DĚLAT
DOBRÝ SOFTWARE
NÁS BAVÍ

PROFINIT

Spark

Jan Hučín

December 14th, 2018

Agenda

1. What is it (for)
2. How to learn it
3. How to work with it
4. How it works
   - logical / technical level
   - transformations, actions, caching
   - examples
   - architecture, sources

Later:

› Spark SQL

› Spark streaming

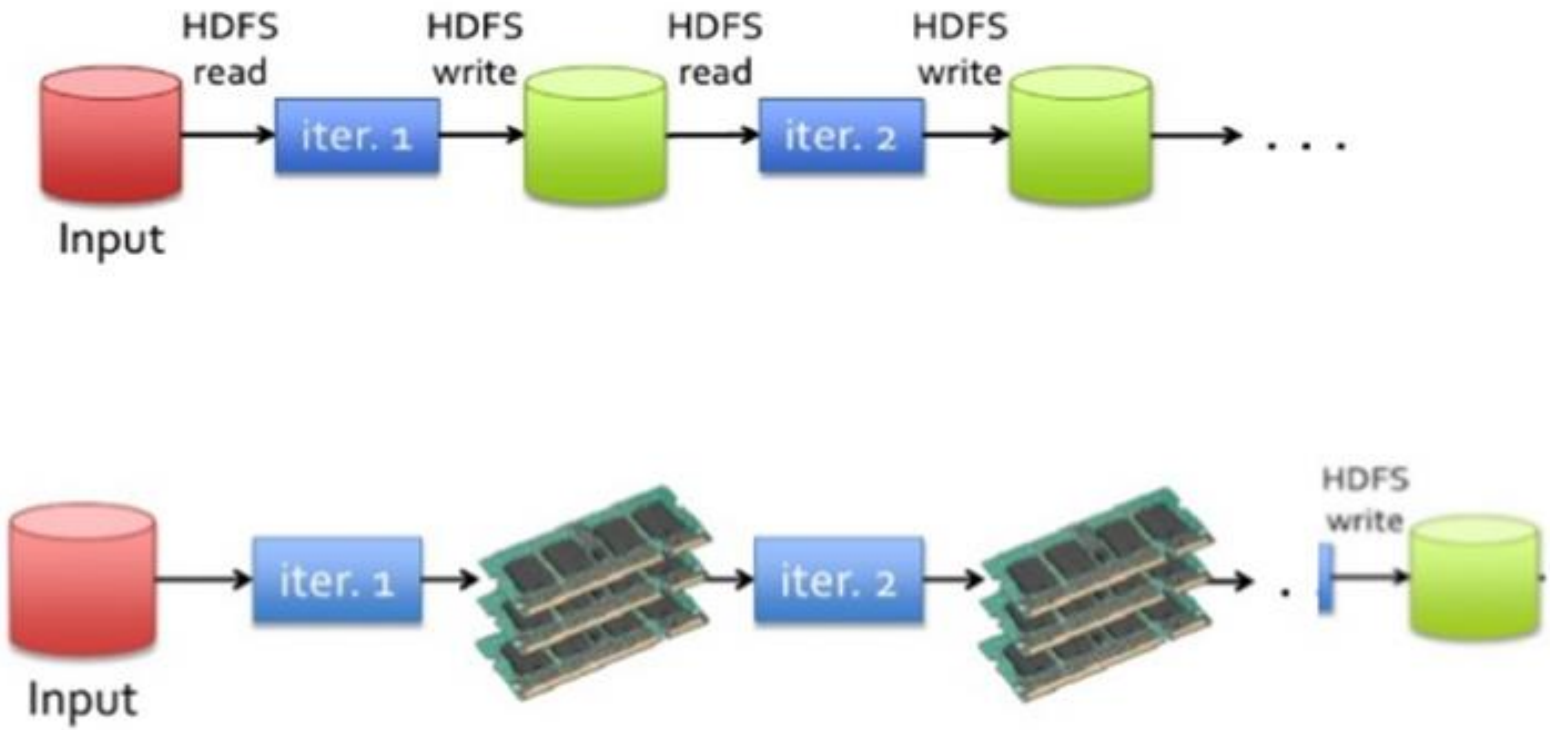
What is Spark

(For) what is Spark

- framework for distributed computations
- improved map-reduce – by 2 orders
  - **in-memory processing** – less I/O operations, good for iterations and data analysis
  - **operation optimisation** before processing
  - enhanced by **SQL-like** commands
- API for Scala, Java, Python, R
- on Hadoop (using HDFS, YARN) or standalone
- written in Scala
- processing in JVM
- the most active (2017) opensource Big Data project

PROFINIT

Spark vs. map-reduce



PROFINIT

Suitable tasks

- big enough, but not extreme
- able to be parallelized
- iterative
- not resolvable by traditional technologies

E. g.

1. sophisticated client features (risk score)
2. complicated SQL tasks for DWH
3. compute once (night), use many (day)
4. graph/network analysis
5. text-mining

PROFINIT

Not suitable tasks

- too small
- with extreme memory demands
- custom-tailored for other technologies (SQL, Java)
- unable to be parallelized
- strictly real-time

E. g.

1. small data modelling
2. median computation, random skipping in the file
3. JOIN of really big tables

PROFINIT

How to learn it

- http://spark.apache.org
- basics of Python | Scala | Java | R
- **self-practice**
- advice of experienced, StackOverflow etc.

