Spark 2

Alexey Zinovyev, Java/BigData Trainer in EPAM

About

With IT since 2007

With Java since 2009

With Hadoop since 2012

With EPAM since 2015

Secret Word from EPAM

itsubbotnik

Contacts

E-mail : Alexey_Zinovyev@epam.com

Twitter : @zaleslaw @BigDataRussia

Facebook: https://www.facebook.com/zaleslaw

vk.com/big_data_russia **Big Data Russia**

vk.com/java_jvm **Java & JVM langs**

Sprk Dvlprs! Let’s start!

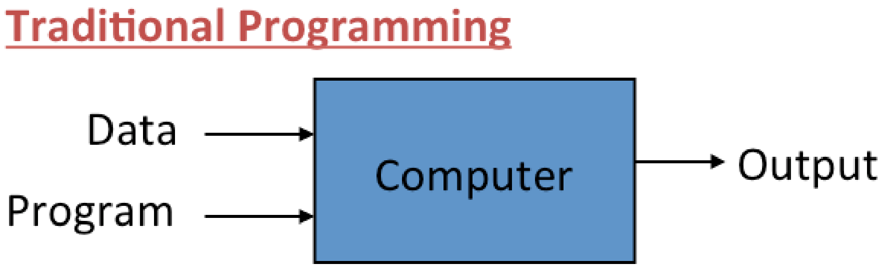
< SPARK 2.0

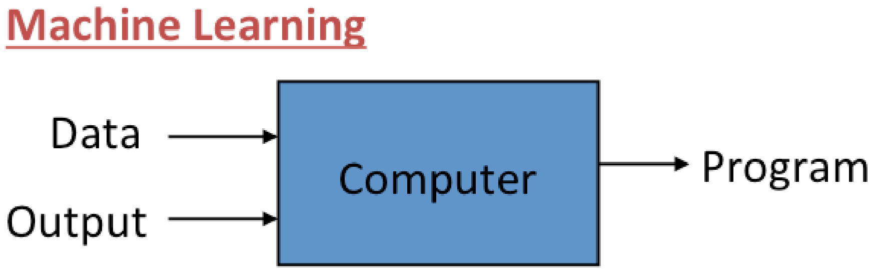
Big Data in 2014

Big Data in 2017

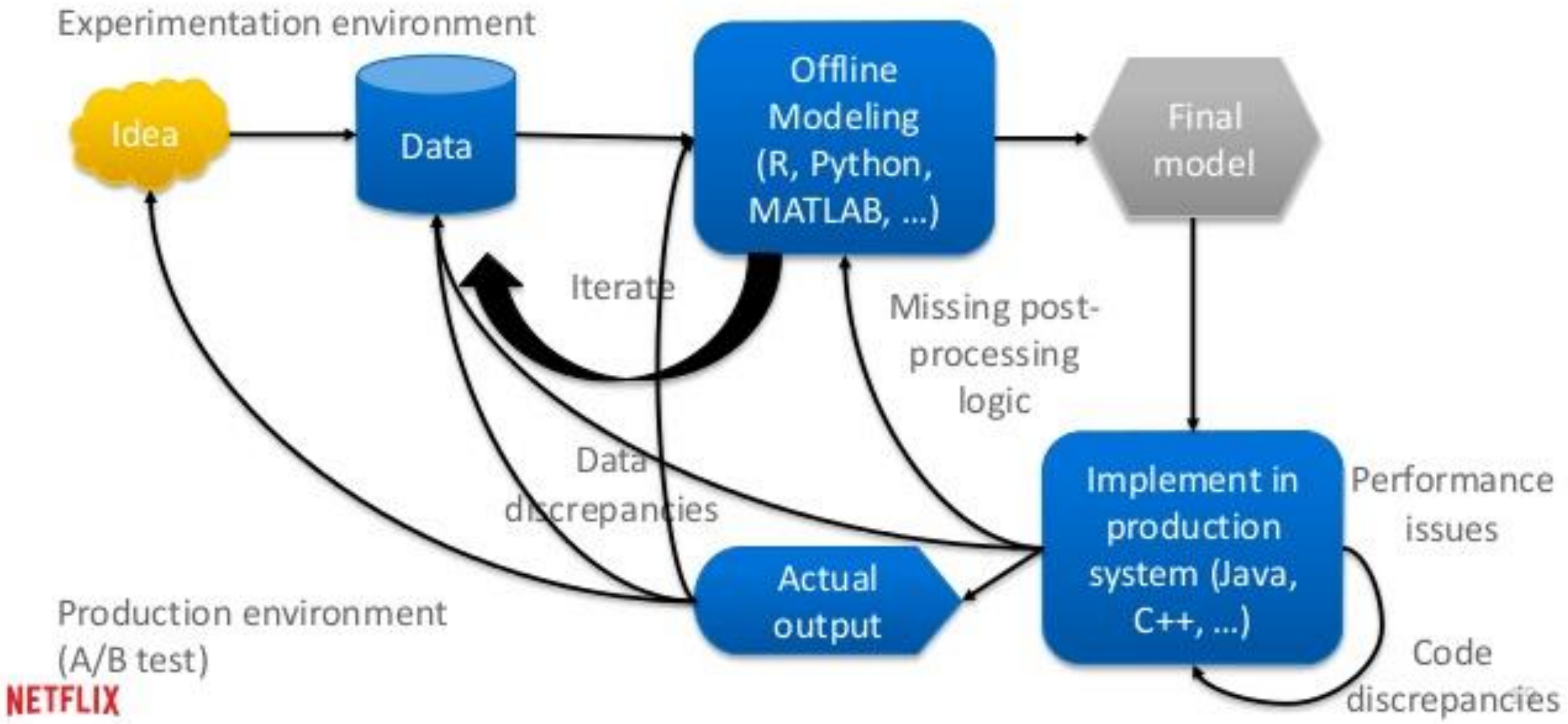
Machine Learning EVERYWHERE

Machine Learning vs Traditional Programming

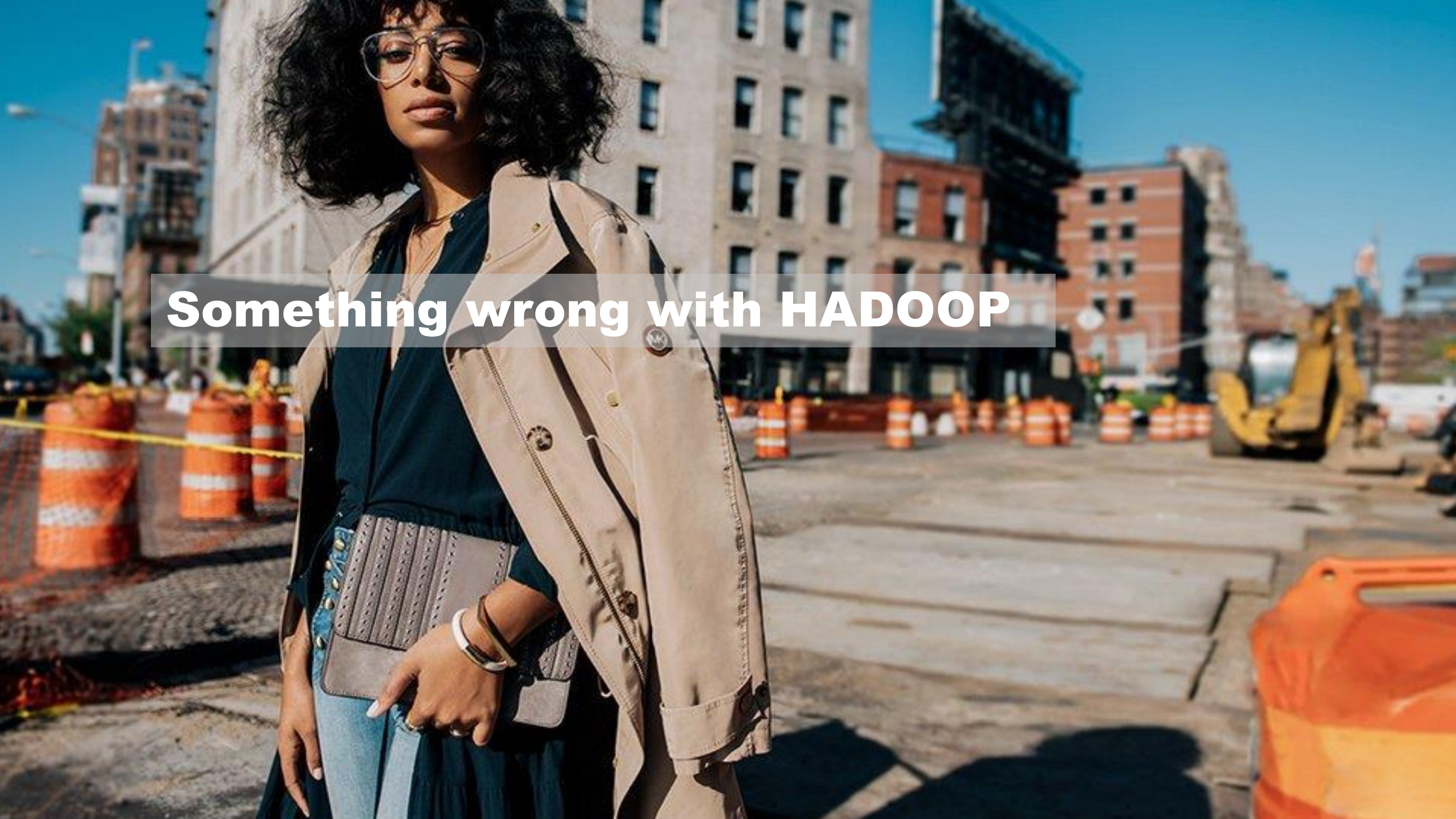




Example development process


Experimentation environment
Idea
Data
Offline Modeling (R, Python, MATLAB, ...)
Final model
Iterate
Missing post-processing logic
Data discrepancies
Implement in production system (Java, C++, ...)
Performance issues
Code discrepancies
Actual output
Production environment (A/B test)
NETFLIX

Something wrong with HADOOP

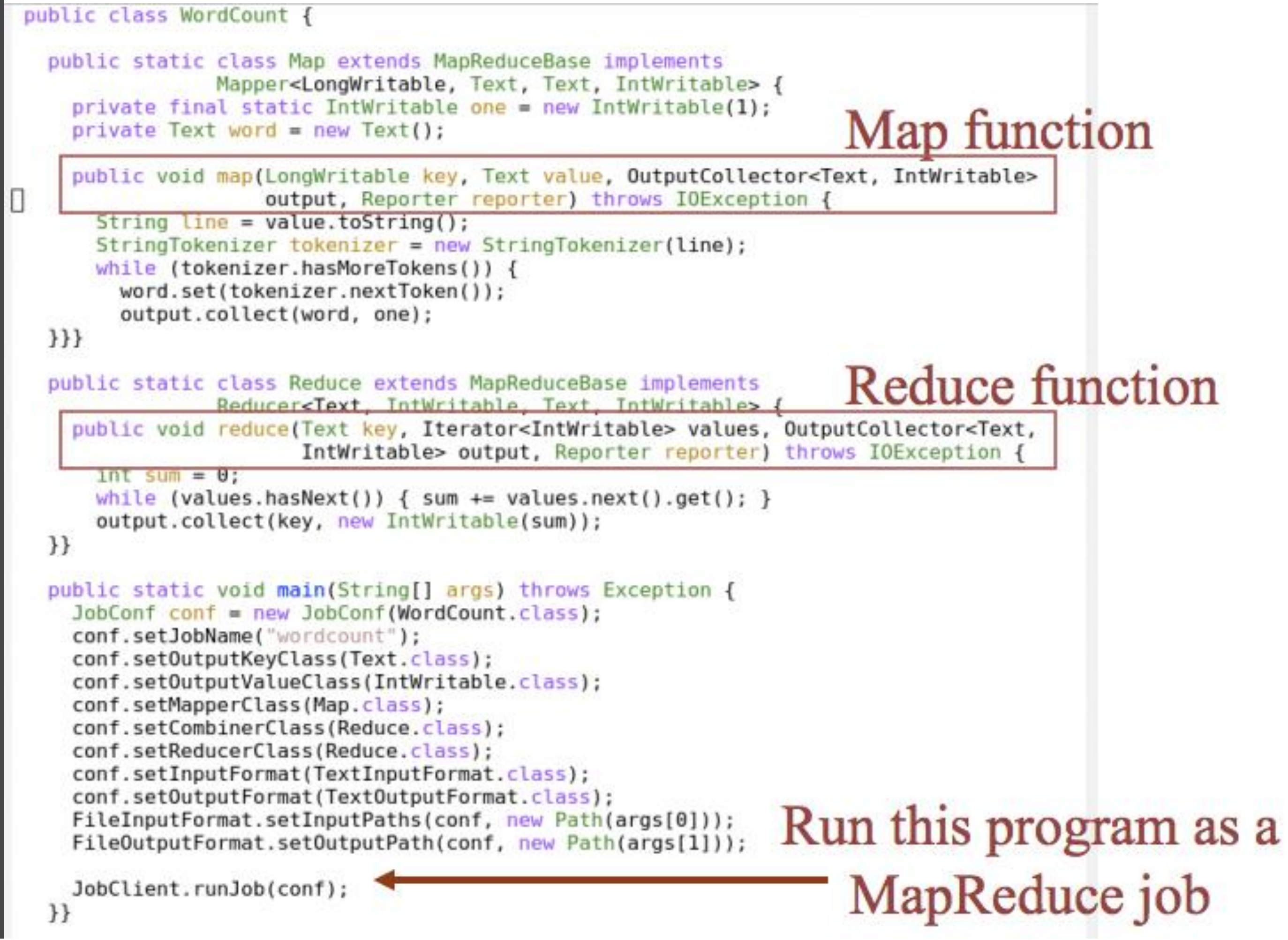


Hadoop is not SEXY

Whaaaat?

