DĚLAT
DOBRÝ SOFTWARE
NÁS BAVÍ

PROFINIT

Spark

Jan Hučín

5. dubna 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
   - Nadstavby
   - Architektura a alokace zdrojů
5. Spark streaming

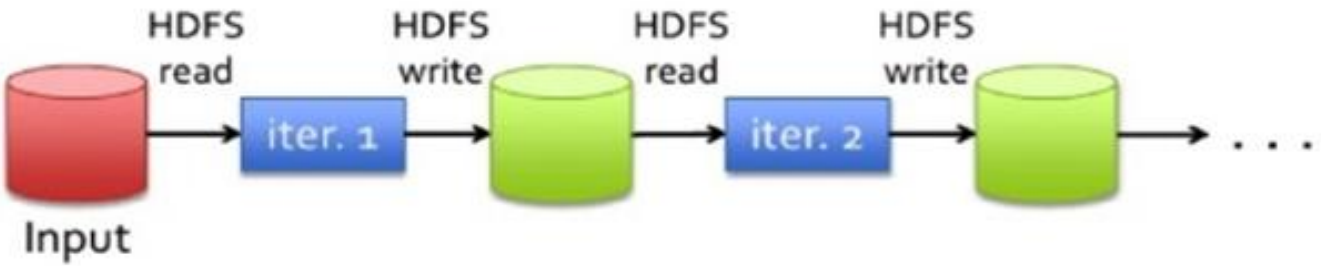
PROFINIT

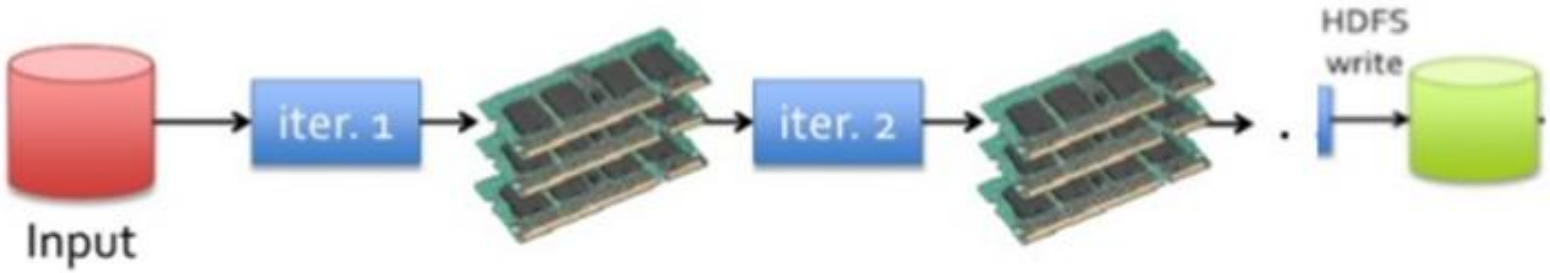
Úvod do Sparku

Co je to Spark a k čemu slouží

- systém pro distribuované výpočty na clusteru
- 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

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. hledání vazeb v síti
4. 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.

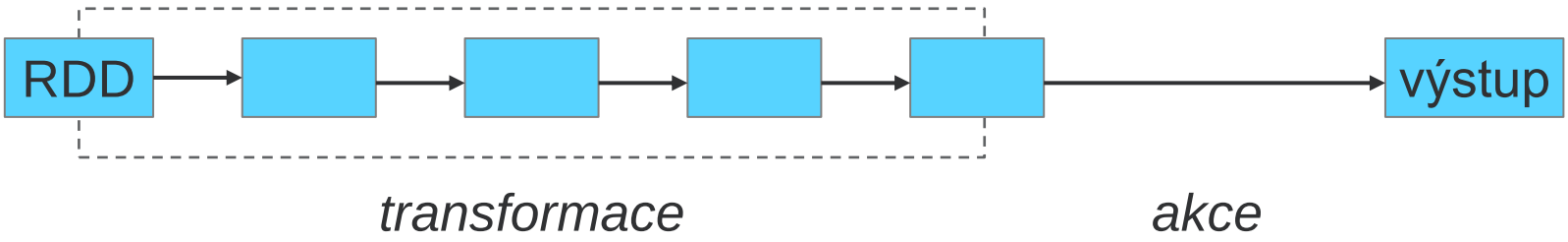
PROFINIT

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)



