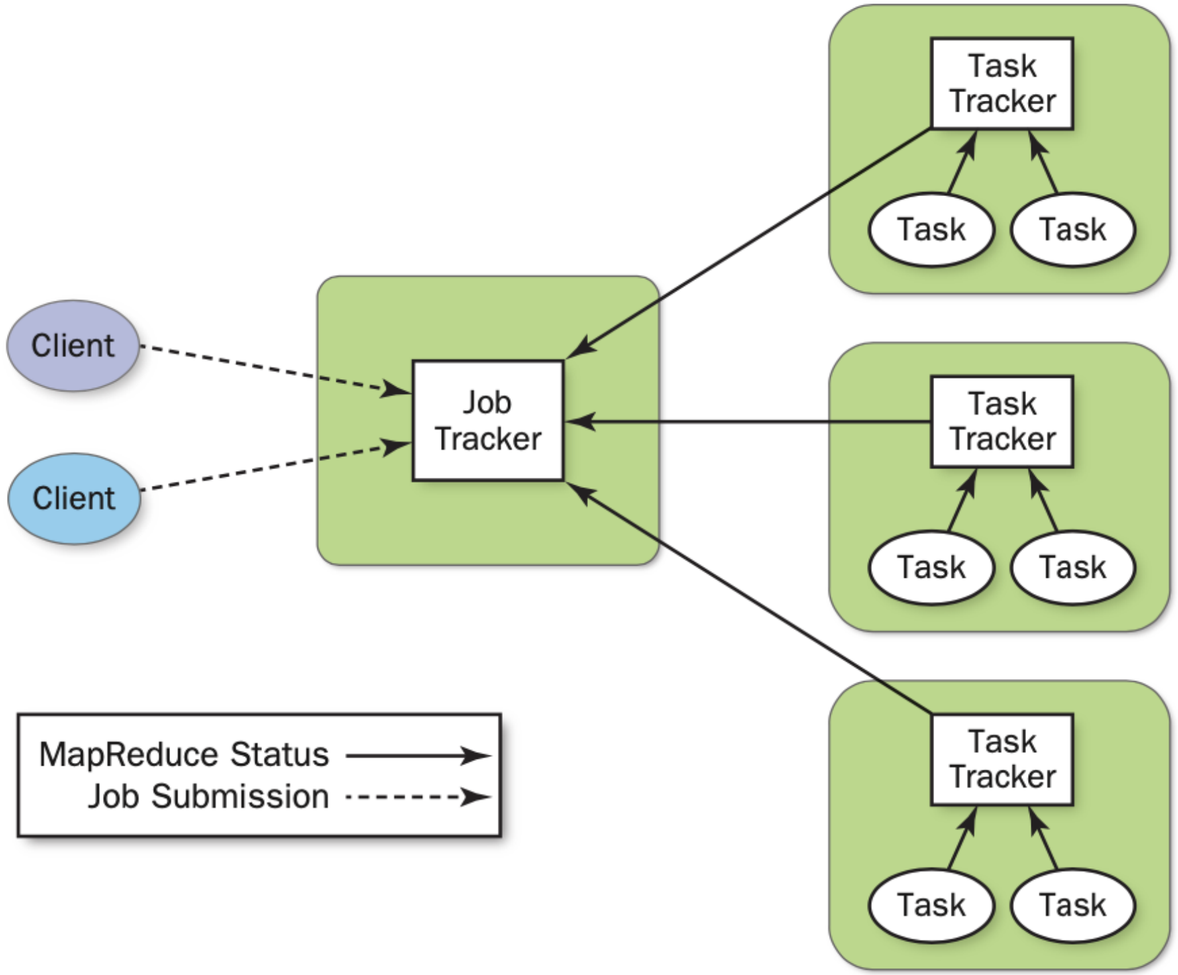



MAPREDUCE (Processing using different languages)
HIVE & DRILL (Analytical SQL-on-Hadoop)
MAHOUT & SPARK MLlib (Machine learning)
PIG (Scripting)
HBASE (NoSQL Database)
ZOOKEEPER & AMBARI (Management & Coordination)
SPARK (In-Memory, Data Flow Engine)
KAFKA & STORM (Streaming)
SOLR & LUCENE (Searching & Indexing)
OOZIE (Scheduling)
Resource Management
YARN
Storage
HDFS
Flume
Unstructured/ Semi-structured Data
Sqoop
Structured Data

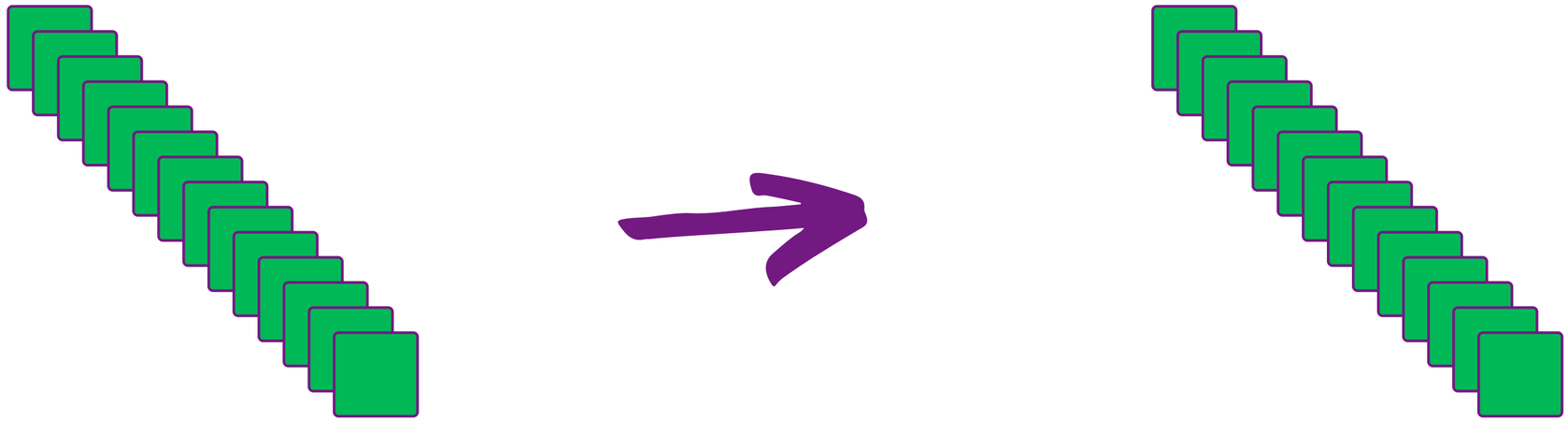
# MapReduce




Jeffrey Dean, Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. Proceedings of OSDI, 2004.


Client
Client
Job Tracker
Task Tracker
Task
Task
Task Tracker
Task
Task
Task Tracker
Task
Task
MapReduce Status
Job Submission

# Функция Map

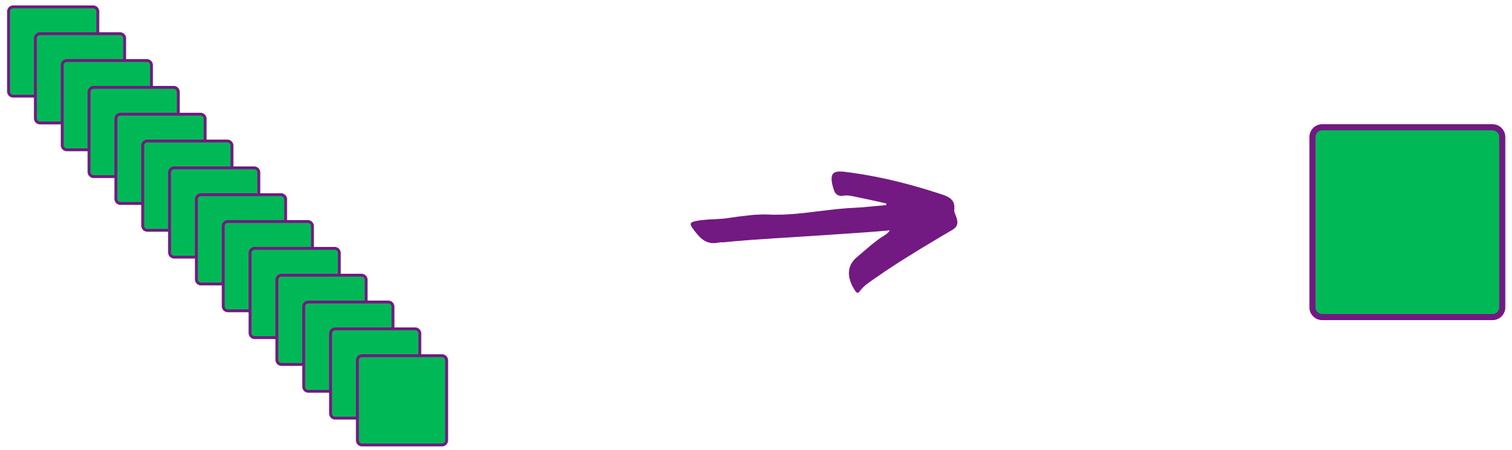


```
function map(String name, String document):
  // name: document name
  // document: document contents
  for each word w in document:
                emit (w, 1)
```

# Mapper.py

```
#!/usr/bin/env python
import sys

for line in sys.stdin:
        line = line.strip()
        words = line.split()