PROFINIT

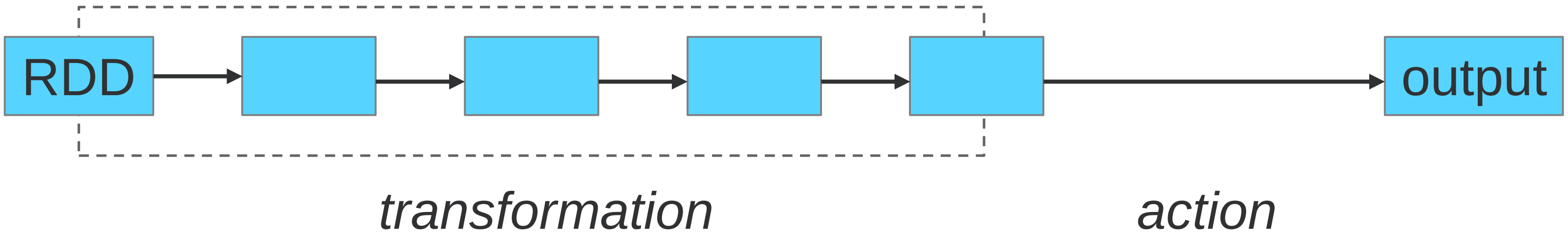
How to work with it

- interactively
  - command line (shell for Python and Scala)
  - Zeppelin/Jupyter notebook
- batch / application
  - compiled .jar file
  - Python script

PROFINIT

How Spark works

Logical level (high)

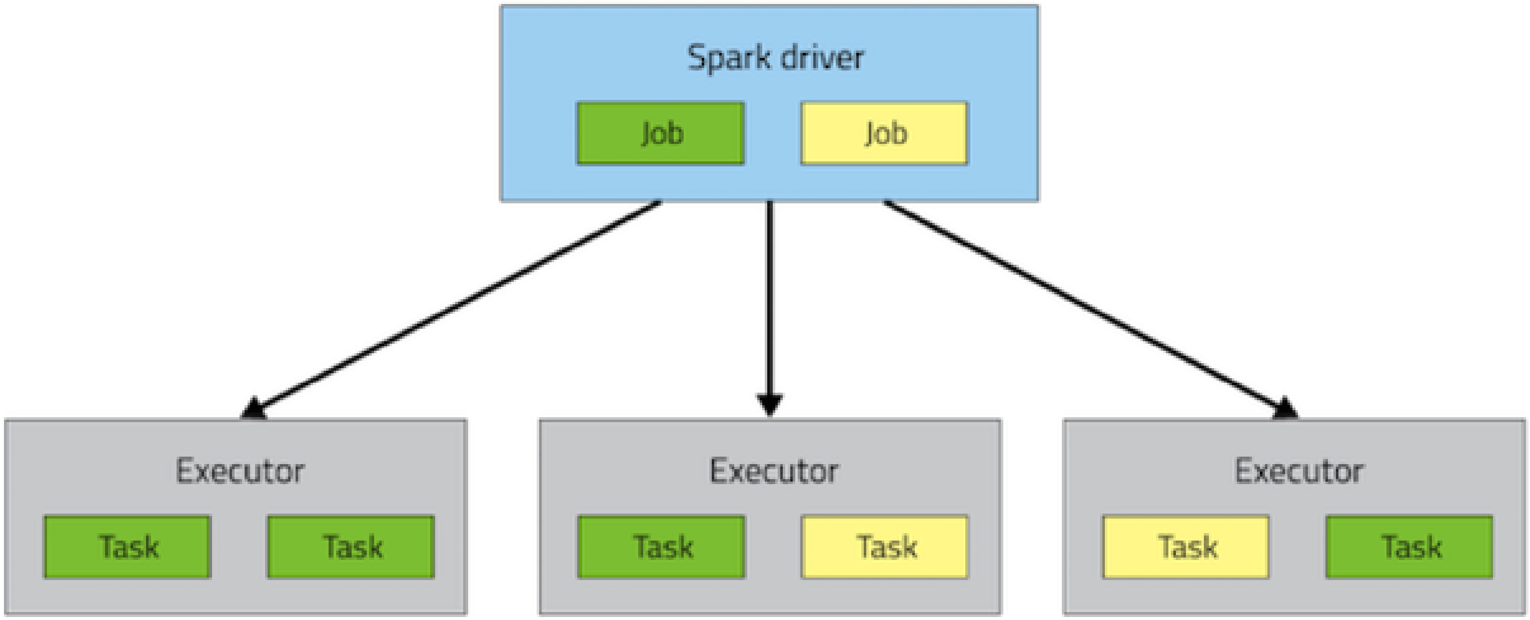


- series of transformations finished by action
- transformation are **planned** and **optimized**, but **not made**
- **lazy evaluation:** action starts all process

What is RDD?

- resilient distributed dataset
- item collections (line of a text file, files in a direktory, data matrix, etc.)
- must be divisible to parts – Spark does the division itself

Technical level (mid)



- creates JVM on nodes (=executors)
- application → jobs; job → tasks
- task (and data) distribution to nodes
- process control
- **more later – Spark architecture**

PROFINIT

Spark RDD – transformations

RDD1 ⇨ RDD2, item by item („row by row“)

- **map** (item ⇨ transforming function ⇨ new item)
- **flatMap** (item ⇨ transforming function ⇨ 0 až N new items)
- **filter**, **distinct** (only items meeting condition / unique items)
- **join** (joining other RDD by key)
- **union**, **intersection**
- **groupByKey**, **reduceByKey** (items agregation by key)
- ... and many others

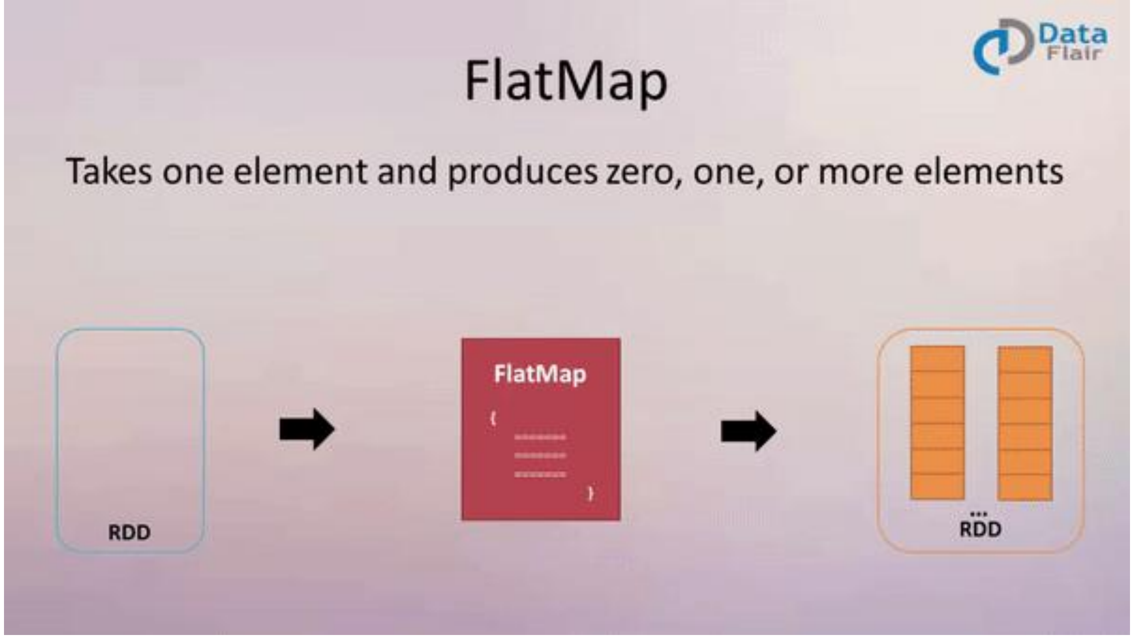
How to get „key“?