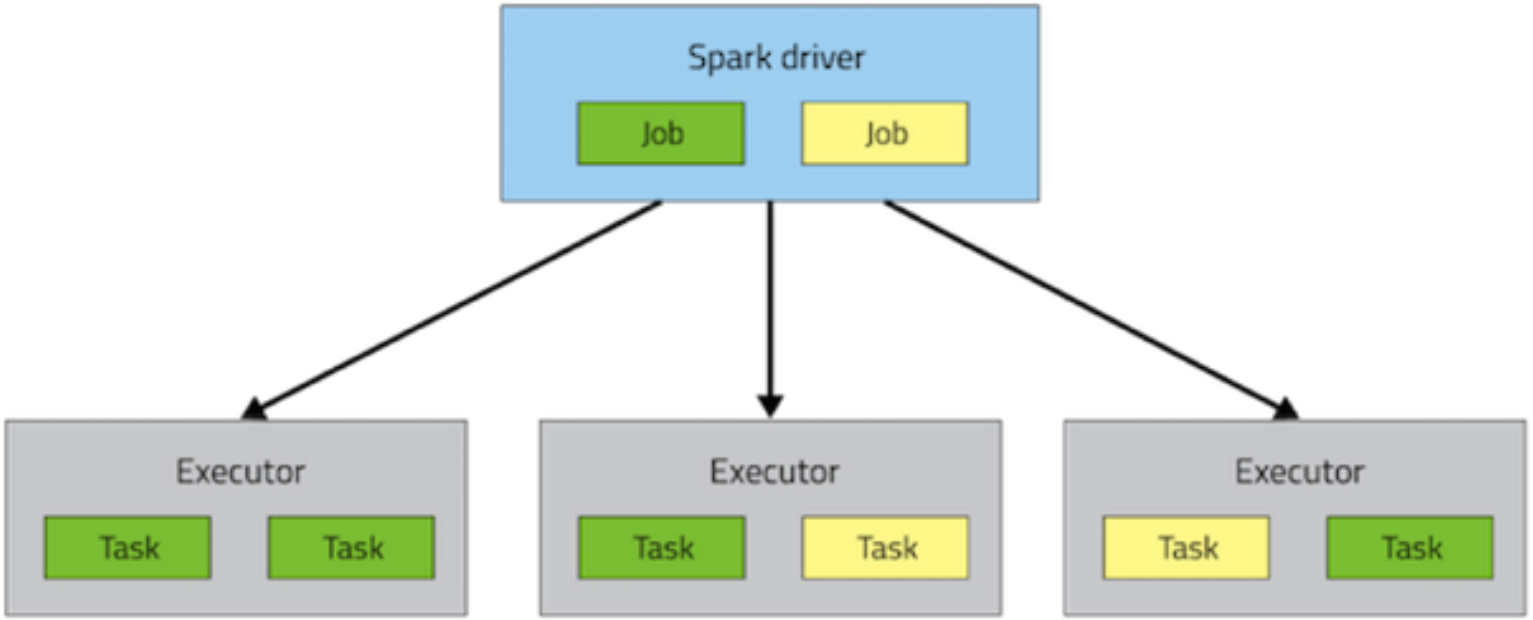
- série transformací zakončená akcí
- transformace se **plánují** a **optimalizují**, ale zatím **neprovádějí**
- **lazy evaluation:** až první akce spustí celý proces

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

Tradiční RDD přístup – 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 další

Kde vzít klíč?

"tuple"
(má Scala i Python)

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

PROFINIT

Příklad – transformace

- 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)

Vsuvka – interaktivní shell

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

- spouští se z příkazové řádky některého serveru bigdata*N*
- funguje lokálně nebo v YARNu:
  - `pyspark --master local`
  - `pyspark --master yarn`
- vytvoří důležité objekty, např. `sc` (SparkContext), `sqlContext`
- ukončuje se `exit()`

PROFINIT

Příklad – transformace

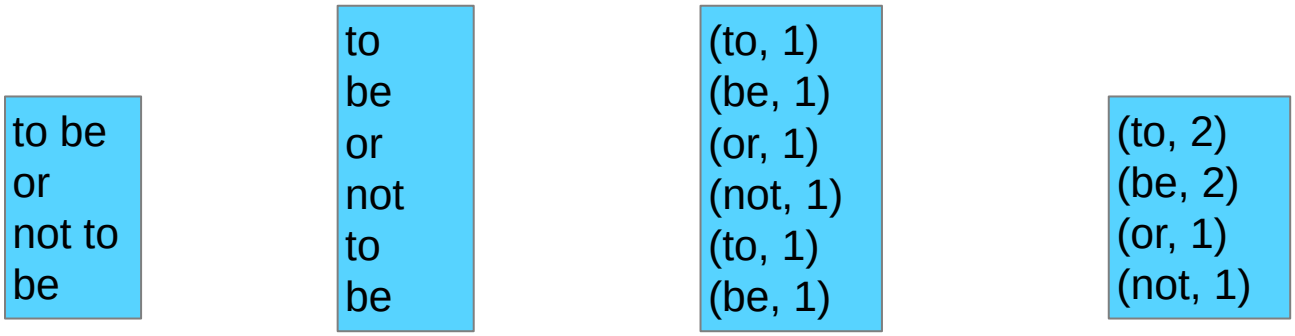
Úkol: spočítat četnosti slov v dokumentu

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.