        for word in words:
                print '%s\t%s' % (word, 1)
```

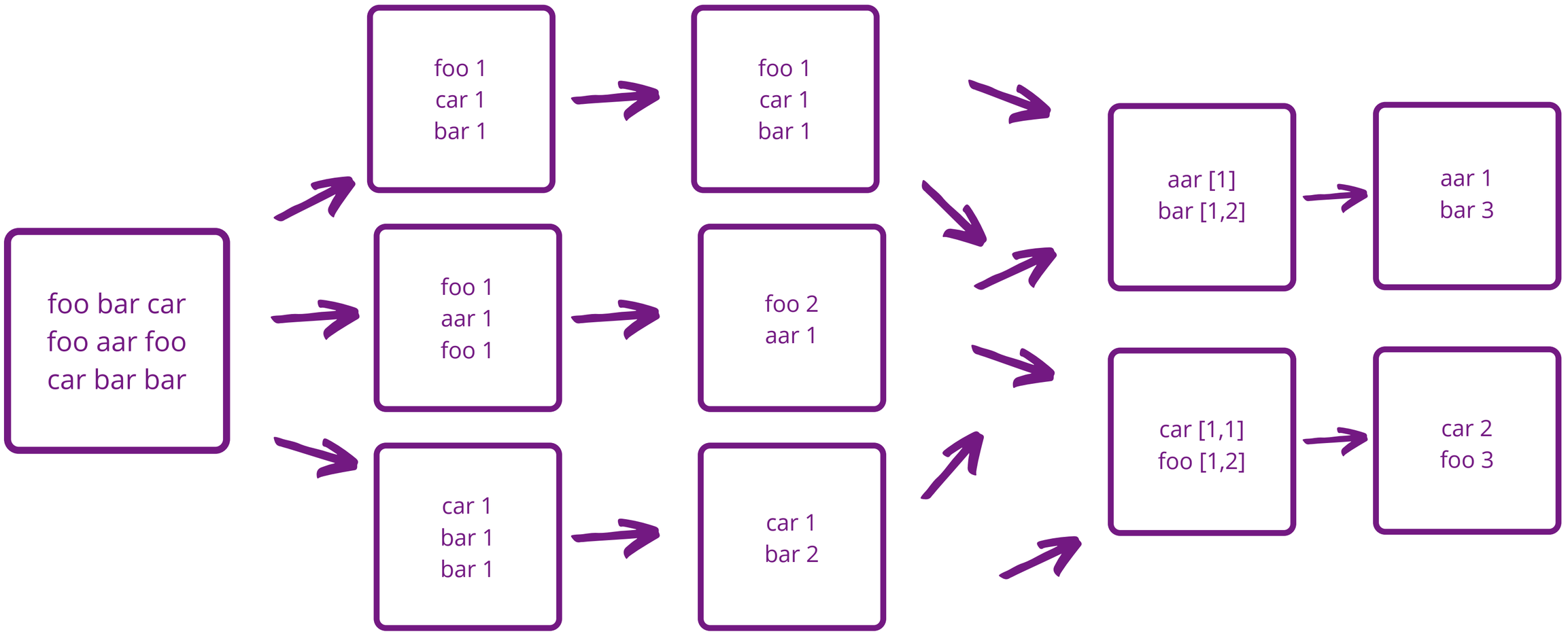
# Функция Reduce



```
function reduce(String word, Iterator partialCounts):
  // word: a word
  // partialCounts: a list of aggregated partial counts
  sum = 0
  for each pc in partialCounts:
    sum += pc
                emit (word, sum)
```

# Reducer.py

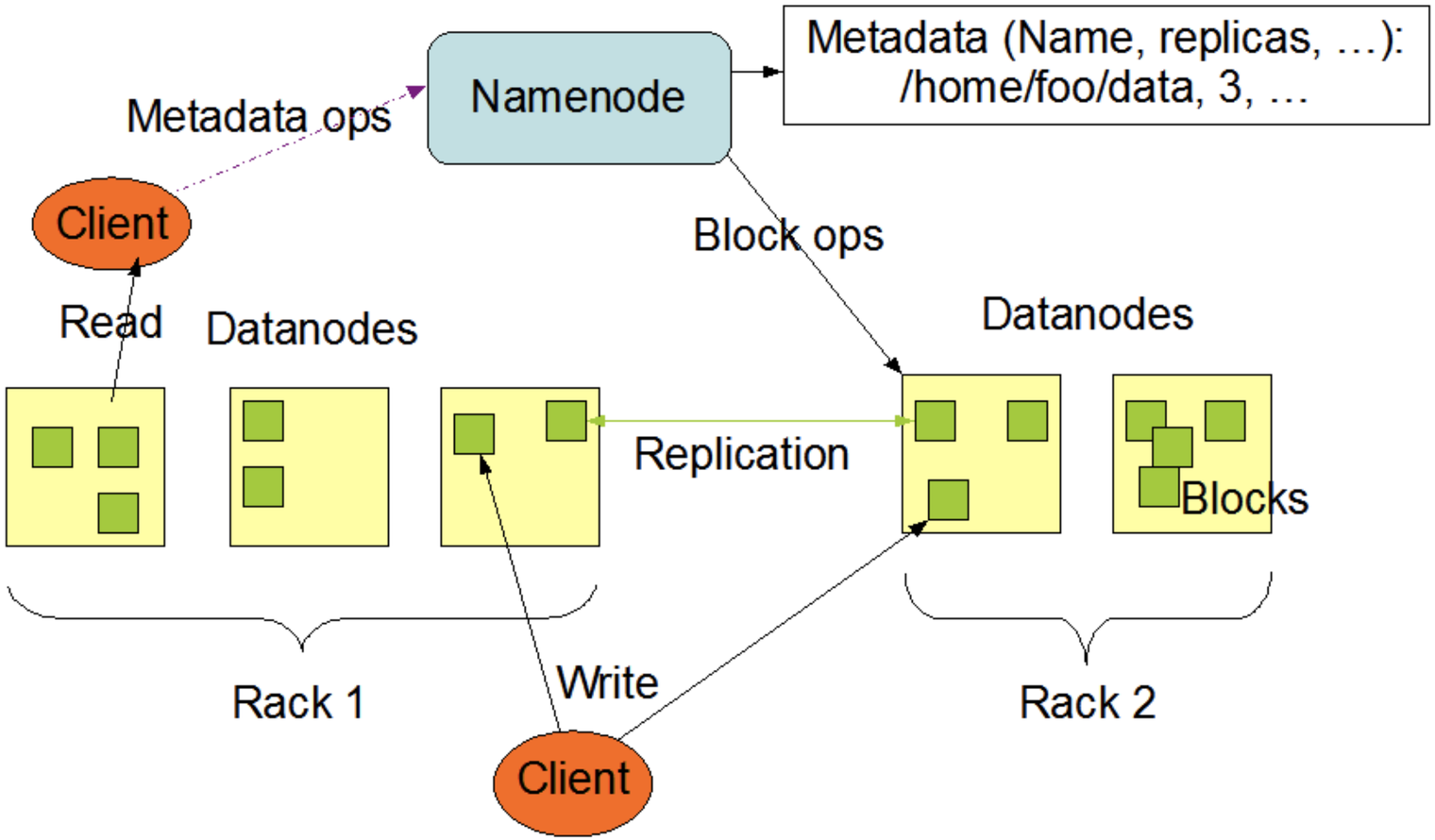
```
#!/usr/bin/env python
from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None
for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 1)
    try:
        count = int(count)
    except ValueError:
        continue
    if current_word == word:
        current_count += count
    else:
        if current_word:
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word

if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```


foo bar car
foo aar foo
car bar bar
foo 1
car 1
bar 1
foo 1
aar 1
foo 1
car 1
bar 1
bar 1
foo 1
car 1
bar 1
foo 2
aar 1
car 1
bar 2
aar [1]
bar [1,2]
car [1,1]
foo [1,2]
aar 1
bar 3
car 2
foo 3

# HDFS

- Hardware Failure
- Streaming Data Access
