DĚLAT
DOBRÝ SOFTWARE
NÁS BAVÍ

PROFINIT

Spark

Jan Hučín

15. listopadu 2017

Osnova

1. Co to je a k čemu slouží
2. Jak se to naučit
3. Jak se s tím pracuje
4. Jak to funguje
   - Logický a technický pohled
   - Transformace, akce, kešování
   - Příklady
   - Architektura a alokace zdrojů

Příště:

- Spark SQL
- Spark streaming

PROFINIT

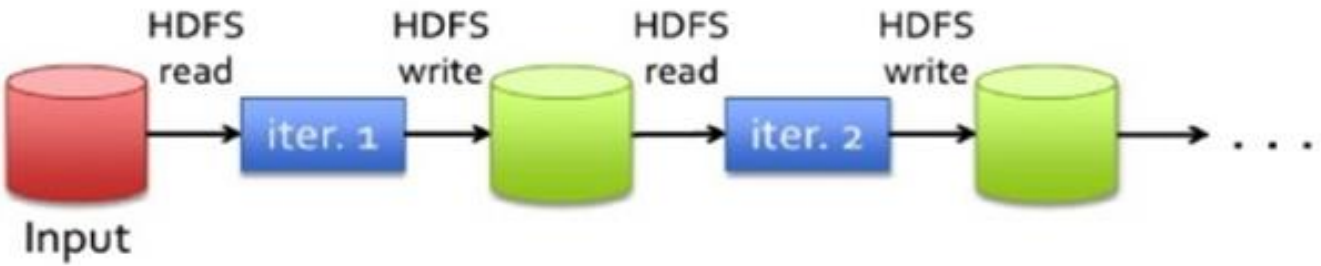
Co je Spark

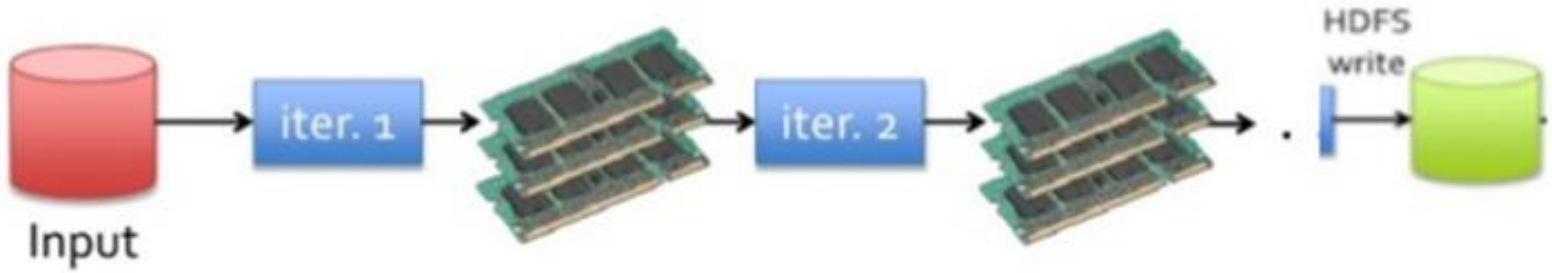
Co je to Spark a k čemu slouží

- framework pro distribuované výpočty
- vylepšení původního map-reduce, o 2 řády rychlejší
  - **zpracování v paměti** – méně I/O operací, vhodné pro iterativní algoritmy nebo analýzu dat
  - **optimalizace operací** před zpracováním
  - nyní i **pseudo-SQL** příkazy
- API pro Scalu, Javu, Python a R
- s Hadoopem (využívá HDFS, YARN) i samostatně
- napsán ve Scale, běží na JVM
- nejaktivnější opensource Big Data projekt

PROFINIT

Spark vs. map-reduce





PROFINIT

Vhodné úlohy

- dostatečně velké, ale ne extrémně
- dobře paralelizovatelné
- iterační
- obtížné pro stávající technologie

Např.

1. výpočty charakteristik klientů (riskové skóre, zůstatky)
2. ETL pro odlehčení DWH
3. noční výpočet – celodenní využití
4. hledání vazeb v síti
5. text-mining

PROFINIT

Nevhodné úlohy

- příliš malé
- s extrémními požadavky na paměť
- šité na míru jiné technologii (SQL, Java)
- špatně paralelizovatelné
- real-time

Např.

1. modelování na malých datech
2. výpočet mediánu, náhodné přeskoky mezi řádky souboru
3. JOIN několika opravdu velkých tabulek

Jak se to naučit

- http://spark.apache.org
- aspoň základy Python | Scala | Java | R
- **vlastní praxe**
- rady zkušených, StackOverflow apod.

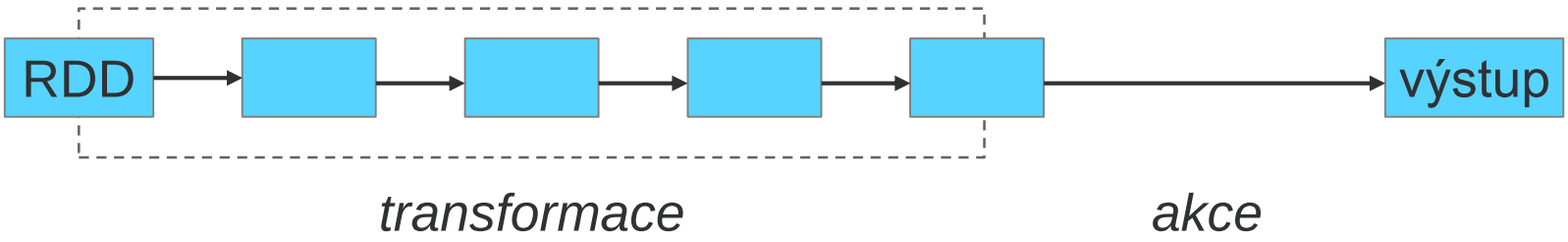
PROFINIT

Jak se s tím pracuje

- interaktivně
  - z příkazové řádky (shell pro Python i Scalu)
  - Zeppelin/Jupyter notebook
- dávka / aplikace
  - zkompilovaný .jar soubor
  - skript Pythonu

