DĚLAT
DOBRÝ SOFTWARE
NÁS BAVÍ

PROFINIT

Spark

Jan Hučín

15. listopadu 2017

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

PROFINIT

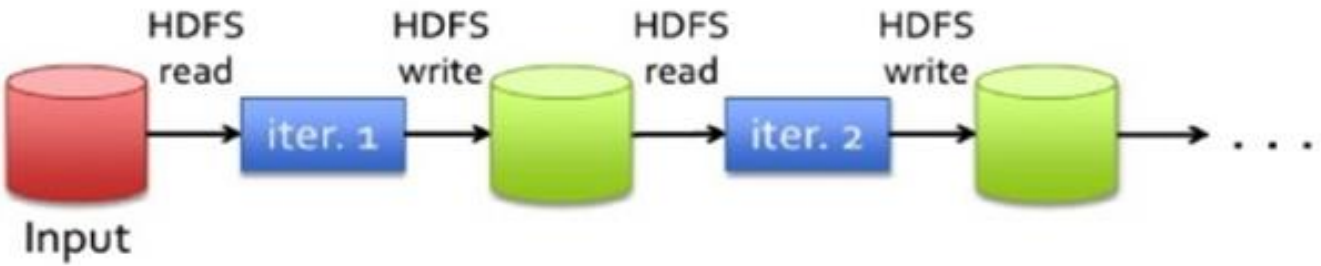
What is Spark

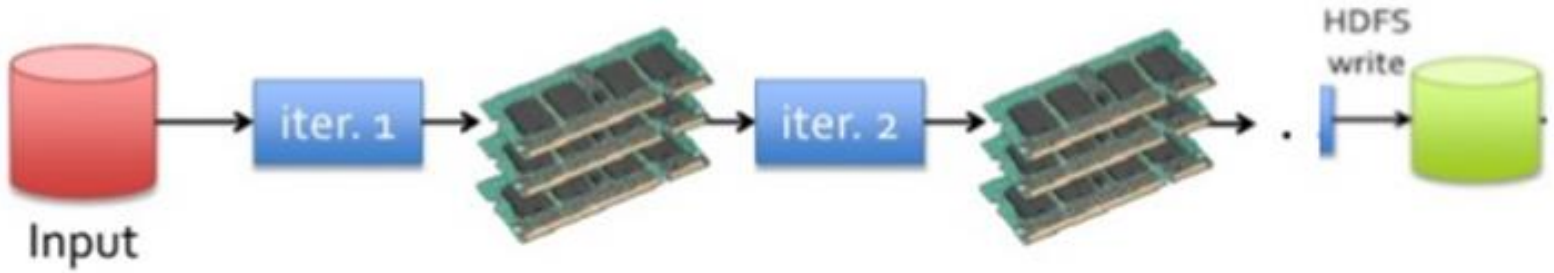
(For) what is Spark

- 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

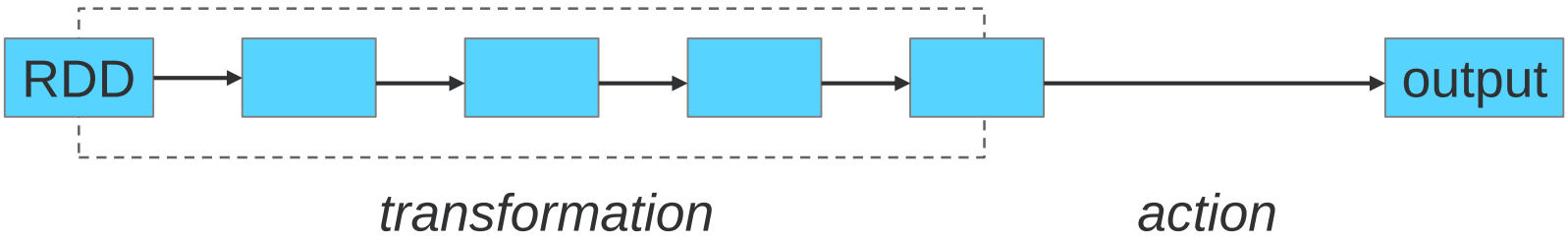
How to work with it

- 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)