- Large Data Sets
- Simple Coherency Model
- “Moving Computation is Cheaper than Moving Data”
- Portability Across Heterogeneous Hardware and Software Platforms

# HDFS Architecture

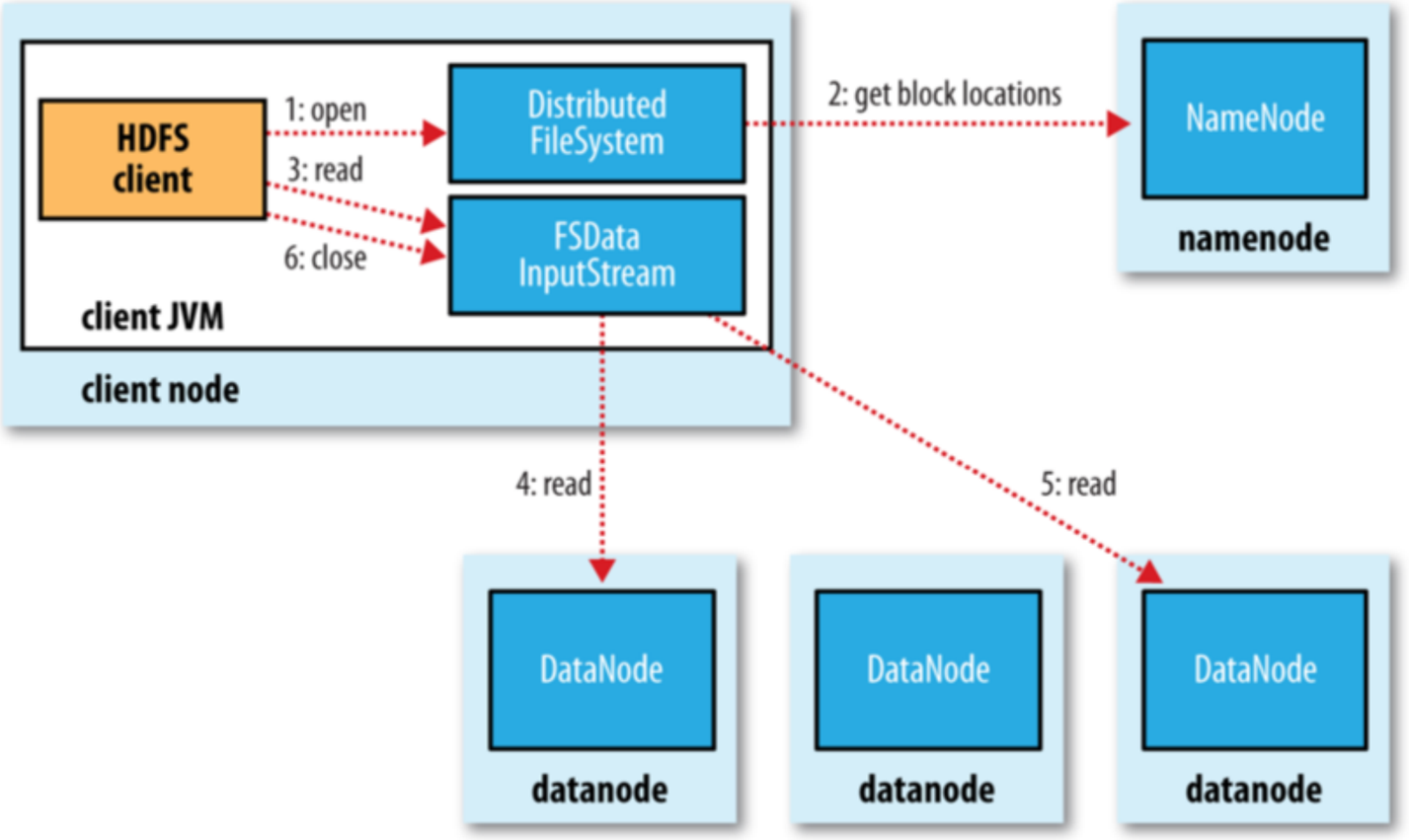


# Высокая доступность HDFS

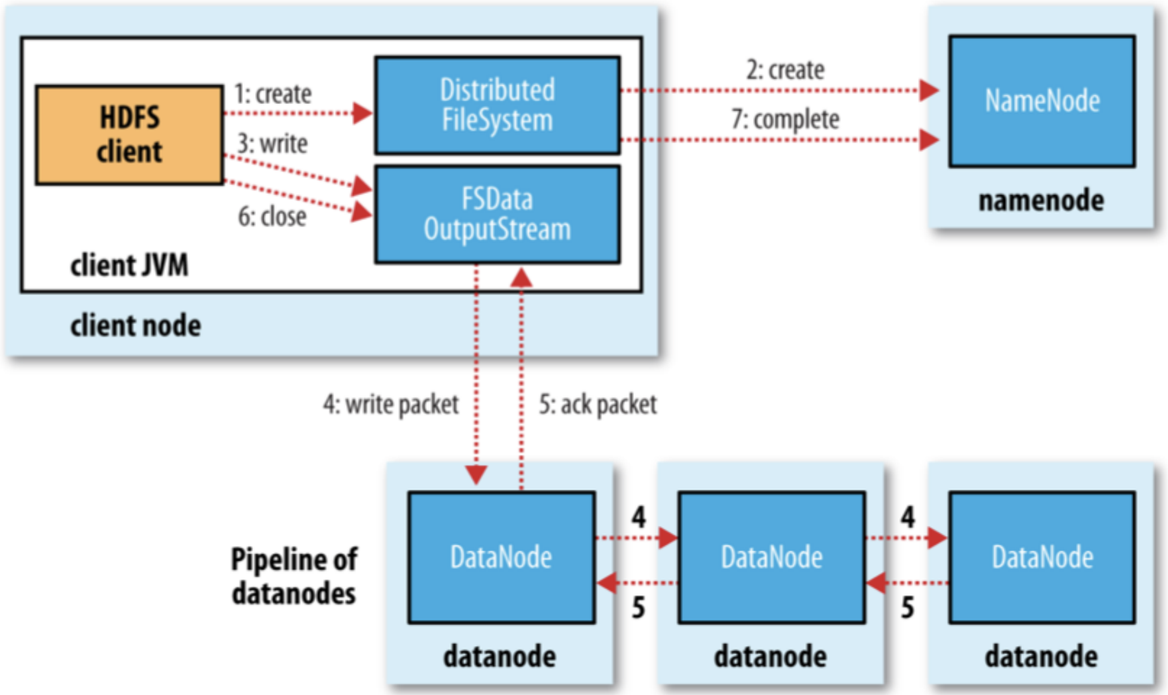
# Консольные команды HDFS

- hadoop distch
- hadoop distcp
- hadoop fs -put
- hadoop fs -get
- hadoop fs -appendToFile
- hadoop fs -cat
- hadoop fs -du
- и т.д.

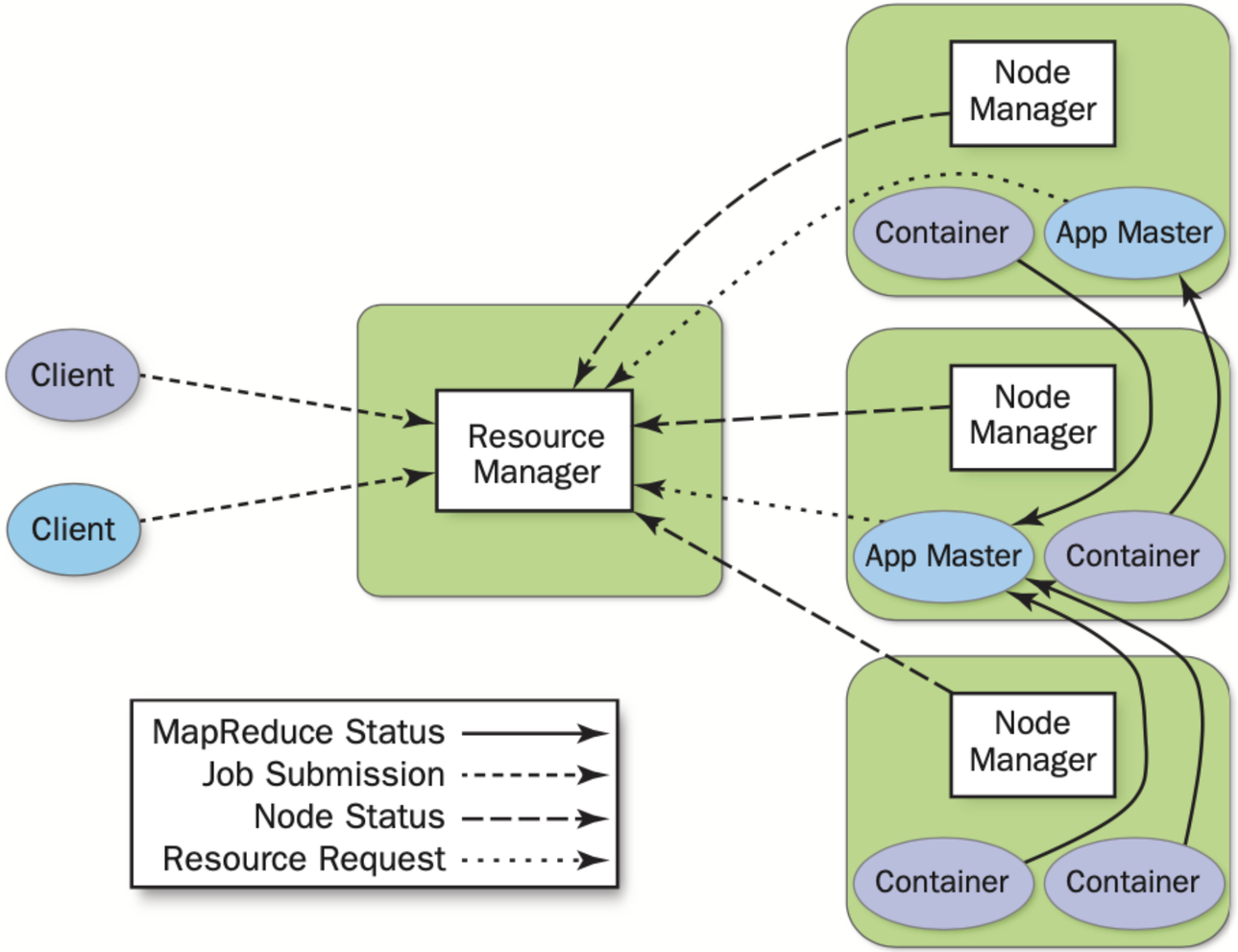
# Чтение файла в HDFS

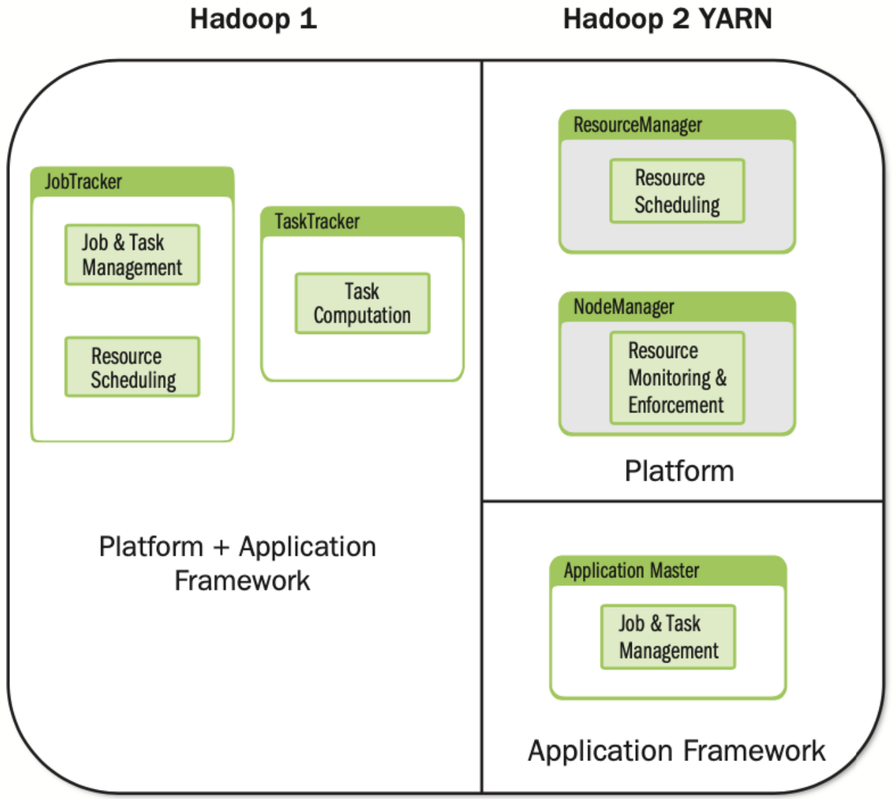

HDFS client
1: open
3: read
6: close
Distributed FileSystem
FSData InputStream
client JVM
client node
2: get block locations
NameNode
namenode
4: read
5: read
DataNode
datanode
DataNode
datanode
DataNode
datanode

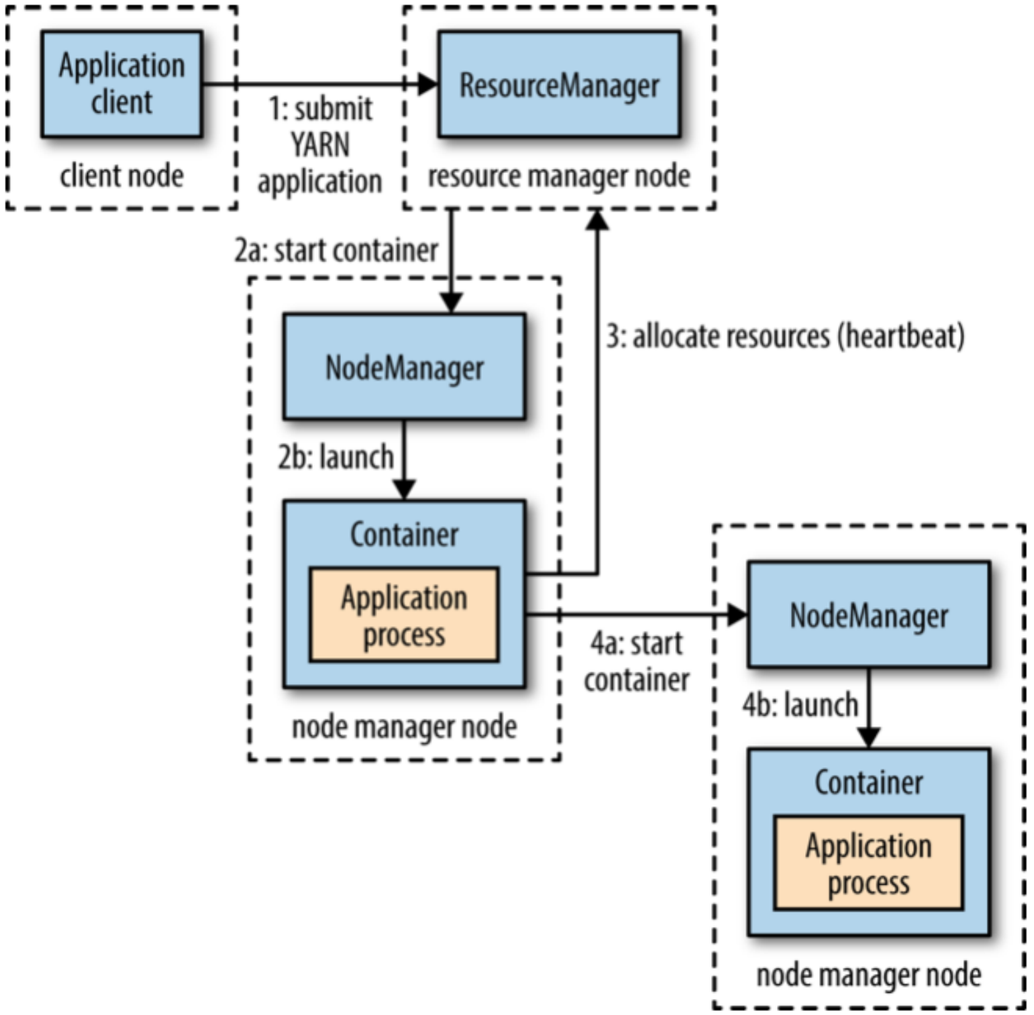
# Запись файла в HDFS


HDFS client
1: create
3: write
6: close
Distributed FileSystem
FSData OutputStream
client JVM
client node
2: create
7: complete
NameNode
namenode
4: write packet
5: ack packet
Pipeline of datanodes
DataNode
DataNode
DataNode
4
5
4
5
datanode
datanode
datanode