Map Reduce Job Writing

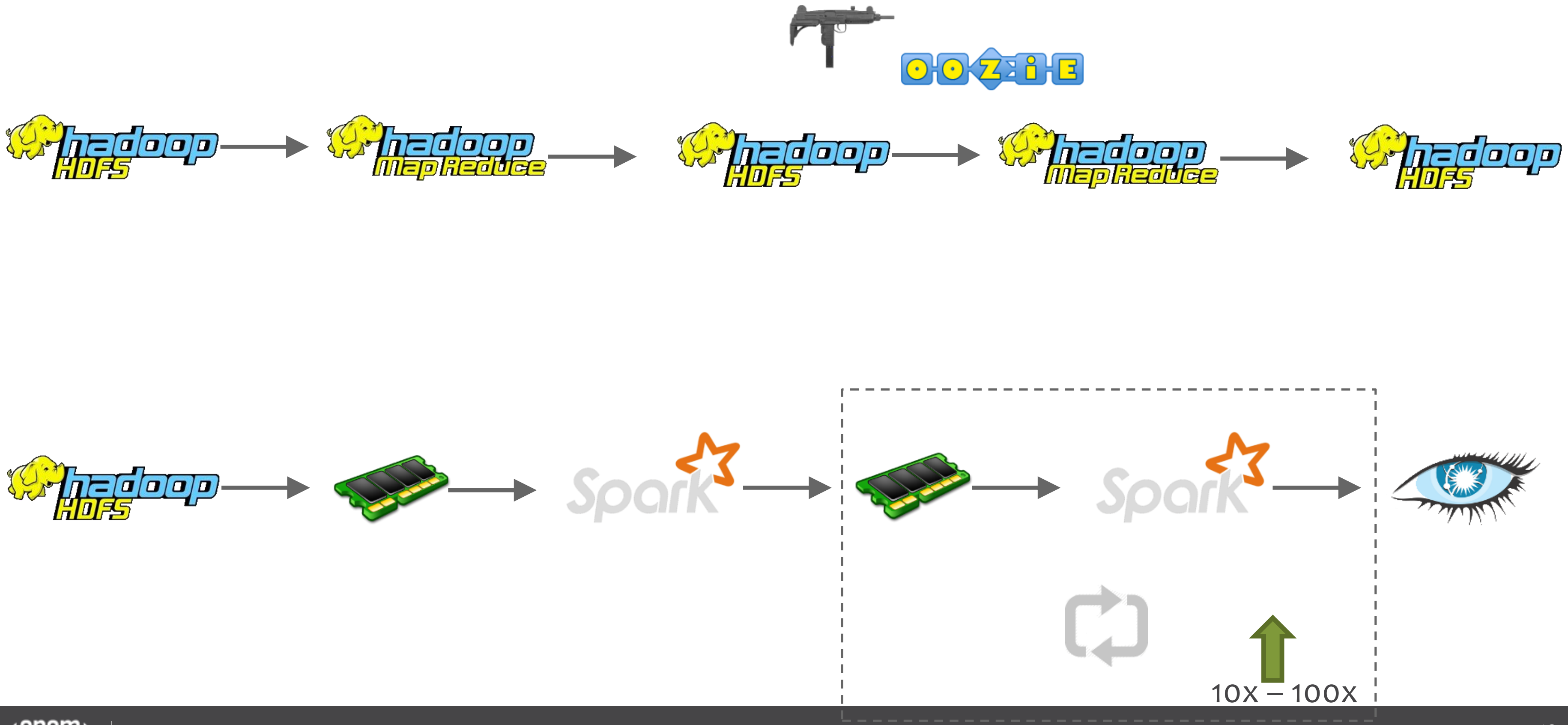
MR code



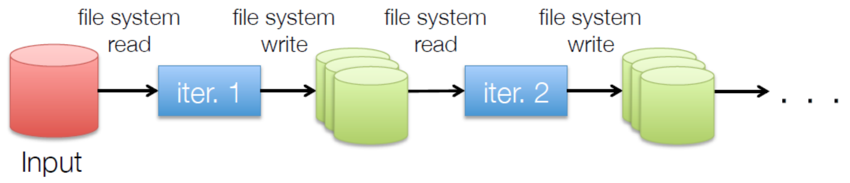
```
public class WordCount {

  public static class Map extends MapReduceBase implements
                Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable>
                    output, Reporter reporter) throws IOException {
      String line = value.toString();
      StringTokenizer tokenizer = new StringTokenizer(line);
      while (tokenizer.hasMoreTokens()) {
        word.set(tokenizer.nextToken());
        output.collect(word, one);
  }}}

  public static class Reduce extends MapReduceBase implements
                Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,
                       IntWritable> output, Reporter reporter) throws IOException {
      int sum = 0;
      while (values.hasNext()) { sum += values.next().get(); }
      output.collect(key, new IntWritable(sum));
  }}

  public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);
    conf.setMapperClass(Map.class);
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);
    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);
    FileInputFormat.setInputPaths(conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));

    JobClient.runJob(conf);
  }}
```