Tradiční RDD přístup – 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)
- ... 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 SQL a DataFrames (DataSets)

- Alternativa k tradičnímu RDD přístupu
- Datová struktura DataFrame = RDD + sloupcové operace
  - obdoba databázové relační tabulky
  - obsahuje i schéma
  - operace nad rámec RDD – práce se sloupci
  - možnost použití syntaxe podobné SQL
- Výhody oproti RDD:
  - optimalizace dotazů
  - rychlejší běh
  - stručnější a jednodušší kód

Spark SQL a DataFrames – co je navíc (např.)

- **sql** (SQL dotazování, např. do Hive)
- **select** (omezení počtu sloupců)
- **filter** (omezení počtu řádků)
- **groupBy** (seskupení)
- **agg** (agregační funkce)
- **show** (hezčí výpis několika řádků)
- **read**, **write** (I/O operace)

Příklad – SparkSQL

- Úkol: který stát USA má na meteostanicích nejvyšší průměrný normál v létě?
- Vstup: tabulka Hive
- Postup:
  - načtení dat dotazem do Hive
  - omezit data na letní měsíce
  - ponechat jen důležité sloupce
  - seskupit po státech, spočítat průměry
  - seřadit podle průměrů sestupně
  - vypsat první řádek

Příklad – SparkSQL

- Úkol: který stát USA má na meteostanicích nejvyšší průměrný normál v létě?
- Vstup: tabulka Hive
- Postup:
  - načtení dat dotazem do Hive
    ```
    df=sqlContext.sql('select * from sstamenov.pocasi')
    ```
  - omezit data na letní měsíce
    ```
    df2=df.filter((df.mesic>5) & (df.mesic<9))
    ```
  - ponechat jen důležité sloupce (+ vyhodit null)
    ```
    df3=df2.select('stat','teplota').na.drop()
    ```
  - seskupit po státech, spočítat průměry (+ přejmenovat sloupec)
    ```
    df4=df3.groupBy('stat').avg().toDF('stat','prum')
    ```
  - seřadit podle průměrů sestupně
    ```
    df5=df4.sort(df4.prum.desc())
    ```
  - omezit na první řádek
    ```
    df5.limit(1)
    ```

