Full Orbit

# Introductie Spark

Your Professionals 20 september 2018

# Agenda

1. Wat is Spark?
2. RDD's
3. Acties
4. Transformaties
5. Groeperen en sorteren
6. Joins
7. Caching en persistency





# 1. Wat is Spark?

"Apache Spark™ is a unified analytics engine for large-scale data processing."

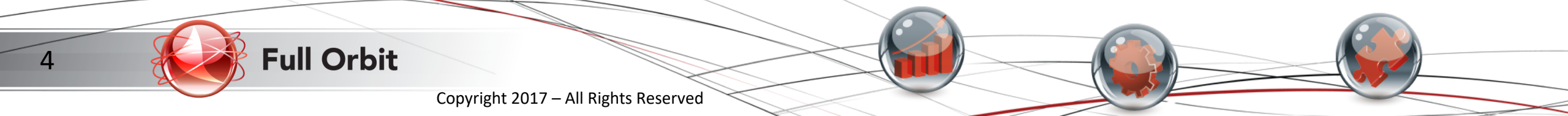
- Open source
- In-memory
- Distributed/clustered
- Parallel execution
- Scalable



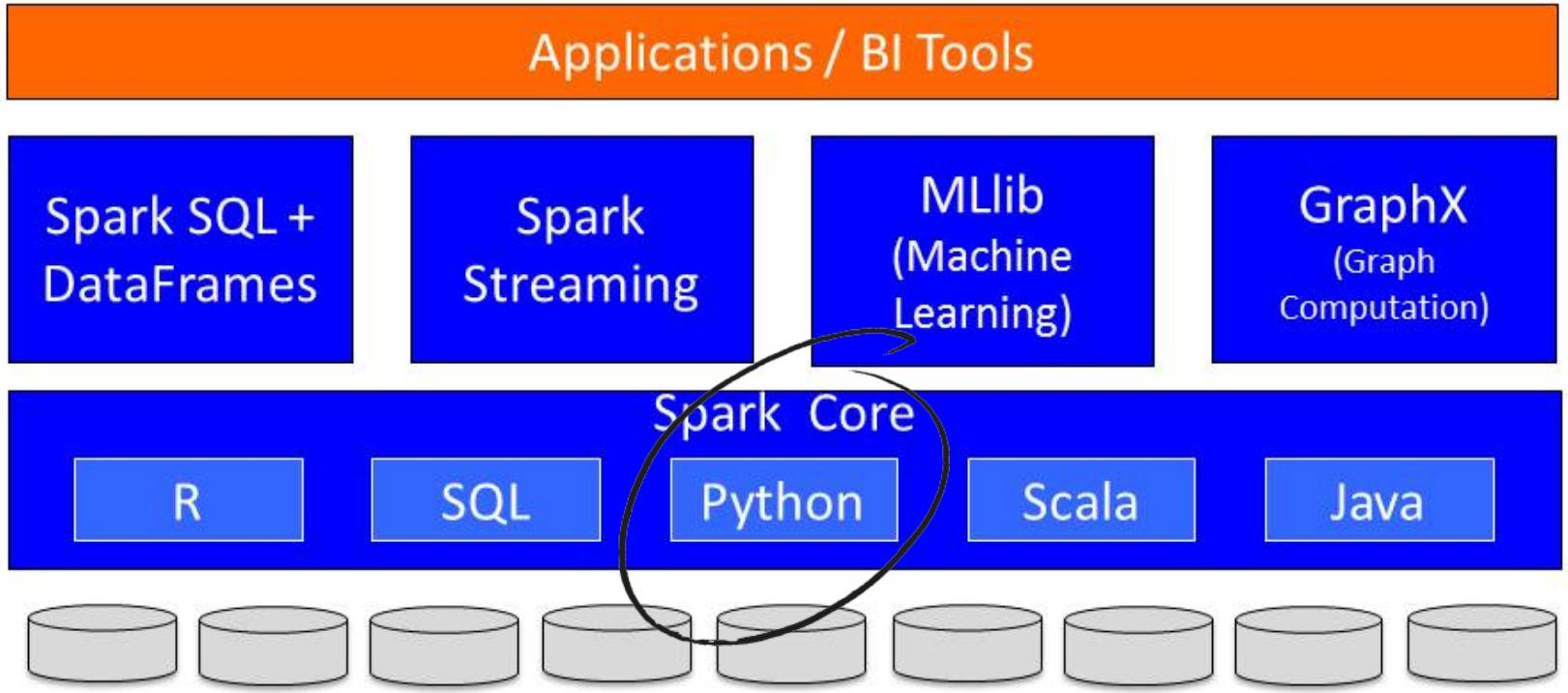


# Kenmerken van Spark?

- Processing in geheugen
- Gedistribueerd, dus meer rekenkracht en geheugen beschikbaar dan een enkele computer
- Spark is geschreven in Scala
- API's beschikbaar voor Java, Python, R en SQL
- Libraries beschikbaar voor ML, SQL en Streaming en GraphX
- Maakt gebruik van Hadoop client libraries voor HDFS en YARN