Jak funguje Spark

Pohled high level (logický)



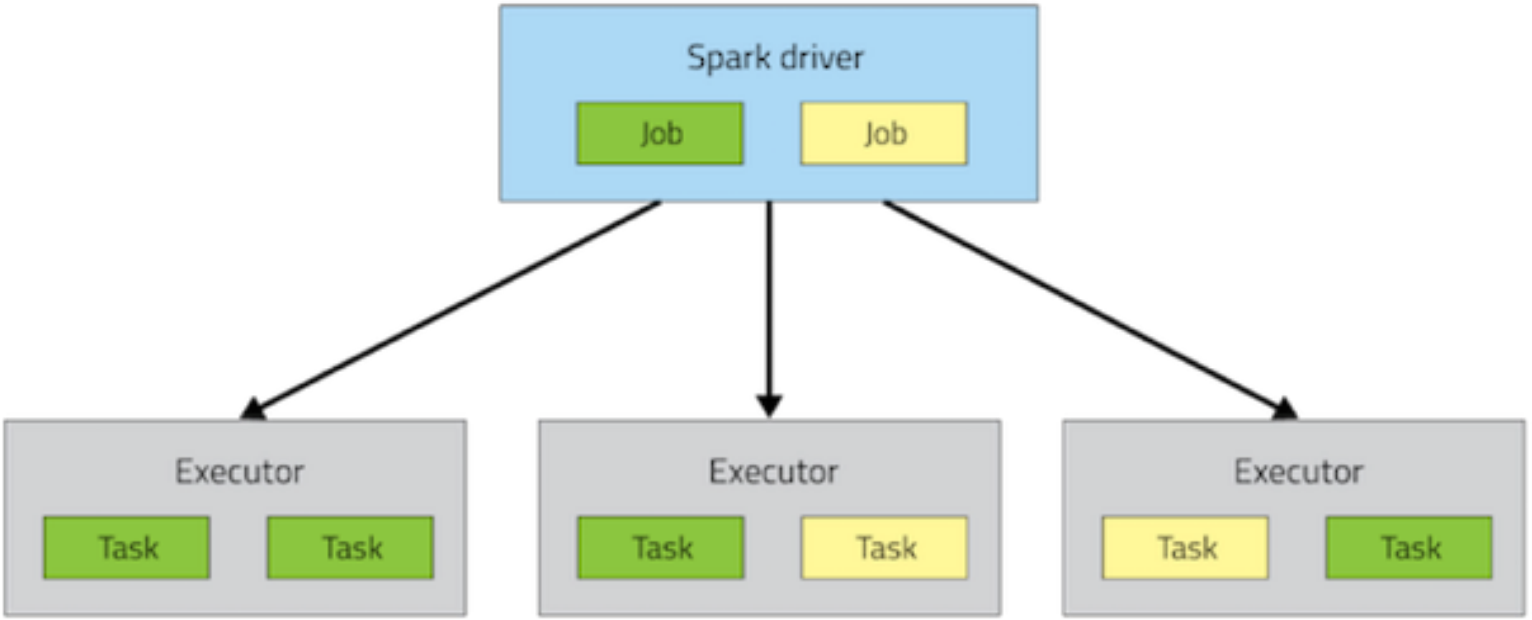
- série transformací zakončená akcí
- transformace se **plánují** a **optimalizují**, ale zatím **neprovádějí**
- **lazy evaluation:** až první akce spustí celý proces

Co je to RDD?

- resilient distributed dataset
- kolekce prvků (např. řádky v textovém souboru, datová matice, posloupnost binárních dat)
- musí být dělitelné na části – místo rozdělení (spolu)určí Spark!

PROFINIT

Pohled mid level (technický)



- vytvoření JVM na nodech (exekutory)
- rozdělení úlohy na joby a jobů na tasky
- distribuce tasků a případně dat na nody
- řízení procesu
- **více v architektuře Sparku**

PROFINIT

Spark RDD – transformace

RDD1 ⇨ RDD2, po prvcích („řádcích“)

- **map** (prvek ⇨ transformační funkce ⇨ nový prvek)
- **flatMap** (prvek ⇨ transformační funkce ⇨ 0 až N nových prvků)
- **filter**, **distinct** (pustí se jen řádky vyhovující podmínce / unikátní)
- **join** (připojí řádky jiného RDD podle hodnot klíče)
- **union**, **intersection** (sjednocení a průnik s jiným RDD)
- **groupByKey**, **reduceByKey** (setřídí / agreguje prvky podle klíče)
- ... a mnoho dalších

