Funkcionalno programiranje

Apache Spark

Motivacija

- Primer sistema implementiranog u jeziku Scala
- Primena funkcionalnog programiranja u distribuiranom sistemu
  - Distribuirani sistemi => velike količine podataka
- Primeri se mogu pokretati
  - Na jednom računaru
  - Databricks (https://databricks.com/) daje besplatno pristup malom distribuiranom sistemu

Apache Spark

- MapReduce
  - Komercijalni alat za obradu velikih količina podataka u distribuiranom sistemu
  - Google napravio za potrebe pretraživanja svih sajtova
  - Koriste se obični računari
- Hadoop
  - Isto što i MapReduce, samo Open Source verzija
- Apache Spark
  - Ista namena kao i Hadoop
  - Napisan u jeziku Scala
  - API sličan kolekcijama iz jezika Scala (višejezičan Scala, Java, Python, …)
  - Funkcionalan API omogućava bolje performanse nego Hadoop

Distribuirana obrada podataka

- Na raspolaganju su obični računari
  - Naziv na engleskom Node
- Povezani su običnom mrežom
- Tipična obrada:
  - data.map(x => doSomething(x)).reduce(somehow)
- Ideja
  - Rasuti podatke po nodovima
  - Svaki node svoje podatke nezavisno obrađuje
  - Kombinovati rezultate po potrebi
- Ideja je slična kao i kod paralelnih kolekcija
  - Samo delove kolekcije obrađuju nezavisni računari, a ne posebne niti
  - Prilikom kombinovanja rezultata komunikacija se odvija preko mreže, a ne preko deljene memorije

Implementacija distribuirane obrade

- Distribuirana obrada je teška za implementaciju
  - Node može da se pokvari, da izgubi konekciju, …
  - Komunikacija preko mreže je spora u odnosu na brzinu procesora, memorije i diska
  - Poruke preko mreže mogu da se zagube, da se isporuče više puta, …
- Apache Spark (MapReduce, Hadoop,…) pokušava da olakša obradu podataka u distribuiranom sistemu
- Osnovna kolekcija za podatke je RDD
  - Resilient Distributed Dataset
  - Korisnik samo treba da zada operacije koje treba da se izvrše, Spark će odraditi (skoro) sve ostalo
  - API sličan kao kod Scala kolekcija

Primer na visokom nivou abstrakcije

- Na raspolaganju ocena proizvoda od strane korisnika
  - Dati projekat sadrži manji deo da bi obrada mogla da se izvrši na jednom računaru
- Kako pretvoriti sva slova u mala u tekstu koji predstavlja ocenu proizvoda na distribuiranom sistemu?
  - val reviews: RDD[SoftwareReview] = ...
    reviews.map {
      review => review. reviewText.toLowerCase
    }

RDD

- Nepromenljiva distribuirana kolekcija podataka
- RDD se kreira na dva načina:
  - Transformacijom postojećeg RDD
  - Pomoću objekta tipa SparkContext (SparkSession)
    - parallelize – metoda koja pretvara običnu Scala kolekciju u RDD
    - textFile – metoda koja učitava podatke iz fajla u RDD

```
abstract class RDD[T] {
  def map[U](f: T => U): RDD[U] = ...
  def flatMap[UJ(f: T => TraversableOnce[U]): RDD[U] = ...
  def filter(f: T => Boolean): RDD[T] = ...
  def reduce(f: (T, T) => T): T = ...
  …
}
```

RDD metode

- Dve vrste metoda
  - Transformacije
  - Akcije
- Transformacije
  - Vraća RDD kao rezultat
  - Samo se pamti koja je zahtevana obrada, ali se obrada ne pokreće
- Akcije
  - Računa rezultat na osnovu RDD-ja
  - Vrši sve prethodno zahtevane obrade