"tuple"
(Scala, Python)

- first member of *tuple* = key
- result of transformation: word ⇨ (word, 1)

PROFINIT

map a flatMap



PROFINIT

Example 1 – word count

- Task: count frequency of words in document
- Input: text file divided to lines
- Output: RDD with items (word, frequency)
- Workflow:
  - import text file as RDD
  - lines transformation: line ⇨ words ⇨ tuples (word, 1)
  - grouping items with same key, summing ones = count

PROFINIT

How to launch the interactive shell

pyspark (Python) **| spark-shell** (Scala)

- from Linux shell in console
- Spark on master node (local) or on YARN:
  - `pyspark --master local`
  - `pyspark --master yarn`
- creates important objects: `sc` (SparkContext), `sqlContext`
- many parameters – later
- closed by `exit()`

PROFINIT

Example 1 – word count

- Task: count frequency of words in document
- Input: text file divided to lines
- Workflow:
- import text file as RDD
```
lines = sc.textFile("/user/pascepet/bible.txt")
```
- lines transformation: line ⇨ words
```
words = lines.flatMap(lambda line: line.split(" "))
```
- lines transformation: words ⇨ tuples (word, 1)
```
pairs = words.map(lambda word: (word, 1))
```
- grouping items with same key, summing ones = count
```
counts = pairs.reduceByKey(lambda a, b: a + b)
```

to be or not to be	to be or not to be	(to, 1) (be, 1) (or, 1) (not, 1) (to, 1) (be, 1)	(to, 2) (be, 2) (or, 1) (not, 1)

PROFINIT

Why does it not count anything?

Because we have done no action so far.

PROFINIT

Spark RDD – actions

- **count**
- **take** (gets first *n* RDD items)
- **collect** (gets all items in RDD)
- **reduce** (aggregation of all RDD items with function provided)
- **saveAsTextFile**
- ... and others

- Action starts all chain from the beginning!
  - After run, all results are „forgotten“.
  - **If we don‘t want this, we have to cache some RDD.**

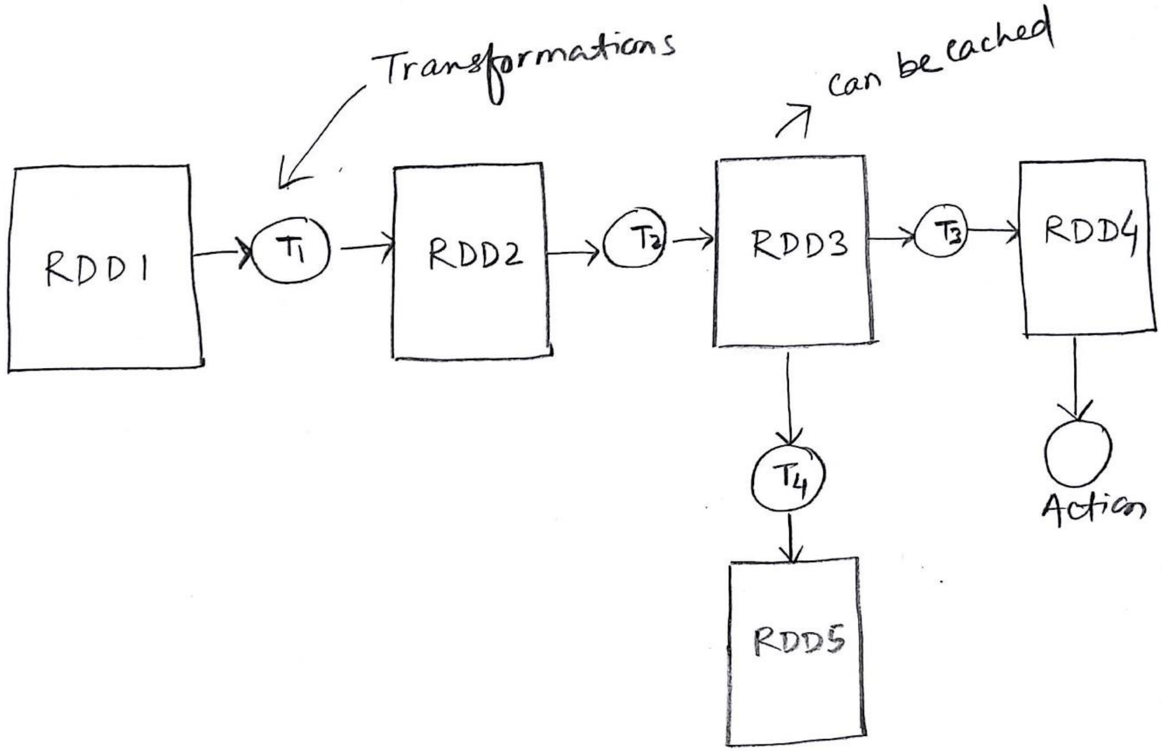
PROFINIT

Caching

- Caching: RDD is not forgotten but saved into memory / on disk.
- **Caching methods:**
  - **cache** (try to save in memory)
  - **persist** (more general – can control serialization, disk/memory)
  - **unpersist** (release RDD from memory/disk)
- **Caching types:**
  - MEMORY_ONLY
  - MEMORY_AND_DISK
  - MEMORY_ONLY_SER
  - MEMORY_AND_DISK_SER
- SER = serialization – less memory, high computation demands
  - Useful only for Java and Scala; Python makes serialization always
- Caching is not an action!

