spark

et consortes

czyli wstęp do ekosystemu big data

Kornel Kiełczewski, k.kielczewski@mini.pw.edu.pl

HDP

- http://pastebin.com/YvhZz26X
- na vm:
  - vi /etc/hadoop/conf/mapred-site.xml
    - mapreduce.map.memory.mb: 2048
    - mapreduce.map.java.opts: -Xmx1024m
    - analogicznie dla .reduce.
  - vi /etc/hadoop/conf/yarn-site.xml
    - yarn.nodemanager.resource.memory-mb: 5000
- lokalnie:
  - /etc/hosts -> 127.0.0.1 sandbox.hortonworks.com
  - dla windows: Windows\System32\drivers\etc\hosts

Agenda

1. Duże dane
2. Java / Scala - wstęp
3. Spark
4. Hive
5. mllib

Duże dane

- Ile to jest dużo:
  - 500MB, 500GB, 500TB?
- Gdzie możemy zapisać *jakiekolwiek* dane:
  - rejestry CPU
  - CPU cache
  - RAM
  - Dysk
- Gdzie możemy zapisać *dużo danych*?

HDFS

- Rozproszony system plików działający na “commodity hardware”

- NameNode trzyma(ją) metadane (np.: gdzie znajdują się bloki danego pliku)

- DataNode trzymają bloki danych + replikacja

Formaty plików

- Text, Sequence File, ORC (Optimized Row Columnar), Parquet, Avro, …
- Odczyt czy zapis?
- Kolumny czy wiersze?
- Ewolucja schematu, brak schematu?

HDFS - map/reduce

- Skoro mamy dane na wielu węzłach, wyślijmy program do danych, a nie dane do programu
- rdzeń aplikacji map/reduce:
  - map
  - shuffle
  - reduce