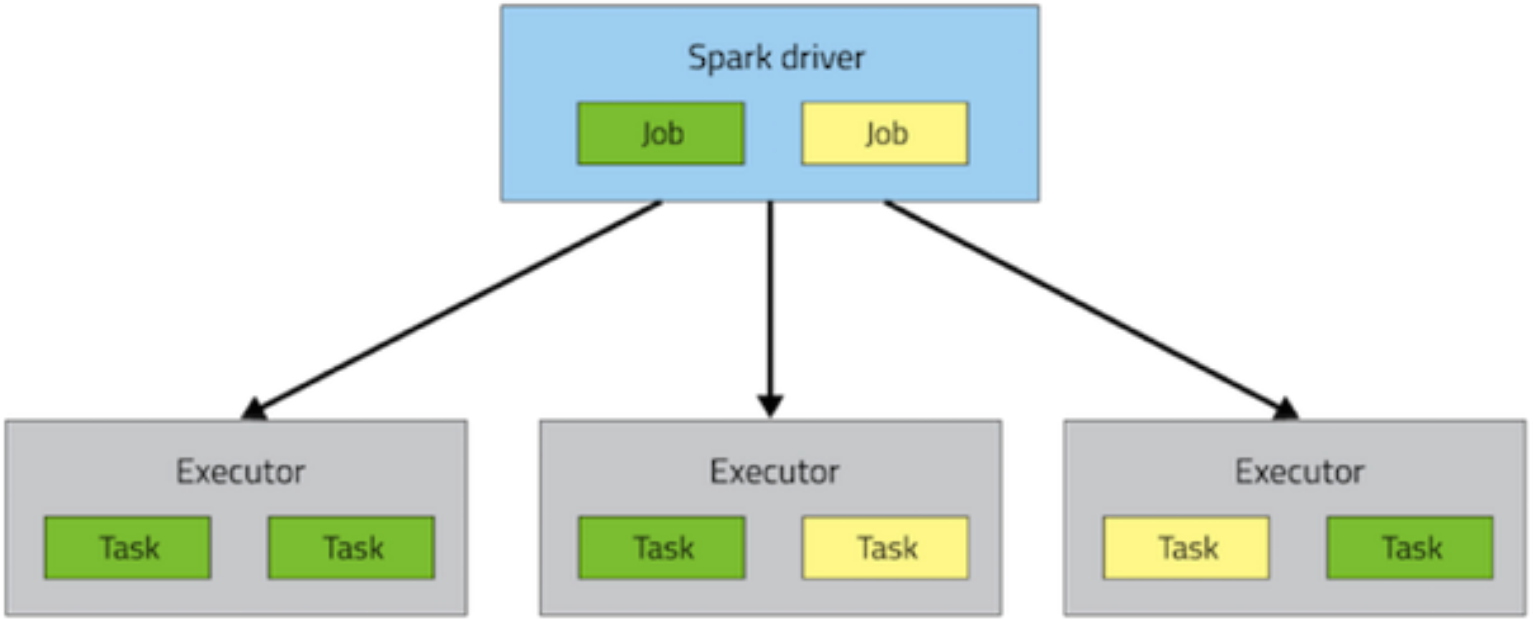
- series of transformations finished by action
- transformation are **planned** and **optimized**, but **not made**
- **lazy evaluation:** action starts all process

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

PROFINIT

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

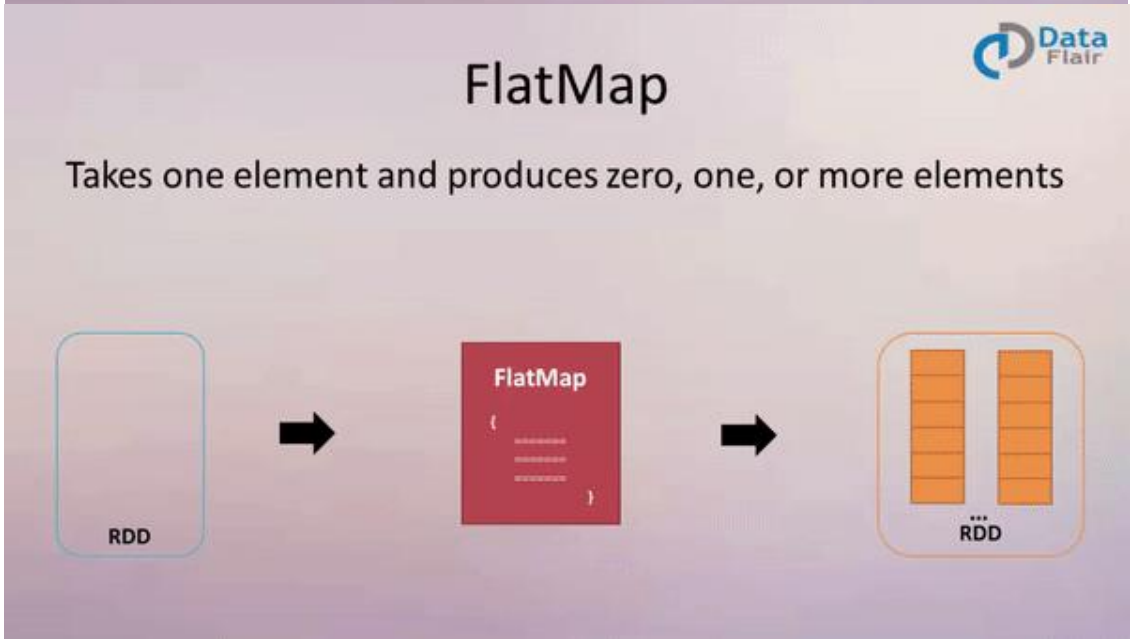
How to get „key“?

"tuple"
(Scala, Python)

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

PROFINIT

map a flatMap



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()`

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

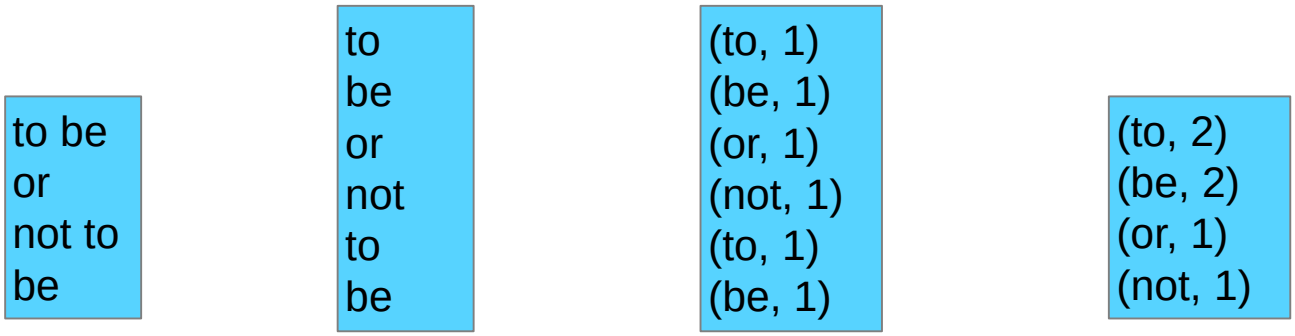
```
words = lines.flatMap(lambda line: line.split(" "))
```