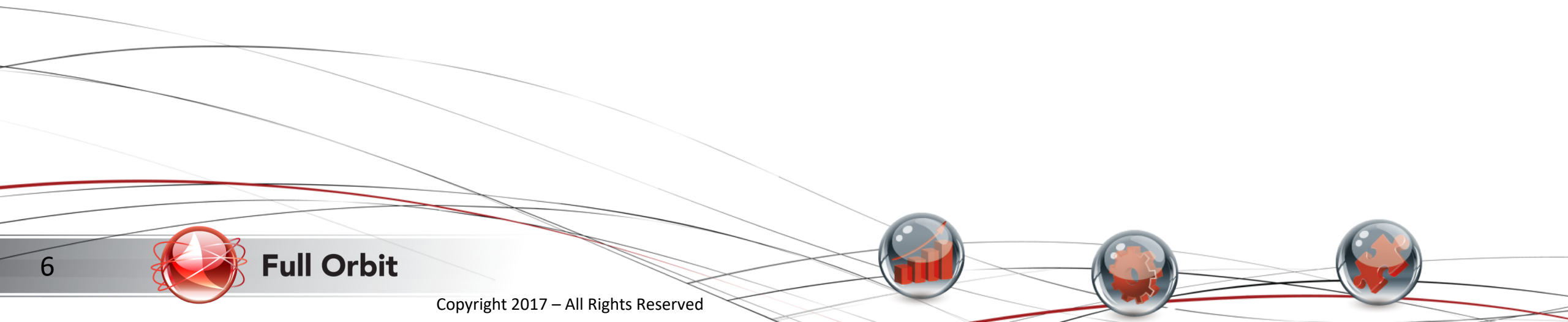


# Componenten van Spark



# Spark vs. Hadoop

- Hadoop is ecosysteem rond HDFS (filesystem) en MapReduce
- Spark draait op Hadoop
- Spark is sneller dan MapReduce

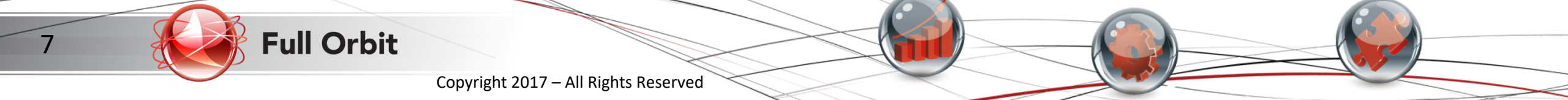


# Wanneer gebruik je Spark?

- Als datasets te groot worden voor analyses met Python en Java
- Data preparatie voor analyse m.b.v. R
- Data preparatie voor streaming applicaties
- Data preparatie voor Machine Learning

Maar:

- Spark complexer dan Python en Java



# Extra informatie

- Databricks biedt (cloud) platform gebaseerd op Spark
  - www.databricks.com
  - Databricks Unified Analytics Platform
- Tools using spark
  - HortonWorks Stream Analytics
  - Oracle Stream Analytics
  - Oracle Data Integrator
- Resources
  - https://spark.apache.org/docs/latest/rdd-programming-guide.html
  - https://spark.apache.org/docs/latest/api/python/index.html
  - https://sparkhub.databricks.com/resources/

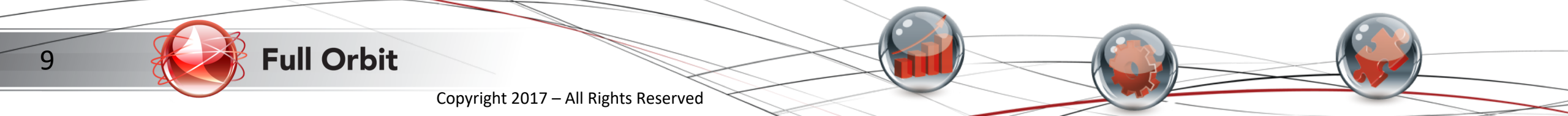




# 2. RDD's

## Een RDD is een Resilient Distributed Dataset

- Resilient
  - Woordenboek: veerkrachtig, elastisch, verkrachtig
  - Niet aanpasbaar (immutable)
- Distributed
  - Verdeeld over nodes (partioned)
  - Parallelle uitvoering
- Dataset
  - Verzameling gegevens