PROFINIT

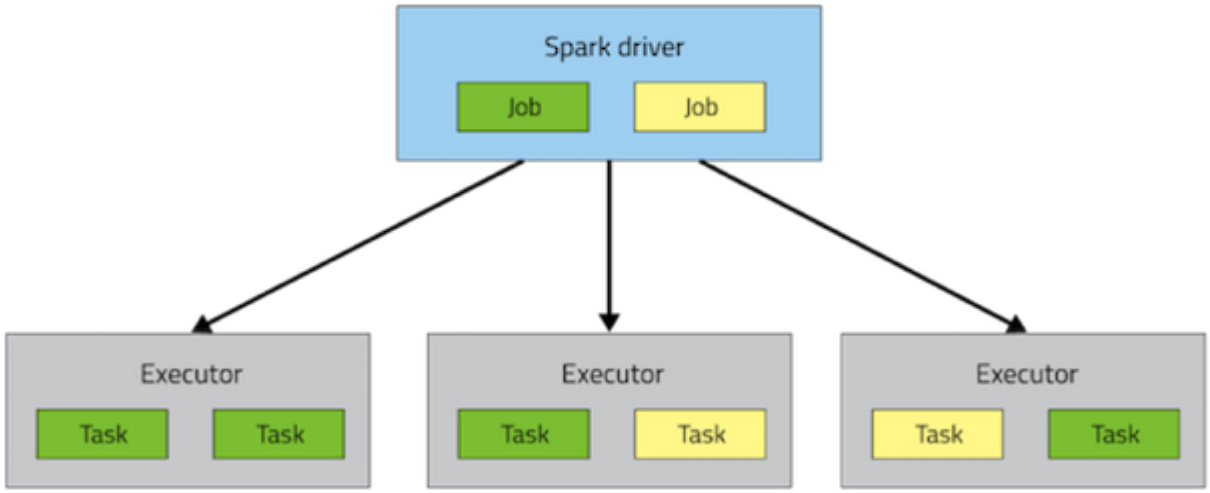
Nadstavby

- machine-learning (MLlib)
- zpracování grafů (GraphX)
- streaming

Architektura Sparku

Důležité pojmy 1

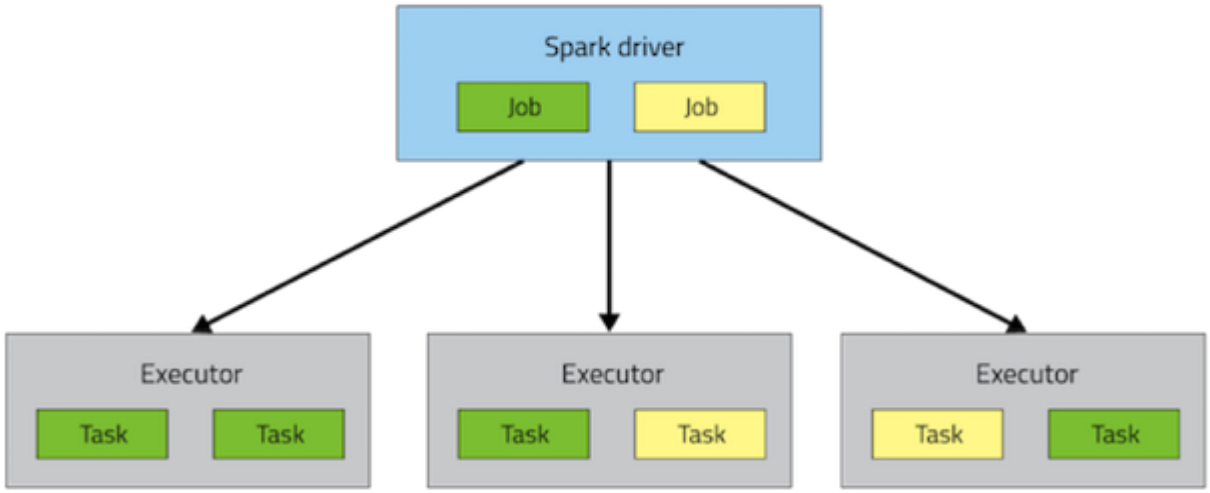
- **Application master**
  - proces zodpovědný za vyjednání výpočetních zdrojů od res. manageru
- **Driver**
  - hlavní proces
  - plánuje workflow
  - distribuuje práci do exekutorů
- **Executor**
  - proces běžící na některém z nodů (ideálně na každém)
  - provádí tasky (může i několik paralelně)