- 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)
```



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

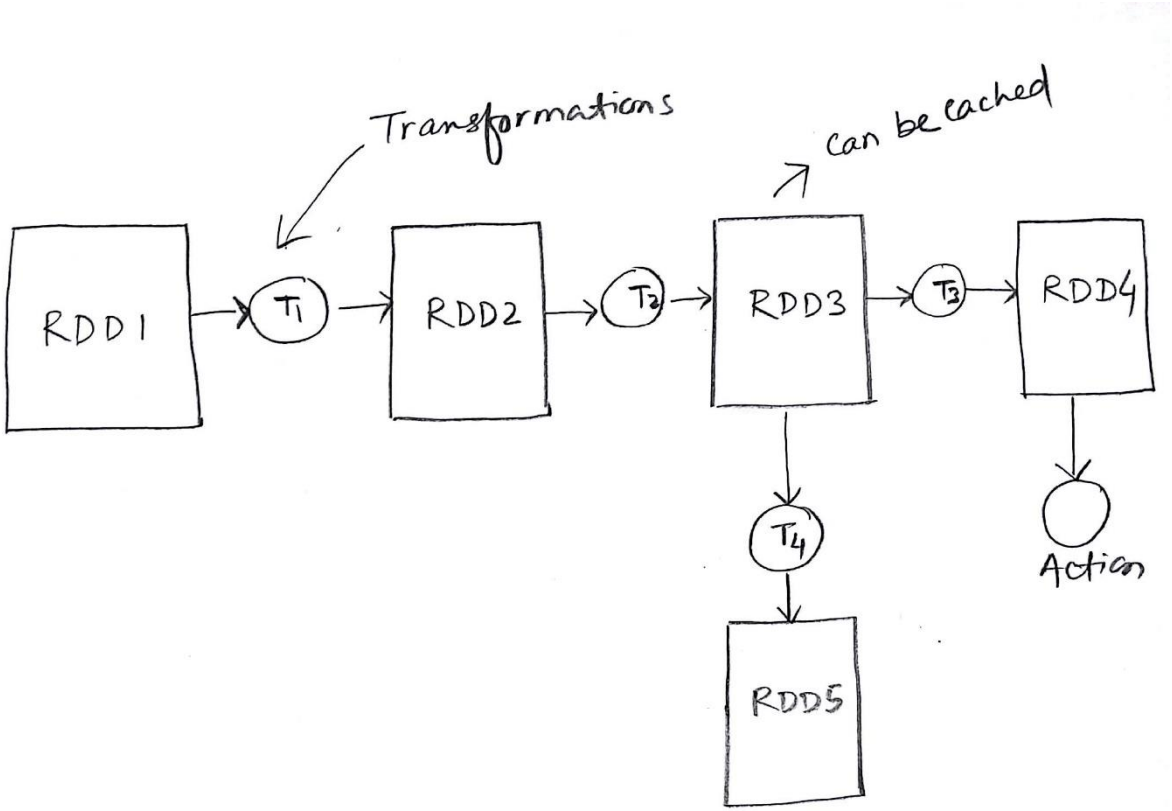
- Action starts all chain from the beginning!
  - After run, all results are „forgotten“.
  - **If we don‘t want this, we have to cache some RDD.**

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!

Spark program as a graph



PROFINIT

Příklad 2 – podobnost obrázků

- Úkol: spočítat podobnosti mezi dvojicemi obrázků
- Vstup: čb soubory BMP (4×4, 256 barev) – prvky RDD
- Postup:
  - parsing: binární BMP ⇨ posloupnost 16 čísel 0/1
  - vytvoření dvojic obrázků
  - výpočet podobnosti dvojic obrázků
- Výsledek transformace:
  RDD s prvky (soubor1, soubor2, podobnost)
- Akce na konci – např. uložení do textového souboru

Příklad 2 – podobnost obrázků

Úkol: spočítat podobnosti mezi dvojicemi obrázků

Vstup: čb soubory BMP (4×4, 256 barev) – prvky RDD

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

Postup:

- parsing: binární BMP ⇨ posloupnost 16 čísel 0/1

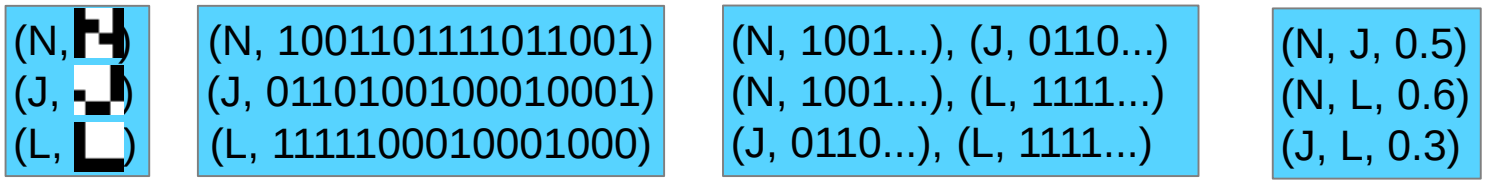
```
filesParsed = files.map(parseBMP)
```

- vytvoření dvojic obrázků

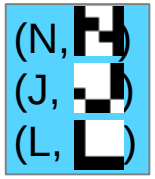
```
filesPairs = filesParsed.cartesian(filesParsed) \
    .filter(lambda f: f[0][0]<f[1][0])
```

- výpočet podobnosti dvojic obrázků