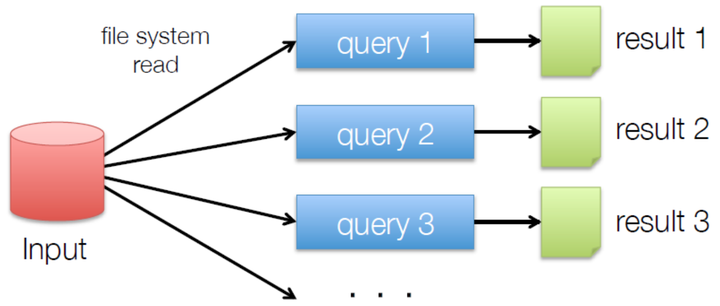
Hadoop Developers Right Now

Iterative Calculations

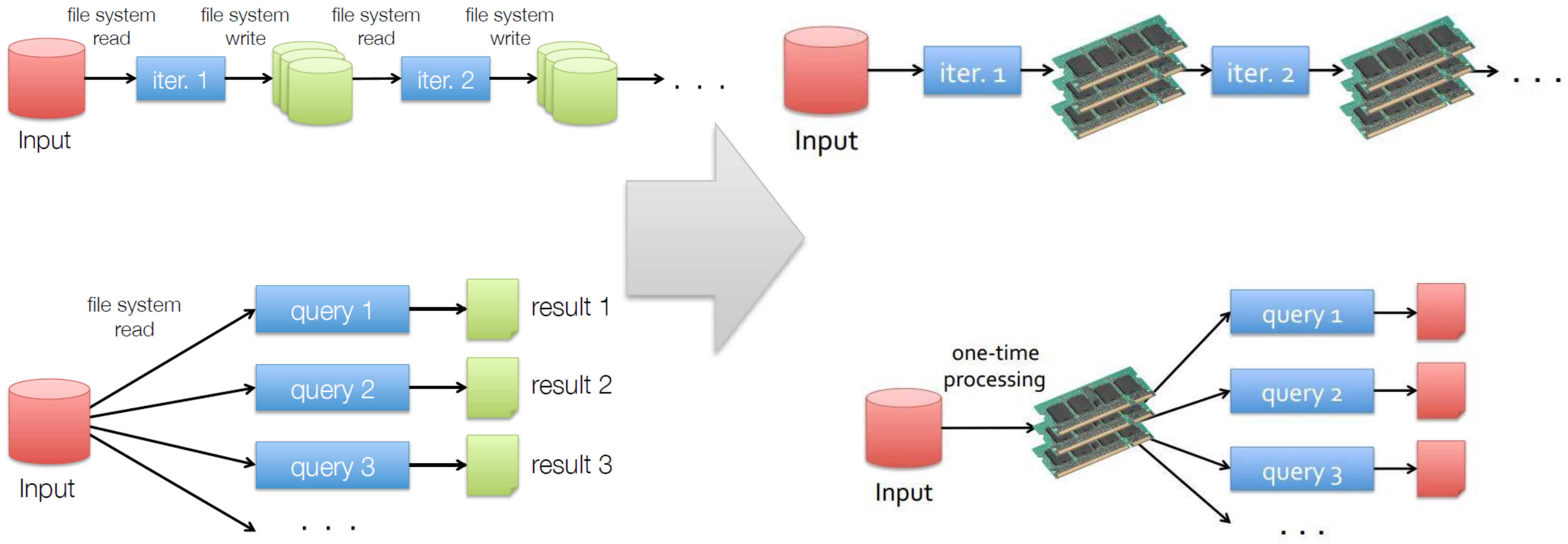


MapReduce vs Spark

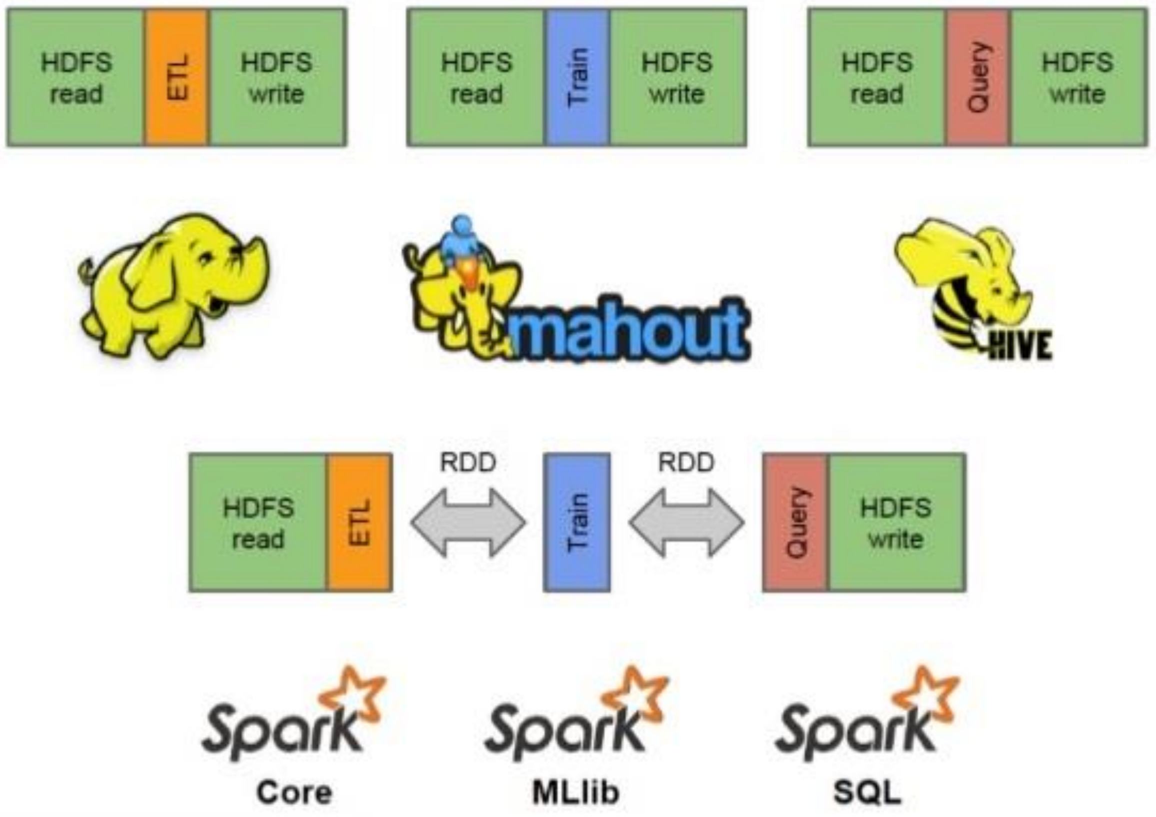




MapReduce vs Spark

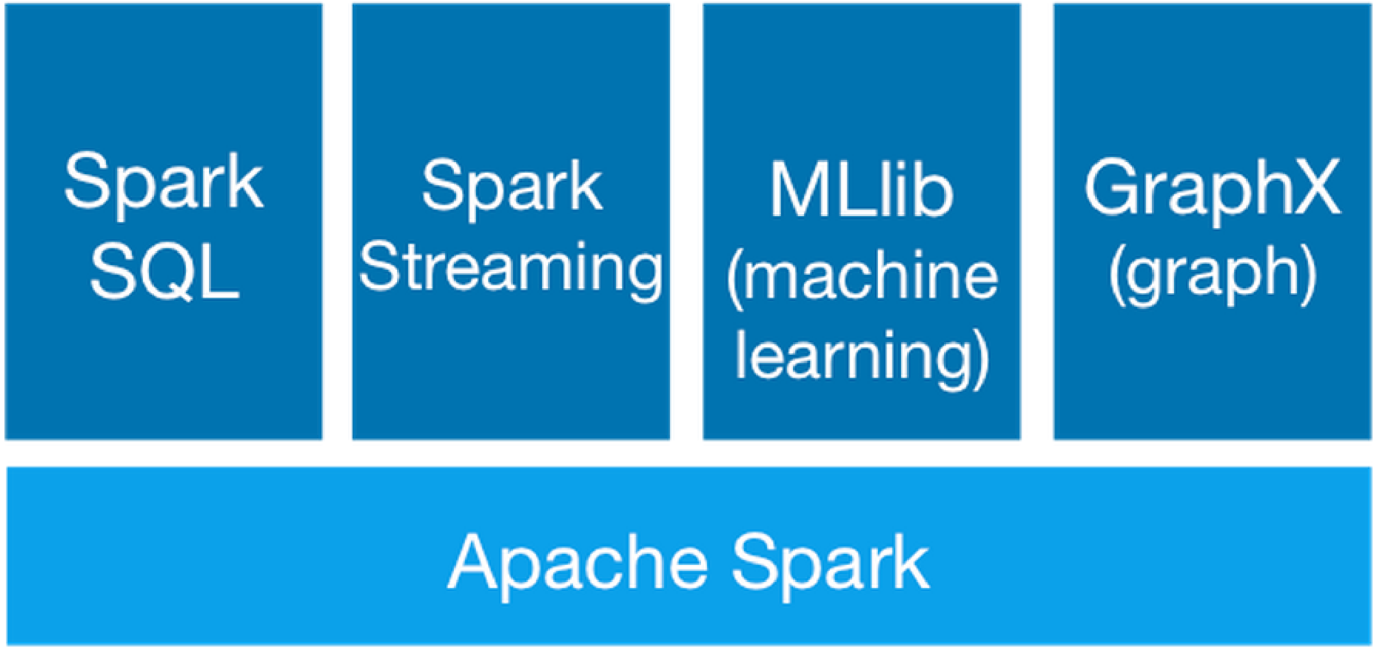


MapReduce vs Spark

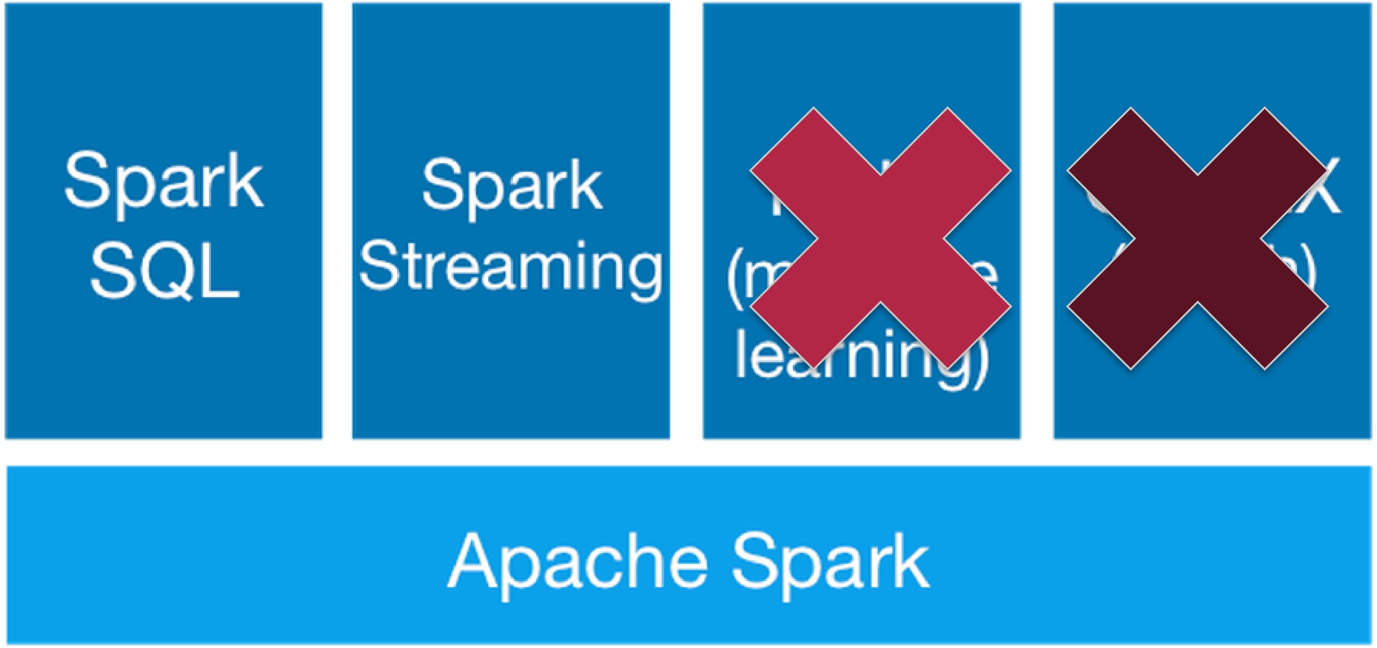


SPARK 2.0 DISCUSSION

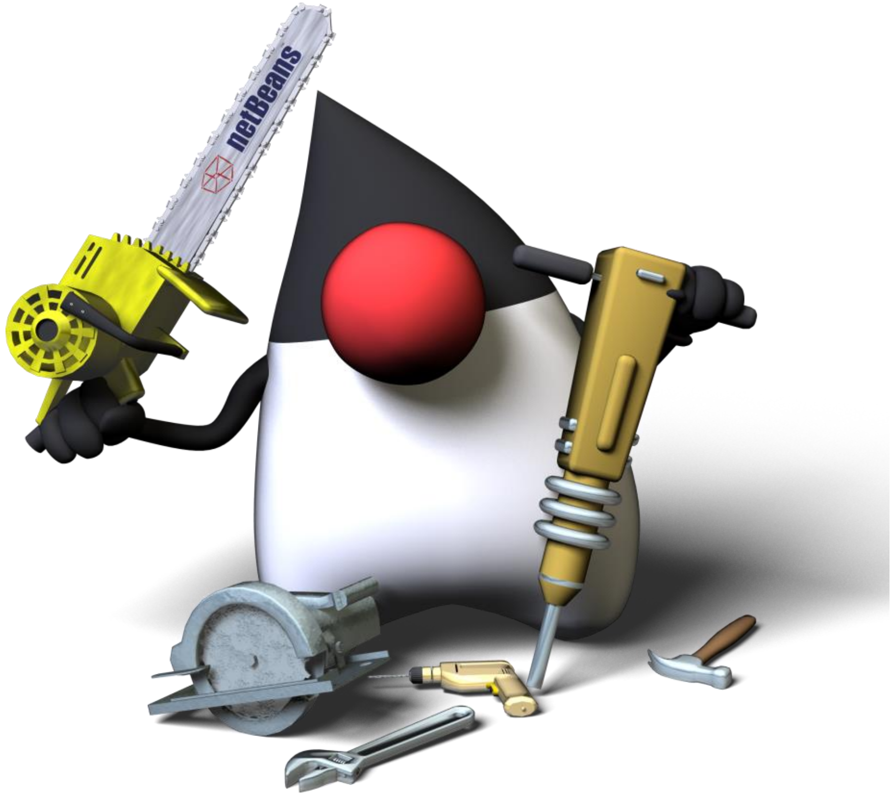
Spark Family



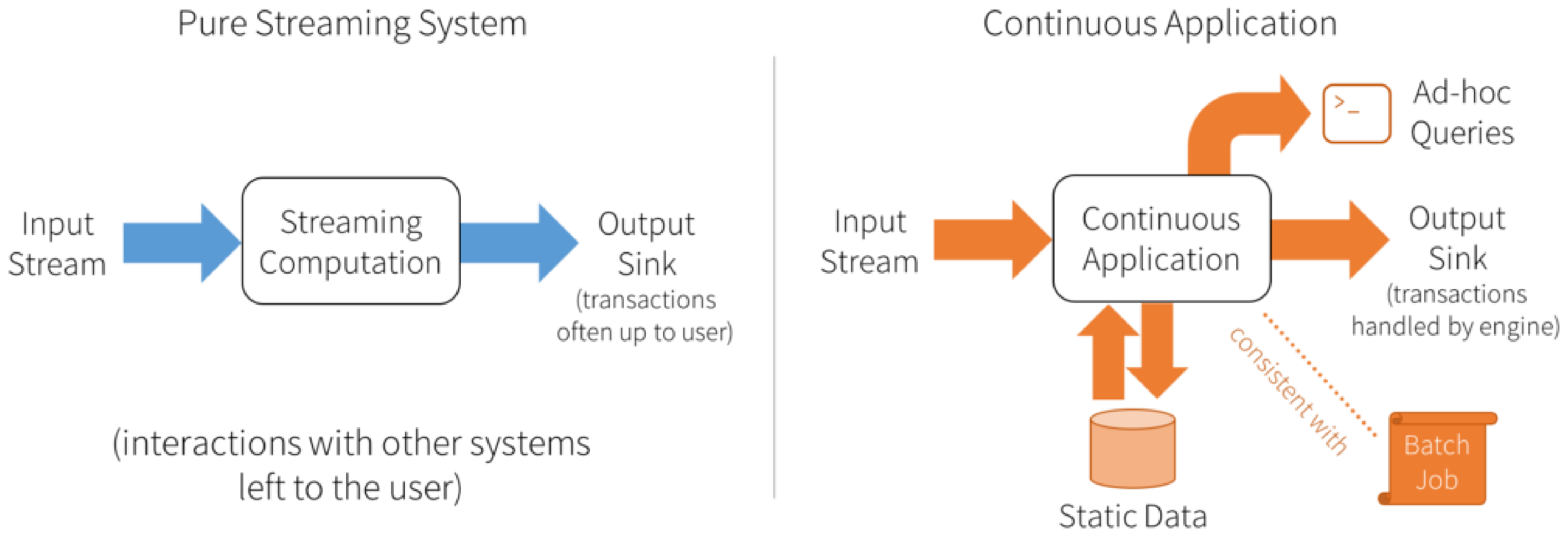
Spark Family



Case #0 : How to avoid DStreams with RDD-like API?



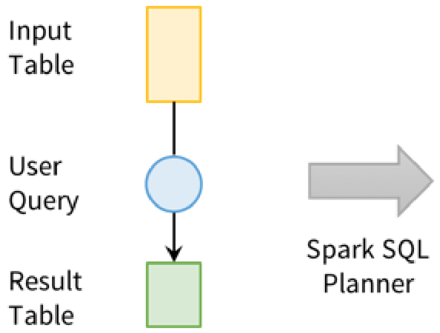
Continuous Applications



Continuous Applications cases

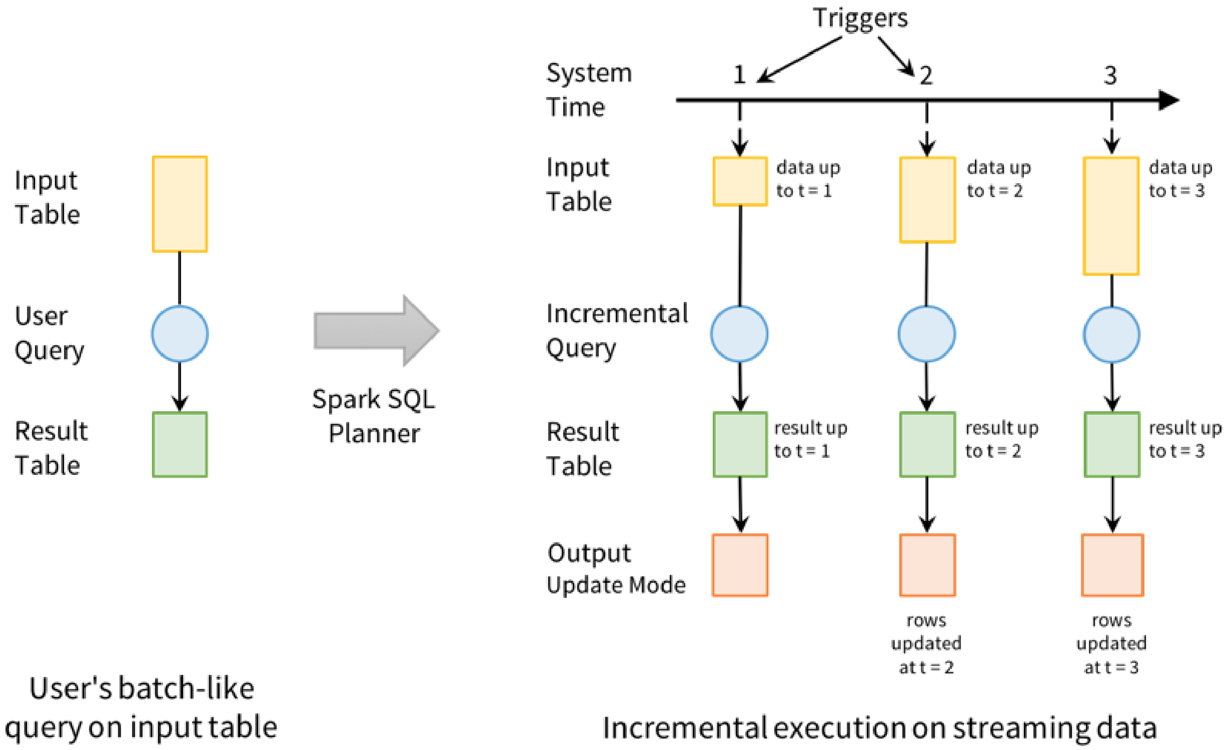
- Updating data that will be served in real time
- Extract, transform and load (ETL)
- Creating a real-time version of an existing batch job
- Online machine learning

Write Batches



User's batch-like query on input table

Catch Streaming



The main concept of Structured Streaming

You can express your streaming computation the same way you would express a batch computation on static data.

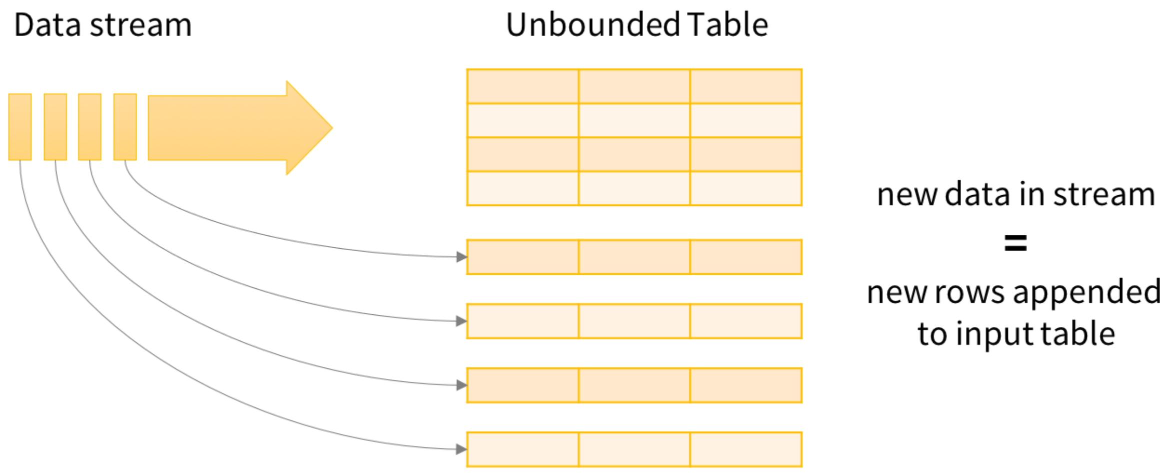
Batch

```
// Read JSON once from S3
logsDF = spark.read.json("s3://logs")

// Transform with DataFrame API and save
logsDF.select("user", "url", "date")
      .write.parquet("s3://out")
```

Real Time

```
// Read JSON continuously from S3
logsDF = spark.readStream.json("s3://logs")

// Transform with DataFrame API and save
logsDF.select("user", "url", "date")
      .writeStream.parquet("s3://out")
      .start()
```

Unlimited Table



Data stream as an unbounded Input Table

WordCount from Socket

```
val lines = spark.readStream
  .format("socket")
  .option("host", "localhost")
  .option("port", 9999)
  .load()
```

WordCount from Socket

```
val lines = spark.readStream
  .format("socket")
  .option("host", "localhost")
  .option("port", 9999)
  .load()

val words = lines.as[String].flatMap(_.split(" "))
```

WordCount from Socket

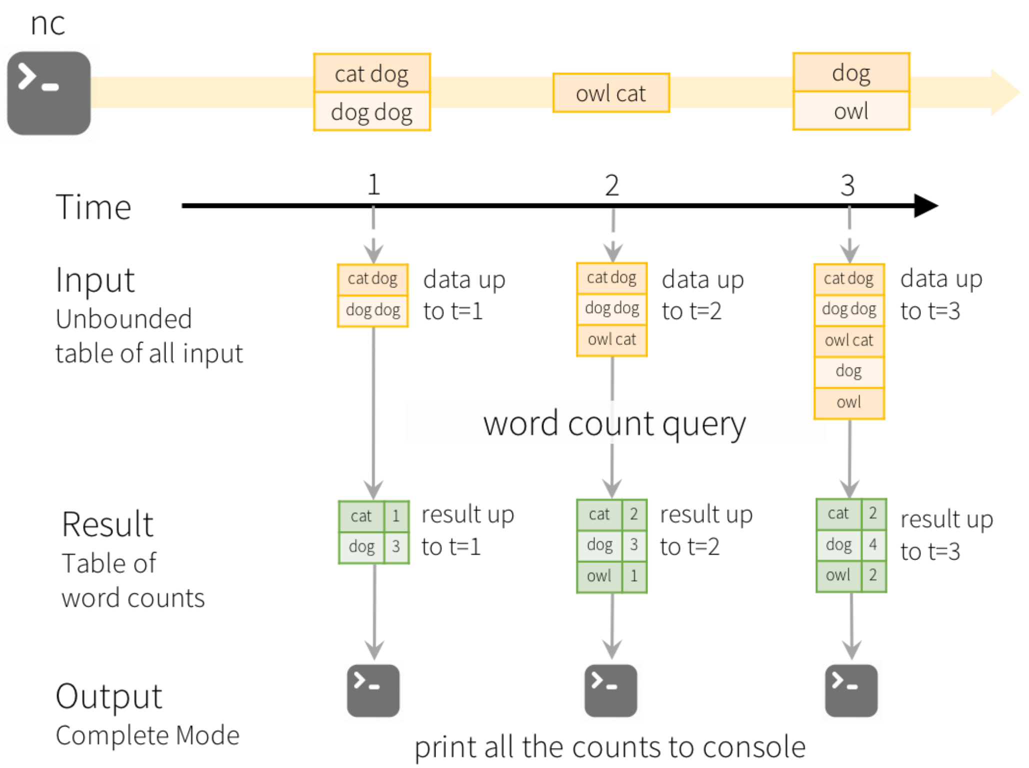
```
val lines = spark.readStream
  .format("socket")
  .option("host", "localhost")
  .option("port", 9999)
  .load()

val words = lines.as[String].flatMap(_.split(" "))


val wordCounts = words.groupBy("value").count()
```