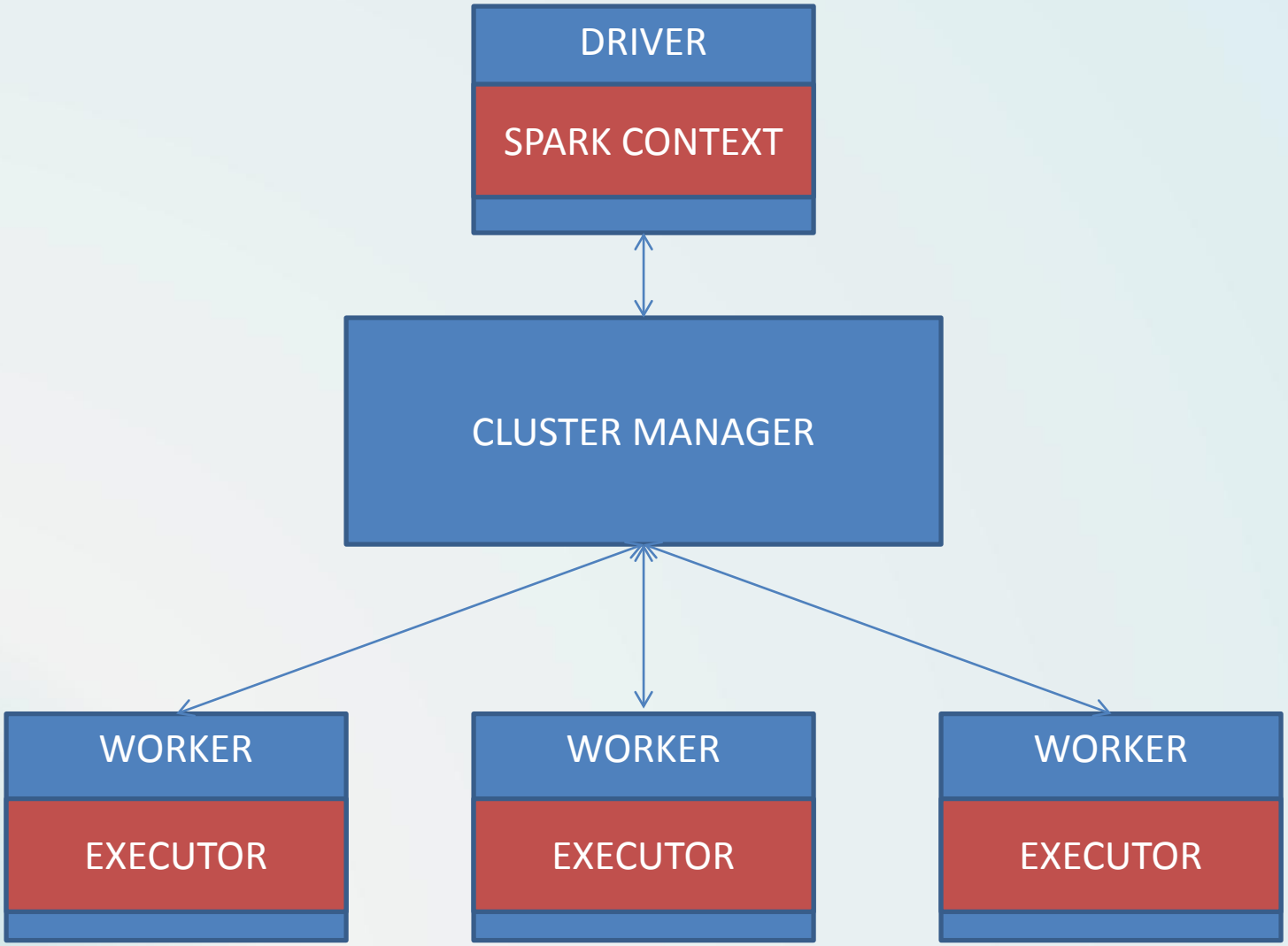
Primeri

- Ispis nekoliko ocena proizvoda
- Broj reči u ocenama proizvoda
- Koliko se koja reč ponavlja u ocenama proizvoda
- Najčešće reči u ocenama proizvoda

Lenja evaluacija

- Obrada podataka se obično radi u iteracijama
- Tolerancija otkaza se postiže samo pamćenjem transformacija u RDD-jevima
  - Nema potrebe za čuvanjem podataka
- Akcije računaju samo deo podataka u RDD-ijevima, koji su potrebni za rezultat

Kluster


DRIVER
SPARK CONTEXT
CLUSTER MANAGER
WORKER
EXECUTOR
WORKER
EXECUTOR
WORKER
EXECUTOR

Pair RDD

- Tretiraju se kao Key-Value Pair (distribuirana mapa)
- Imaju svoje transformacije
  - Paralelna obrada na osnovu ključa
  - Regrupisanje preko mreže
- Oblik je RDD[(K, V)]