map/reduce przykład

- przykład: word count

- przykład: ile słów średnio per dokument

Map Reduce

- HDFS in
- Map
- Shuffle
- Reduce
- HDFS out

Map Reduce - Problem

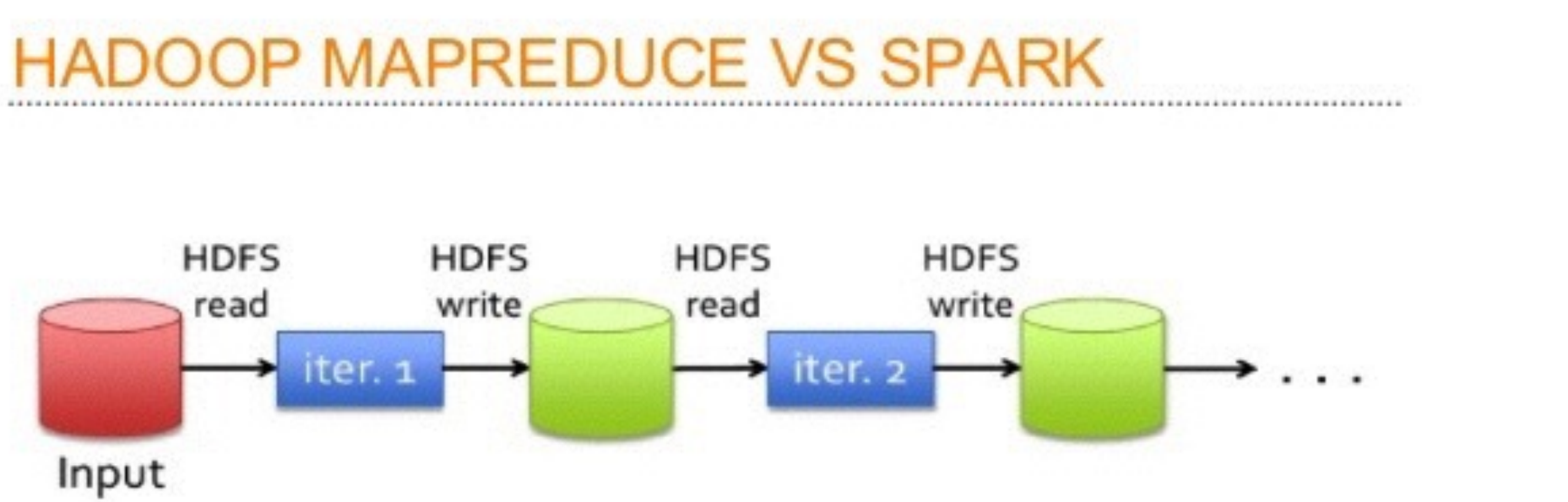
- I/O: dysk vs RAM

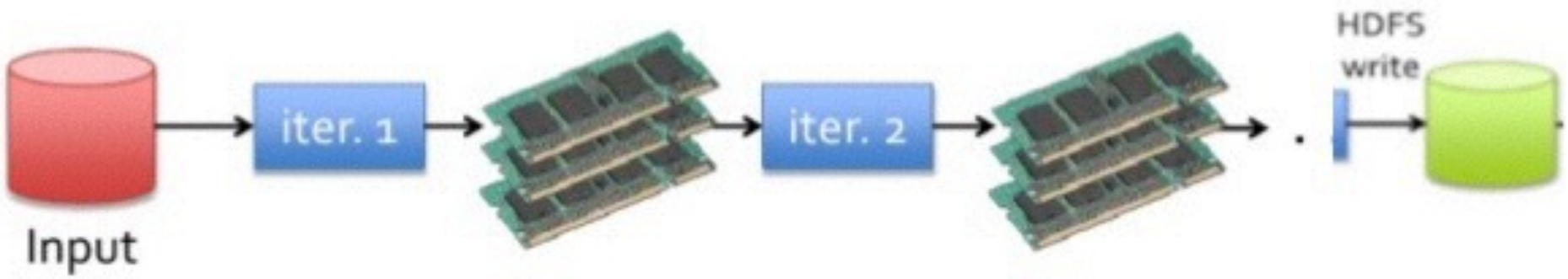
- Dane z Jupitera: http://blog.codinghorror.com/the-infinite-space-between-words/

Map Reduce - Rozwiązanie

- Trzymajmy dane w RAM’ie, zrzucając na dysk tylko wtedy, gdy to *konieczne*

Map-Reduce vs Spark





Spark - kiedy nie warto?

- Don't use Hadoop - your data isn't that big: https://www.chrisstucchio.com/blog/2013/hadoop_hatred.html
- 600 MB - pandas + numpy
- 10 GB - pandas
- 100GB - postgres?

Spark: Alternatywy

- Samza - Kafka
- Storm - “real time” streaming
  - Trident - micro-batch
- Flink - streaming
- Google Cloud DataFlow
- …?

Spark: zalety

- Powszechny (w tym dużo “dodatków”)
  - np.: Cassandra connector
- MLLib (+ sparkling water i h2o)
- Batch + Streaming
- YARN, Mesos, Standalone

YARN

- Yet Another Resource Negotiatior

- Zarządzanie zasobami w klastrze: czyli kontenery i kolejki oraz komu dać a komu zabrać zasoby