WordCount from Socket

```
val lines = spark.readStream
  .format("socket")
  .option("host", "localhost")
  .option("port", 9999)
  .load()


val words                   ing].flatMap(_.split(" "))


val wordCounts = words.groupBy("value").count()
```

Don't forget to start Streaming

WordCount with Structured Streaming



Structured Streaming provides …

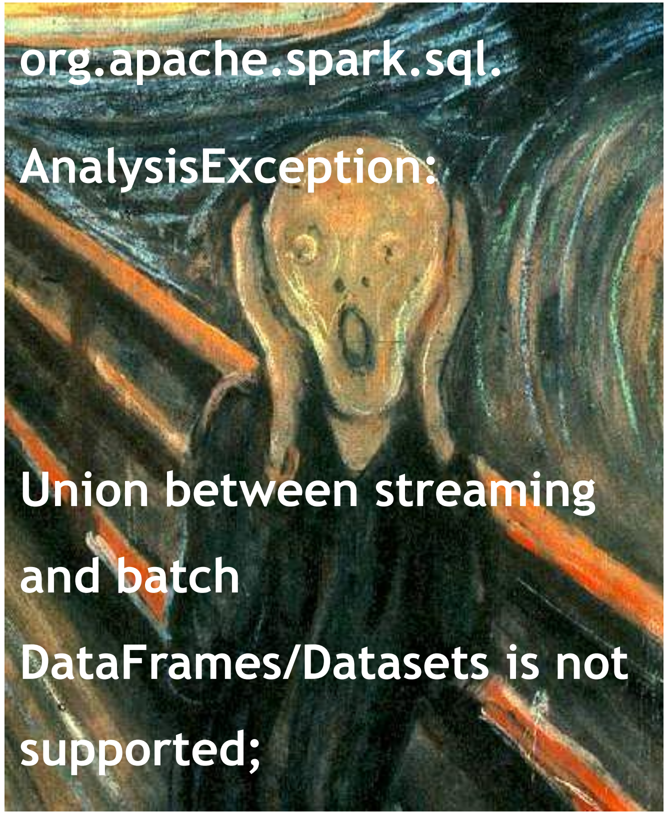
- fast & scalable
- fault-tolerant
- end-to-end with exactly-once semantic
- stream processing
- ability to use DataFrame/DataSet API for streaming

Structured Streaming ~~provides~~ (in dreams) …

- fast & scalable
- fault-tolerant
- end-to-end with exactly-once semantic
- stream processing
- ability to use DataFrame/DataSet API for streaming

Let’s UNION streaming and static DataSets

Let’s UNION streaming and static DataSets



org.apache.spark.sql.

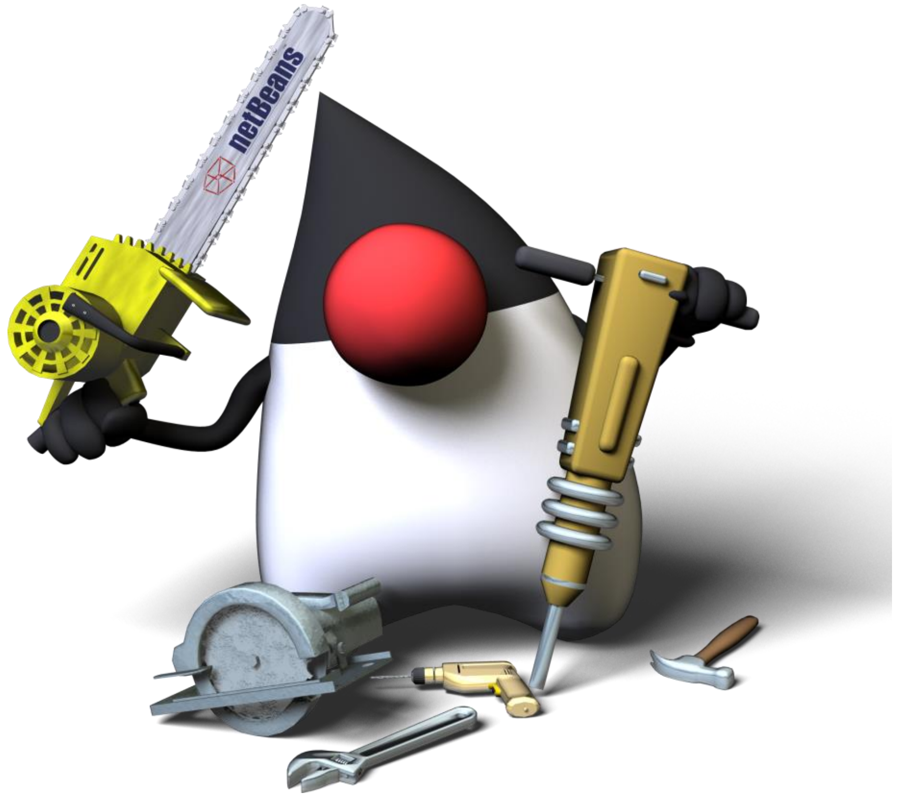
AnalysisException:

Union between streaming and batch DataFrames/Datasets is not supported;

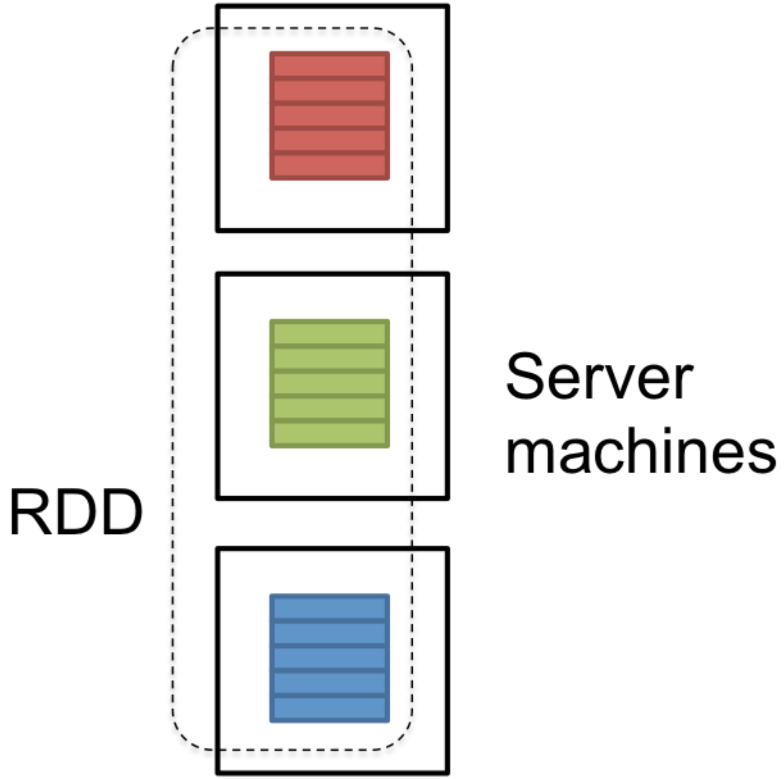
Let’s UNION streaming and static DataSets

Go to UnsupportedOperationChecker.scala and check your operation

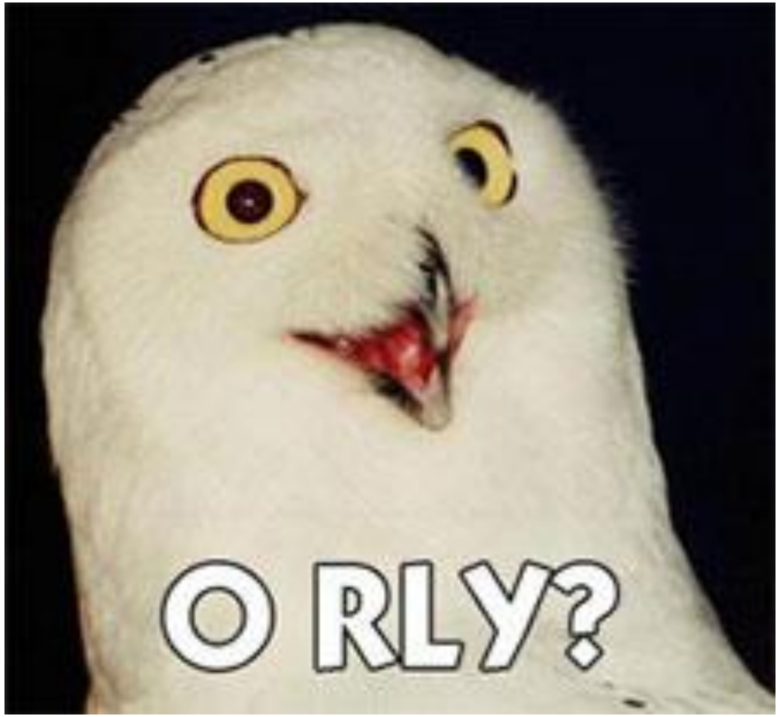
Case #1 : We should think about optimization in RDD terms



Single Thread collection



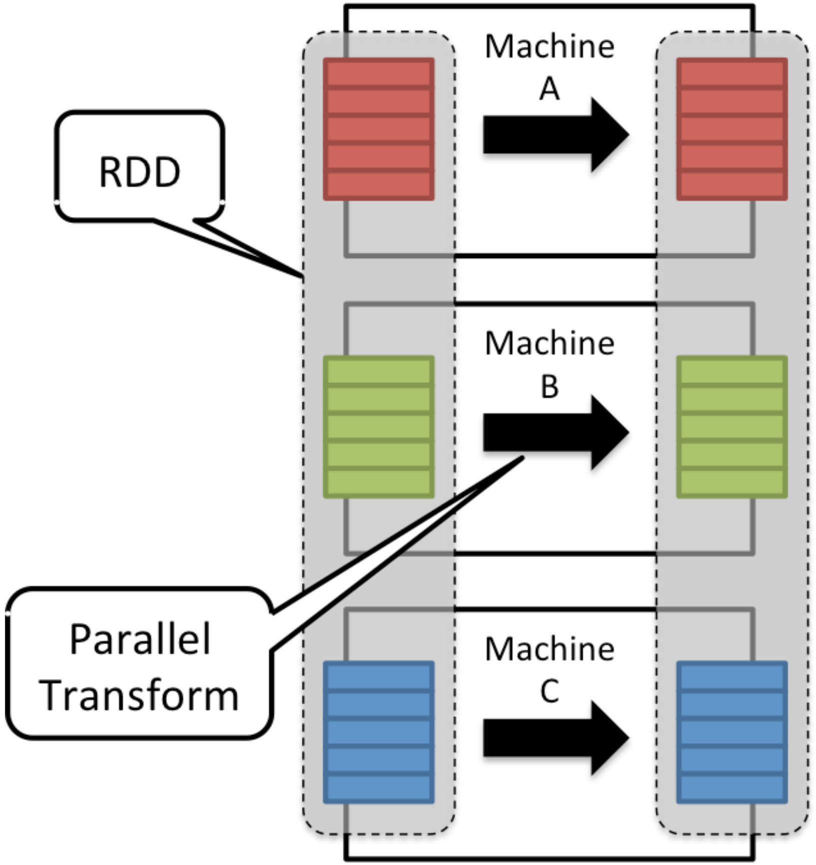
No perf issues, right?



The main concept

more partitions = more parallelism

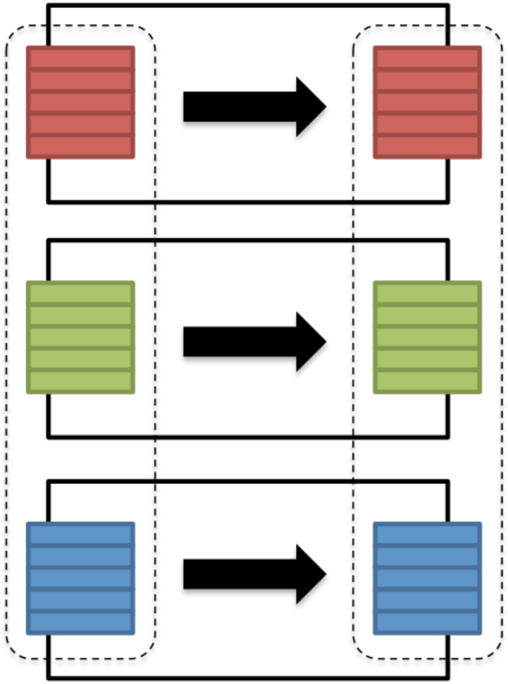
Do it parallel



I’d like NARROW

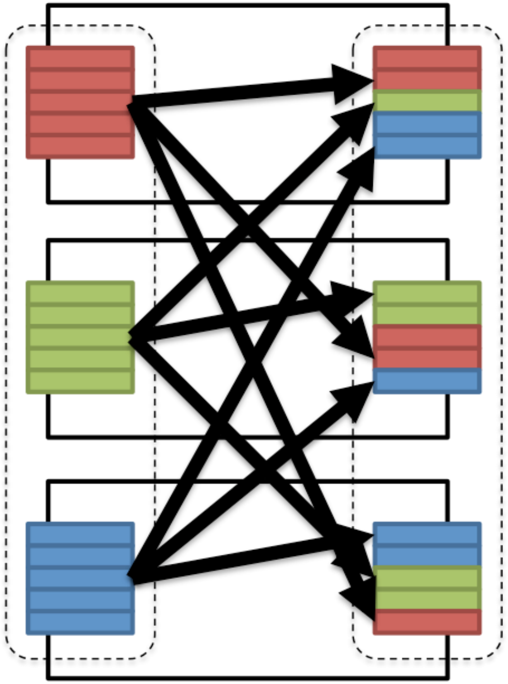
Narrow transformation

- Input and output stays in same partition
- No data movement is needed



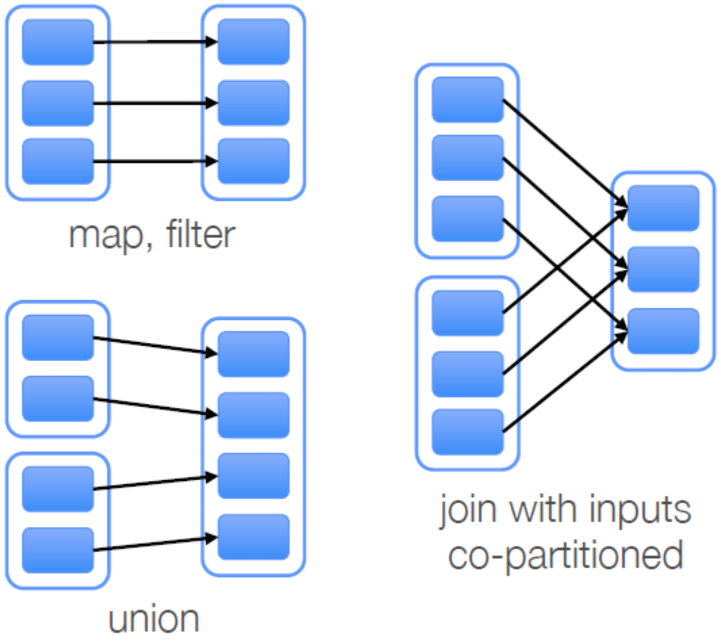
Wide transformation

- Input from other partitions are required
- Data shuffling is needed before processing