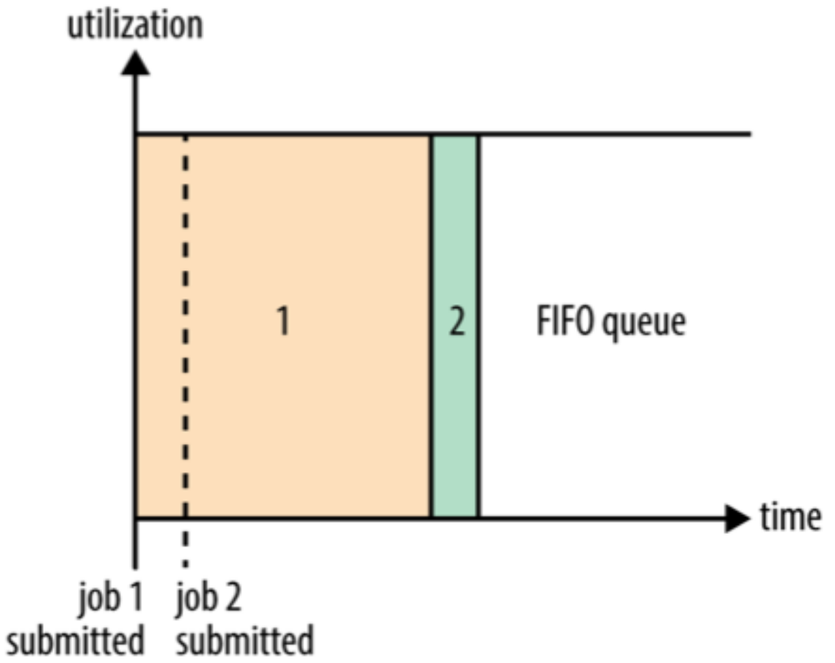
# YARN


Node Manager
Container
App Master
Client
Client
Resource Manager
Node Manager
App Master
Container
Node Manager
Container
Container
MapReduce Status
Job Submission
Node Status
Resource Request

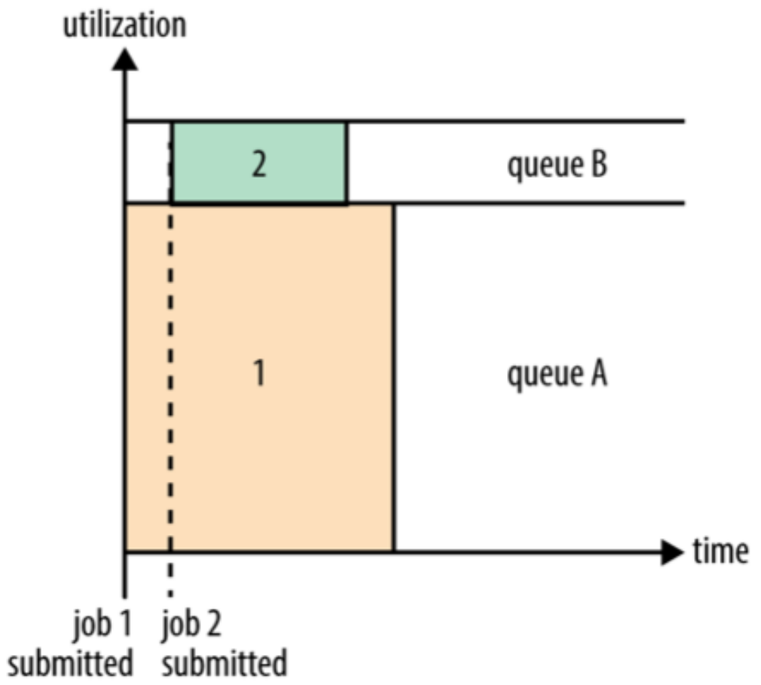

Hadoop 1
Hadoop 2 YARN
JobTracker
Job & Task Management
Resource Scheduling
TaskTracker
Task Computation
Platform + Application Framework
ResourceManager
Resource Scheduling
NodeManager
Resource Monitoring & Enforcement
Platform
Application Master
Job & Task Management
Application Framework


Application client
client node
1: submit YARN application
ResourceManager
resource manager node
2a: start container
3: allocate resources (heartbeat)
NodeManager
2b: launch
Container
Application process
node manager node
4a: start container
NodeManager
4b: launch
Container
Application process
node manager node

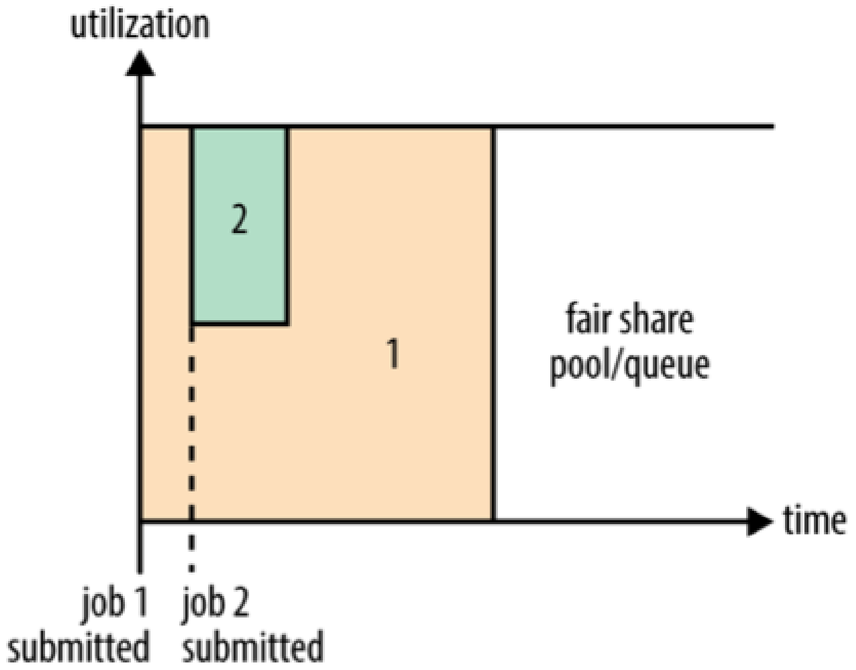
**i. FIFO Scheduler**

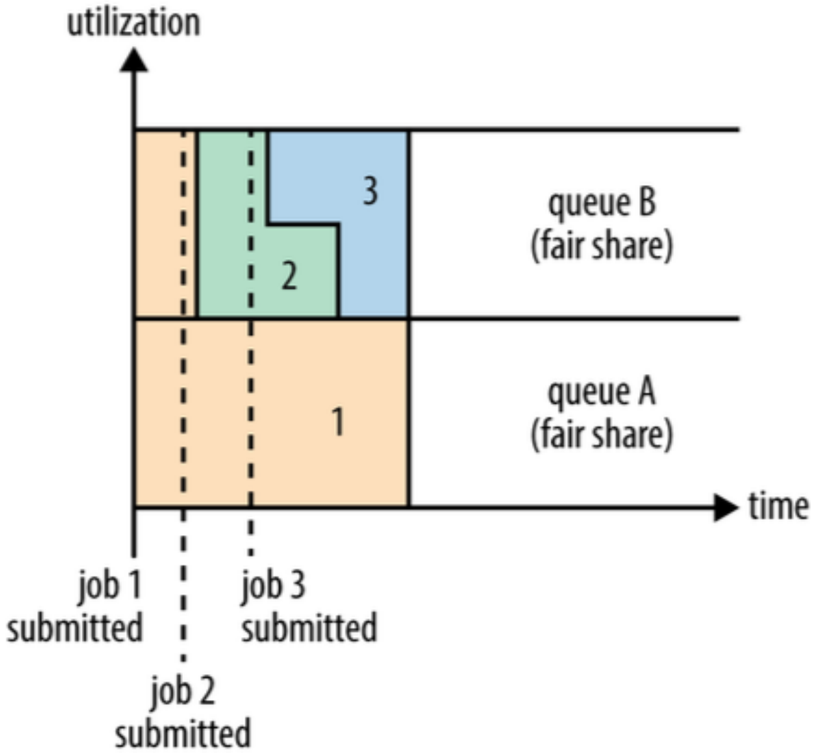


**ii. Capacity Scheduler**

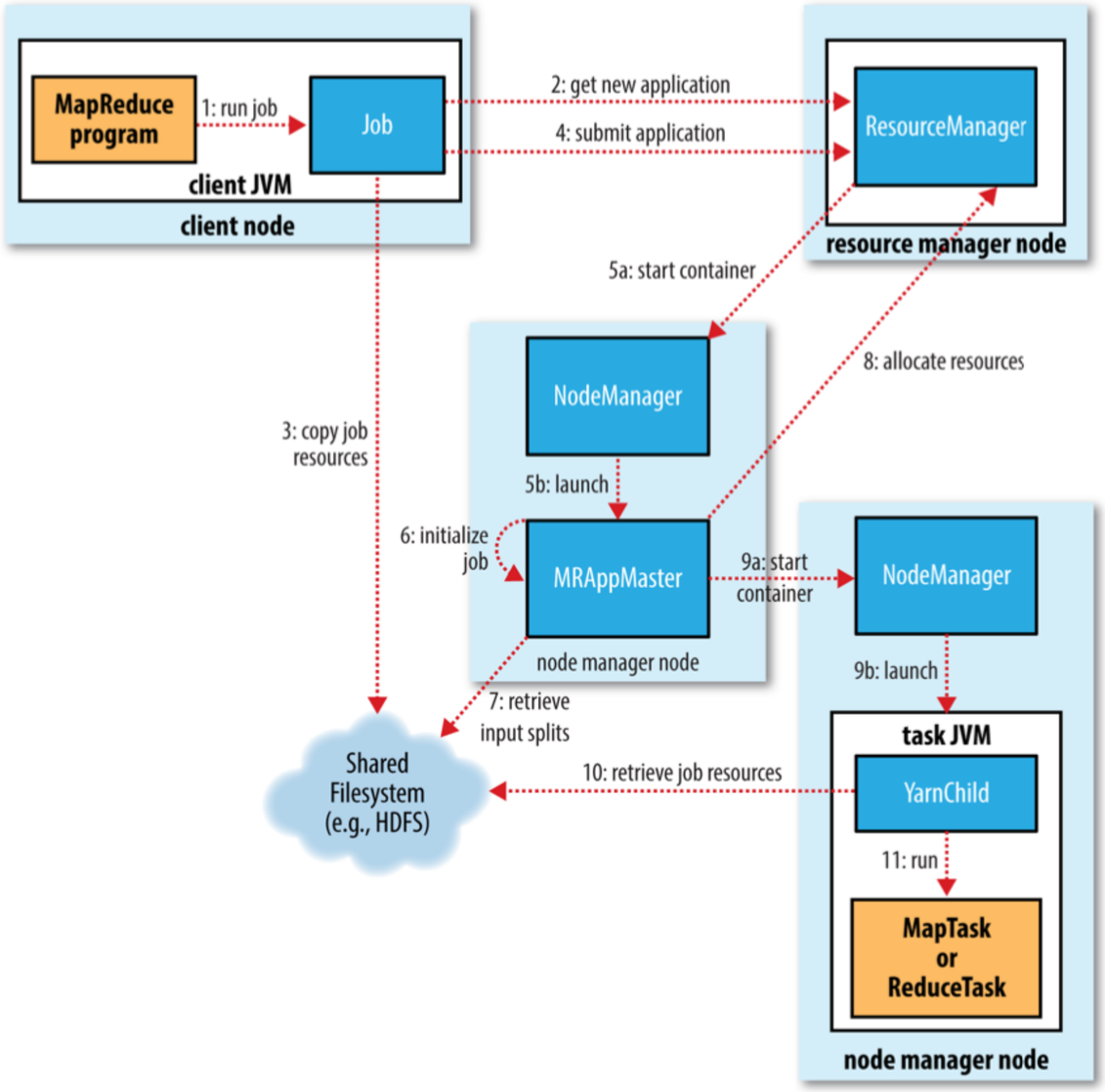


### iii. Fair Scheduler

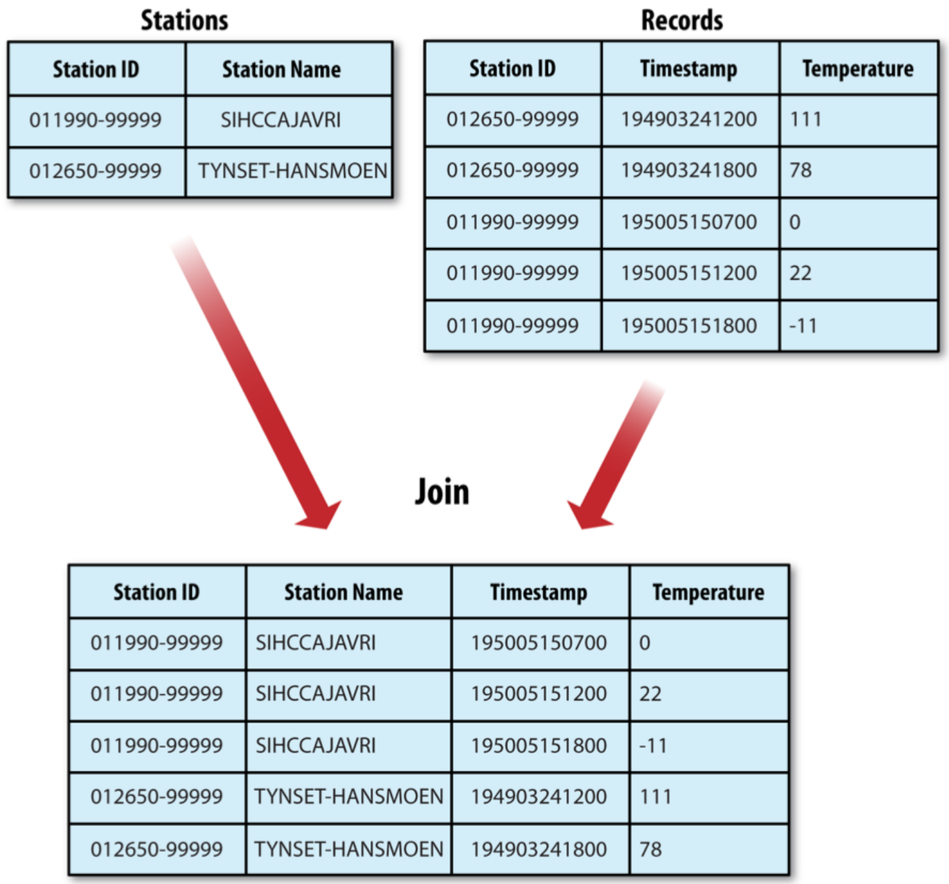