Trasformacije i akcije za Pair RDD

- Transformacije
  - groupByKey
  - reduceByKey
  - mapValues
  - keys
  - join
  - leftOuterJoin/rightOuterJoin
- Akcije
  - countByKey
- Primeri
  - def groupByKey(): RDD[(K, Iterable[V])]
  - def countByKey(): Map[K, Long]

Join

- Operacija nad dva RDD-a
- Svaki je Pair RDD
- Uparivanje se radi po ključevima
  - Samo ako ključ postoji u oba RDD-ja

- def join[W](other: RDD[(K, W)]): RDD[(K, (V, W))]

- Primer:

RDD1	**RDD2**	**Rezultujući RDD**
(1, (10, “Belgrade”)) **(1, (100, “Nis”))** **(2, (20, “Belgrade”))** **(3, (30, “Nis”))**	**(1, “Pera”)** **(3, “Mika”)**	**(1, ((10, “Belgrade”), “Pera”))** **(1, ((100, “Nis”), “Pera”))** **(3, ((30, “Nis), “Mika”)**

Outer Joins

def leftOuterJoin[W](other: RDD[(K, W)]): RDD[(K, (V, Option[W]))]
def rightOuterJoin[W](other: RDD[(K, W)]): RDD[(K, (Option[V], W))]

- Zadrzavaju se i ključevi koji nemaju parnjaka u drugom RDD-iju
  - left – iz RDD-ija nad kojim se poziva metoda
  - right – iz RDD-ija koji je parametar metode
- Primer za left:

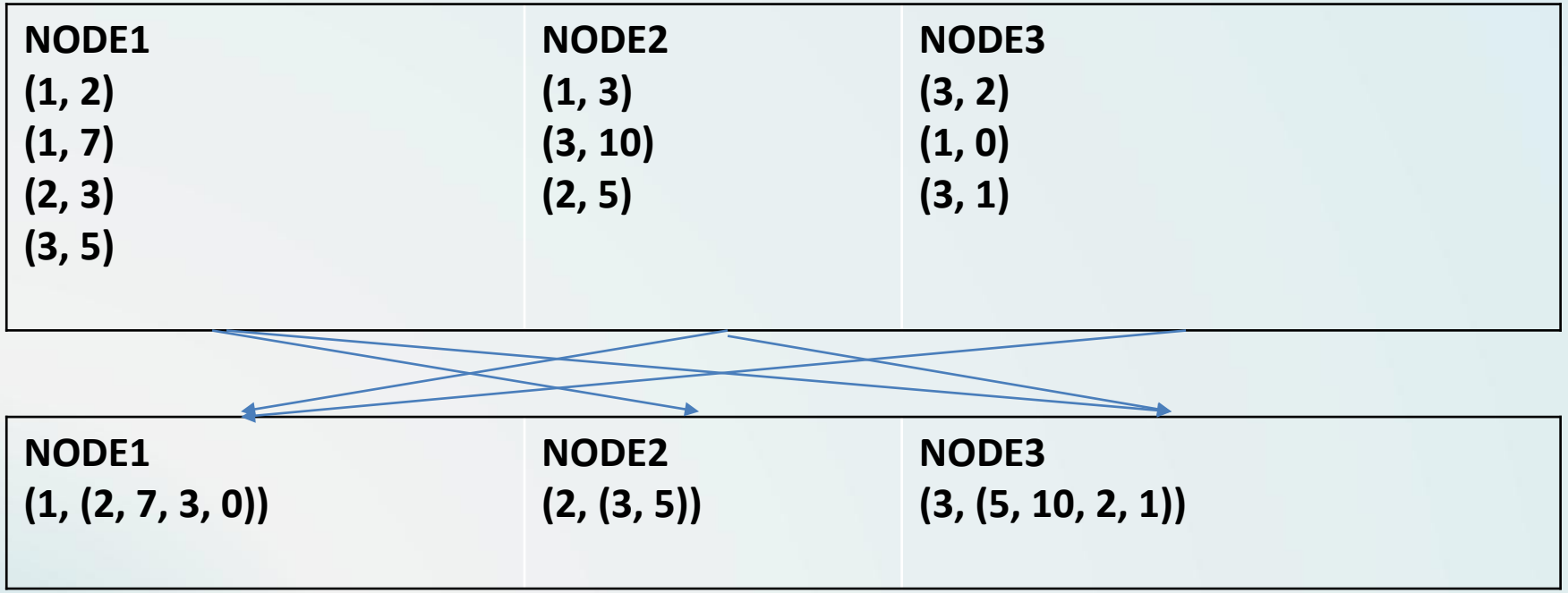
RDD1	**RDD2**	**Rezultujući RDD**
(1, (10, “Belgrade”))	**(1, “Pera”)**	**(1, ((10, “Belgrade”), Some(“Pera”)))**
(1, (100, “Nis”))	**(3, “Mika”)**	**(1, ((100, “Nis”), Some(“Pera”)))**
(2, (20, “Belgrade”))		**(2, ((20, “Belgrade”), None)**
(3, (30, “Nis”))		**(3, ((30, “Nis), Some(“Mika”))**