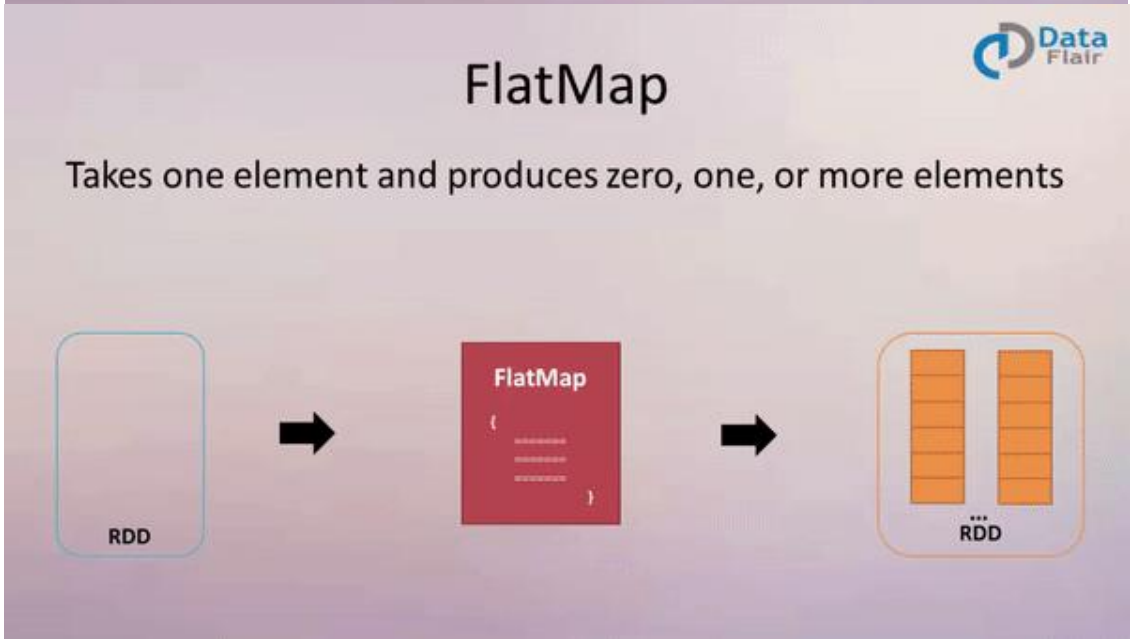
Kde vzít klíč?

"tuple"
(má Scala i Python)

- výsledek transformace, např. slovo ⇨ (slovo, 1)
- první prvek *tuple* se bere jako klíč

PROFINIT

map a flatMap



Příklad 1 – word count

- Úkol: spočítat četnosti slov v dokumentu
- Vstup: textový soubor rozdělený do řádků (RDD)
- Postup:
  - transformace řádků: řádek ⇨ rozdělení na slova ⇨ prvky typu (slovo, 1)
  - seskupení prvků se stejným klíčem a sečtení jedniček
- Výsledek transformace: RDD s prvky (slovo, četnost)

PROFINIT

Vsuvka – interaktivní shell

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

- spouští se z příkazové řádky
- funguje lokálně nebo v YARNu:
  - `pyspark --master local`
  - `pyspark --master yarn`
- vytvoří důležité objekty, např. `sc` (SparkContext), `sqlContext`
- má další parametry – o nich později
- ukončuje se `exit()`

PROFINIT

Příklad 1 – word count

Úkol: spočítat četnosti slov v dokumentu

Vstup: textový soubor rozdělený do řádků (RDD)

```
lines = sc.textFile("/user/pascepet/bible.txt")
```

Postup:

- transformace řádků: řádek ⇨ jednotlivá slova (více prvků)

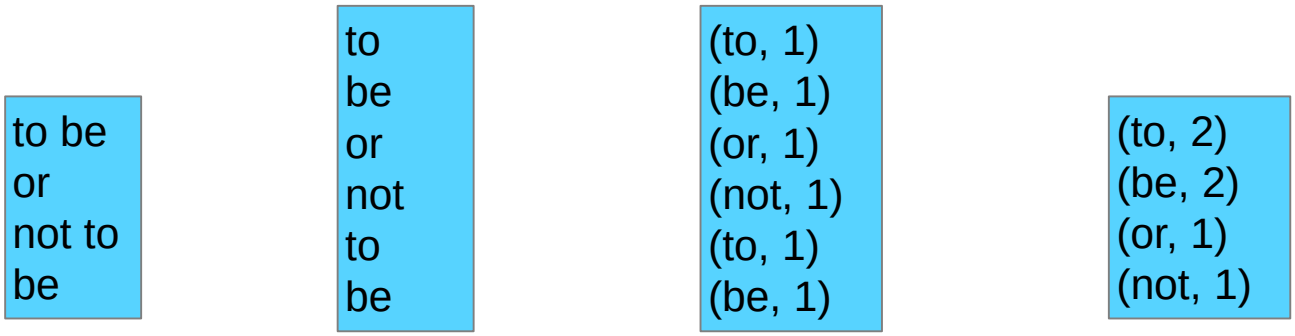
```
words = lines.flatMap(lambda line: line.split(" "))
```

- transformace řádků: řádek čili slovo ⇨ struktura (slovo, 1)

```
pairs = words.map(lambda word: (word, 1))
```

- seskupení prvků se stejným klíčem a sečtení jedniček

```
counts = pairs.reduceByKey(lambda a, b: a + b)
```



PROFINIT

Proč se nic nespočítalo?

Protože jsme zatím neprovedli žádnou akci.