utilization
3
2
1
queue B
(fair share)
queue A
(fair share)
time
job 1
submitted
job 2
submitted
job 3
submitted

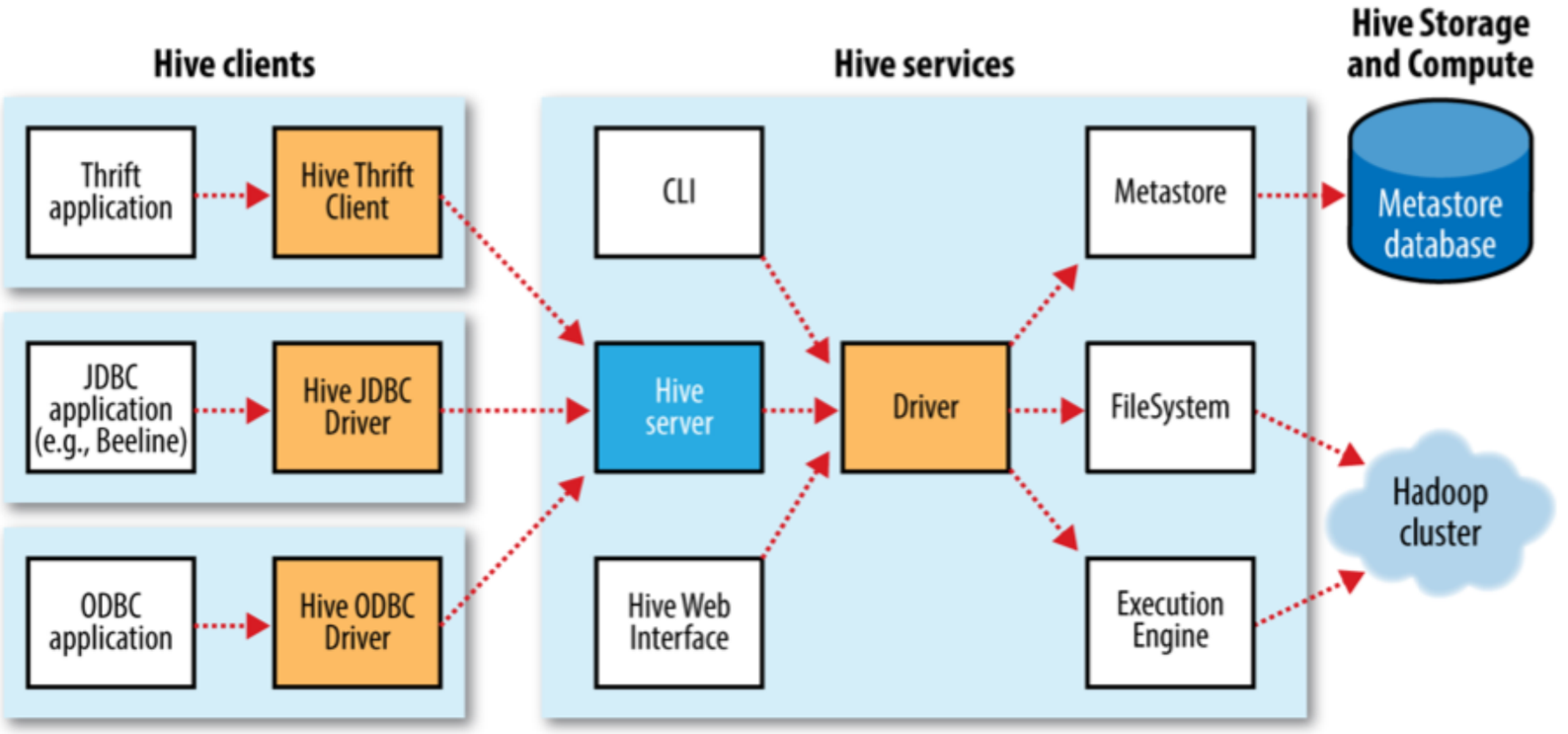
# СОБЕРЕМ ВСЕ ВМЕСТЕ


MapReduce program
1: run job
Job
client JVM
client node
2: get new application
4: submit application
ResourceManager
resource manager node
5a: start container
NodeManager
5b: launch
3: copy job resources
8: allocate resources
6: initialize job
MRAppMaster
node manager node
9a: start container
NodeManager
9b: launch
task JVM
YarnChild
11: run
MapTask or ReduceTask
node manager node
7: retrieve input splits
Shared Filesystem (e.g., HDFS)
10: retrieve job resources

# JOIN


Stations
Station ID
Station Name
011990-99999
SIHCCAJAVRI
012650-99999
TYNSET-HANSMOEN
Records
Station ID
Timestamp
Temperature
012650-99999
194903241200
111
012650-99999
194903241800
78
011990-99999
195005150700
0
011990-99999
195005151200
22
011990-99999
195005151800
-11
Join
Station ID
Station Name
Timestamp
Temperature
011990-99999
SIHCCAJAVRI
195005150700
0
011990-99999
SIHCCAJAVRI
195005151200
22
011990-99999