PROFINIT

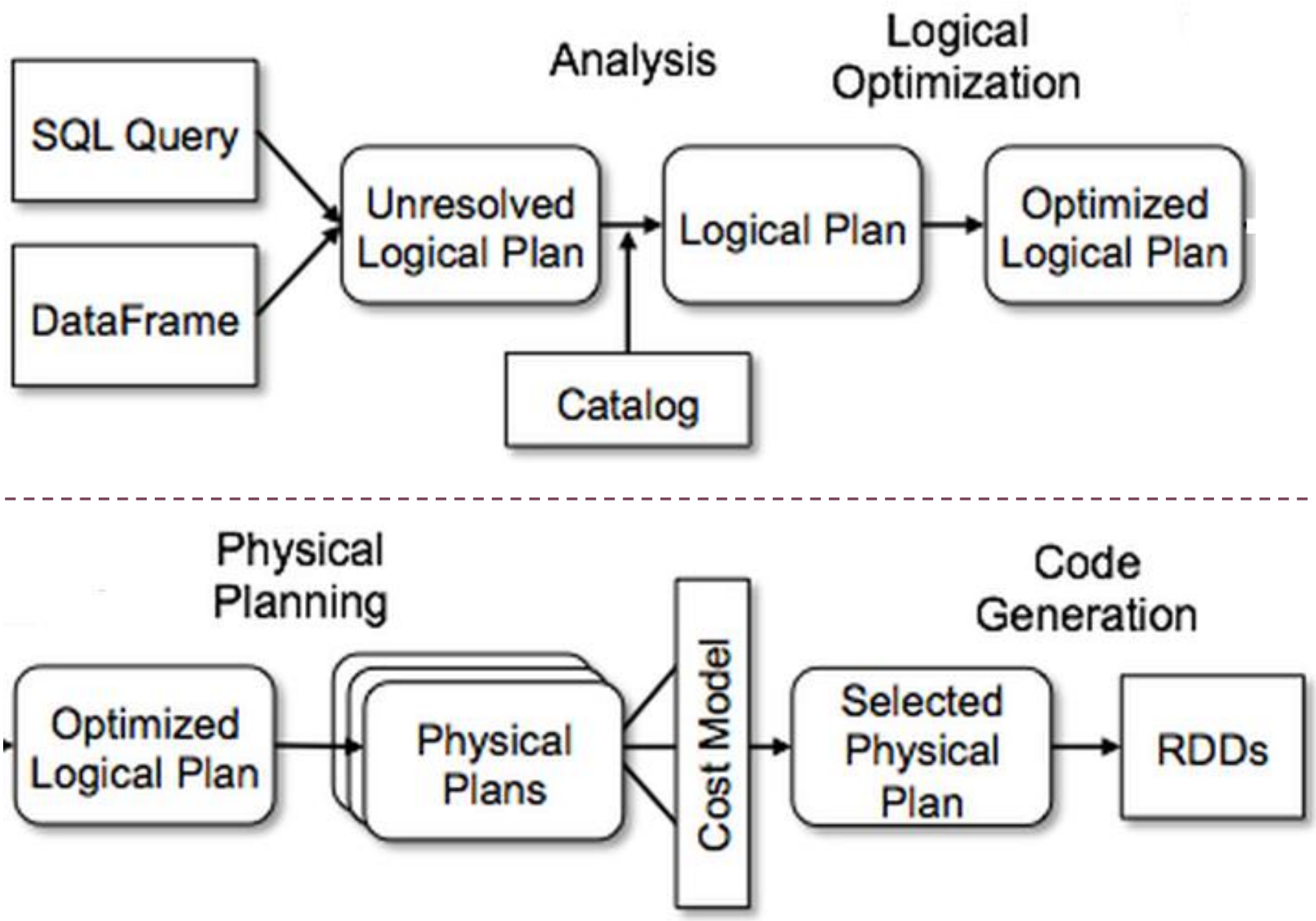
Důležité pojmy 2

- **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



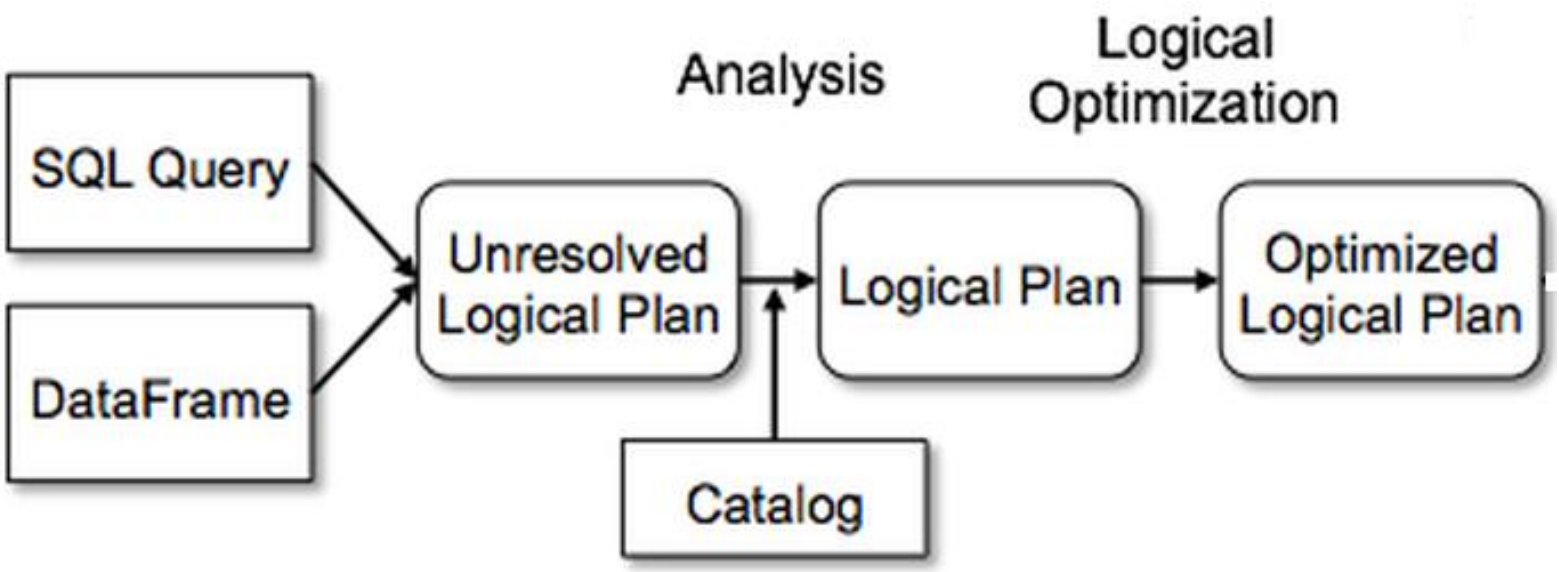
PROFINIT

Plánování a optimalizace