```
simil = filesPairs.map(similPair)
```



Příklad 2 – parsing



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

```
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)
```

PROFINIT

Příklad 2 – podobnost

```
((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

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

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

- takeOrdered

- 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

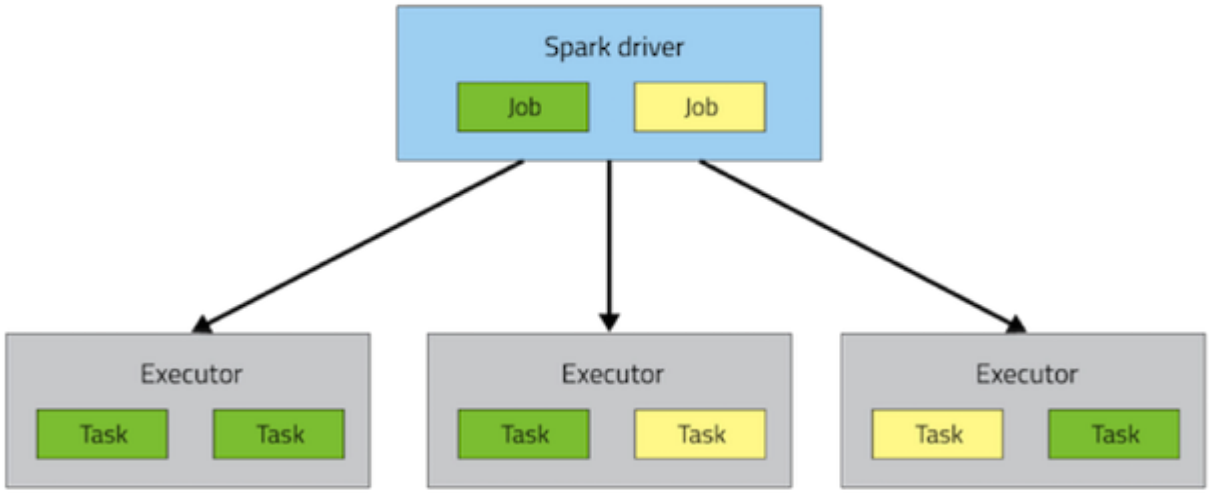
- keys
- values

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

Spark architecture

Důležité pojmy 1

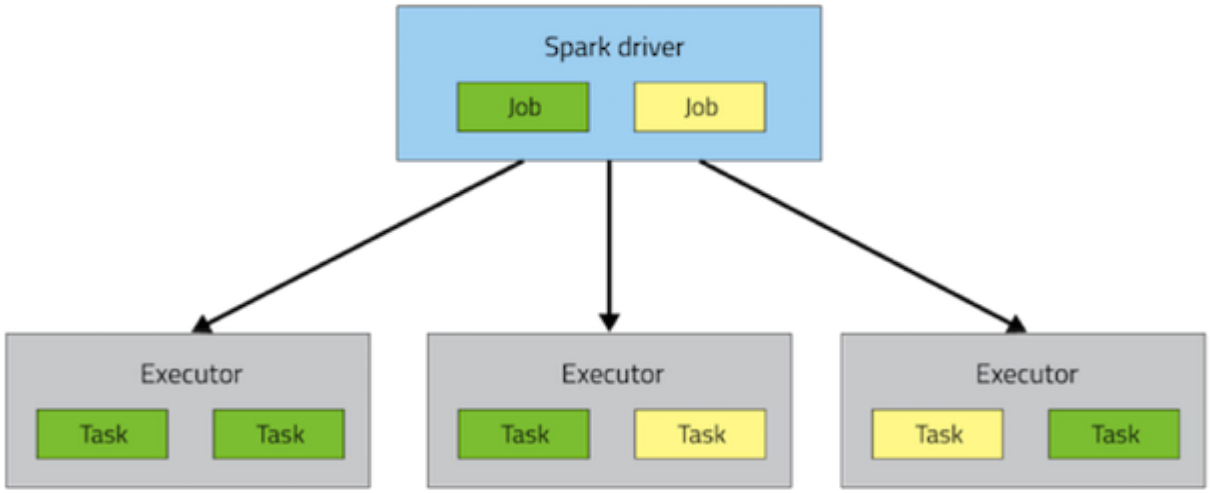
- **Application master**
  - proces zodpovědný za vyjednání výpočetních zdrojů od res. manageru
- **Driver**
  - hlavní proces
  - plánuje workflow
  - distribuuje práci do exekutorů
- **Executor**
  - proces běžící na některém z nodů (ideálně na každém)
  - provádí tasky (může i několik paralelně)