SIHCCAJAVRI
195005151800
-11
012650-99999
TYNSET-HANSMOEN
194903241200
111
012650-99999
TYNSET-HANSMOEN
194903241800
78

# HIVE


Hive clients
Hive services
Hive Storage and Compute
Thrift application
Hive Thrift Client
JDBC application (e.g., Beeline)
Hive JDBC Driver
ODBC application
Hive ODBC Driver
CLI
Hive server
Hive Web Interface
Driver
Metastore
FileSystem
Execution Engine
Metastore database
Hadoop cluster

# Managed Tables and External Tables

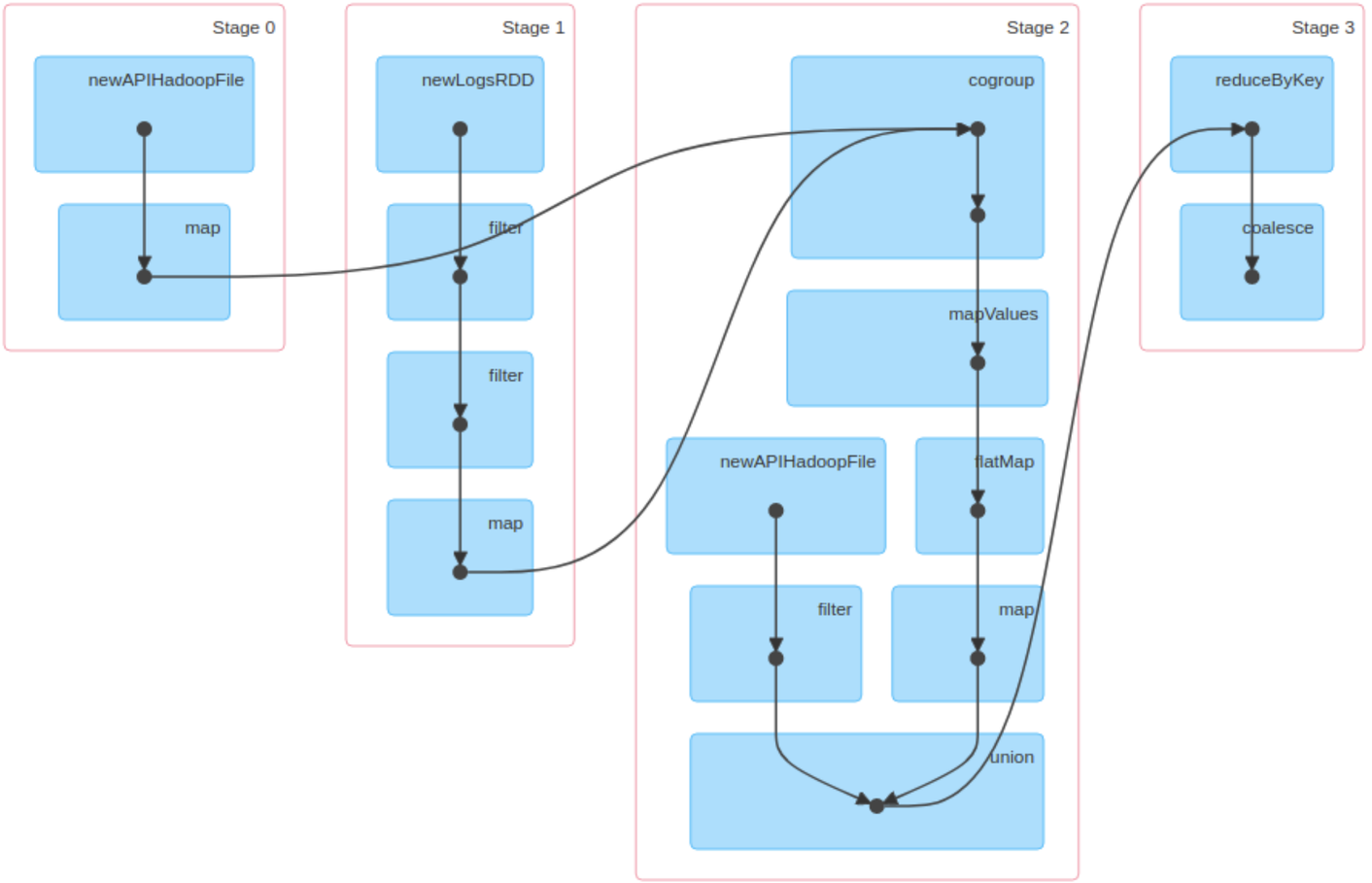
```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name
  [(col_name data_type [column_constraint_specification] [COMMENT col_comment], ... [constraint_specification])]
  [COMMENT table_comment]
  [PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
  [CLUSTERED BY (col_name, col_name, ...) [SORTED BY (col_name [ASC|DESC], ...)] INTO num_buckets BUCKETS]
   [ROW FORMAT row_format]
   [STORED AS file_format]
     | STORED BY 'storage.handler.class.name' [WITH SERDEPROPERTIES (...)]  -- (Note: Available in Hive 0.6.0 and later)
  ]
  [LOCATION hdfs_path]
  [AS select_statement];
```

*указано не все!