Spark RDD – akce

- **reduce** (pomocí zadané funkce agreguje všechny prvky RDD)
- **take** (vypíše prvních *n* prvků RDD)
- **count** (počet prvků)
- **collect** (vypíše RDD jako pole prvků)
- **saveAsTextFile** (uloží jako textový soubor, resp. více txt souborů)
- ... a další

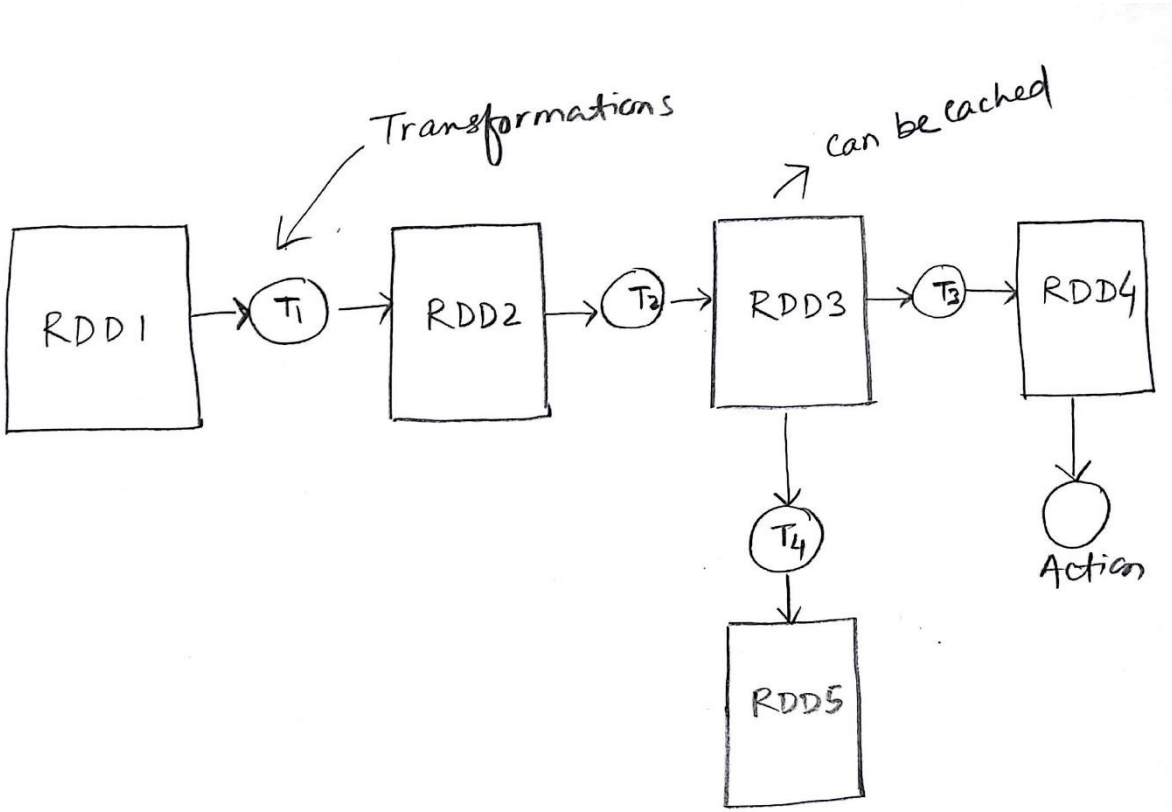
- Akce spustí celý řetězec od začátku!
  - Všechny mezivýsledky se zapomenou.
  - **Pokud to nechceme, musíme některé RDD uložit do keše.**

PROFINIT

Kešování

- Kešování: RDD se nezapomene, ale uchová v paměti / na disku.
- **Metody pro kešování:**
  - **cache** (pokusí se uchovat v paměti)
  - **persist** (obecnější, např. serializace, využití disku atd.)
  - **unpersist** (uvolnění RDD z paměti)
- **Typy kešování:**
  - MEMORY_ONLY
  - MEMORY_AND_DISK
  - MEMORY_ONLY_SER
  - MEMORY_AND_DISK_SER
- SER = serializace – úspora paměti, ale vyšší výpočetní náročnost
  - Volby se SER pouze Java a Scala, v Pythonu serializace vždy
- Kešování není akce!

Spark program jako graf



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

PROFINIT

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

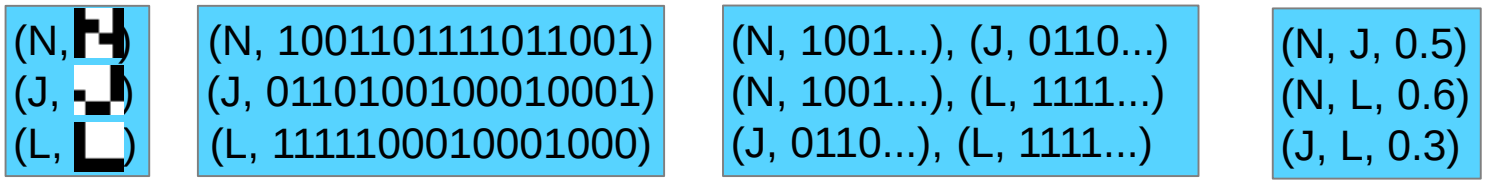
```
filesParsed = files.map(parseBMP)
```