PROFINIT

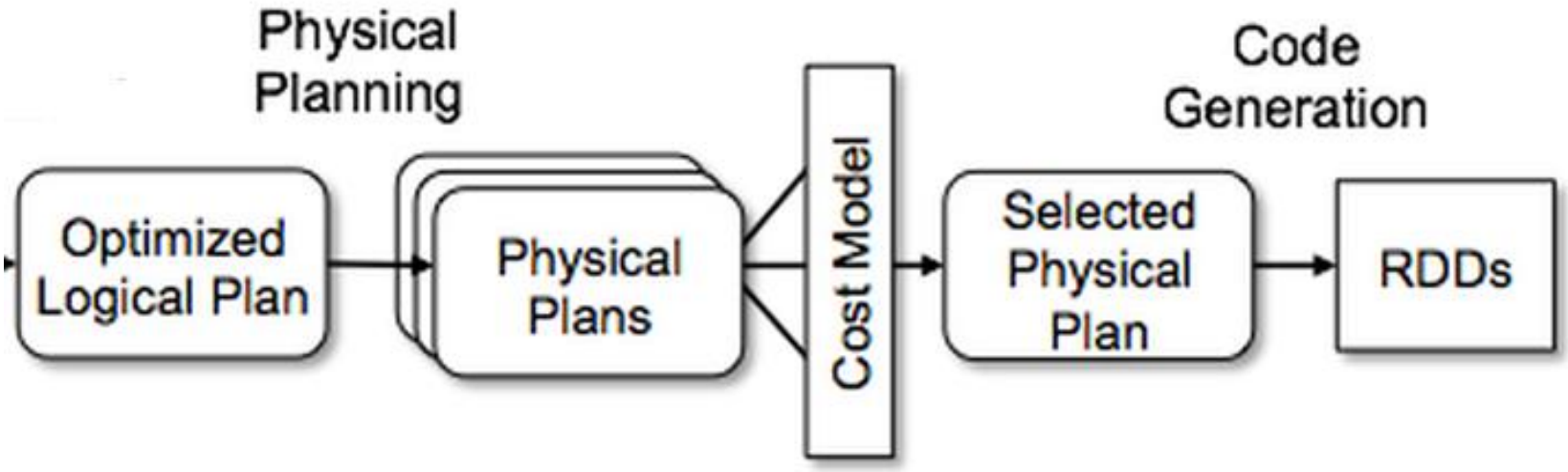
Plánování a optimalizace



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

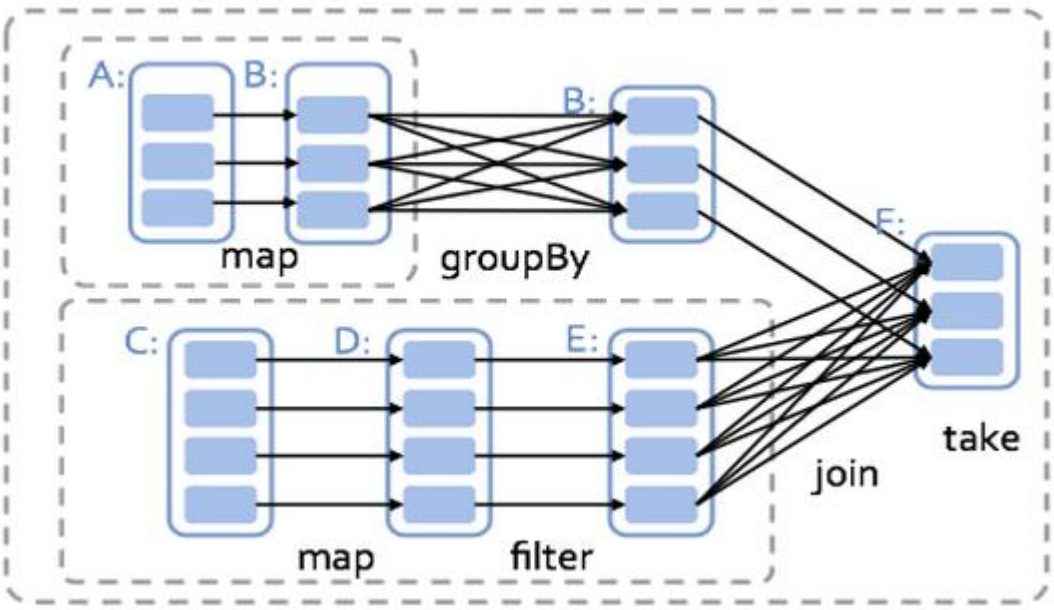
PROFINIT