# JOIN

```
SELECT /*+ MAPJOIN(b) */ a.key, a.value
FROM a JOIN b ON a.key = b.key
```

# SPARK


Stage 0
newAPIHadoopFile
map
Stage 1
newLogsRDD
filter
filter
map
Stage 2
cogroup
mapValues
newAPIHadoopFile
flatMap
filter
map
union
Stage 3
reduceByKey
coalesce

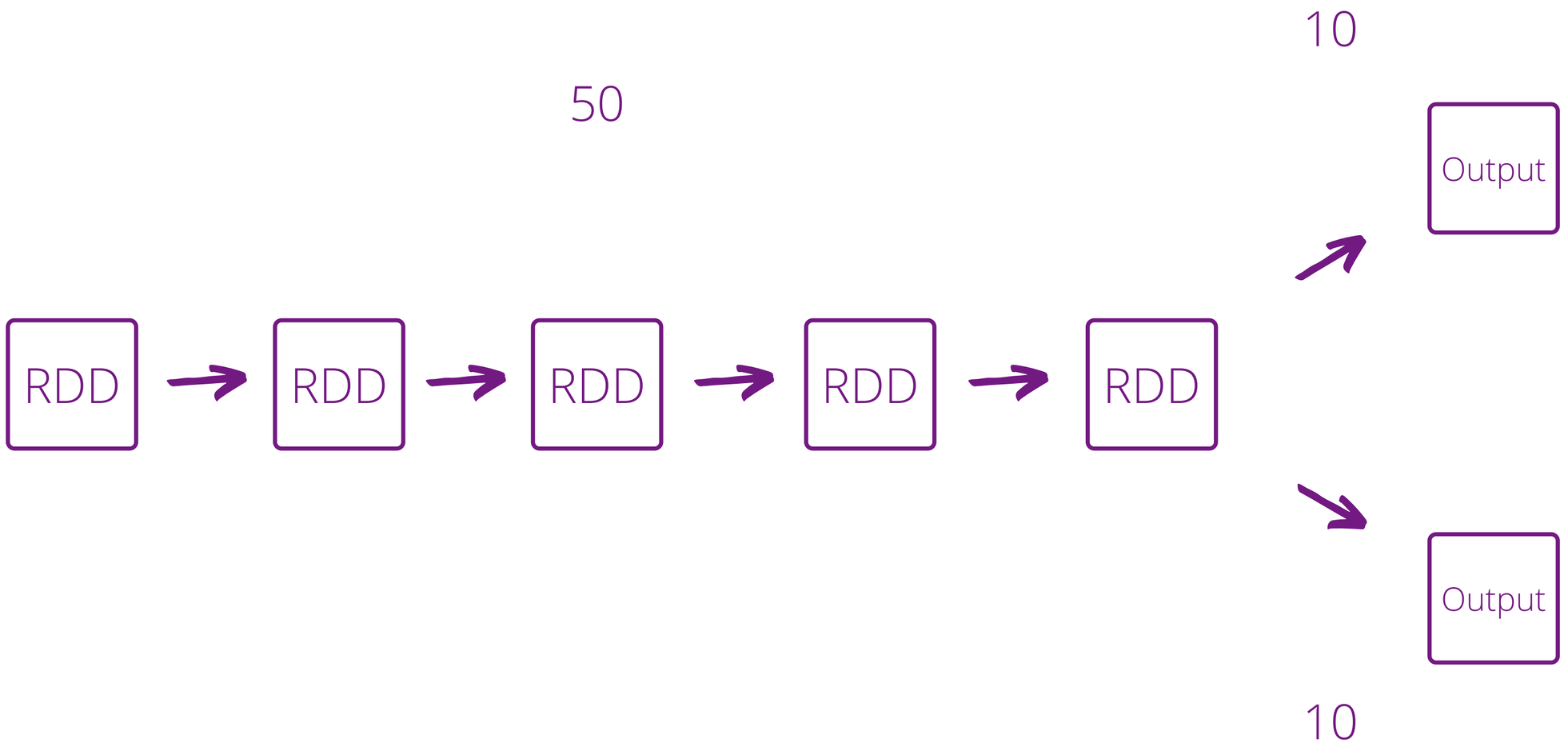
# Resilient Distributed Datasets (RDD)

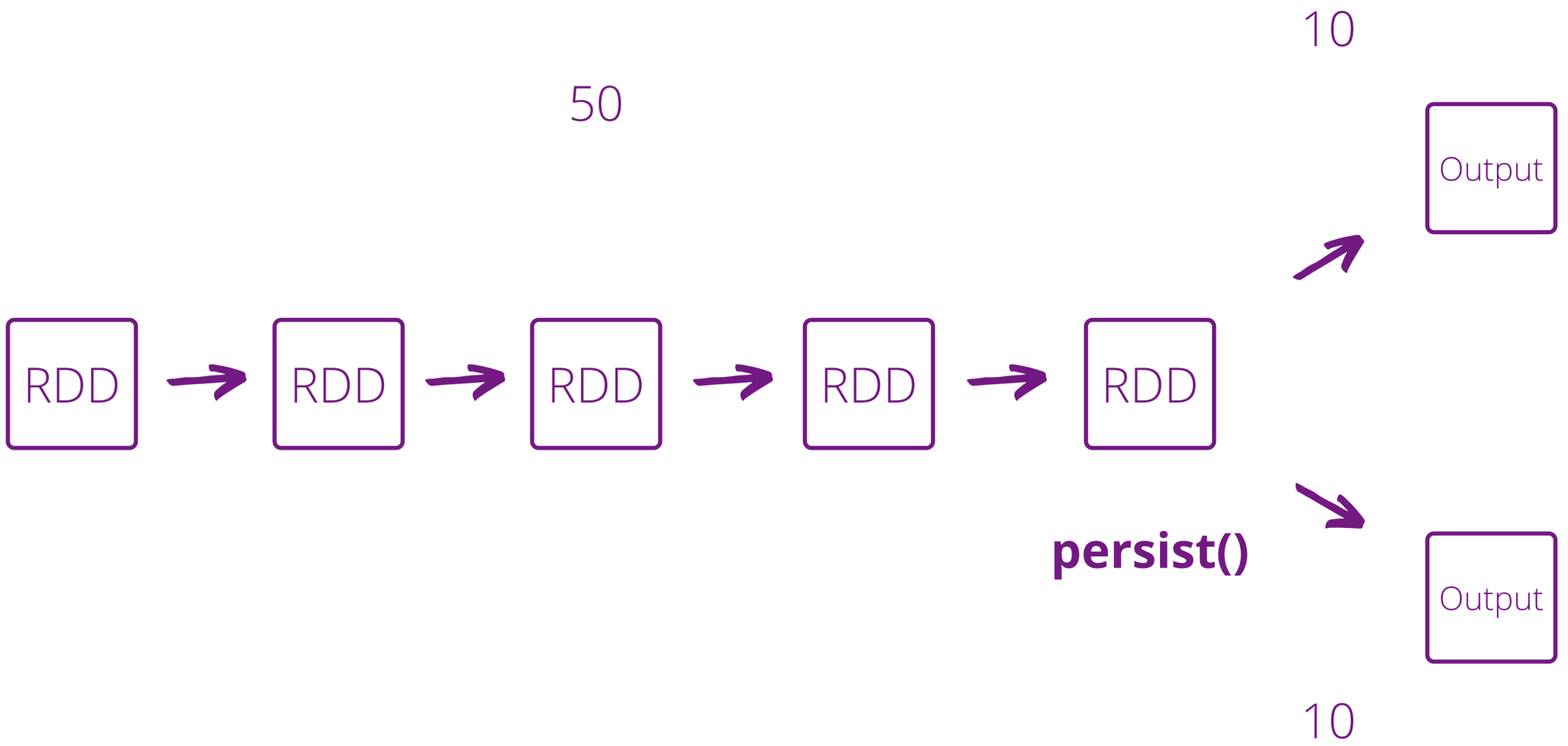
# Transformations and Actions

- map(func)
- filter(func)
- flatMap(func)
- sample(withReplacement, fraction, seed)
- union(otherDataset)
- distinct([numPartitions]))
- reduceByKey(func, [numPartitions])
- join(otherDataset, [numPartitions])
- repartition(numPartitions)
- coalesce(numPartitions)

- collect()
- count()
- take(n)
- takeSample(withReplacement, num, [seed])
- saveAsTextFile(path)

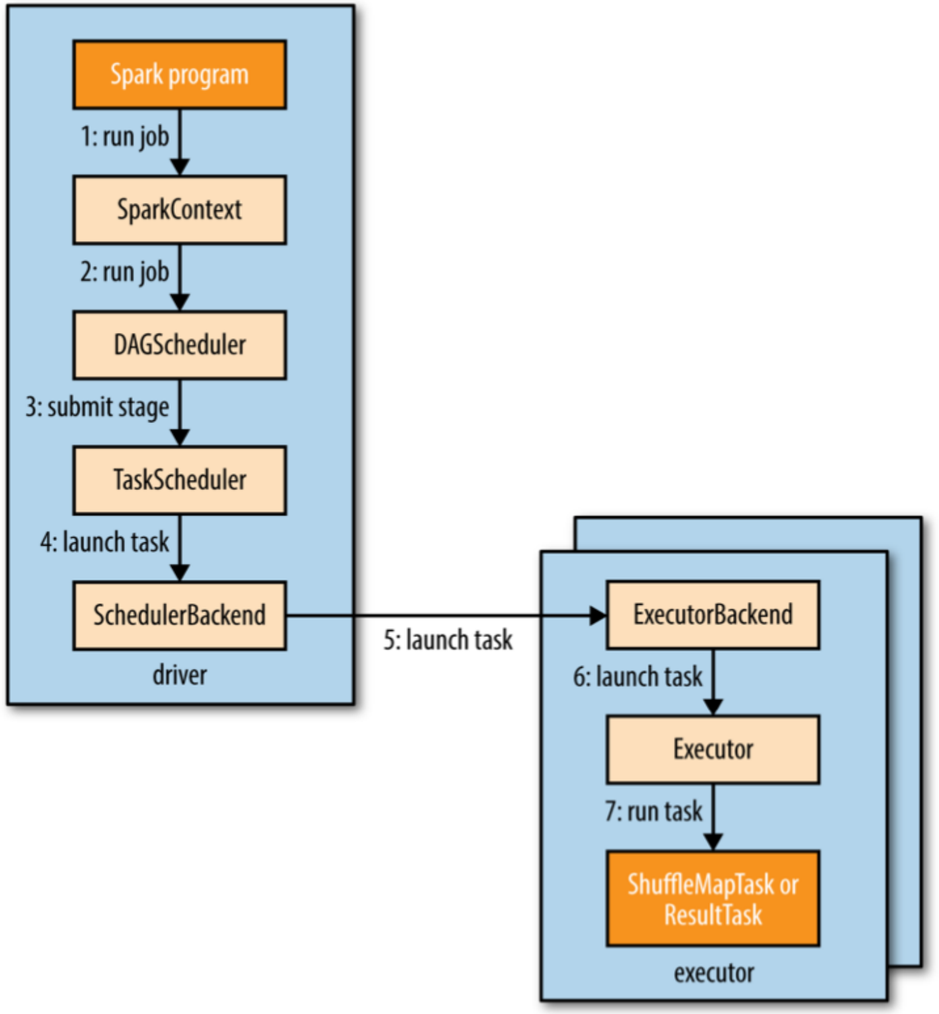
```
lines = sc.textFile("data.txt")
pairs = lines.map(lambda s: (s, 1))
counts = pairs.reduceByKey(lambda a, b: a + b)
```

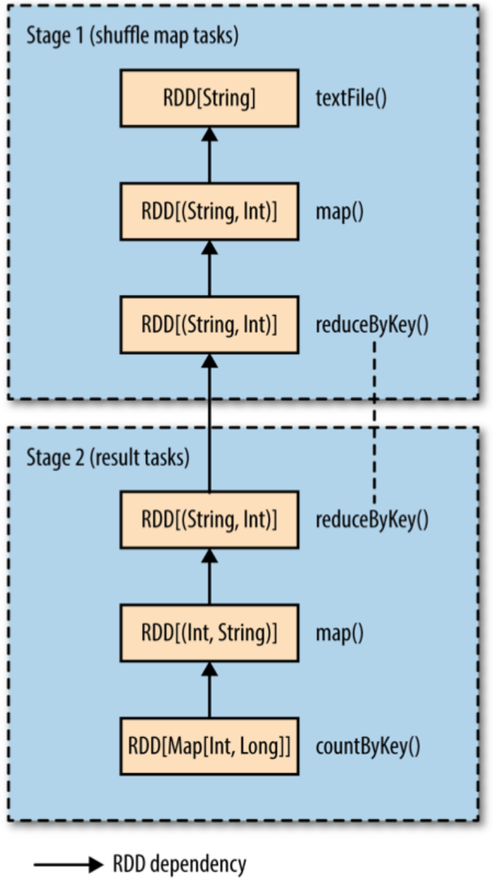

10
50
Output
RDD
RDD
RDD
RDD
RDD
Output
10


10
50
Output
RDD
RDD
RDD
RDD
RDD
persist()
Output
10

- MEMORY_ONLY
- MEMORY_AND_DISK
- MEMORY_ONLY_SER (Java and Scala)
- MEMORY_AND_DISK_SER (Java and Scala)
- DISK_ONLY
- MEMORY_ONLY_2
- MEMORY_AND_DISK_2

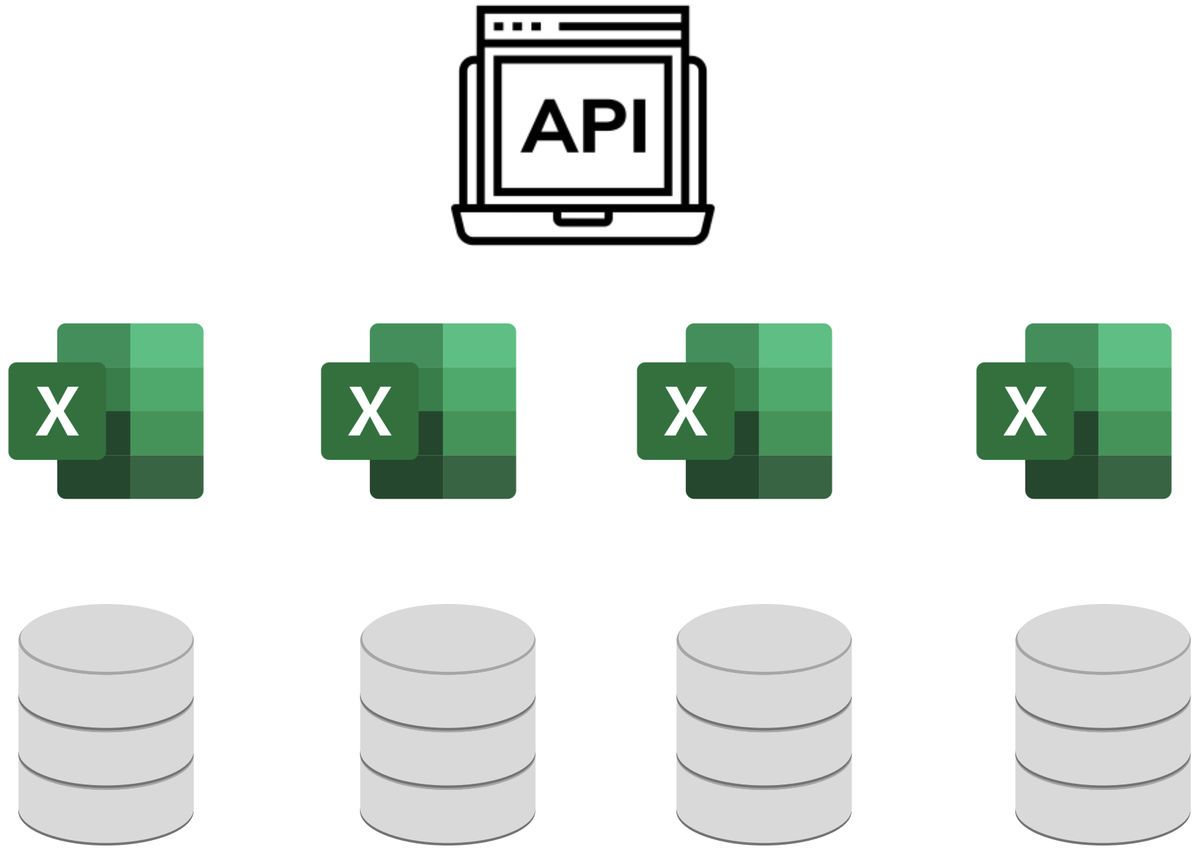
```
sc.broadcast([1, 2, 3])
```


Spark program
1: run job
SparkContext
2: run job
DAGScheduler
3: submit stage
TaskScheduler
4: launch task