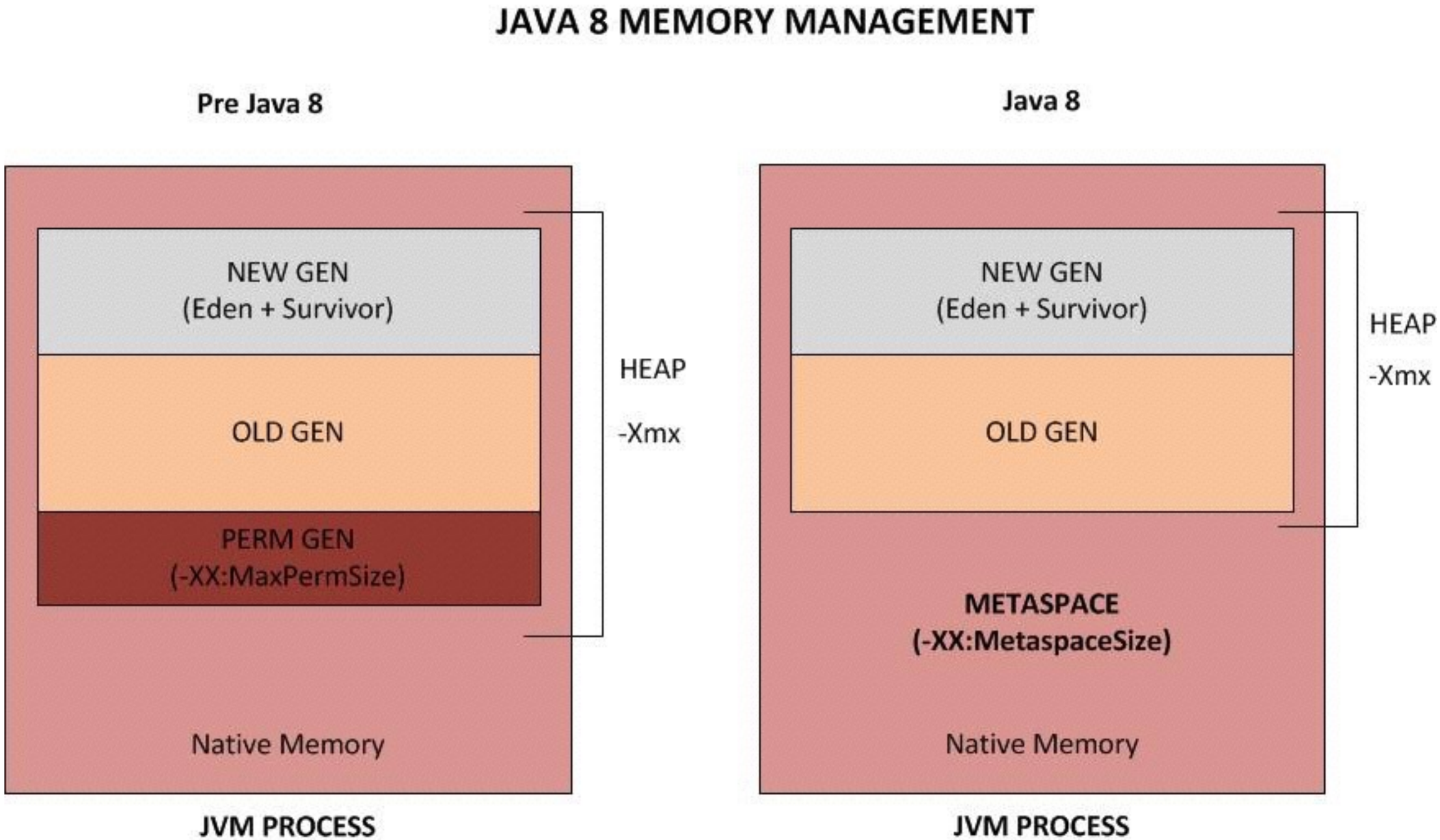
Spark: zalety

- DataFrame, DataSet
- Project Tungsten
- spark-shell
- Scala, Java, Python, R
- Platformy:
  - Cloudera
  - Hortonworks