PROFINIT

Spark program as a graph



Example 2 – similarity of images

- Task: evaluate similarity in pairs of images
- Input: b&w BMP files (4×4px, 256 colors) – RDD items
- Workflow:
  - parsing: binary BMP ⇨ sequence of 16 bits (0/1)
  - generating list of image pairs
  - evaluating similarity in pairs
- Result:
  RDD with items (file1, file2, similarity)
  save as text file

PROFINIT

Example 2 – similarity of images

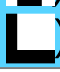
Task: evaluate similarity in pairs of images

Input: b&w BMP files (4×4px, 256 colors) – RDD items

```
files = sc.binaryFiles("/user/pascepet/pismena/*.bmp")
```

Workflow:

- parsing: binary BMP ⇨ sequence of 16 bits (0/1)
  ```
  filesParsed = files.map(parseBMP)
  ```
- generating list of image pairs
  ```
  filesPairs = filesParsed.cartesian(filesParsed) \
      .filter(lambda f: f[0][0]<f[1][0])
  ```
- evaluating similarity in pairs
  ```
  simil = filesPairs.map(similPair)
  ```

(N, )
(J, )
(L, )

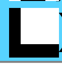
(N, 1001101111011001)
(J, 0110100100010001)
(L, 1111100010001000)

(N, 1001...), (J, 0110...)
(N, 1001...), (L, 1111...)
(J, 0110...), (L, 1111...)

(N, J, 0.5)
(N, L, 0.6)
(J, L, 0.3)

PROFINIT

Example 2 – parsing

(N, )
(J, )
(L, )

```
def parseBMP(file):
    name = file[0]
    bytes = file[1]
    bytesLast = bytes[-16:]
    bits = []
    for z in bytesLast:
        if z=='\x00':
            bits.append(1)
        elif z=='\xff':
            bits.append(0)
        else:
            pass
    return (name, bits)
```

(N, 1001101111011001)
(J, 0110100100010001)
(L, 1111100010001000)

Example 2 – similarity

```
((N, 1001101111011001), (J, 0110100100010001))
```

```
def similPair(pair):
    file1 = pair[0]
    file2 = pair[1]
    return (file1[0],file2[0],
            similarity(file1[1],file2[1])
           )


def similarity(bits, pattern=[0]*16):
    sum = 0
    for i in range(0, len(bits)):
        sum += (bits[i]==pattern[i])
    return sum*1.0/len(bits)
```

```
(N, J, 0.5)
```

PROFINIT

Other Spark RDD operations

https://training.databricks.com/visualapi.pdf

Essential Core & Intermediate Spark Operations

TRANSFORMATIONS

General

- map
- filter
- flatMap
- mapPartitions
- mapPartitionsWithIndex
- groupBy
- sortBy

Math / Statistical

- sample
- randomSplit

Set Theory / Relational

- union
- intersection
- subtract
- distinct
- cartesian
- zip

Data Structure / I/O

- keyBy
- zipWithIndex
- zipWithUniqueID
- zipPartitions
- coalesce
- repartition
- repartitionAndSortWithinPartitions
- pipe

ACTIONS

General

- reduce
- collect
- aggregate
- fold
- first
- take
- forEach
- top
- treeAggregate
- treeReduce
- forEachPartition
- collectAsMap

Math / Statistical

- count
- takeSample
- max
- min
- sum
- histogram
- mean
- variance
- stdev
- sampleVariance
- countApprox
- countApproxDistinct

Set Theory / Relational

- takeOrdered

Data Structure / I/O

- saveAsTextFile
- saveAsSequenceFile
- saveAsObjectFile
- saveAsHadoopDataset
- saveAsHadoopFile
- saveAsNewAPIHadoopDataset
- saveAsNewAPIHadoopFile

https://training.databricks.com/visualapi.pdf

Essential Core & Intermediate PairRDD Operations

TRANSFORMATIONS

General

- flatMapValues
- groupByKey
- reduceByKey
- reduceByKeyLocally
- foldByKey
- aggregateByKey
- sortByKey
- combineByKey

Math / Statistical

- sampleByKey

Set Theory / Relational

- cogroup (=groupWith)
- join
- subtractByKey
- fullOuterJoin
- leftOuterJoin
- rightOuterJoin

Data Structure

- partitionBy

ACTIONS

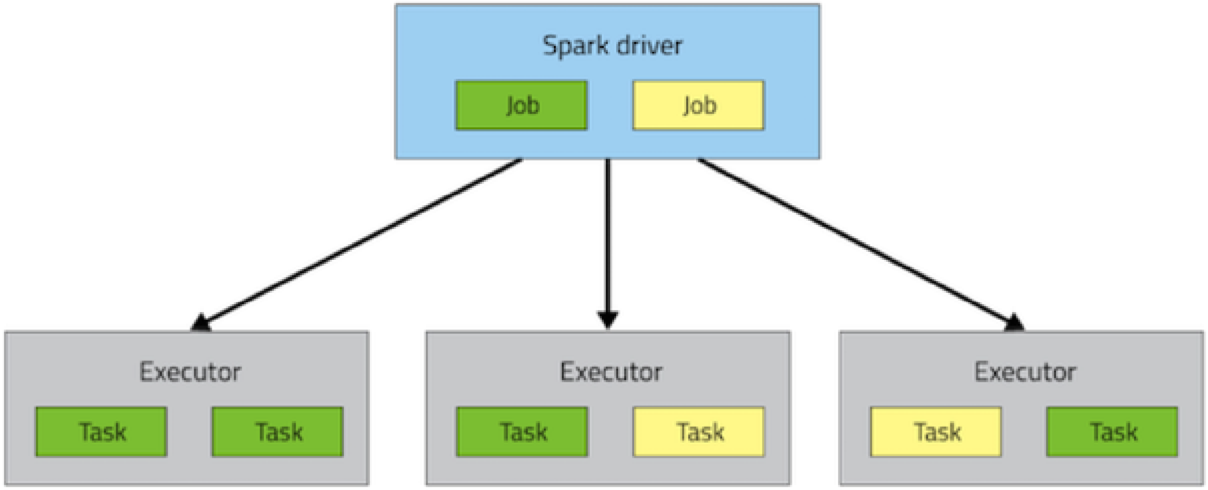
- keys
- values

- countByKey
- countByValue
- countByValueApprox
- countApproxDistinctByKey
- countApproxDistinctByKey
- countByKeyApprox
- sampleByKeyExact

