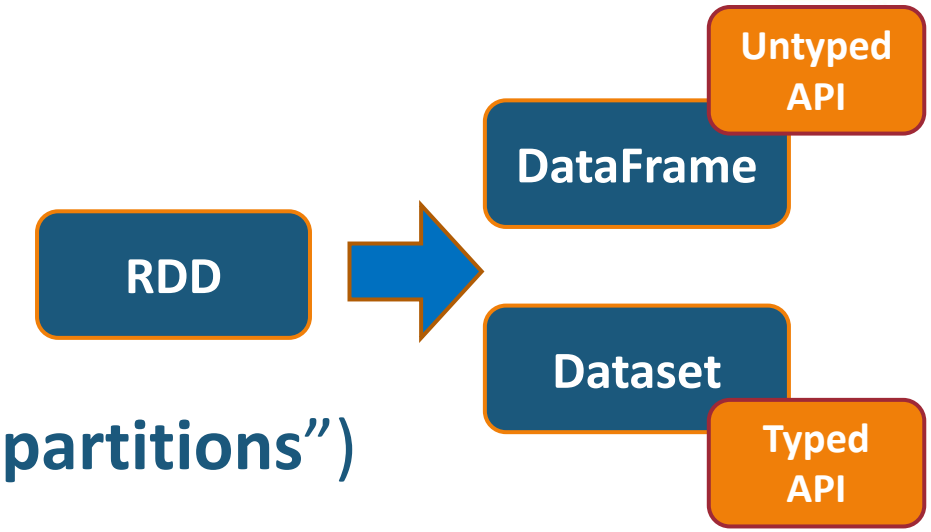
Low Level API, RDD

МОДУЛЬ 3

ШКОЛА БОЛЬШИХ ДАННЫХ

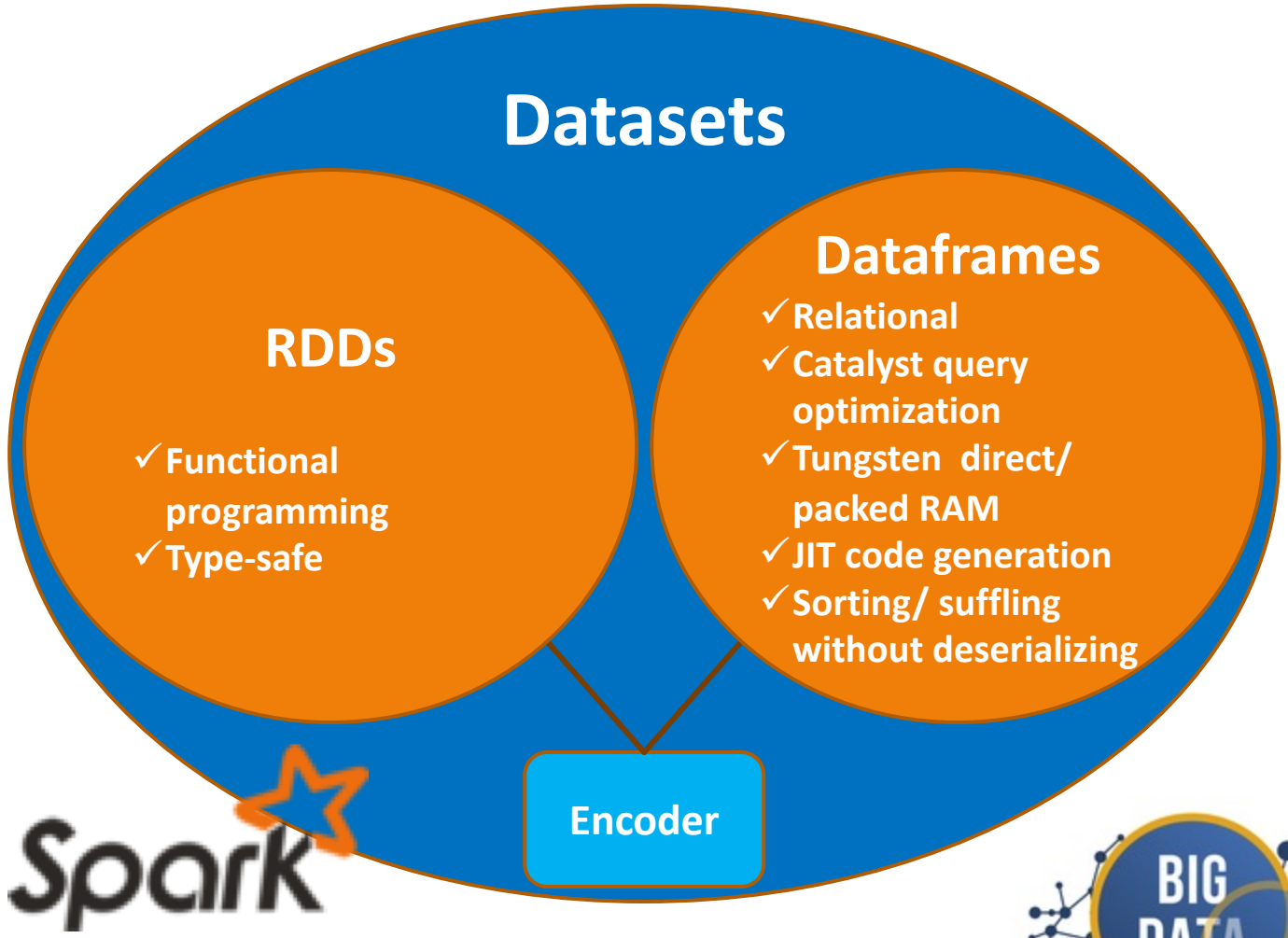
Что такое RDD



- набор **объектов**, разбитых на разделы (“**partitions**”)
- часть “**Low Level API**”
- “**dataframe**” "компилируются" в “**RDD**”
- **менее** функционально полная абстракция “**spark**”

Что такое RDD

- java objects
- garbage collection
- Tungsten
  - ✓ memory management
  - ✓ code generation



Зачем нам нужны

- для понимания концепции (как **“SQL”**)
- работы с legacy кодом
- если нужна функциональность, которой нет в “Structured API”
  - ✓ например, управление размещением данных по разделам
  - ✓ преобразование из JSON
- фокус все равно на **“Structured API”**

Формальное определение RDD

RDD характеризуется следующим набором атрибутов:

- списком разделов (**“partitions”**)
- функцией для вычисления каждого “split”
- списком зависимостей от других RDD
- (опционально) функцией “Partitioner” для **“key-value RDD”**
- (опционально) списком предпочтений узлов, на которых обрабатывать каждый **“split”** (например, список расположений блоков файла HDFS)

Трансформации и действия

- та же модель, что и с **“dataframe”**
- набор трансформаций **меньше** (и он - другой)
- используется **“lazy evaluation”**

Lazy evaluation (or call-by-name) is an evaluation strategy which delays the evaluation of an expression until its value is needed

Создание RDD