SchedulerBackend
driver
5: launch task
ExecutorBackend
6: launch task
Executor
7: run task
ShuffleMapTask or
ResultTask
executor


Stage 1 (shuffle map tasks)
RDD[String]
textFile()
RDD[(String, Int)]
map()
RDD[(String, Int)]
reduceByKey()
Stage 2 (result tasks)
RDD[(String, Int)]
reduceByKey()
RDD[(Int, String)]
map()
RDD[Map[Int, Long]]
countByKey()
RDD dependency

# Shuffle

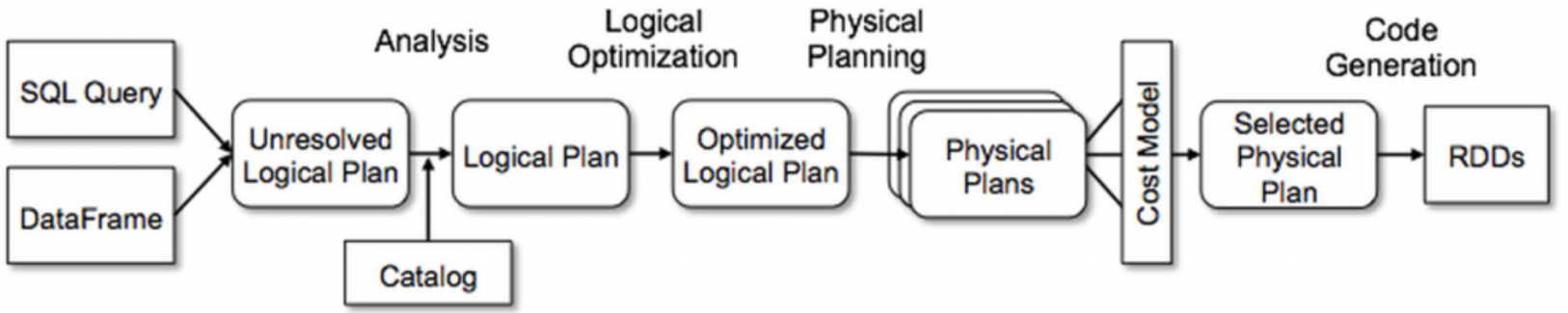
# DataFrame


API

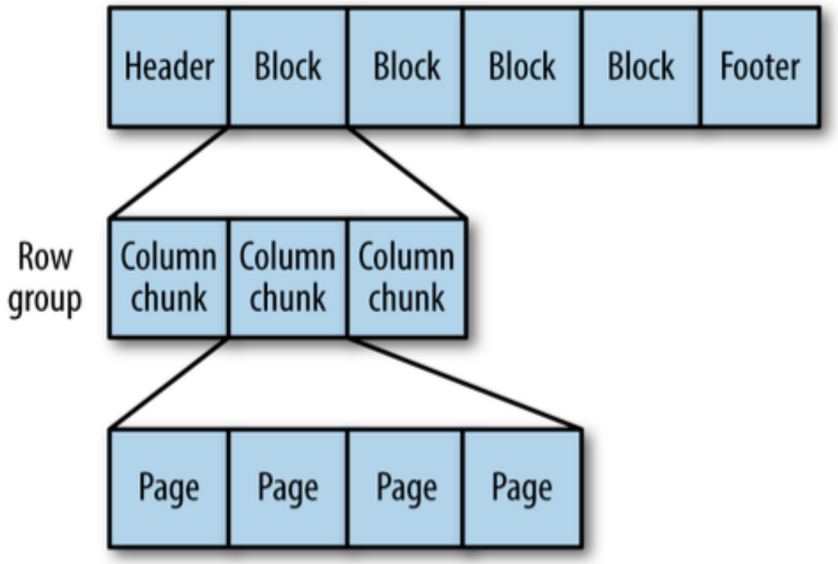
- df = spark.read.json("examples/src/main/resources/people.json")
- df.filter(df['age'] > 21).show()
- df.groupBy("age").count().show()
- sqlDF = spark.sql("SELECT * FROM people")

```
jdbcDF = spark.read \
    .format("jdbc") \
    .option("url", "jdbc:postgresql:dbserver") \
    .option("dbtable", "schema.tablename") \
    .option("user", "username") \
    .option("password", "password") \
    .load()
```

- df = spark.read.parquet("examples/src/main/resources/users.parquet")
- (df
  .write
  .partitionBy("favorite_color")
  .bucketBy(42, "name")
  .saveAsTable("people_partitioned_bucketed") )


SQL Query
DataFrame
Analysis
Unresolved Logical Plan
Catalog
Logical Optimization
Logical Plan
Physical Planning
Optimized Logical Plan
Physical Plans
Cost Model
Code Generation
Selected Physical Plan
RDDs

# PARQUET


Header
Block
Block
Block
Block
Footer
Row group
Column chunk
Column chunk
Column chunk
Page
Page
Page
Page

@almorozovv

# Thank you!