Spark architecture

Important terms 1

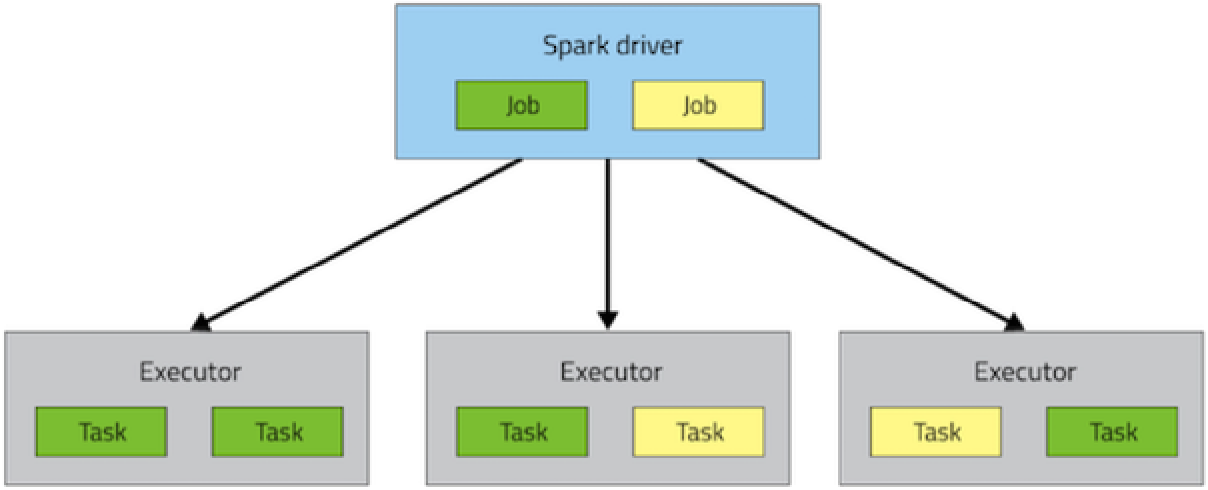
- **Application master**
  - process for negotiation of sources
- **Driver**
  - main process
  - workflow planning
  - distributing work to executors
- **Executor**
  - process running on a node – JVM
  - doing tasks (possibly in parallel if it has multiple cores)



PROFINIT

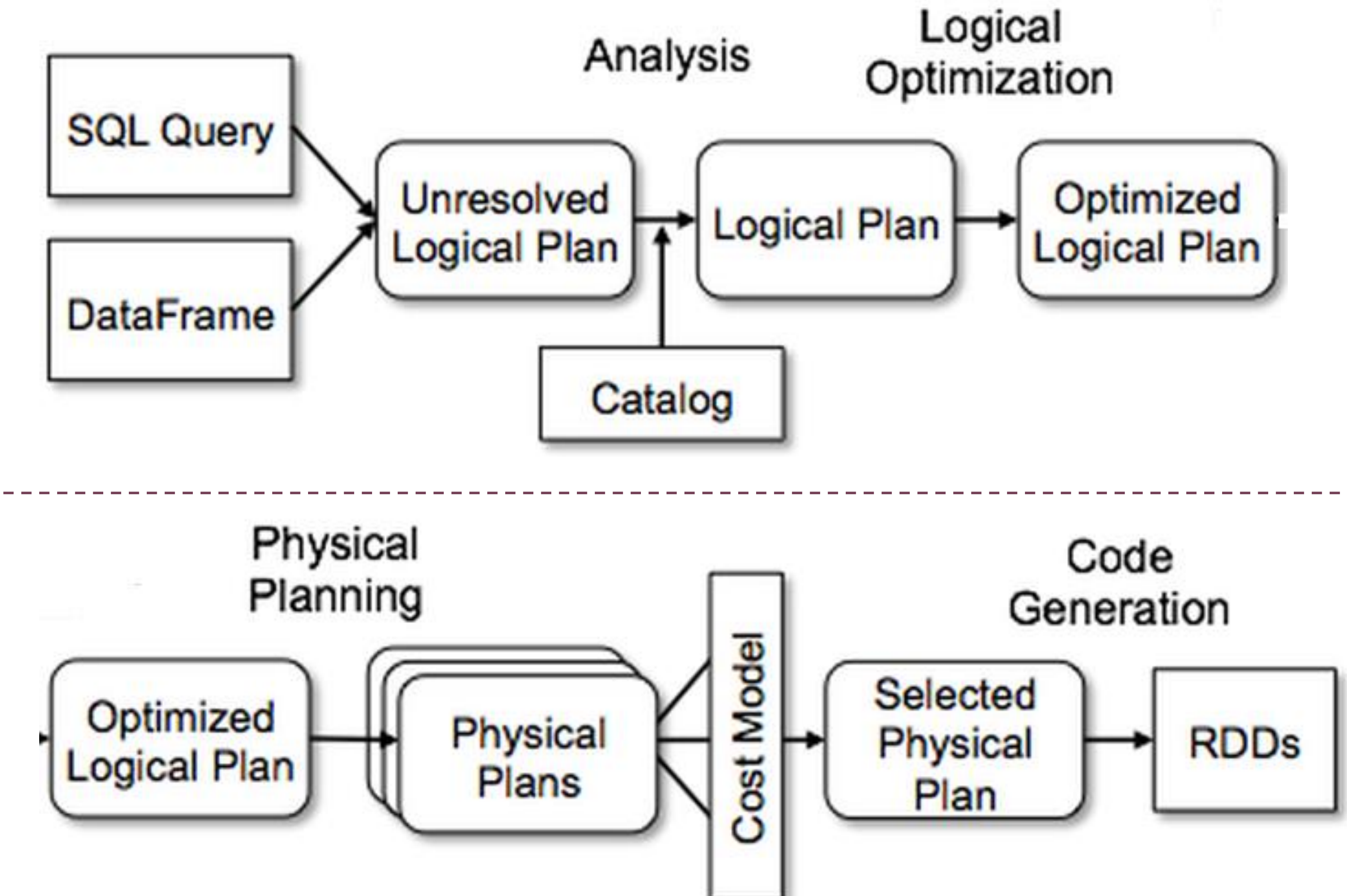
Important terms 2

- **Job**
  - part of application, driven by driver
- **Stage**
  - set of transformations which can be done without a shuffle
- **Task**
  - unit of work which is carried out by executor on some pice of data