- vytvoření dvojic obrázků

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

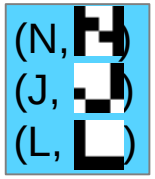
- výpočet podobnosti dvojic obrázků

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



PROFINIT

Příklad 2 – parsing




```
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)

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

Další operace Spark RDD

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

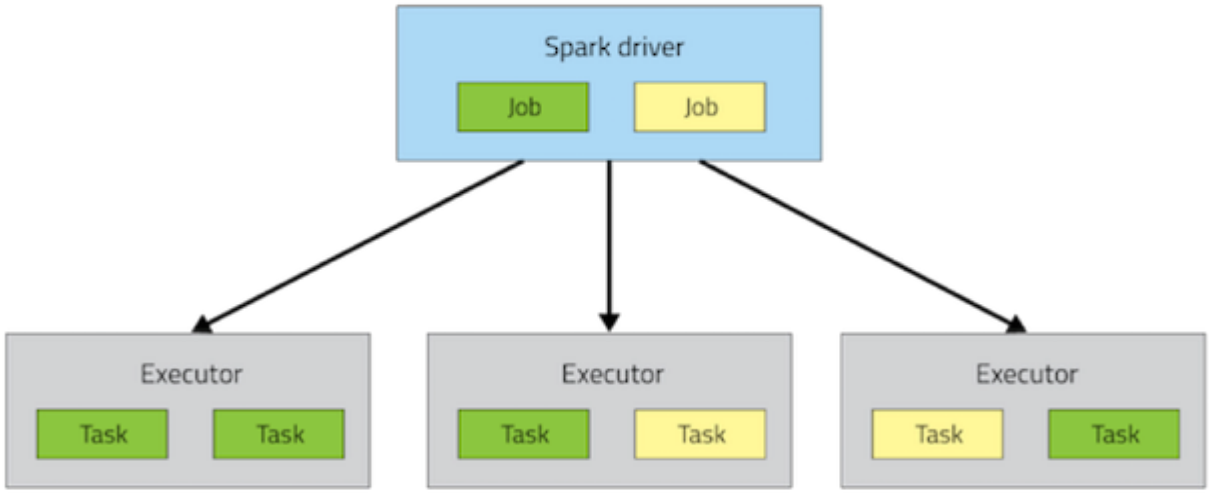
- keys
- values

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

Architektura Sparku

Důležité pojmy 1

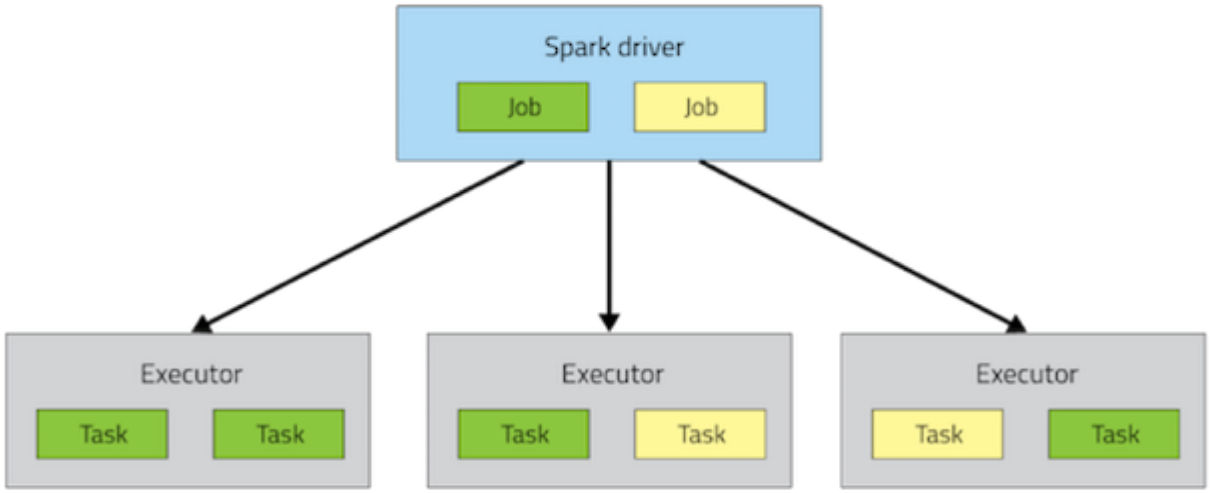
- **Application master**
  - proces zodpovědný za vyjednání výpočetních zdrojů od res. manageru
- **Driver**
  - hlavní proces
  - plánuje workflow
  - distribuuje práci do exekutorů