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

Spuštění Sparku a konfigurace běhu

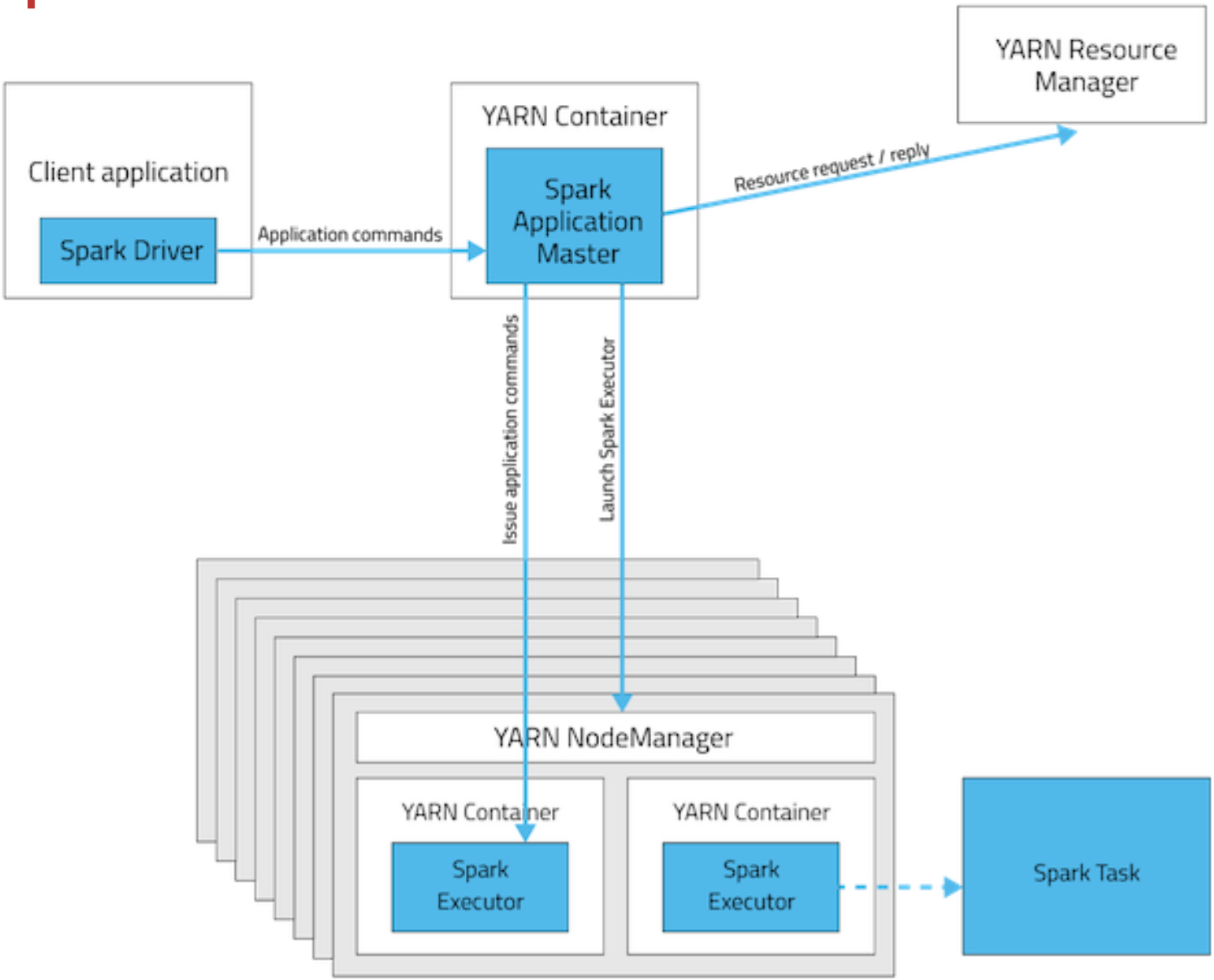
```
pyspark | spark-shell | spark-submit --param value
```