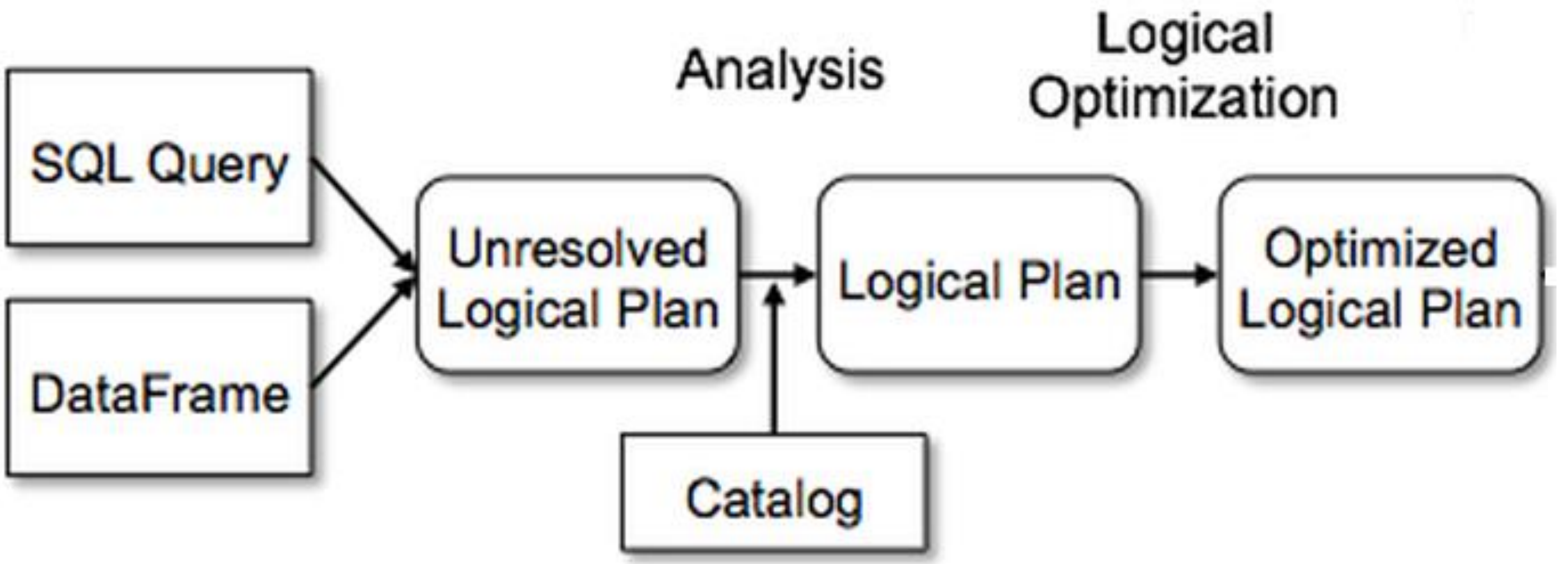
PROFINIT

Planning and optimization



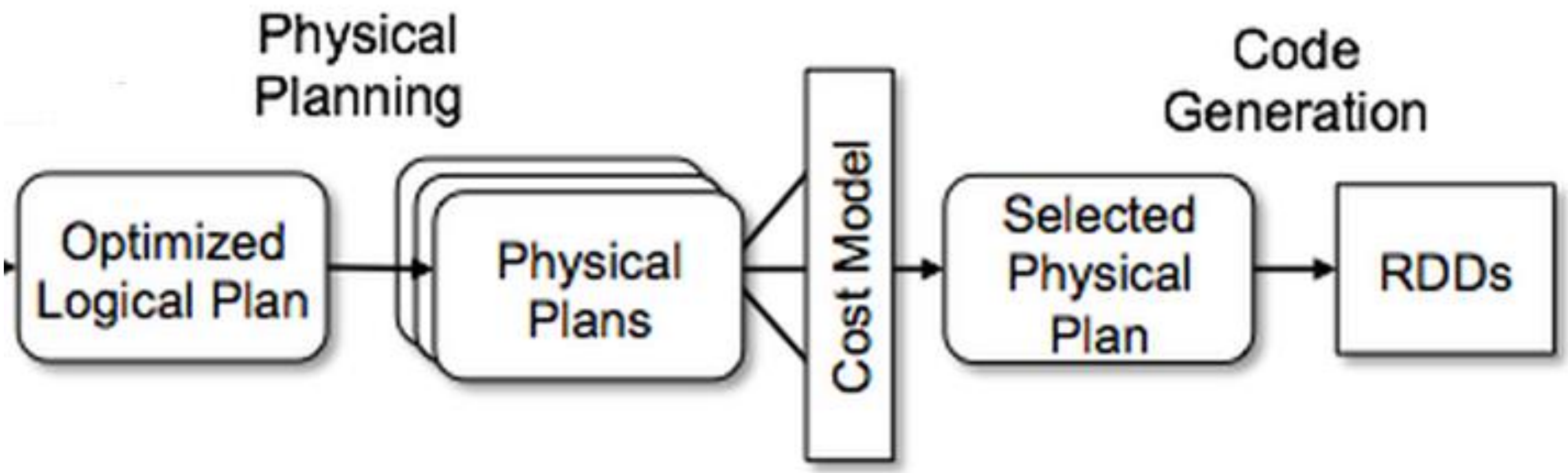
PROFINIT

Planning and optimization



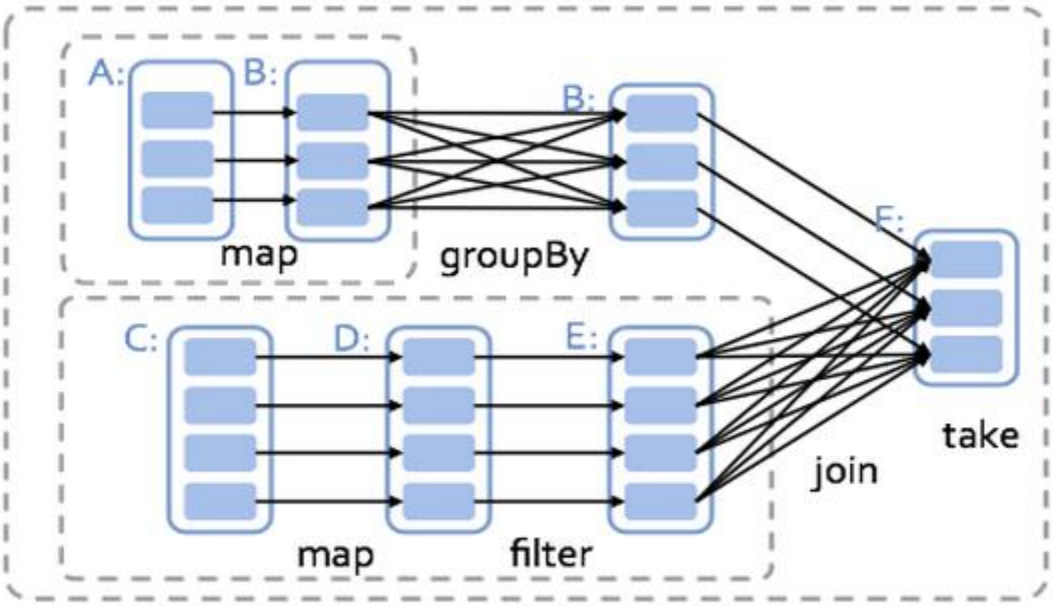
- typecasting
- rearranging of transformations (e. g. swap FILTER and JOIN)
- JOIN type selection, exploiting clustering, partitions, data skew
- etc.

Planning and optimization



- partitioninga and data distribution
- translating of transformations and actions to JVM language
- etc.

Example of graphic planning



- DAG – directed acylic graph
- describes computation flow
- dependencies (X must be done before Y)
- optimisation with respect to dependencies