- **Executor**
  - proces běžící na některém z nodů (ideálně na každém)
  - provádí tasky (může i několik paralelně)



PROFINIT

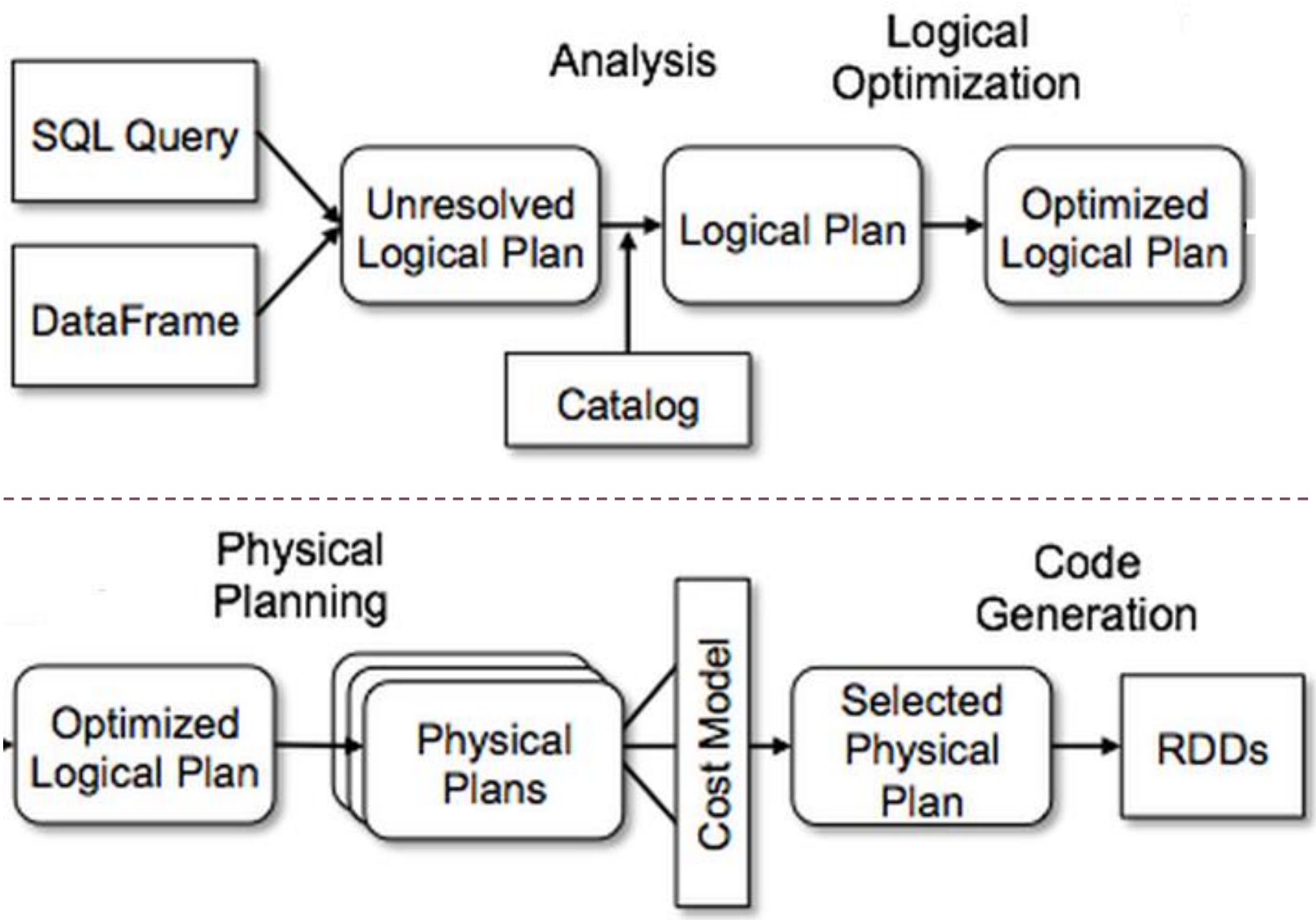
Důležité pojmy 2

- **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



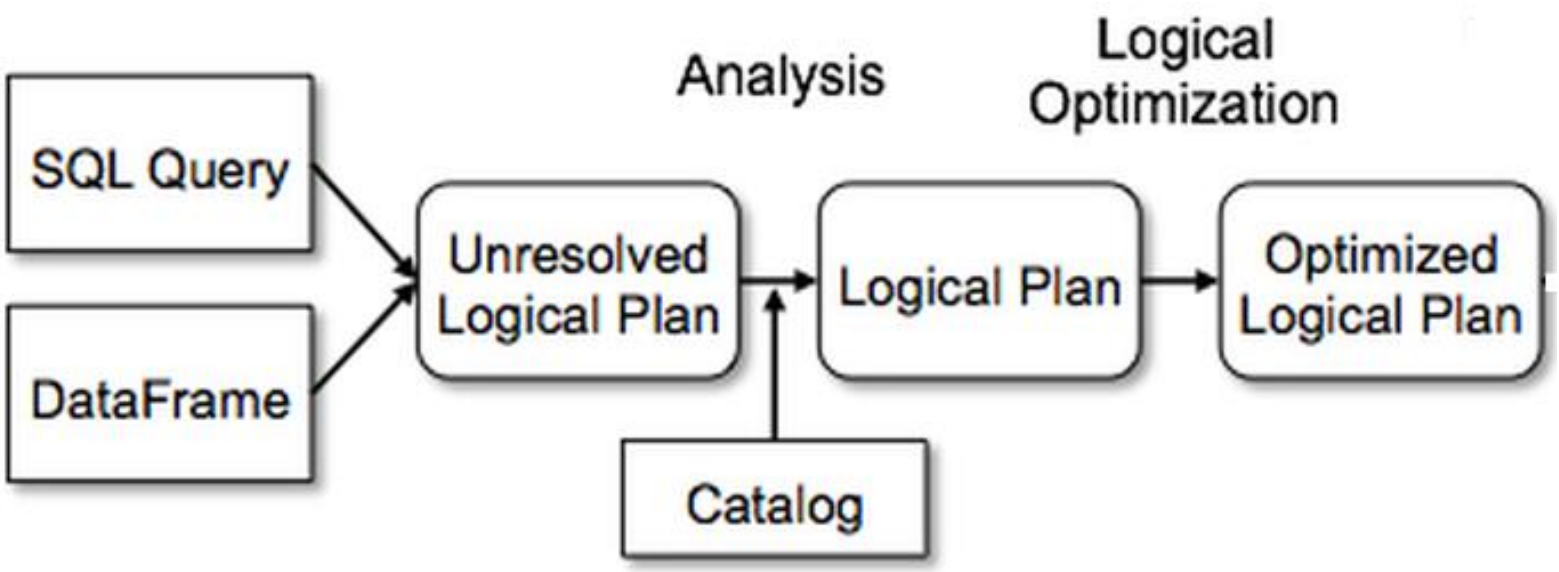
PROFINIT

Plánování a optimalizace



PROFINIT

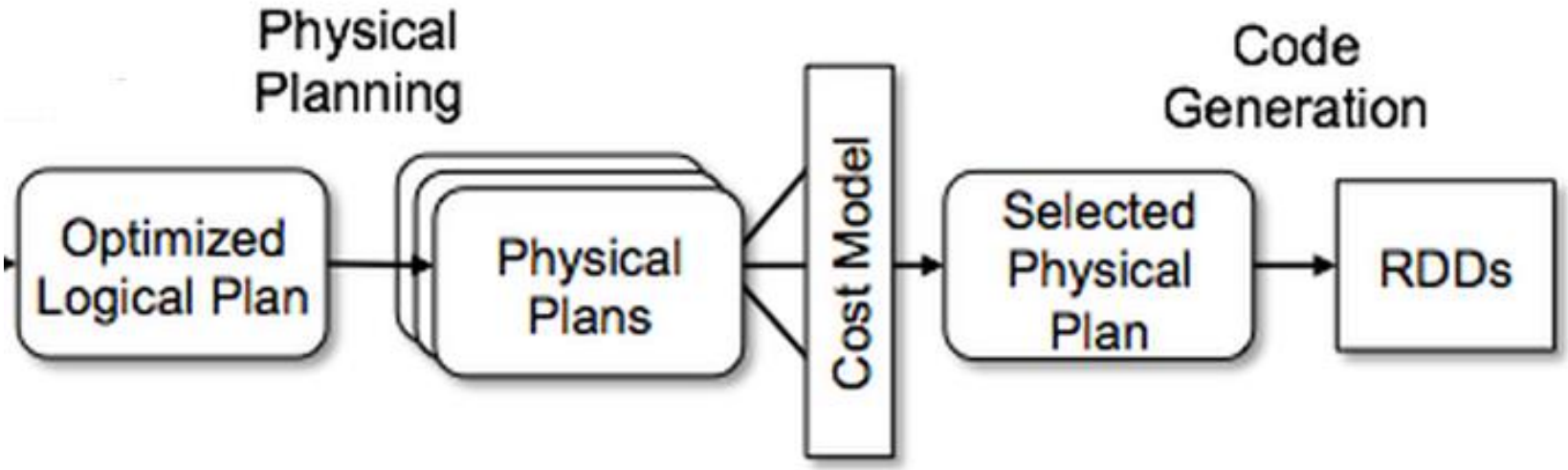
Plánování a optimalizace