- **“spark.sparkContext”**: точка входа в Low Level API
- **“parallelize()”** - создание RDD из «списка»
- **“textFile()”** - создание RDD из файлов (каждая строка - элемент RDD)
- **“wholeTextFiles()”** - каждый файл - элемент RDD (например, наши сложные XML или JSON)

Основные трансформации

- **“distinct()”**- оставить только уникальные элементы RDD
- **“filter()”** - отфильтровать элементы с помощью заданной функции
- **“map()”** - преобразовать элементы с помощью заданной функции
- **“flatMap()”** - преобразовать с добавлением элементов RDD
- **“sort()”** - упорядочить элементы RDD
- **“glom()”** - собрать элементы раздела в список

Основные действия

- **“reduce()”** - сворачивает элементы RDD, используя коммутативную и ассоциативную функцию (с двумя параметрами)
- **“count()”** - посчитать количество элементов в RDD
- **“first()”** - собрать первые N элементов RDD на драйвер
- **“min()/max()”** - найти максимум и минимум
- **“collect()/take()”** - собрать/собрать N элементов RDD на драйвер
- **“saveAsTextFile()”** - сохранить RDD в виде текстового файла

Python и RDD

При использовании **python-а** и **RDD** происходит потеря производительности (**в отличие от RDD + java или использования Structured API**): RDD - это как использование python UDF для каждого элемента RDD

- ✓ сначала данные **десериализуются** в python структуры данных
- ✓ обрабатываем их на **python-**е
- ✓ **сериализуем** результат в формат JVM

Вопросы?