Data partitioning

- **partition** – piece of data processed in one task
- by default 1 partition = 1 HDFS block = 1 task = 1 core
- partition processed on the node where is saved
- more partitions ⇨ more tasks ⇨ higher parallelism ⇨ smaller partition ⇨ lower efficiency ⇨ higher overhead
- ... and the other way around

Is it possible to set? And how?

- data input: ***`sc.textFile(file, partition_number`***`)`
- at run: ***`coalesce, repartition, partitionBy`***
- Every repartitioning causes a shuffle!

PROFINIT

Launching and configuration

Spark launching

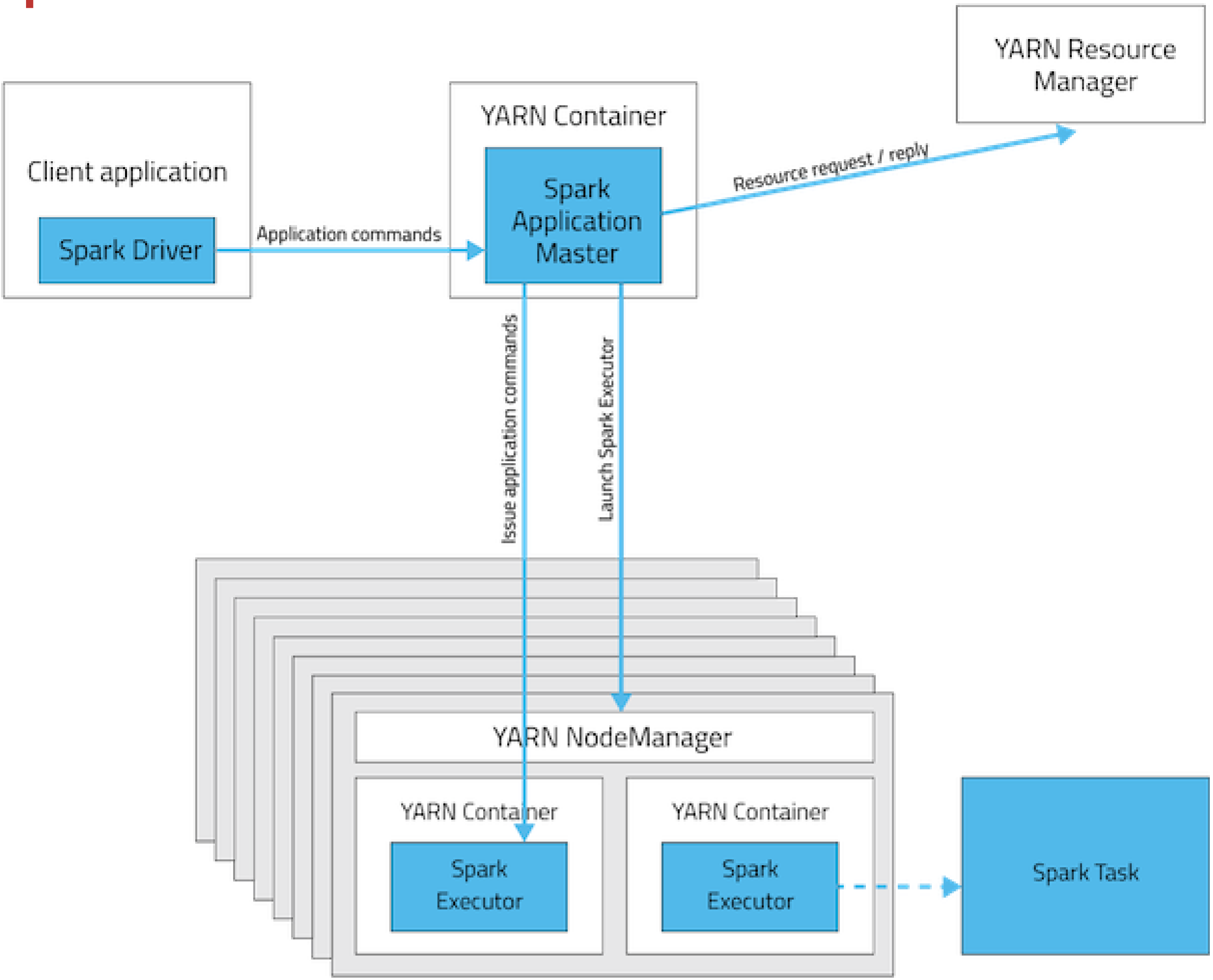
```
pyspark | spark-shell | spark-submit --param value
```