Primeri

- Prikazati ime proizvoda i ocenu
- Koliko je prosečno proizvoda dodatno kupljeno za proizvode po ocenama

Shuffles

- groupByKey i join operacije prouzrokuju razmenu podataka preko mreže (shuffles)

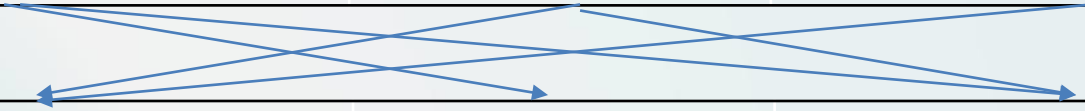


Smanjene podataka za prebacivanje

- rešiti problem sa reduceByKey

NODE1	NODE2	NODE3
(1, 2) (1, 7) (2, 3) (3, 5)	(1, 3) (3, 10) (2, 5)	(3, 2) (1, 0) (3, 1)

NODE1	NODE2	NODE3
(1, (2, 9)) (2, (1, 3)) (3, (1, 5))	(1, (1, 3)) (3, (1, 10)) (2, (1, 5))	(3, (2, 3)) (1, (1, 0))



NODE1	NODE2	NODE3
(1, (4, 12))	(2, (2, 8))	(3, (4, 18))

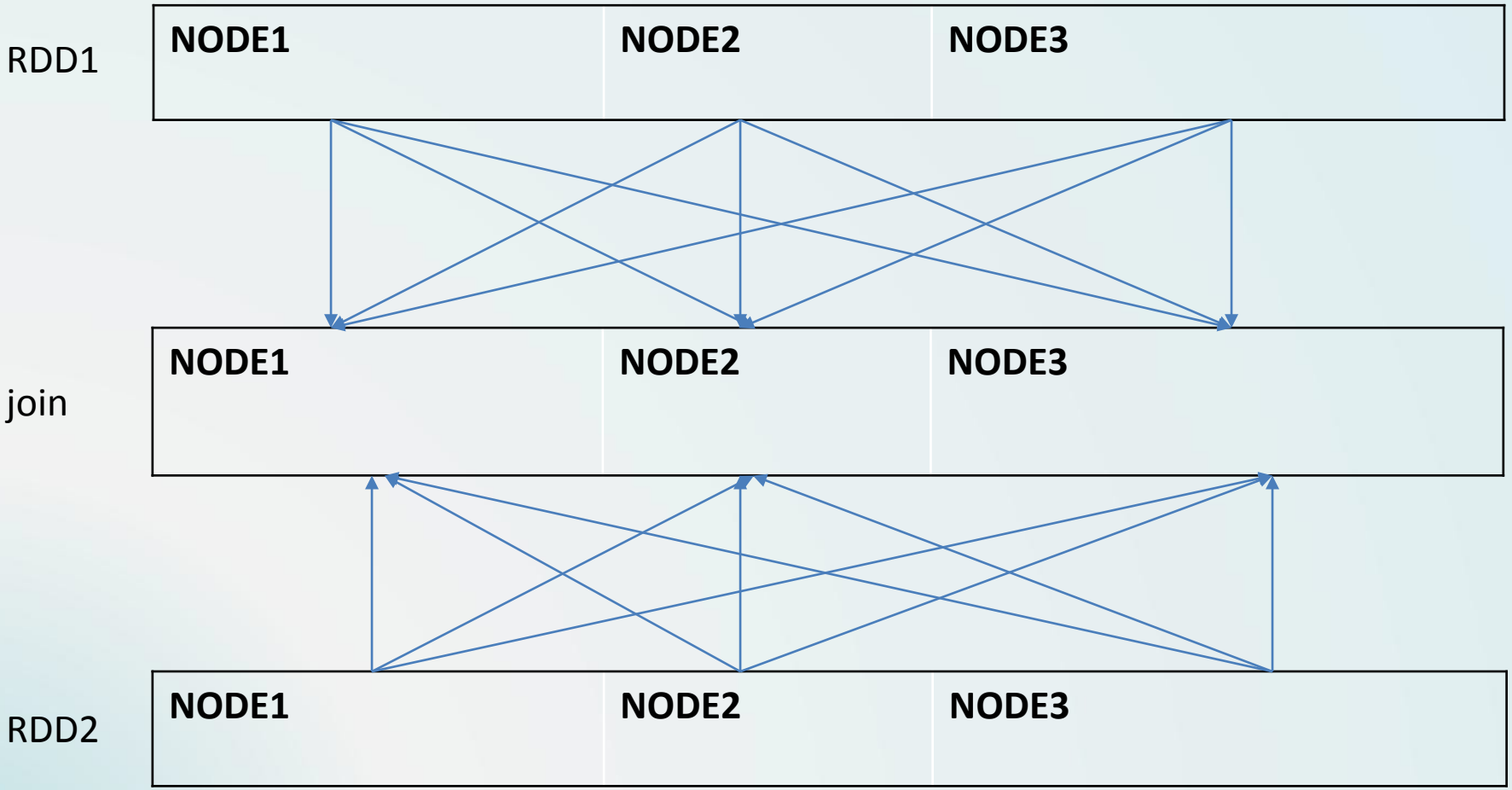
Particionisanje

- Podaci su raspodeljeni u particije
  - Jedna particija sadrži podatke na jednom računaru
  - Svaki računar sadrži jednu ili više particija
  - Broj particija je konfigurabilan – podrazumevano broj računara
- Dva načina za particionisanje
  - Heš
  - Opseg
- Primer:

```
val partitioner = new RangedPartitioner(5, allOverall)
val partitionedAllOverall =
        allOverall.partitionBy(partitioner).persist()
```