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

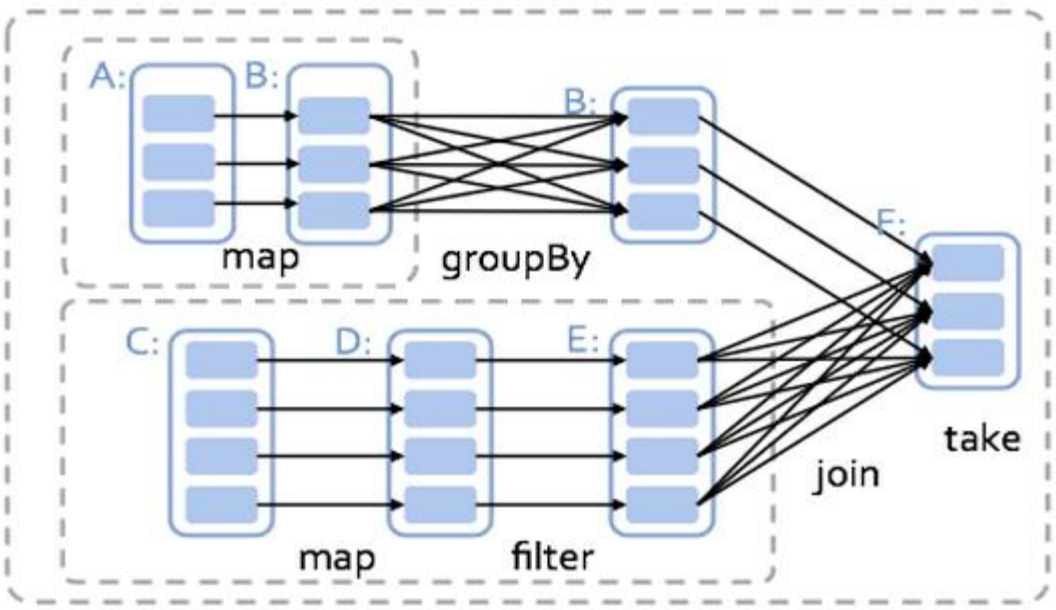
PROFINIT

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

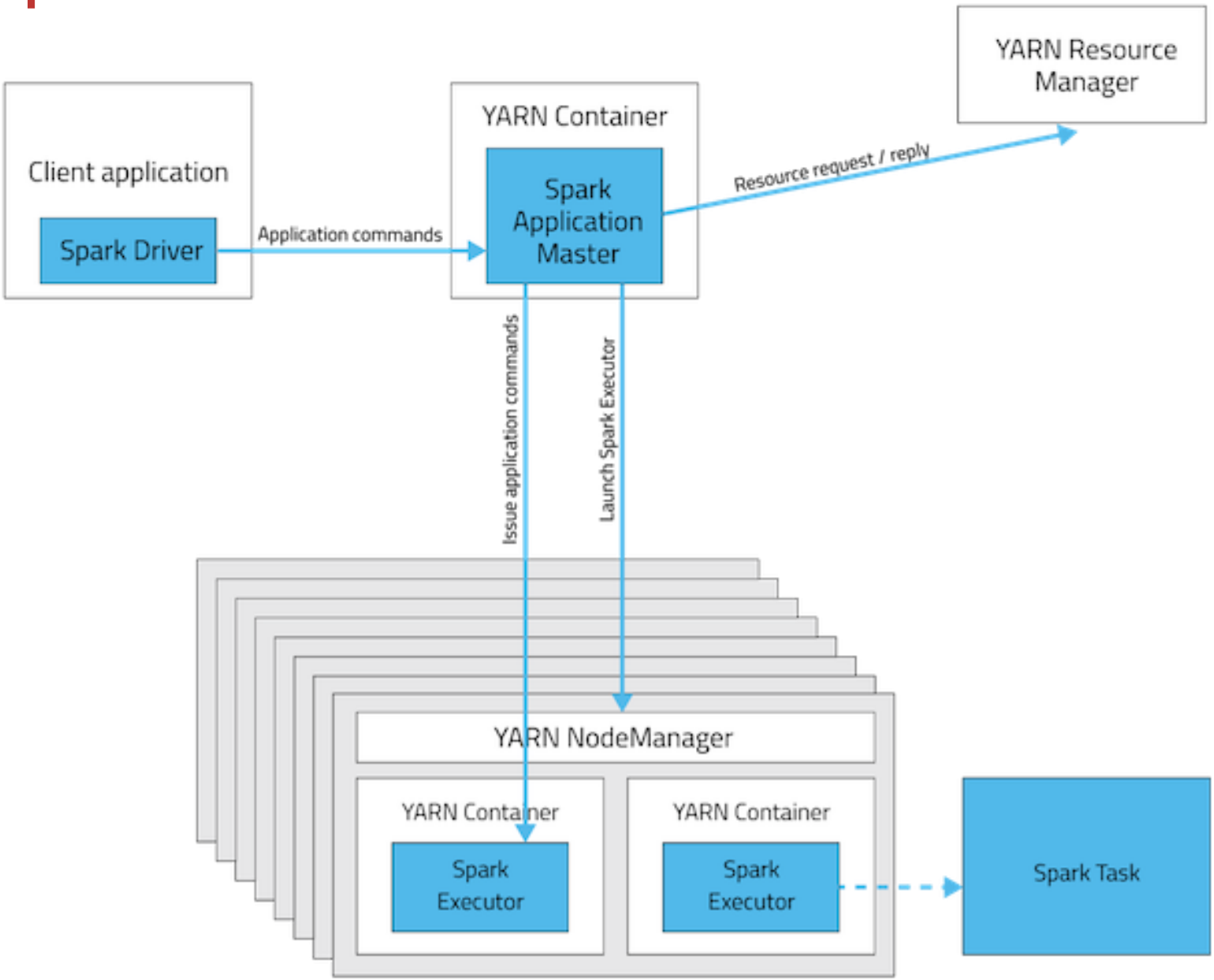
```
pyspark | spark-shell | spark-submit --param value
```

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

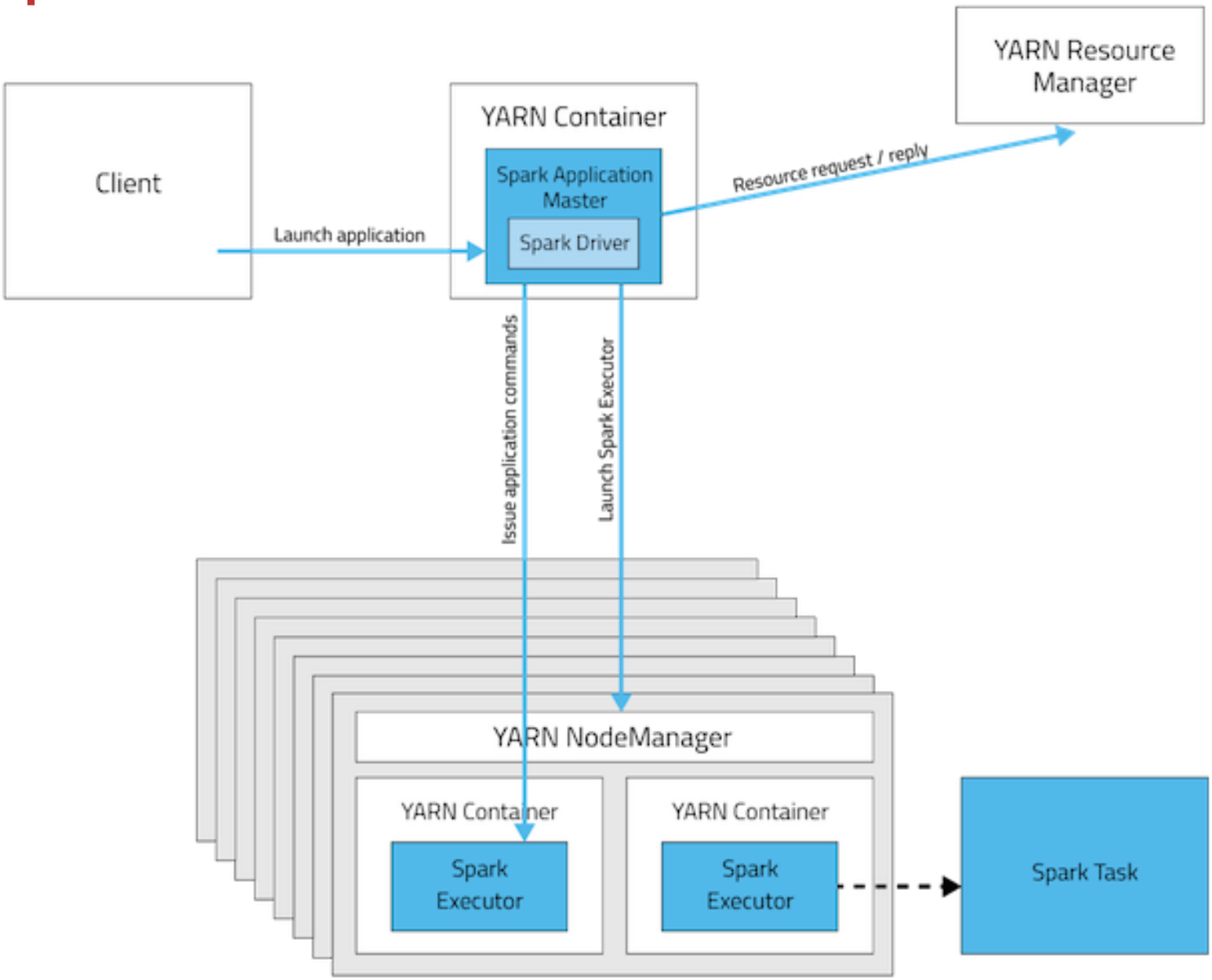
PROFINIT

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