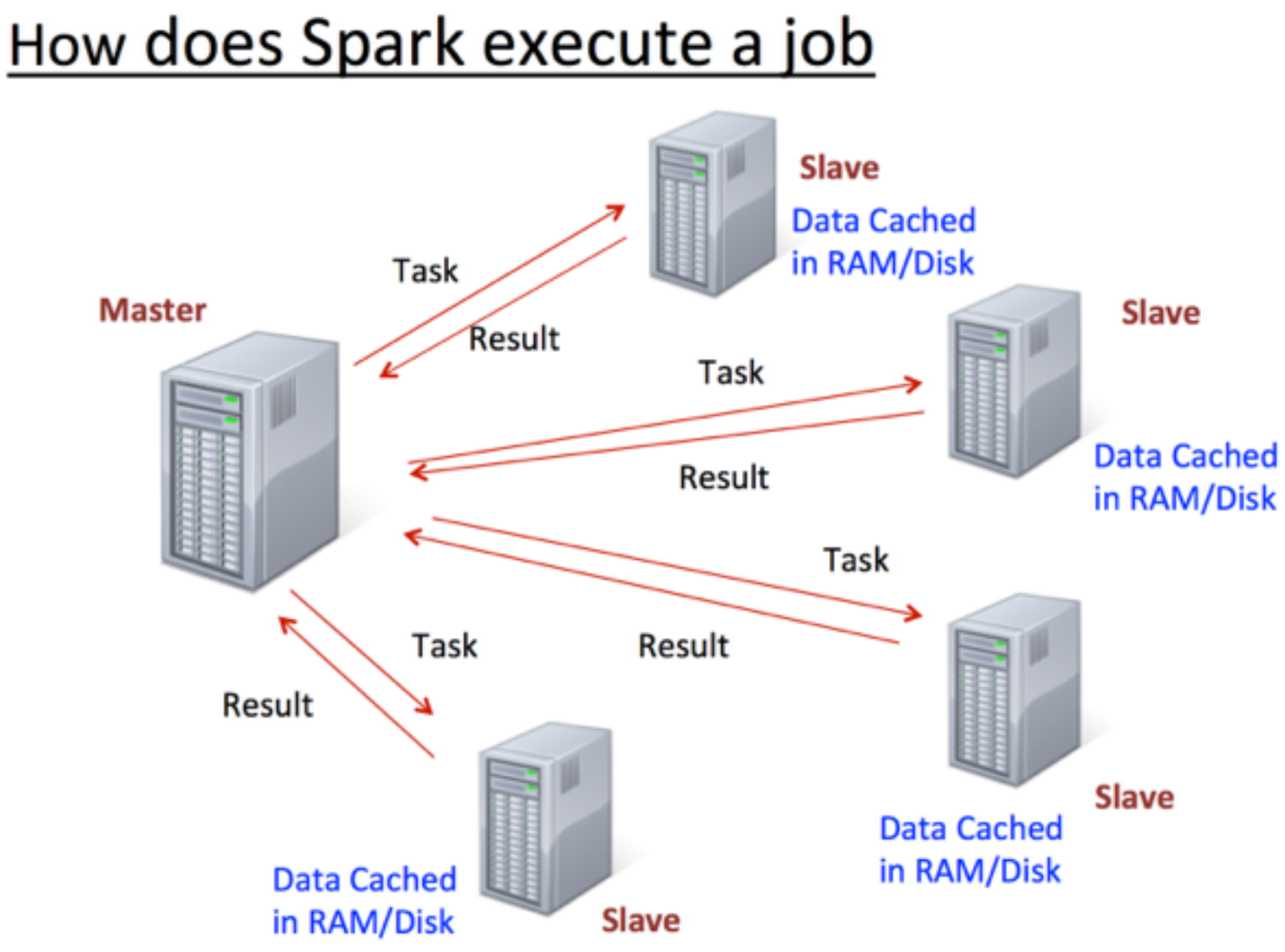
Spark: wady

- JVM ?
- micro-batch ?

Java: co warto wiedzieć



Model działania



Model działania

- Przykład dla YARN:
  - yarn client
  - yarn cluster
- a co z High Availability?

Spark RDD

- Resilient Distributed Dataset: http://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf

- RDD - interface
  - Zbiór partycji
  - Lista zależności do “rodziców”, gdzie rodzic to też RDD
  - Funkcja compute: oblicza dane na podstawie rodziców
  - Opcjonalnie: preferowane lokalizacja
  - Opcjonalnie: partycjonowanie dla RDD typu klucz -> wartość

Hierarchy Class RDD

Scope: All

- Object *(java.lang)*
  - RDD *(org.apache.spark.rdd)*
    - BaseRRDD *(org.apache.spark.api.r)*
    - BlockRDD *(org.apache.spark.rdd)*
    - CartesianRDD *(org.apache.spark.rdd)*
    - CassandraPartitionedRDD *(com.datastax.spark.connector.rdd.partitioner)*
    - CassandraRDD *(com.datastax.spark.connector.rdd)*
    - CheckpointRDD *(org.apache.spark.rdd)*
    - CoalescedRDD *(org.apache.spark.rdd)*
    - CoGroupedRDD *(org.apache.spark.rdd)*
    - EmptyRDD *(org.apache.spark.rdd)*
    - HadoopMapPartitionsWithSplitRDD in HadoopRDD$ *(org.apache.spark.rdd)*
    - HadoopRDD *(org.apache.spark.rdd)*
    - JdbcRDD *(org.apache.spark.rdd)*
    - JDBCRDD *(org.apache.spark.sql.execution.datasources.jdbc)*
    - MapPartitionsRDD *(org.apache.spark.rdd)*
    - NewHadoopMapPartitionsWithSplitRDD in NewHadoopRDD$ *(org.apache.spark.rdd)*
    - NewHadoopMapPartitionsWithSplitRDD in SqlNewHadoopRDD *(org.apache.spark.rdd)*
    - NewHadoopRDD *(org.apache.spark.rdd)*
    - PairwiseRDD *(org.apache.spark.api.python)*
    - ParallelCollectionRDD *(org.apache.spark.rdd)*
    - PartitionerAwareUnionRDD *(org.apache.spark.rdd)*
    - PartitionPruningRDD *(org.apache.spark.rdd)*
    - PartitionwiseSampledRDD *(org.apache.spark.rdd)*
    - PipedRDD *(org.apache.spark.rdd)*
    - PythonRDD *(org.apache.spark.api.python)*
    - SampledRDD *(org.apache.spark.rdd)*
    - ShuffledRDD *(org.apache.spark.rdd)*
    - ShuffledRowRDD *(org.apache.spark.sql.execution)*
    - SpannedByKeyRDD *(com.datastax.spark.connector.rdd)*
    - SpannedRDD *(com.datastax.spark.connector.rdd)*
    - SqlNewHadoopRDD *(org.apache.spark.rdd)*
    - SubtractedRDD *(org.apache.spark.rdd)*
    - UnionRDD *(org.apache.spark.rdd)*
    - ZippedPartitionsBaseRDD *(org.apache.spark.rdd)*
    - ZippedWithIndexRDD *(org.apache.spark.rdd)*