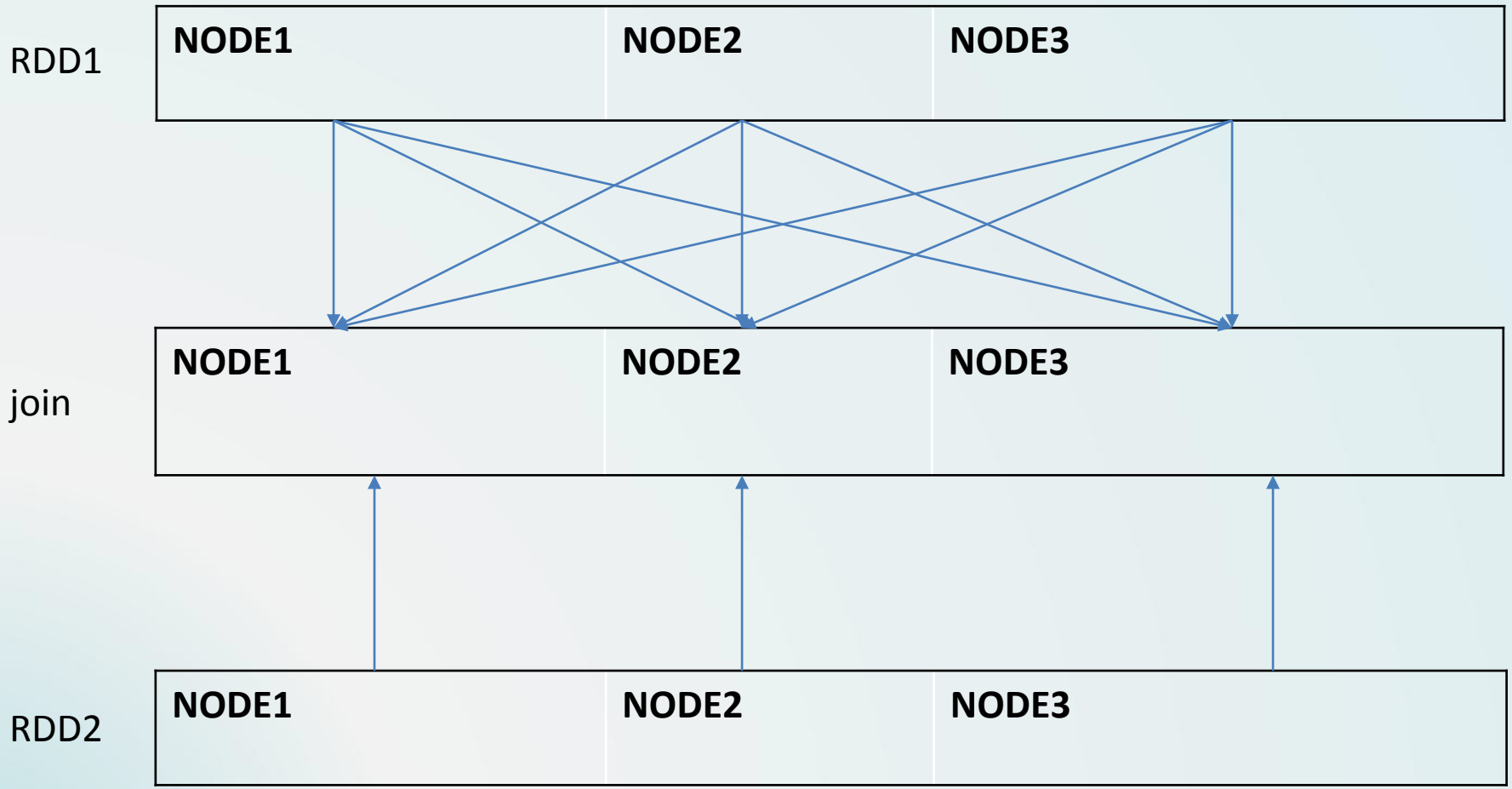
- Particionisanje vrši premeštanje podataka
  - Keširanje poželjno
- Neke operacije zadržavaju particionisanje
  - Okvirno one koje ne menjaju ključ

Join i shuffles



Join i shuffles

- Particionisati veći RDD (recimo RDD2)



Kada se shuffle dešava?

- Okvirno kada se kombinuju različiti elementi RDD-ija/ijeva
- Povratni tip nekih transformacija je ShuffledRDD
- Metoda toDebugString prikazuje plan izvršavanja

Spark SQL

- Komponenta u okviru Spark-a
- Omogućava relaciono procesiranje
  - Ima optimajzer i off-heap serijalizaciju
- Podržava korišćenje podataka iz nekih izvora
  - CSV, JSON, …
  - Baze podataka
- Tri interfejsa
  - SQL sintaksa u stringovima
  - DataFrames
  - Datasets

DataFrame

- DataFrame je isto što i tabela u relacionim bazama podataka
- RDD čiji su elementi rekordi sa poznatom šemom
- Nisu tipizirani
- Kako se dobijaju?
  - Iz RDD-ija kad se obezbedi šema (case klasa je šema sama po sebi)
  - Iz struktuiriranog izvora (JSON)
- Moguće SQL operacije
  - HiveQL
- SQL tipovi se preslikavaju u Spark tipove (pogledati dokumentaciju za detalje)

Operacije

- Neke transformacije

```
def select(col: String, cols: String*): DataFrame
def agg(expr: Column, exprs: Column*): DataFrame
def groupBy(col: String, cols: String*): DataFrame
def join(right: DataFrame): DataFrame
```

Primer

- Učitati JSON fajl u jedan DataFrame
- Prikazati podatke
- Prikazati šemu
- Selektovati samo one proizvode koji imaju nešto u koloni also_buy
- Prebrojati dodatno kupljene proizvode za svaki proizvod

Ogroman API?

- Opsežna dokumentacija:
  - https://spark.apache.org/docs/latest/api/scala/org/apache/spark/sql/Dataset.html