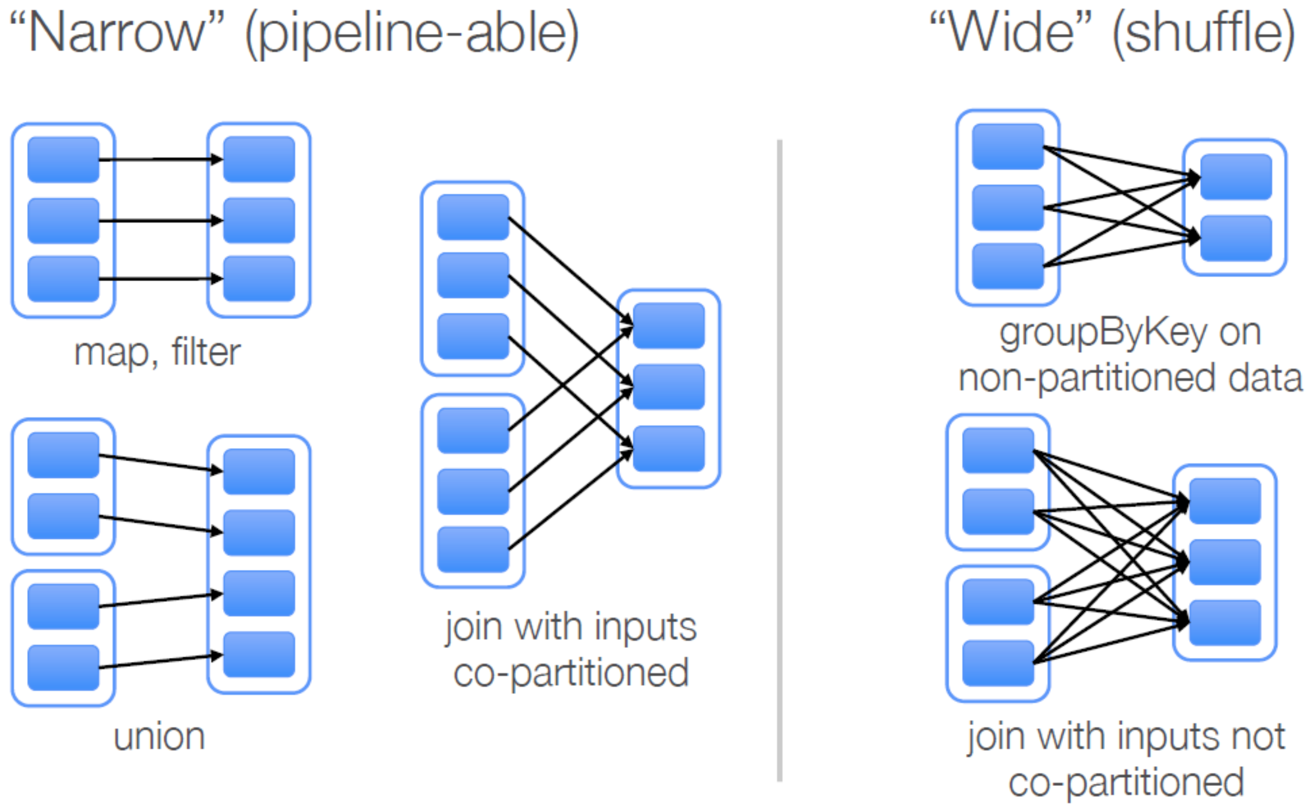


Map, filter, filter

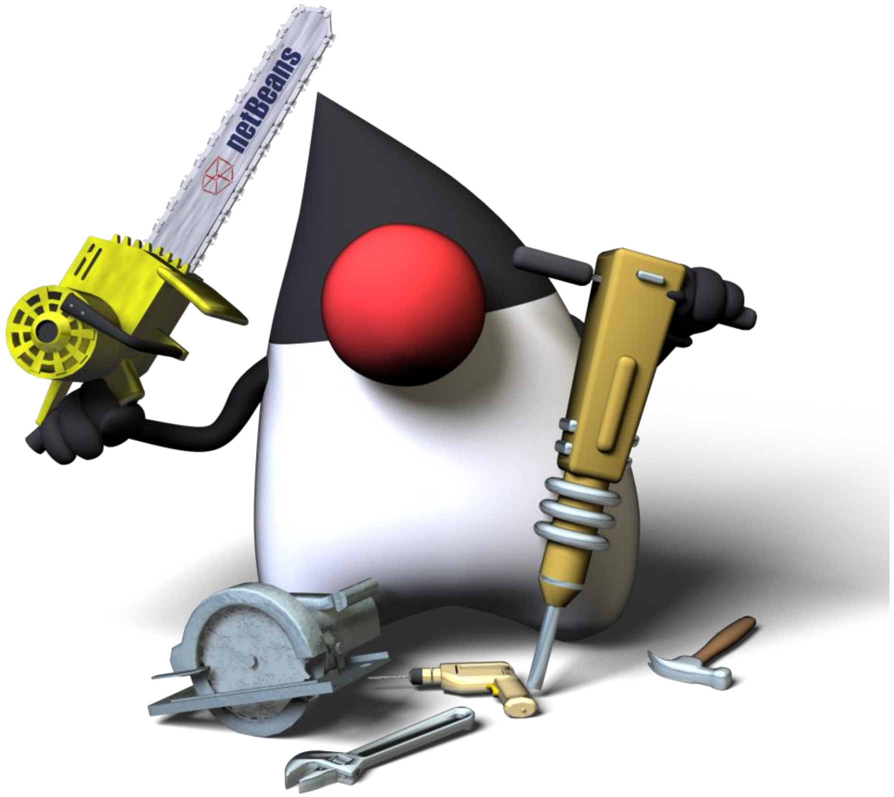
“Narrow” (pipeline-able)



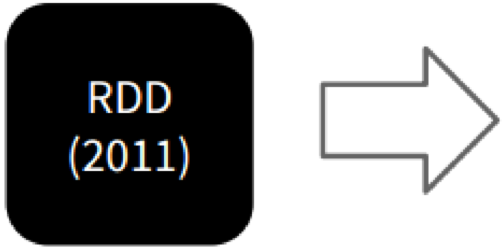
GroupByKey, join


"Narrow" (pipeline-able)
"Wide" (shuffle)
map, filter
union
join with inputs co-partitioned
groupByKey on non-partitioned data
join with inputs not co-partitioned

Case #2 : DataFrames suggest mix SQL and Scala functions



History of Spark APIs



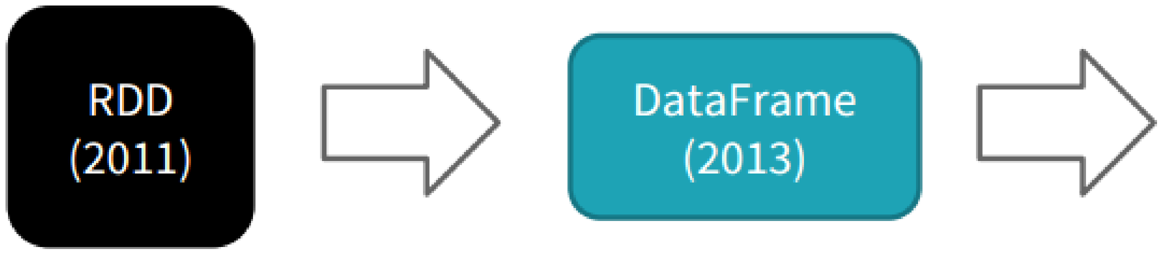
Distribute collection
of JVM objects

Functional Operators (map,
filter, etc.)

RDD

```
rdd.filter(_.age > 21) // RDD
```

History of Spark APIs



Distribute collection
of JVM objects

Functional Operators (map,
filter, etc.)

Distribute collection
of Row objects

Expression-based
operations and UDFs

Logical plans and optimizer

Fast/efficient internal
representations

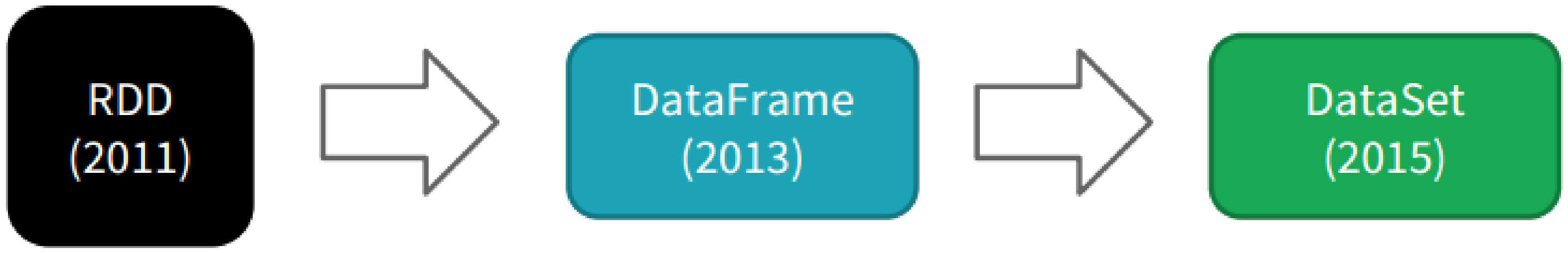
SQL

```
rdd.filter(_.age > 21) // RDD

df.filter("age > 21") // DataFrame SQL-style
```

Expression

```
rdd.filter(_.age > 21) // RDD

df.filter("age > 21") // DataFrame SQL-style

df.filter(df.col("age").gt(21)) // Expression
```

History of Spark APIs



RDD

Distribute collection of JVM objects

Functional Operators (map, filter, etc.)

DataFrame

Distribute collection of Row objects

Expression-based operations and UDFs

Logical plans and optimizer

Fast/efficient internal representations

DataSet

Internally rows, externally JVM objects

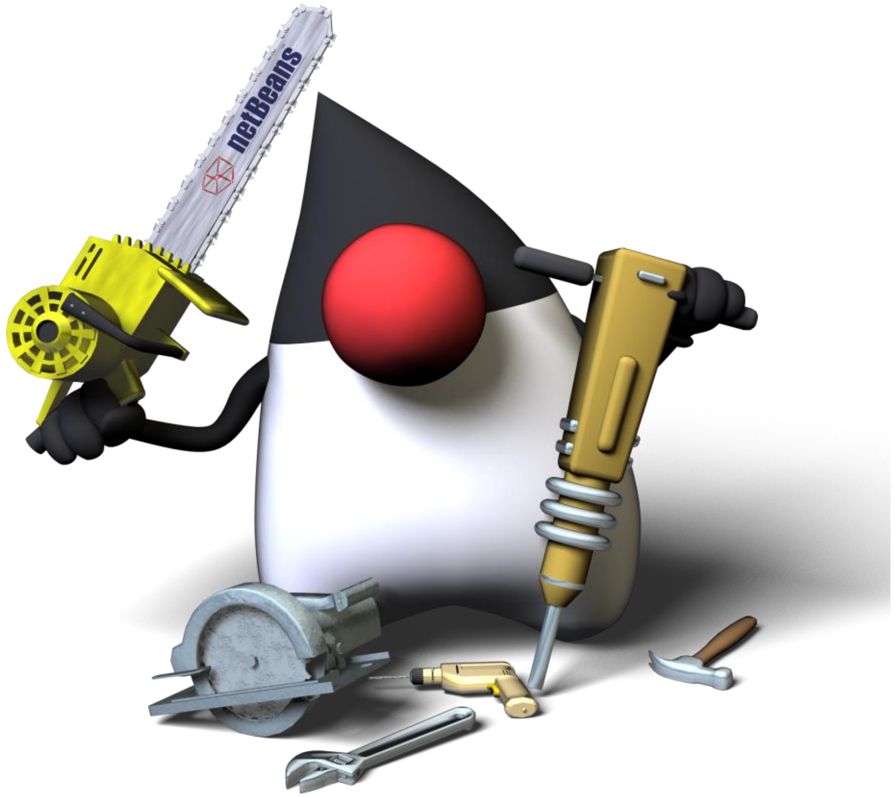
Almost the "Best of both worlds": **type safe + fast**

But slower than DF
Not as good for interactive analysis, especially Python

DataSet

```
rdd.filter(_.age > 21) // RDD

df.filter("age > 21") // DataFrame SQL-style

df.filter(df.col("age").gt(21)) // Expression style

dataset.filter(_.age < 21);
```