Spark RDD

- HadoopRDD:
    - Zbiór partycji: jedna per blok na HDFS
    - Zależności: brak
    - Funkcja: przeczytanie danego bloku
    - Preferowana lokazliacja: lokalizacja bloku na HDFS
    - Partycjonowanie: brak

- MapPartitionsRDD:
    - Zbiór partycji: takie jak rodzica
    - Zależności: relacja 1-1 z rodzicami
    - Funkcja: oblicz wartość rodzica i przekształć
    - Preferowana lokalizacja: zapytaj rodziców
    - Partycjonowanie: brak

spark-shell: ćwiczenie

- $ hdfs dfs -ls -h /demo/data/Customer/acct.txt
- $ hdfs dfs -text /demo/data/Customer/acct.txt
- $ spark-shell
  - val tf = sc.textFile(“/demo/data/Customer/acct.txt")
  - tf.dependencies.foreach(x => println(x.rdd))

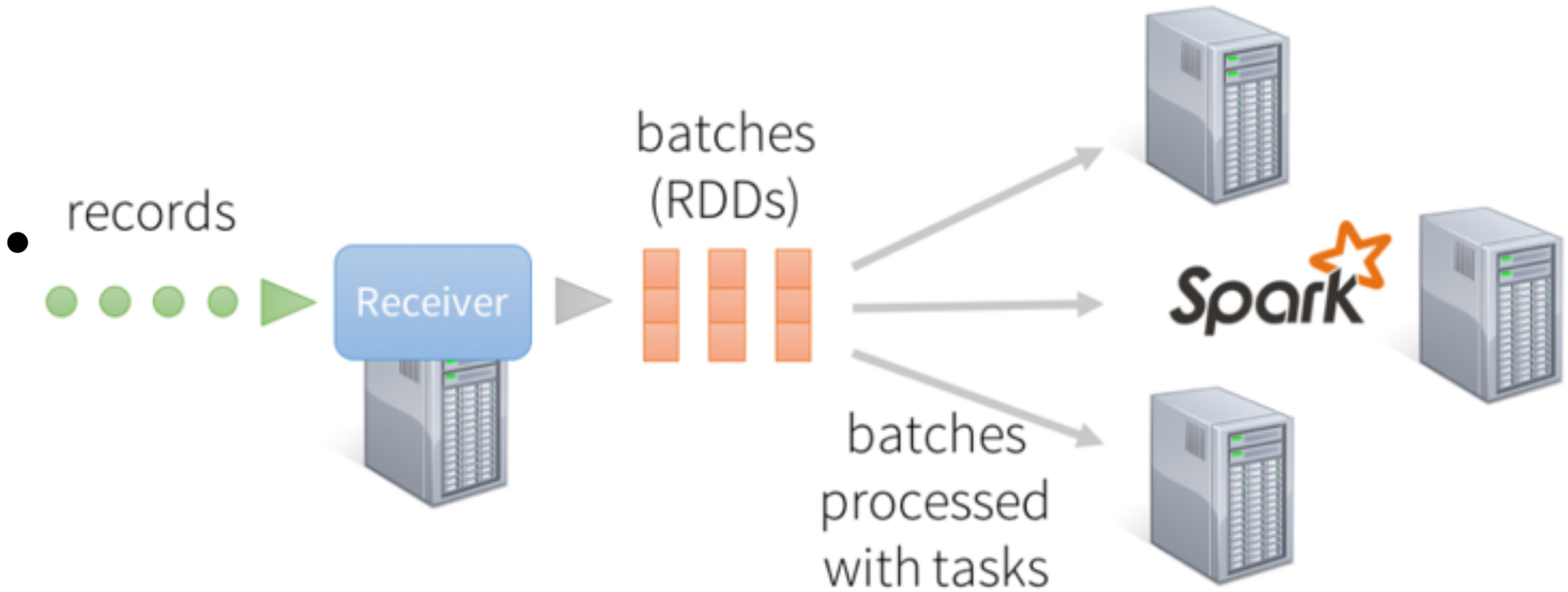
Stream vs Batch

- Bounded vs Unbounded: https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101