Where and how it runs

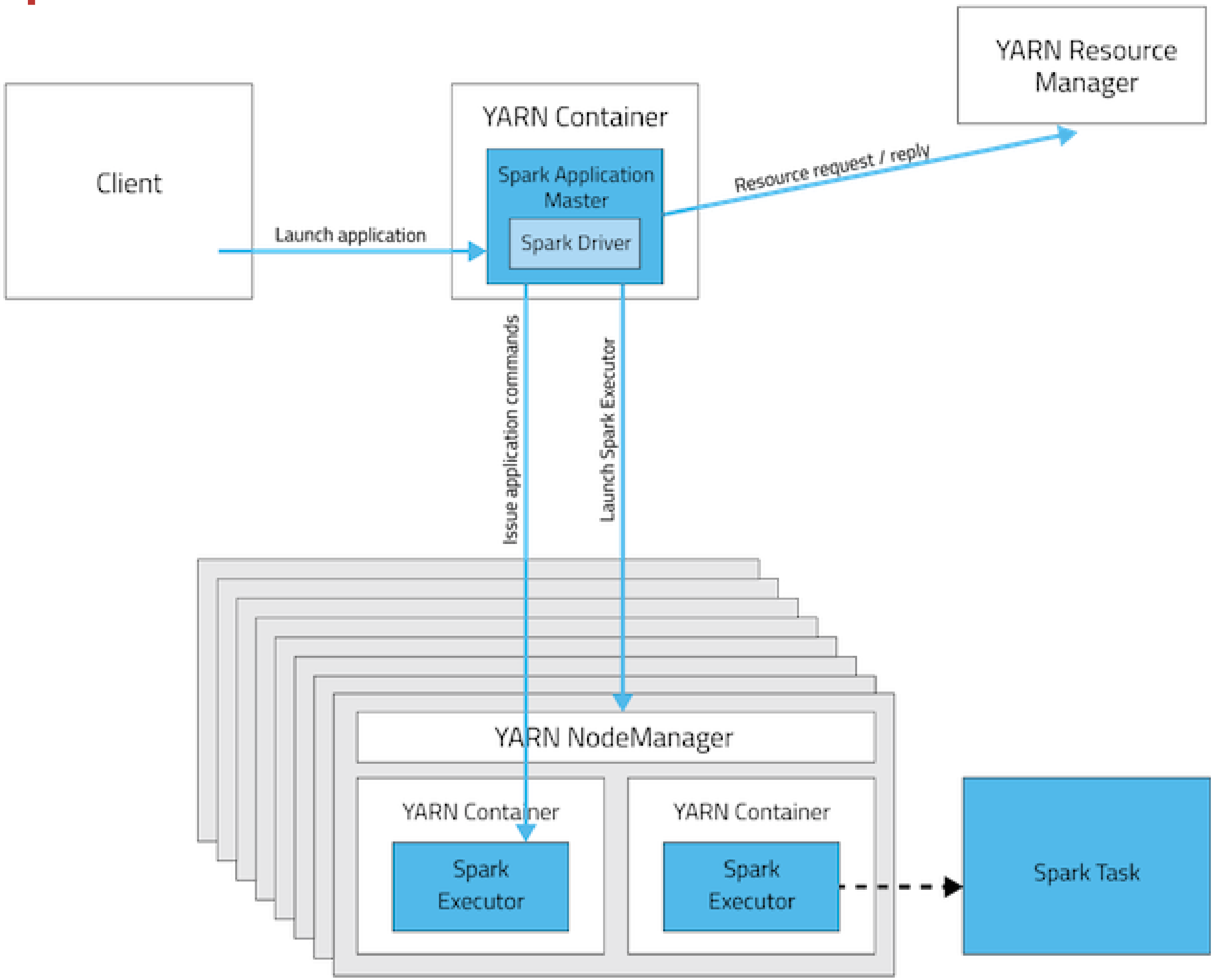
- on the cluster – full parallelism
  - mode client
  - mode cluster
- locally – parallel run on multiple cores
- determined by parameters –master, --deploy-mode

PROFINIT

Spark on YARN client mode



Spark on YARN cluster mode



PROFINIT

Mode client and mode cluster

- default mode = client
- **client** – good for an interactive work and debugging (output to the local console)
- **cluster** – good for production

PROFINIT

Spark configuration – requirement of sources

- --name
- --driver-memory
- --num-executors
- --executor-cores
- --executor-memory

Example

- `pyspark --master yarn --deploy-mode client --driver-memory 1G --num-executors 3 --executor-cores 2 --executor-memory 3G`

PROFINIT

Source allocation plan – example

Generally:

- --num-cores <= 5
- --executor-memory <= 64 GB

Let‘s say… cluster of 6 nodes, each 16 cores and 64 GB RAM

- 1 core and 1GB per node → OS
  6 * 15 cores and 63 GB remaining
- 1 core for Spark Driver: 6 * 15 – 1 = 89 cores remaining
- 89 / 5 ~ 17 executors; each node approx. 3 executors
- 63 GB / 3 ~ 21 GB memory for an executor
- due to memory overhead → set 19 GB for an exekutor

PROFINIT

Thank for your attention

PROFINIT

Profinit, s.r.o.
Tychonova 2, 160 00 Praha 6

Telefon
+ 420 224 316 016

Web
www.profinit.eu

LinkedIn
linkedin.com/company/profinit

Twitter
twitter.com/Profinit_EU