Case #2 : DataFrame is referring to data attributes by name



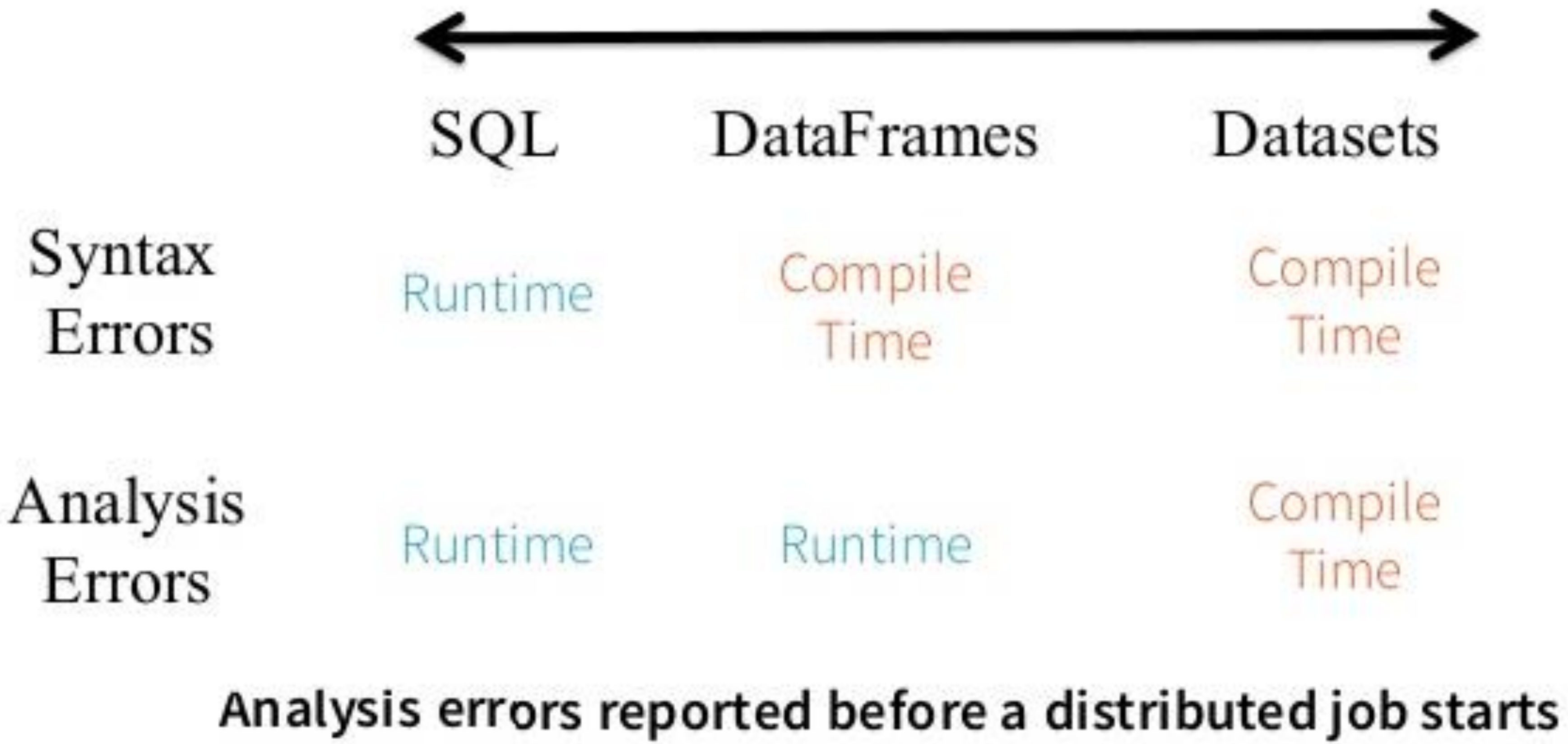
DataSet = RDD’s types + DataFrame’s Catalyst

- RDD API
- compile-time type-safety
- off-heap storage mechanism
- performance benefits of the Catalyst query optimizer
- Tungsten

DataSet = RDD’s types + DataFrame’s Catalyst

- RDD API
- compile-time type-safety
- off-heap storage mechanism
- performance benefits of the Catalyst query optimizer
- Tungsten

Structured APIs in SPARK



	SQL	DataFrames	Datasets
Syntax Errors	Runtime	Compile Time	Compile Time
Analysis Errors	Runtime	Runtime	Compile Time

Analysis errors reported before a distributed job starts

Unified API in Spark 2.0

DataFrame = Dataset[Row]

Dataframe is a schemaless (untyped) Dataset now

Define case class

```
case class User(email: String, footSize: Long, name: String)
```

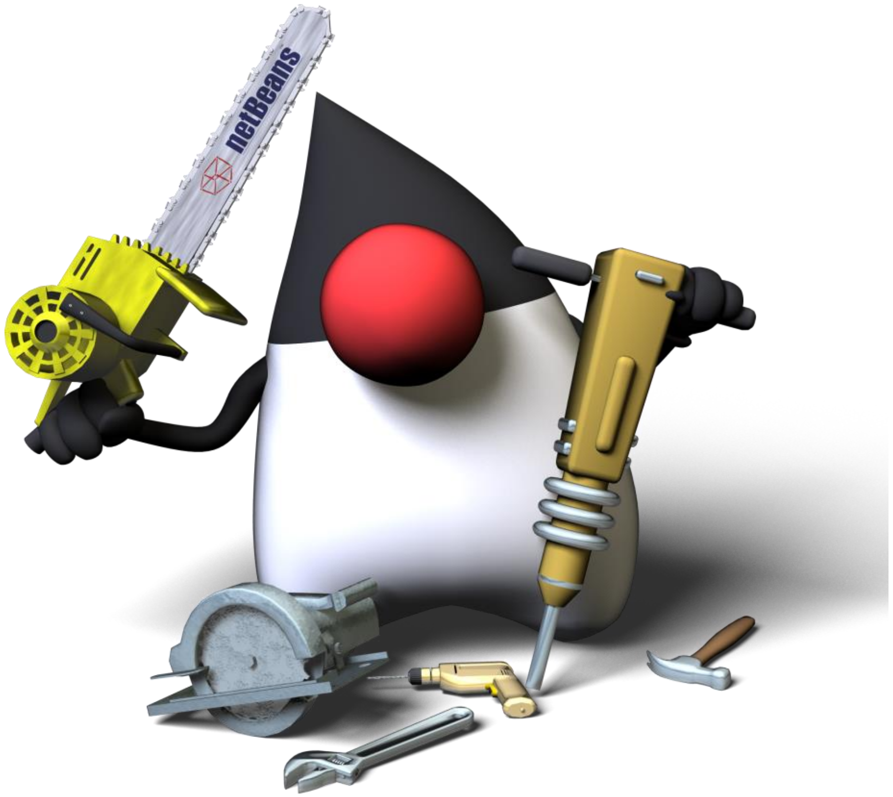
Read JSON

```
case class User(email: String, footSize: Long, name: String)


// DataFrame -> DataSet with Users
val userDS =
spark.read.json("/home/tmp/datasets/users.json").as[User]
```

Filter by Field

```
case class User(email: String, footSize: Long, name: String)


// DataFrame -> DataSet with Users

val userDS =

spark.read.json("/home/tmp/datasets/users.json").as[User]


userDS.map(_.name).collect()


userDS.filter(_.footSize > 38).collect()
```

Case #3 : Spark has many contexts



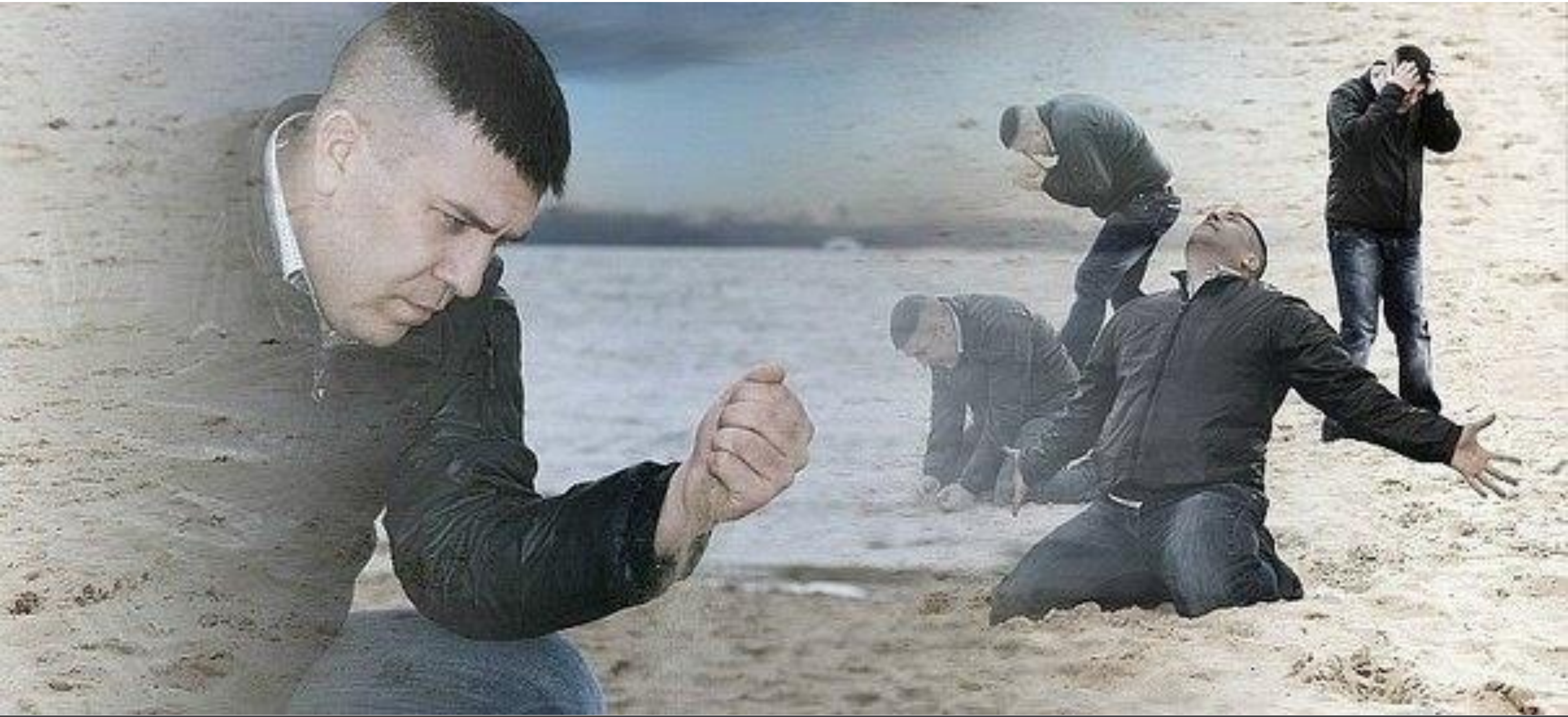
Spark Session

- New entry point in spark for creating datasets
- Replaces SQLContext, HiveContext and StreamingContext
- Move from SparkContext to SparkSession signifies move away from RDD

Spark Session

```
val sparkSession = SparkSession.builder
             .master("local")
             .appName("spark session example")
             .getOrCreate()

val df = sparkSession.read
              .option("header","true")
              .csv("src/main/resources/names.csv")


df.show()
```

No, I want to create my lovely RDDs

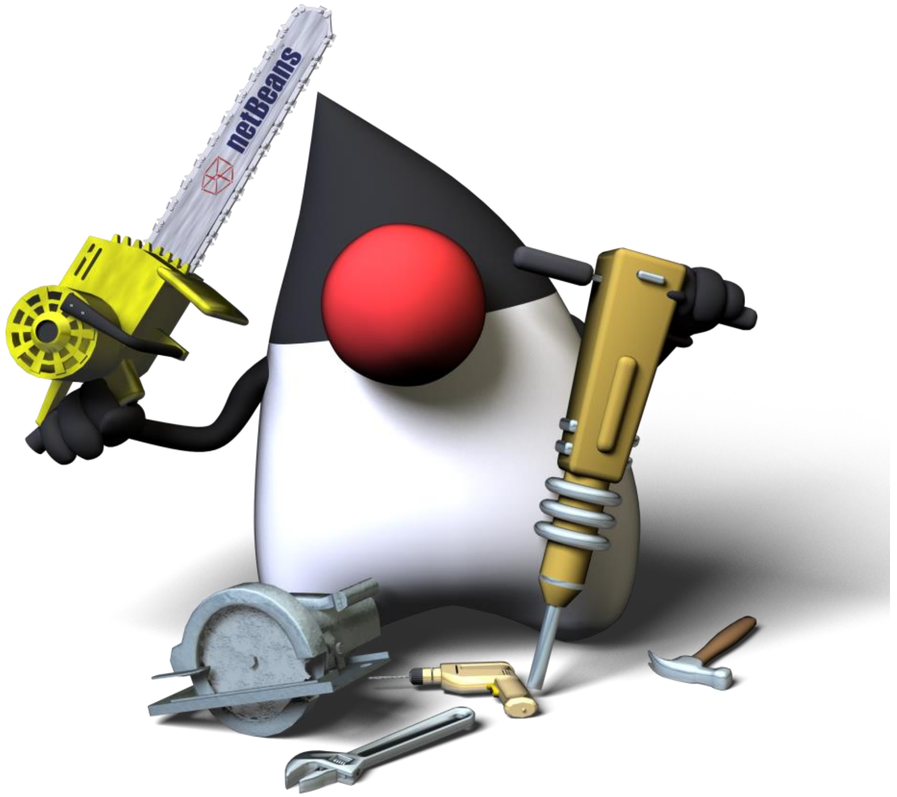


Where's parallelize() method?