# Hoe maak ik een RDD?

- In een ‘spark context’
  - Een spark context is startpunt van je spark-programma
  - Is de connectie naar je spark-cluster
- Op basis van een tekst-bestand
  - textFile(*name*, *minPartitions=None*, *use_unicode=True*)
- Op basis van een collectie
  - parallelize(*collection*, *numSlices=None*)





# Wat kun je met RDD's?

- Een RDD kun je niet wijzigen!
- Maar je heb wel acties
  - Count()
  - saveAsTextFile()
- Je kunt nieuwe RDD's maken met transformaties
  - Map
  - GroupByKey
- Analyses bestaan uit verschillende transformaties afgesloten met een actie





# Spark Lazy Execution

- Alleen acties triggeren executie
- Transformaties zijn alleen beschrijvingen
- Transformaties worden opgeslagen in DAG (Directed Acryllic Graph)
- Betere optimalisatie
- Betere performance





# 3. Acties

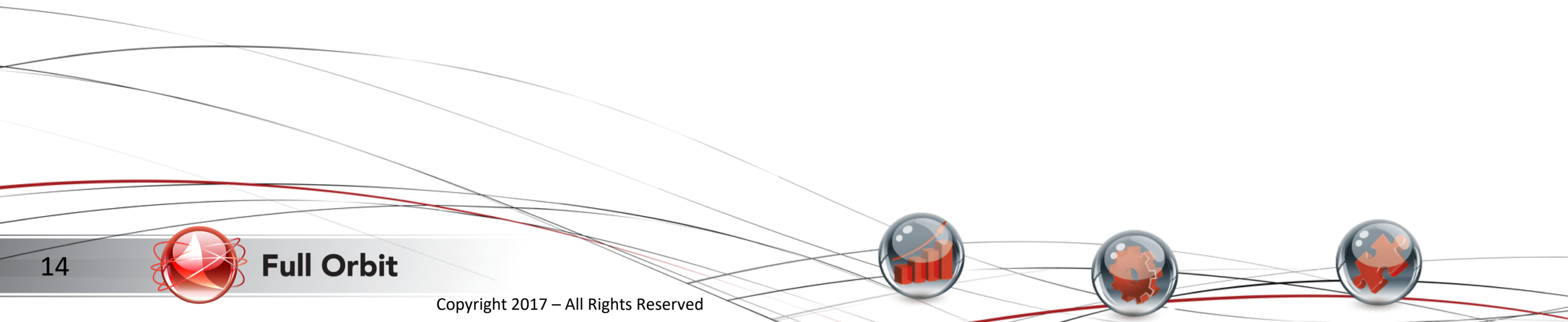
- Acties triggeren uitvoering!
- collect(): retourneert alle elementen van de RDD
- take(n): retourneert de eerste n elementen
- First(): retourneert het eerste element
- count(): telt het aantal elementen
- countByValue(): telt het aantal unieke elementen
- saveAsTextFile(filename, compressie methode)





# 4. Transformaties

- Een transformatie is een beschrijving van operatie op een RDD
- Het resultaat van een transformatie is een nieuwe RDD
- Een nieuwe RDD kan uit meerdere transformaties worden opgebouwd
- Transformaties worden pas uitgevoerd bij een actie!



# Voorbeelden van transformaties

- Map()
  - Operatie op elk element in een RDD
- flatMap()
  - Is combinatie van Map + Flatten operatie
  - Resultaat is een RDD met ander aantal elementen
  - Voorbeeld: tellen van aantal woorden
- Filter()
  - Transformatie om deel van de dataset te selecteren
  - Resultaat is (meestal) een subset van de RDD





# En meer…

- distinct()
- union()
- intersection()
- substract()
- cartesian()





# Lambda function

- Lambda functies zijn kleine ‘anonymous’ functies
- Lambda functies kunnen meerdere argumenten hebben
- Lambda functies hebben slechts één expressie
- Voorbeeld:
  - lambda x: x+ 4
  - lambda x,y: x+y
  - Lambda x:split(“,”)