Spark DStream

Spark Streaming

discretized stream processing



records processed in batches with short tasks
each batch is a RDD (partitioned dataset)

Scala

- Crash course: https://github.com/Kornel/why-choose-scala

Spark: Transformations

- map(func)
- filter(func
- flatMap(func)
- sample(withReplacement, fraction, seed)
- union(otherDataset)
- intersection(otherDataset)
- distinct([numTasks]))
- groupByKey([numTasks])
- reduceByKey(func, [numTasks])
- …

Spark: Actions

- reduce(func)
- collect()
- count()
- first()
- take(n)
- saveAsTextFile(path)
- foreach(func)
- …

Spark word-count

```
np.: http://lipsum.com/feed/html

hdfs dfs –put [src] [dst]
```

```
sc.textFile(“some-file.txt”)
  .flatMap(_.split(“ “))
  .map(word => (word, 1))
  .reduceByKey(_ + _)
```

Spark word-count

- ćwiczenie I: word count + usunąć kropki, sprowadzić wszystko do małych liter, bez słów zaczynających się od a
  - kropki: str.replace(“.”, “”)
  - małe litery: str.toLowerCase
  - filtr: filter(_.startWith(‘a’))
- ćwiczenie II: wyświetlić top 5 słów
  - sortowanie: sortBy
  - top N: take

HDP

- Apache Ambari: http://127.0.0.1:8080/ maria_dev@maria_dev
- Spark-shell UI: http://127.0.0.1:4040/
- Hadoop: http://localhost:8088/cluster
- Zeppelin Notebook http://127.0.0.1:9995/
- cd /usr/hdp/current/spark-client
    - ./bin/spark-submit --class org.apache.spark.examples.SparkPi --master yarn-client --num-executors 1 --driver-memory 512m --executor-memory 512m --executor-cores 1 lib/spark-examples*.jar 10

Hive

- SQL na danych trzymanych w HDFS
- Metastore
- Silniki: M/R, Tez, Spark, …

Hive

- show schemas;
- show tables;
- desc extended call_detail_records;
- hdfs dfs -ls /apps/hive/warehouse/xademo.db/call_detail_records
- sqlContext.sql("select * from xademo.call_detail_records")

```
select * from xademo.call_detail_records;

create table records_clean(amount double, bytes double) stored as avro;

insert overwrite table records_clean
select q.amount, q.bytes from
        (select
            row_number() over (partition by 1) rn,
            c.*
        from
            xademo.call_detail_records c) q
where q.rn != 1;


select * from records_clean;
```

Hive vs Spark

- Z tabeli xademo.customer_details wyświetlimy sumę kolumny BALANCE per REGION:

  A. Za pomocą zapytania HQL

  B. Za pomocą Spark DataFrame

  C. Za pomocą Spark RDD (bez użycia sqlContext'u)

```
show create table xademo.customer_details;
```

- Z tabeli xademo.call_detail_records wyświetlić sumę kolumny BALANCE per REGION:

A. select region, sum(BALANCE) from xademo.customer_details group by region;

B. sqlContext.sql(t.j w pkt A) albo sqlContext
```
.sql("select * from xademo.customer_details")
.groupBy("region")
.agg(sum("balance"))
```

C.
```
sc.textFile("/apps/hive/warehouse/xademo.db/customer_details")
.mapPartitionsWithIndex((idx, par) => if (idx == 0) par.drop(1) else par)
.map(_.split("\\|"))
.map(x => x(6) -> x(4).toDouble)
.reduceByKey(_ + _)
.collect()
```

Spark Job jako projekt “standalone”

- https://github.com/Kornel/spark-mini-template

mllib

- http://spark.apache.org/docs/latest/mllib-linear-methods.html#regression

- https://archive.ics.uci.edu/ml/datasets/Housing

- Dziękuję za uwagę.