PROFINIT

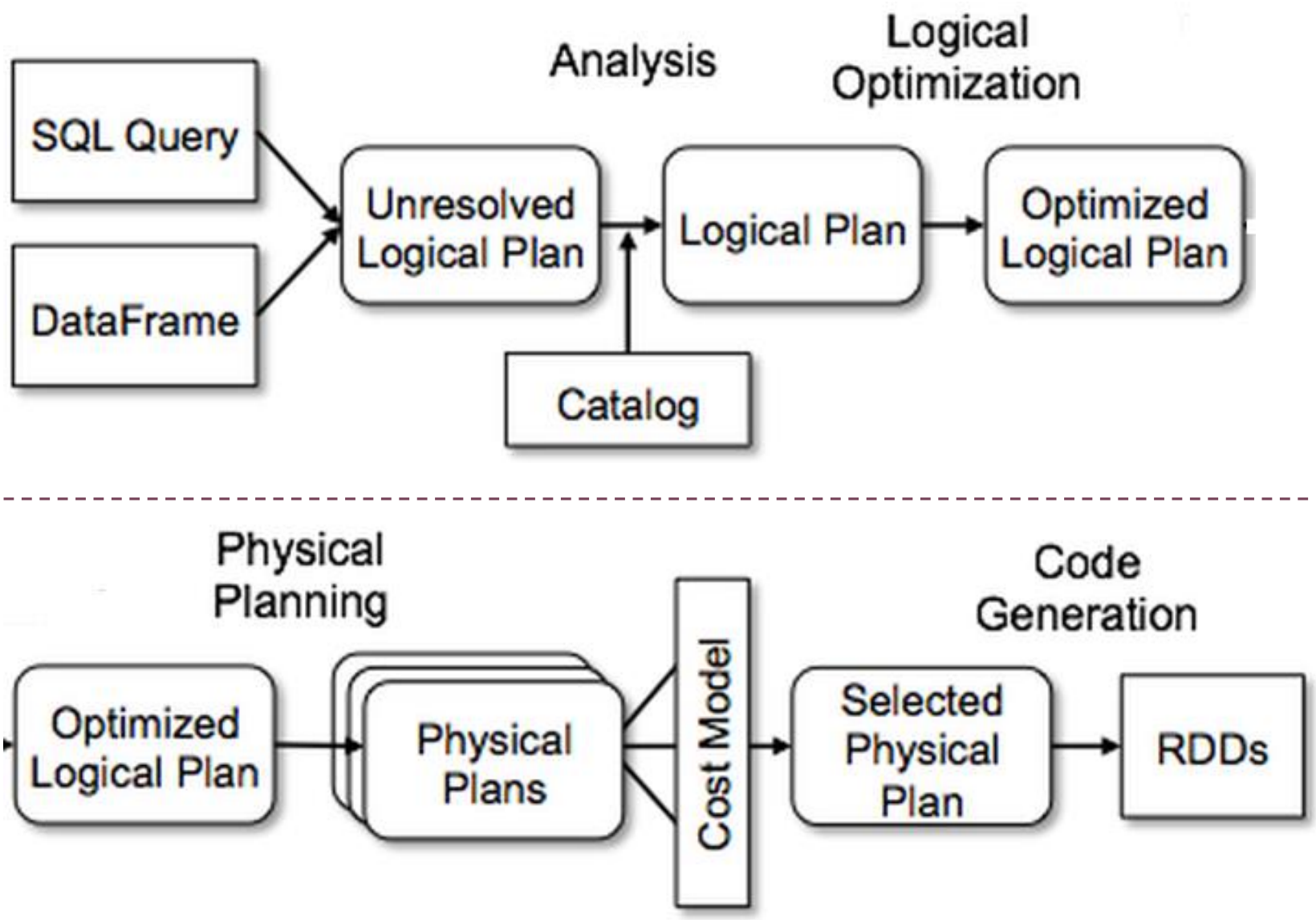
Důležité pojmy 2

- **Job**
  - akce volaná uvnitř programu driveru
- **Stage**
  - sada transformací, které mohou být vykonány bez shuffle
- **Task**
  - jednotka práce, kterou provádí exekutor na nějakém kousku dat