RDD?

```
case class User(email: String, footSize: Long, name: String)


// DataFrame -> DataSet with Users

val userDS =

spark.read.json("/home/tmp/datasets/users.json").as[User]


userDS.map(_.name).collect()


userDS.filter(_.footSize > 38).collect()


ds.rdd // IF YOU REALLY WANT
```

Case #4 : Spark uses Java serialization A LOT



Two choices to distribute data across cluster

- **Java serialization**

By default with ObjectOutputStream

- **Kryo serialization**

Should register classes (no support of Serialazible)

The main problem: overhead of serializing

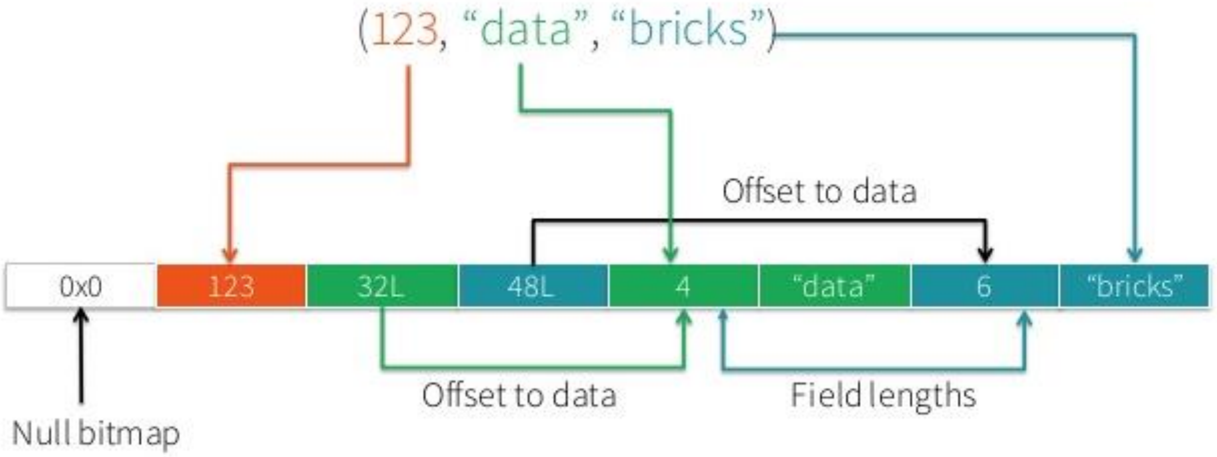
Each serialized object contains the class structure as well as the values

The main problem: overhead of serializing

Each serialized object contains the class structure as well as the values

Don’t forget about GC

Tungsten Compact Encoding



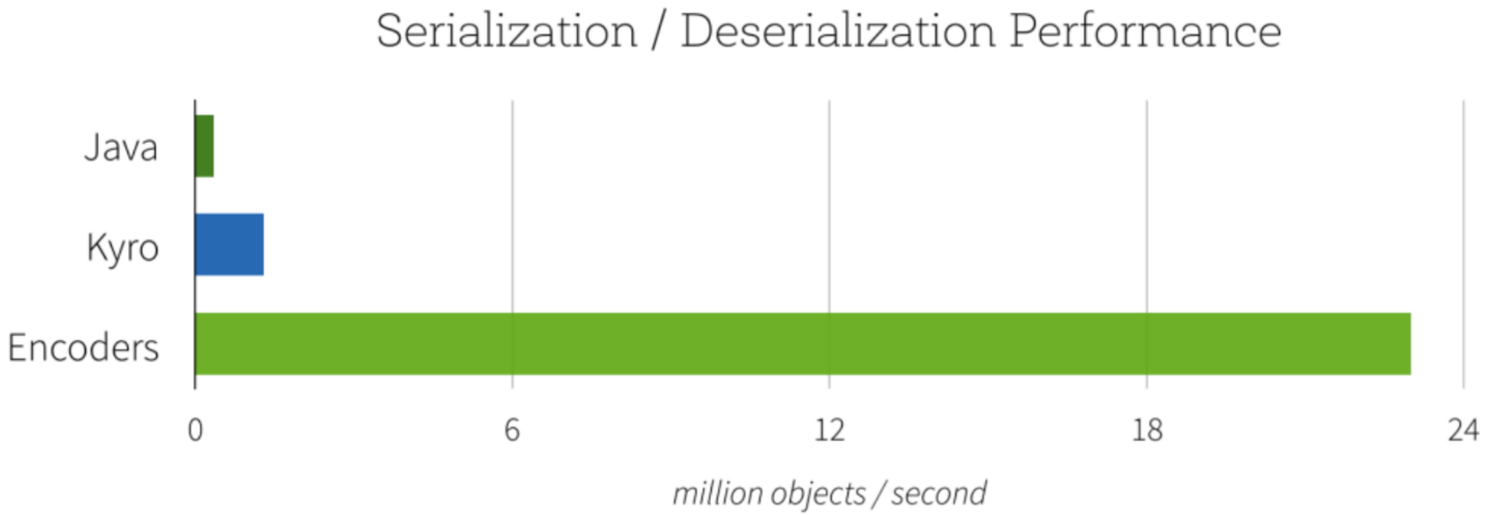
Encoder’s concept

Generate bytecode to interact with off-heap

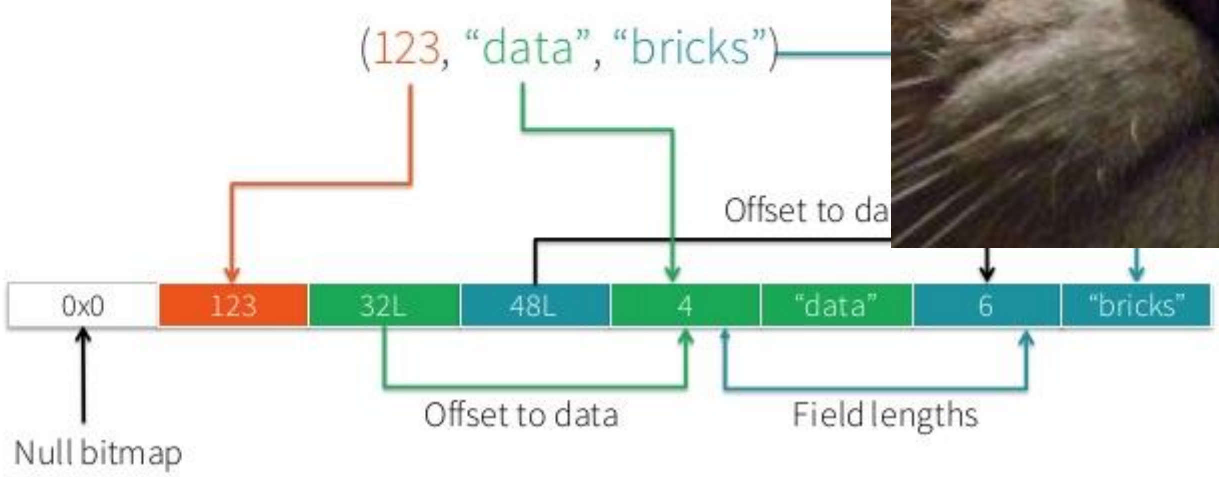
&

Give access to attributes without ser/deser

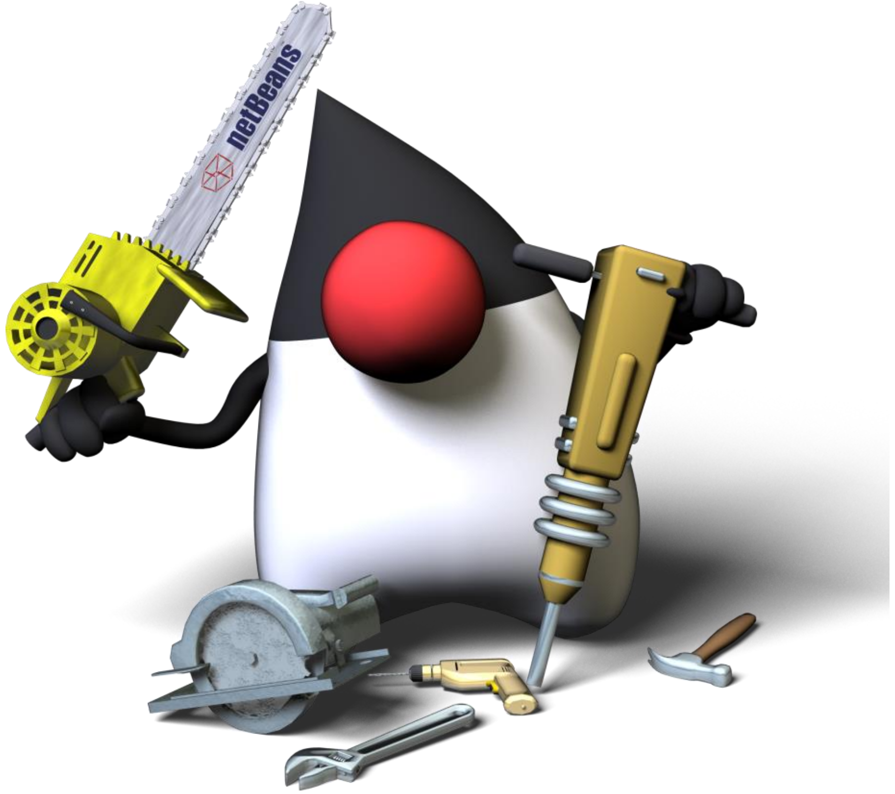
Encoders


Serialization / Deserialization Performance
Java
Kyro
Encoders
0
6
12
18
24
million objects / second

No custom encoders



Case #5 : Not enough storage levels ☺



Caching in Spark

- Frequently used RDD can be stored in memory
- One method, one short-cut: persist(), cache()
- SparkContext keeps track of cached RDD
- Serialized or deserialized Java objects

Full list of options

- MEMORY_ONLY
- MEMORY_AND_DISK
- MEMORY_ONLY_SER
- MEMORY_AND_DISK_SER
- DISK_ONLY
- MEMORY_ONLY_2, MEMORY_AND_DISK_2

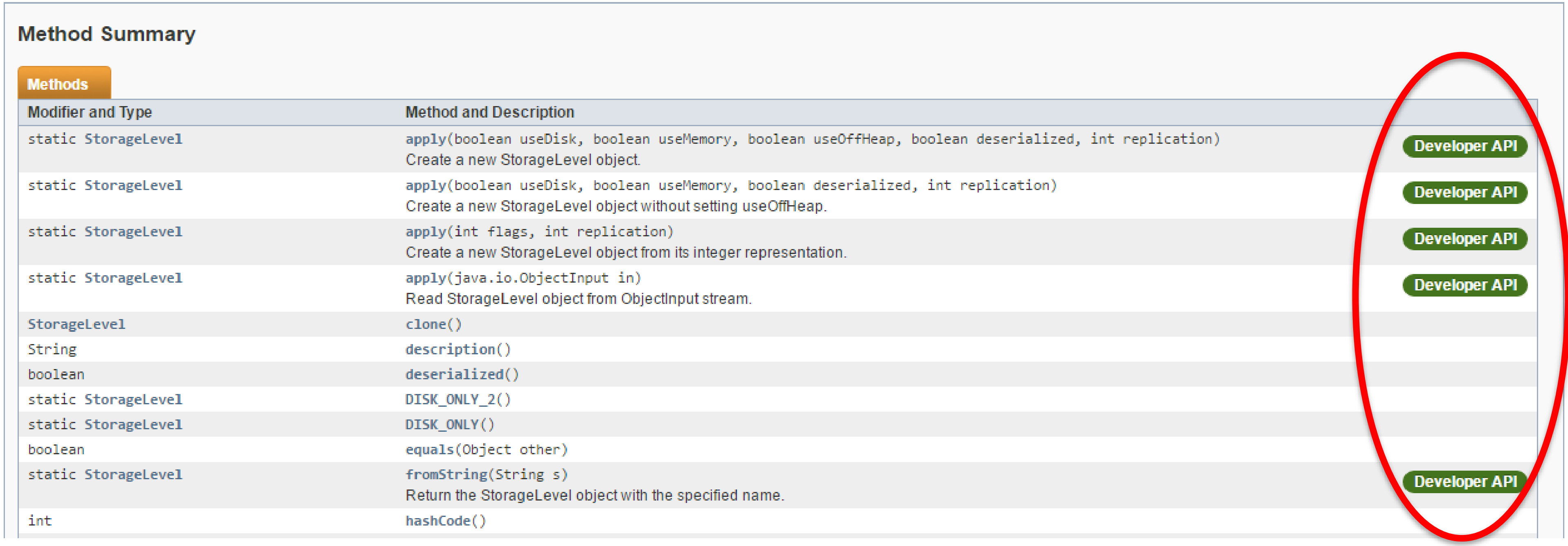
Spark Core Storage Level

- MEMORY_ONLY (default for Spark Core)
- MEMORY_AND_DISK
- MEMORY_ONLY_SER
- MEMORY_AND_DISK_SER
- DISK_ONLY
- MEMORY_ONLY_2, MEMORY_AND_DISK_2

Spark Streaming Storage Level

- MEMORY_ONLY (default for Spark Core)
- MEMORY_AND_DISK
- MEMORY_ONLY_SER (default for Spark Streaming)
- MEMORY_AND_DISK_SER
- DISK_ONLY
- MEMORY_ONLY_2, MEMORY_AND_DISK_2

Developer API to make new Storage Levels



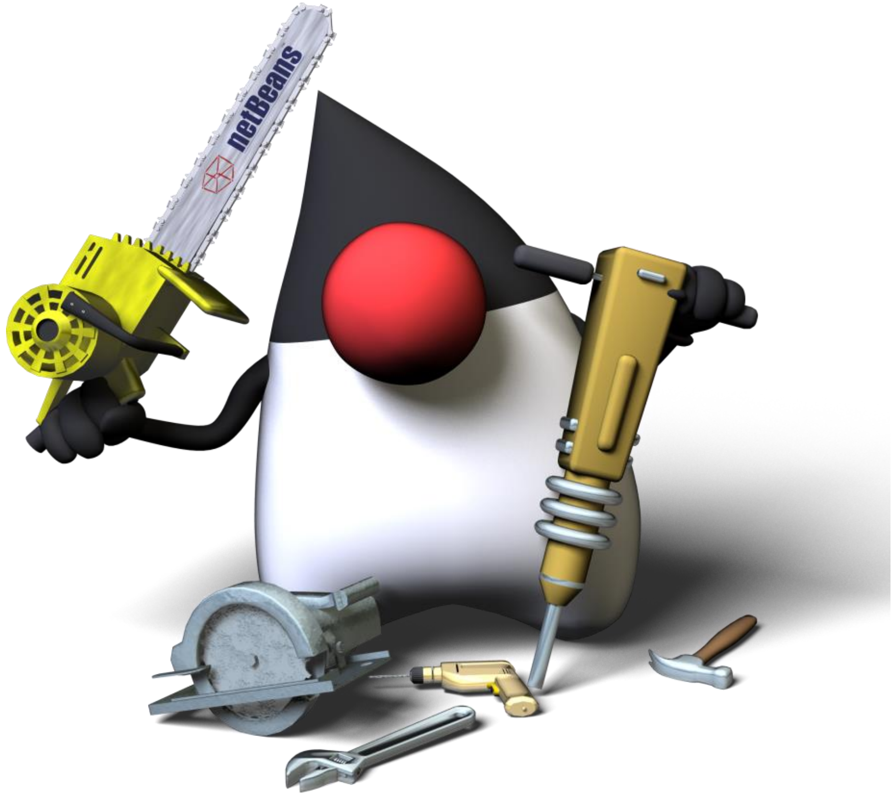
Method Summary

Methods

Modifier and Type	Method and Description	
static StorageLevel	`apply(boolean useDisk, boolean useMemory, boolean useOffHeap, boolean deserialized, int replication)` Create a new StorageLevel object.	Developer API
static StorageLevel	`apply(boolean useDisk, boolean useMemory, boolean deserialized, int replication)` Create a new StorageLevel object without setting useOffHeap.	Developer API
static StorageLevel	`apply(int flags, int replication)` Create a new StorageLevel object from its integer representation.	Developer API
static StorageLevel	`apply(java.io.ObjectInput in)` Read StorageLevel object from ObjectInput stream.	Developer API
StorageLevel	`clone()`	
String	`description()`	
boolean	`deserialized()`	
static StorageLevel	`DISK_ONLY_2()`	
static StorageLevel	`DISK_ONLY()`	
boolean	`equals(Object other)`	
static StorageLevel	`fromString(String s)` Return the StorageLevel object with the specified name.	Developer API
int	`hashCode()`	

What’s the most popular file format in BigData?