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

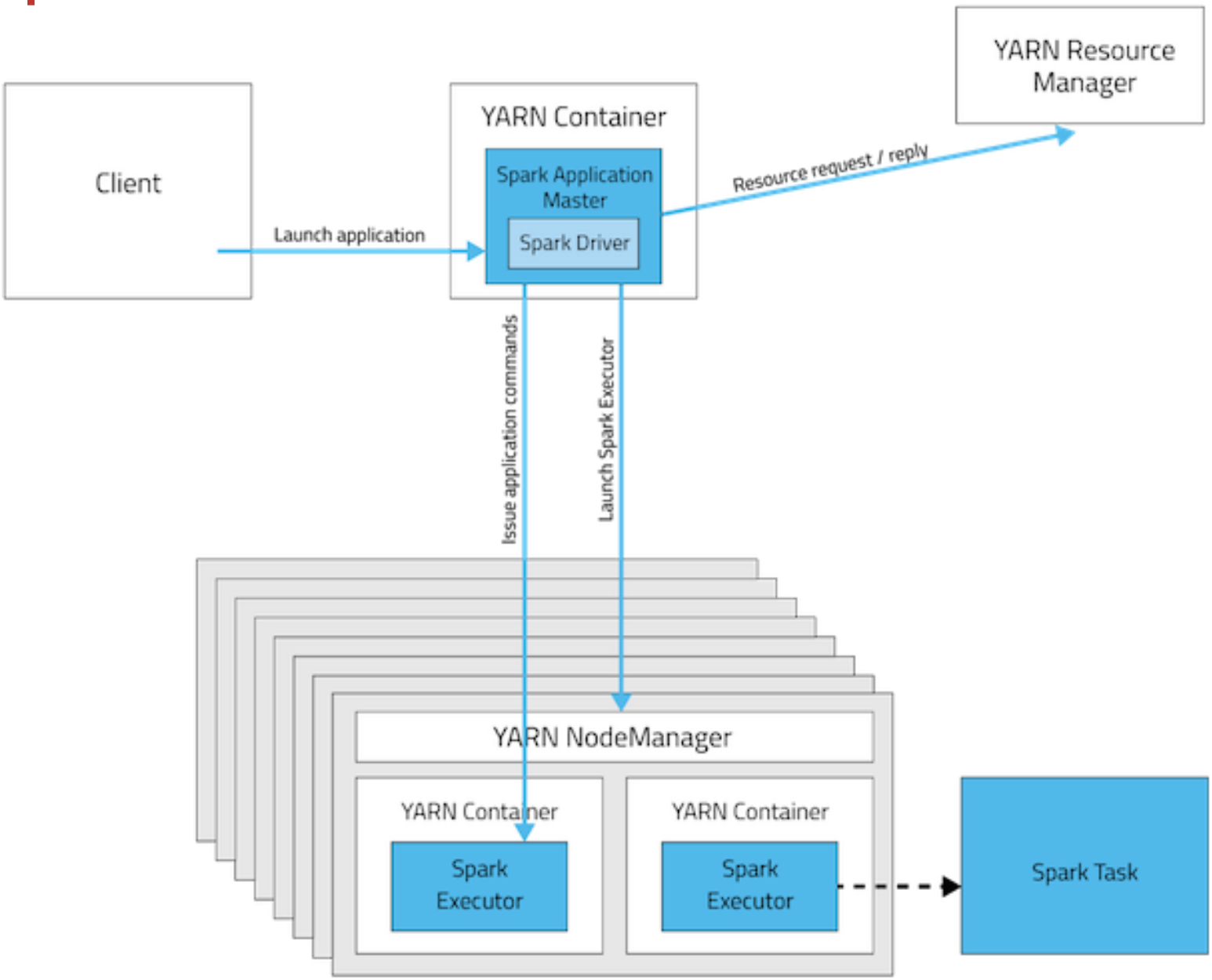
PROFINIT

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

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

Spark Streaming

Co to je a jak to využít

- dávkové zpracování přicházejících dat
- příchozí data neklepou na dveře, sedí v čekárně
- near real-time, pevné nastavení časového okna

Možné využití:

- filtrování logů, zpráv
- monitorování, reakce na událost
- vyhledávání v nakešovaných datech

PROFINIT

Princip zpracování

- streamovací modul dávkuje příchozí data – posloupnost RDD
- klasický Spark postupně odbavuje RDD ve frontě
- API pro Javu, Scalu, s malým omezením i Python

PROFINIT

Příklad

Úkol: pro každou dávku ze socketu spočítat četnosti slov

```
sc = SparkContext(appName="Priklad")
ssc = StreamingContext(sc, 10)

lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))
counts = lines.flatMap(lambda line: line.split(" ")) \
                  .map(lambda word: (word, 1)) \
                  .reduceByKey(lambda a, b: a+b)

counts.pprint()

ssc.start()
ssc.awaitTermination()
```

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