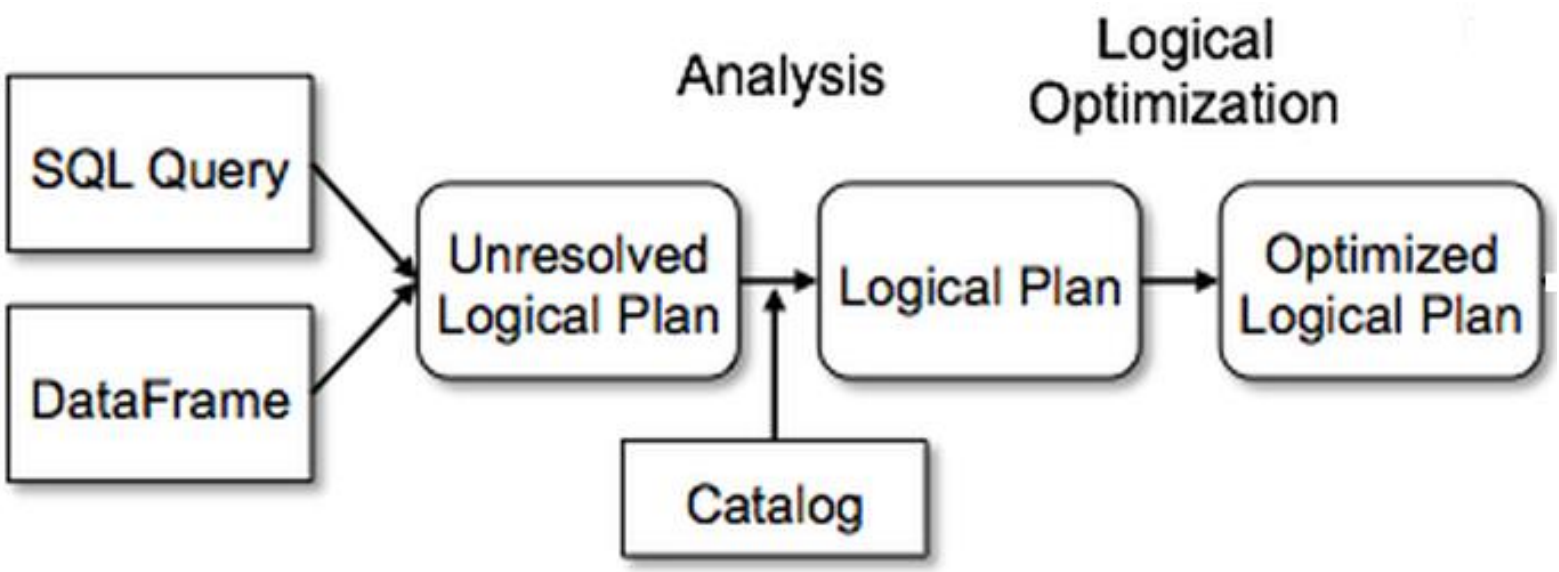
PROFINIT

Plánování a optimalizace



PROFINIT

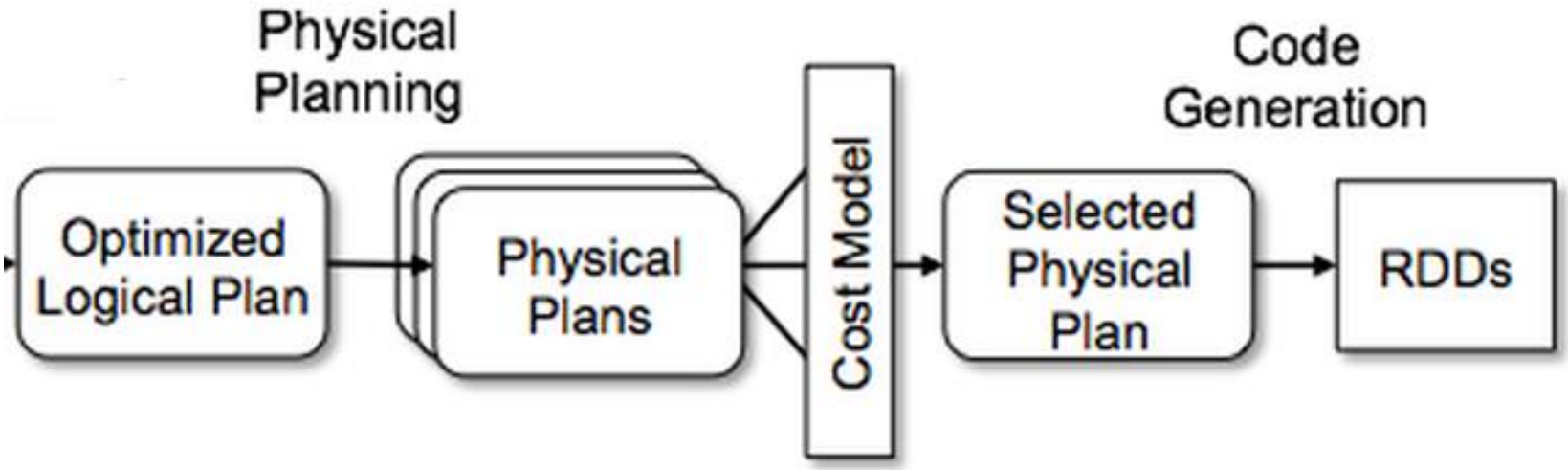
Plánování a optimalizace



- přetypování
- posloupnost transformací (např. přehození FILTER a JOIN)
- volba typu JOINu, využití clusterování, partitions, skew apod.
- atd.

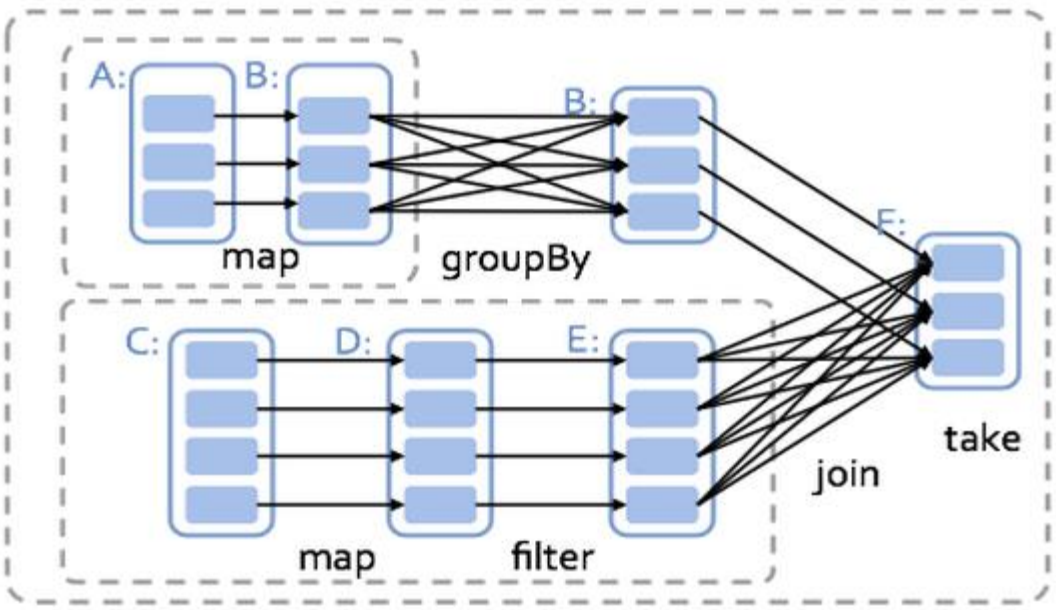
PROFINIT

Plánování a optimalizace



- rozdělení a distribuce dat
- překlad transformací a akcí do příkazů pro JVM
- atd.