# 5. Groeperen en sorteren

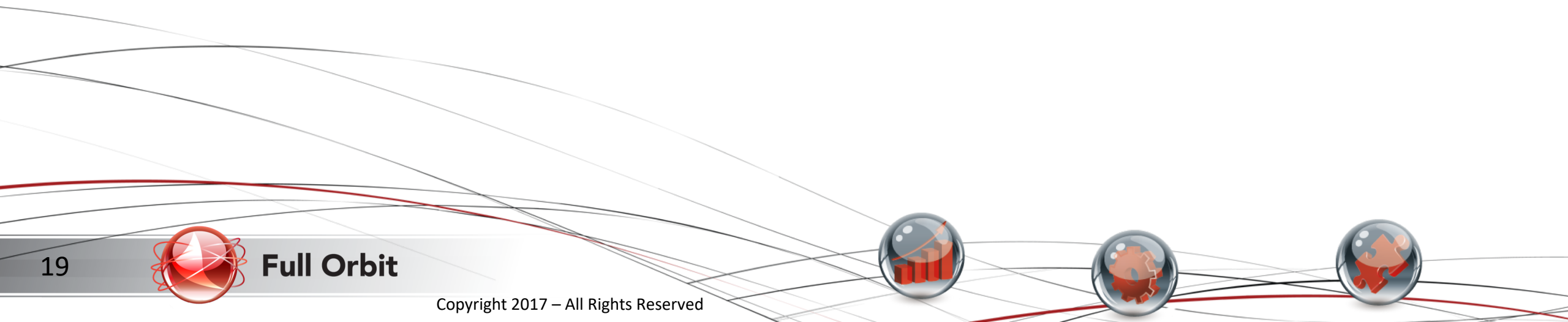
- Dit zijn ook transformaties
- Maar transformaties op een speciaal type RDD:
- Pair RDD
  - Pair RDD is een RDD op basis van key-value pairs
  - Key-value pair heet in Python een tuple
  - (key, value)
  - Voorbeeld:
    - [(Jan,35),(Kees,41),(Ellen,27),(Joop,45)]
    - [([2010,NL],[WII,ACTION,13.2]),([2011,BE],[WII,ACTION,11.2])]





# groupByKey()

- Groepeert data per key
- Resultaat is een nieuwe pair RDD
- Value van nieuwe RDD is een list van values
- Kostbare transformatie
- Gebruik bij voorkeur reduceByKey()



# reduceByKey()

- Reduceert data per key
- Resultaat is een nieuwe pair RDD
- Heeft function nodig als parameter
- Voorbeeld:
  - reduceByKey(lambda x, y: x+y) geeft total per key





# sortByKey()

- Sorteert RDD op basis van key
- Resultaat is een pair RDD
- Parameter voor oplopend of aflopend sorteren
  - True = oplopend
  - False = aflopend





# 6. Joins

- Een join in een transformatie voor pair RDD
- Resultaat van een join is een nieuwe pair RDD op basis van 2 pair RDD's.
- Join is op basis van de key
- Nieuwe RDD bevat de values van beide pair RDD's
- Je kunt in één join slechts één pair RDD aan een andere koppelen





# Verschillende joins

- Vergelijkbaar met SQL
- Join()
  - Inner join
  - Koppelt alleen gemeenschappelijke key's
- leftOuterJoin()
  - Koppelt ook keys die niet voorkomen in 'gekoppelde' pair RDD
- rightOuterJoin()
  - Koppelt ook keys die niet voorkomen in 'koppelende' pair RDD





# 7. Caching en persistency

- In sommige gevallen is het handig de resultaten van een aantal transformaties en/of acties op een RDD op te slaan.
- Voorbeeld:
  - Groot bestand waarop een aantal transformaties gedefinieerd zijn om de data te schonen. Deze ‘schone’ data wordt gebruikt voor verschillende analyses.
    - Header record verwijderd
    - Niet correcte waardes gecorrigeerd of verwijderd
    - Ontbrekende gegevens aangevuld.





# persist()

- Met actie persist() wordt een RDD opgeslagen
- Verschillende methoden van opslag:
  - MEMORY_ONLY
  - MEMORY_AND_DISK
  - MEMORY_ONLY_SER
    (serialized, efficienter wat ruimte betreft)
  - MEMORY_AND_DISK_SER
    (serialized)
  - DISK_ONLY





# cache()

- cache is een speciale vorm van persist()
- cache() = persist(MEMORY_ONLY)





# Wat te kiezen?

- Bij voorkeur in opslaan in geheugen
  - Is meest efficient m.b.t. CPU
  - Past data niet in geheugen, probeer data ‘serialized’ op te slaan.
    Dit neemt minder geheugenruimte in beslag
- Voorkom te veel opslag op disk.
  - Traag