Ukázka



- DAG – graf popisující průběh výpočtu
- určení závislostí (X musí být uděláno před Y)
- optimalizace v rámci dodržení závislostí

PROFINIT

Rozdělení dat – partitions

- **partition** – část dat zpracovaná v jednom tasku
- defaultně 1 partition = 1 HDFS block = 1 task = 1 core
- partition se zpracuje na nodu, kde je uložená
- více partitions ⇨ víc tasků ⇨ vyšší paralelizace ⇨ menší velikost jedné partition ⇨ nižší efektivita ⇨ vyšší overhead
- ... a naopak

Lze ovlivnit? A jak?

- při vstupu: např. `sc.textFile(soubor, počet_part)`
- za běhu: `coalesce, repartition, partitionBy`
- shuffle!

PROFINIT

Spuštění a konfigurace

Spuštění Sparku

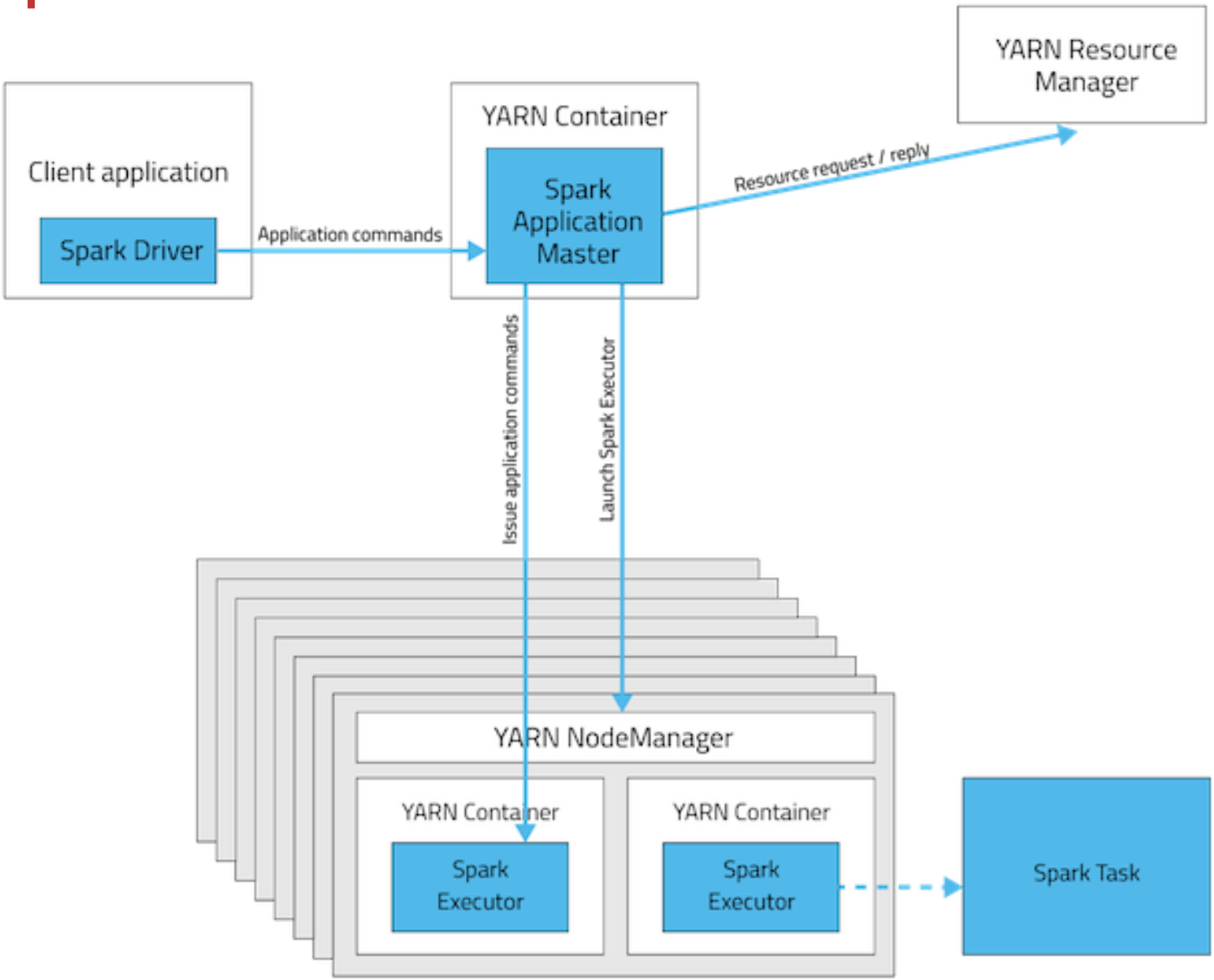
```
pyspark | spark-shell | spark-submit --param value
```

Kde a jak poběží

- na clusteru – plné využití paralelismu
  - mod client
  - mod cluster
- lokálně – paralelní běh na více jádrech
- určeno parametry --master a --deploy-mode

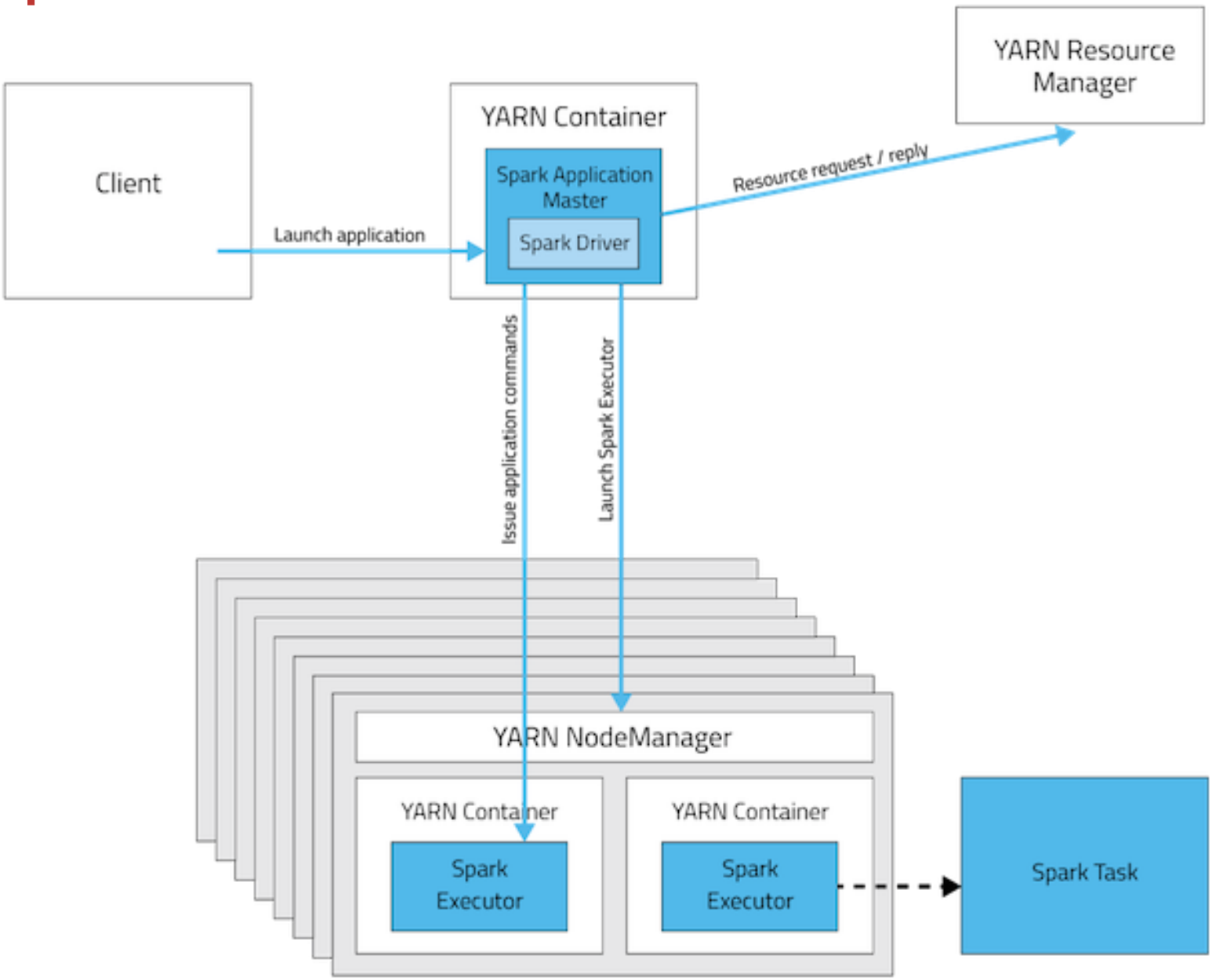
PROFINIT

Spark on YARN client mode



PROFINIT

Spark on YARN cluster mode



Mod client versus mod cluster

- default je client
- mod client je vhodný pro interaktivní práci a debugging (výstup jde na lokální konzolu)
- mod cluster je vhodný pro produkční účely

PROFINIT

Konfigurace běhu Sparku – požadavky na zdroje

- --name *jméno aplikace*
- --driver-memory *paměť pro driver*
- --num-executors *počet exekutorů*
- --executor-cores *počet jader pro exekutor*
- --executor-memory *paměť pro exekutor*

Příklad

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

PROFINIT

Příklad plánu alokace zdrojů

Obecná doporučení:

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

Cluster 6 nodů, každý 16 jader a 64 GB RAM

- Rezervovat 1 jádro a 1GB /node pro OS
  zbývá 6 *15 jader a 63 GB
- 1 jádro pro Spark Driver: 6 * 15 – 1 = 89 jader.
- 89 / 5 ~ 17 exekutorů. Každý node (kromě toho s driverem) bude mít 3 exekutory.
- 63 GB / 3 ~ 21 GB paměti na exekutor. Navíc se musí počítat s memory overhead -> nastavit 19 GB na exekutor

PROFINIT

Díky za pozornost

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