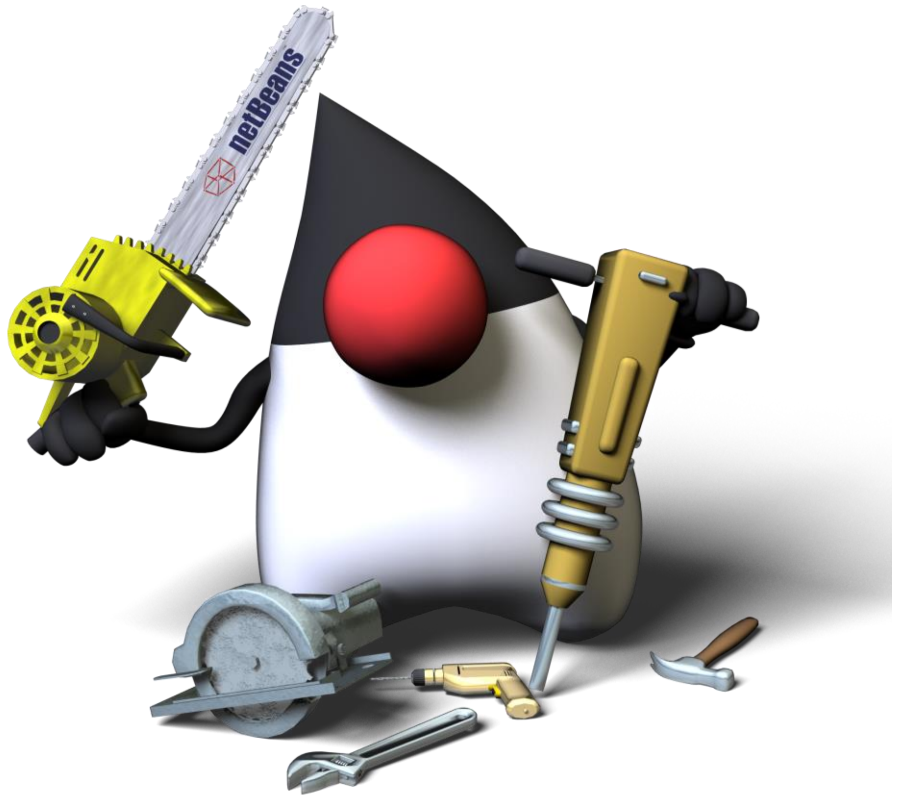
Case #6 : External libraries to read CSV



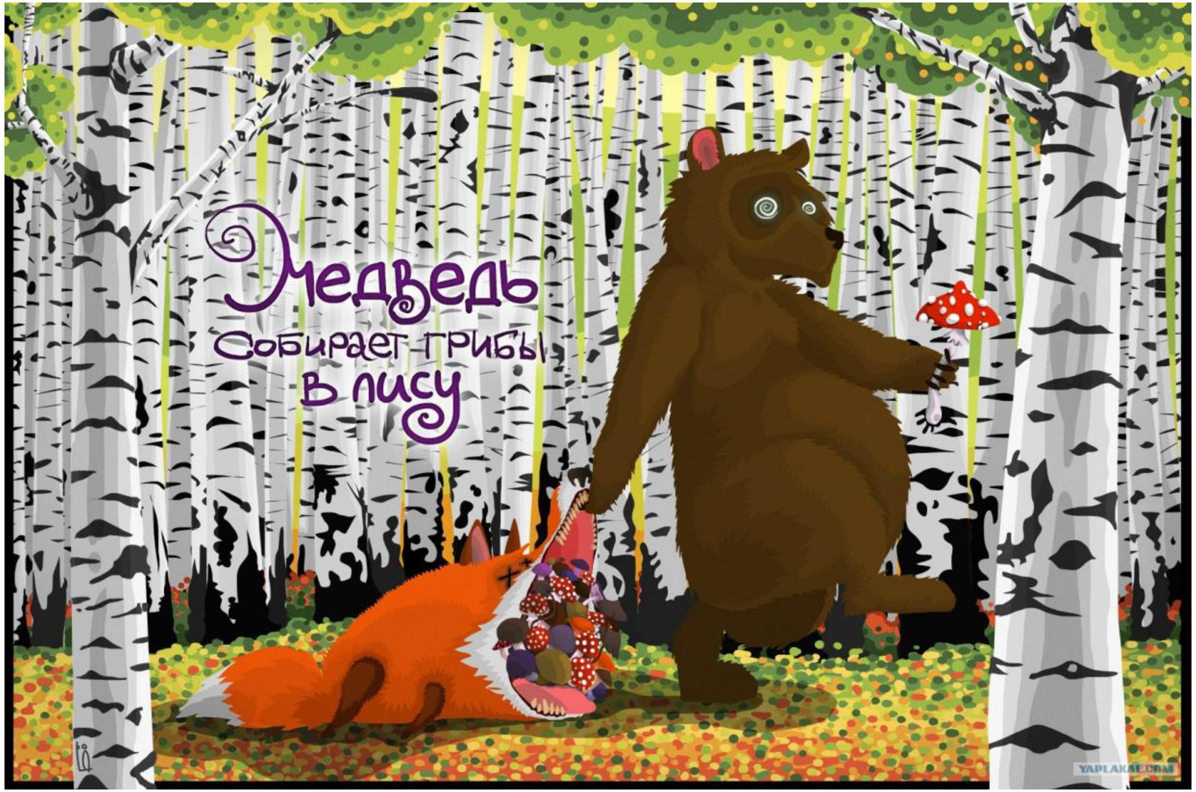
Easy to read CSV

```
data = sqlContext.read.format("csv")
.option("header", "true")
.option("inferSchema", "true")
.load("/datasets/samples/users.csv")


data.cache()


data.createOrReplaceTempView(“users")


display(data)
```

Case #7 : How to measure Spark performance?



You'd measure performance!



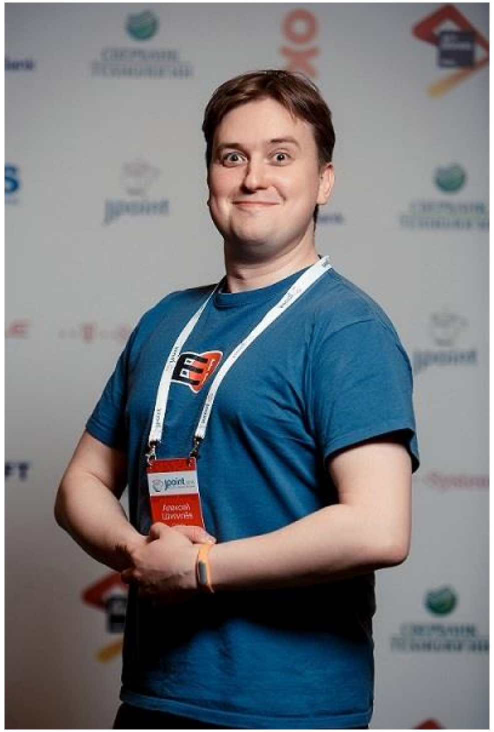
TPCDS

99 Queries

http://bit.ly/2dObMsH

```
              |       v1. cc_name = v1_lead. cc_name and
              |       v1.rn = v1_lag.rn + 1 and
              |       v1.rn = v1_lead.rn - 1)
              | select * from v2
              | where  d_year = 1999 and
              |        avg_monthly_sales > 0 and
              |        case when avg_monthly_sales > 0 then abs(sum_sales - avg_monthly_sales) / avg_monthly_sales else null end > 0.1
              | order by sum_sales - avg_monthly_sales, 3
              | limit 100
              """.stripMargin),
      ("q58", """
              | with ss_items as
              | (select i_item_id item_id, sum(ss_ext_sales_price) ss_item_rev
              | from store_sales, item, date_dim
              | where ss_item_sk = i_item_sk
              |   and d_date in (select d_date
              |                  from date_dim
              |                  where d_week_seq = (select d_week_seq
              |                                      from date_dim
              |                                      where d_date = '2000-01-03'))
              |   and ss_sold_date_sk   = d_date_sk
              | group by i_item_id),
              | cs_items as
              | (select i_item_id item_id
              |        ,sum(cs_ext_sales_price) cs_item_rev
              |  from catalog_sales, item, date_dim
              | where cs_item_sk = i_item_sk
              |  and  d_date in (select d_date
              |                  from date_dim
              |                  where d_week_seq = (select d_week_seq
              |                                      from date_dim
              |                                      where d_date = '2000-01-03'))
```

How to benchmark Spark



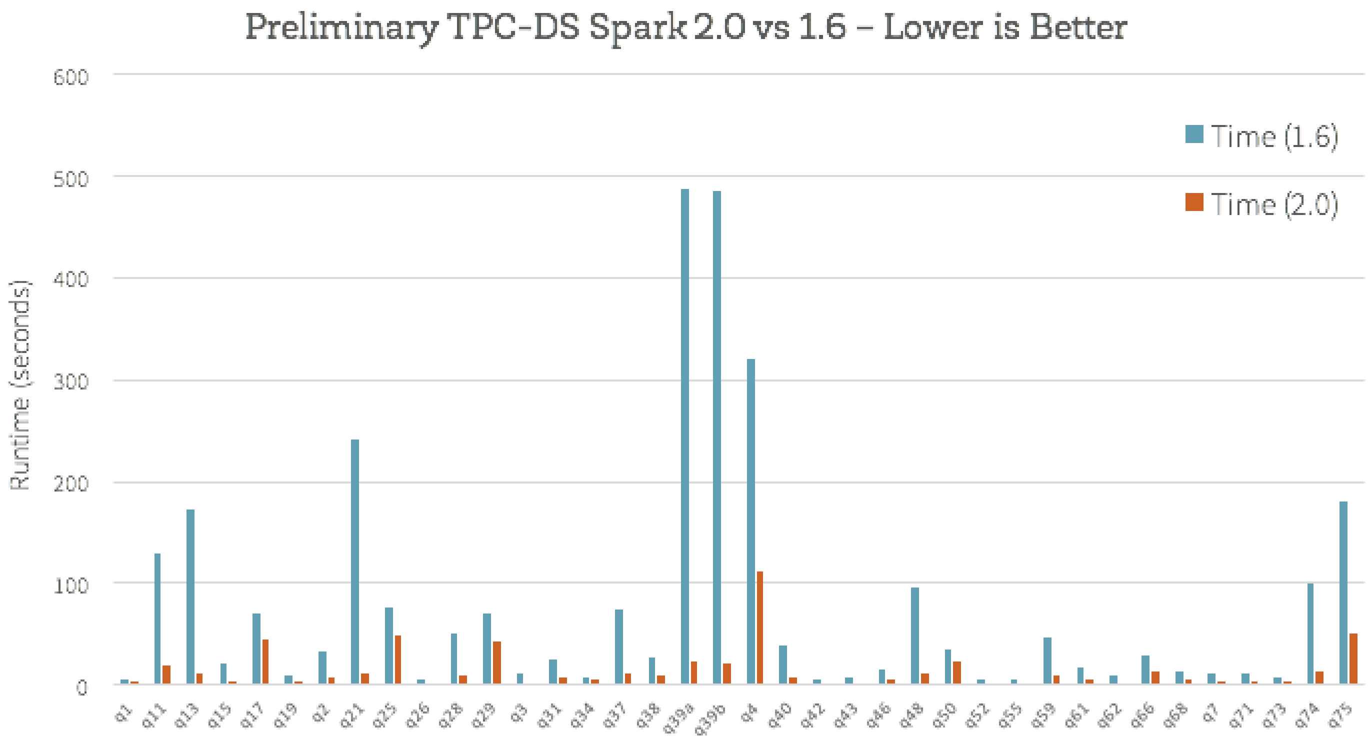
+ Spark = ?

Special Tool from Databricks

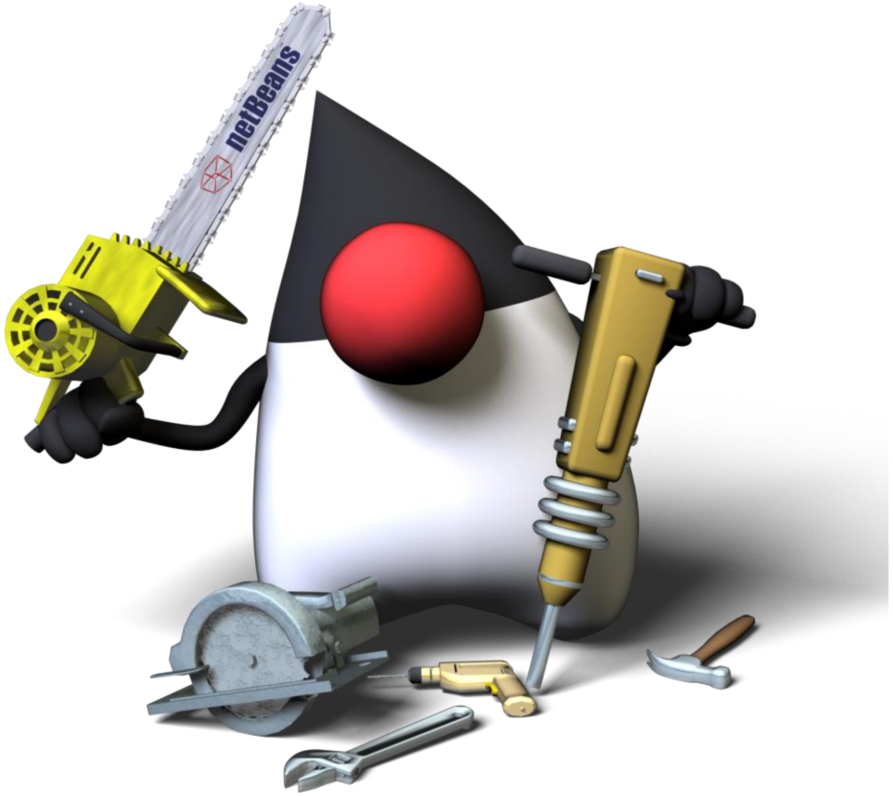
Benchmark Tool for SparkSQL

https://github.com/databricks/spark-sql-perf

Spark 2 vs Spark 1.6

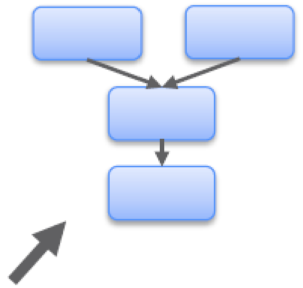


Case #8 : What’s faster: SQL or DataSet API?



Job Stages in old Spark

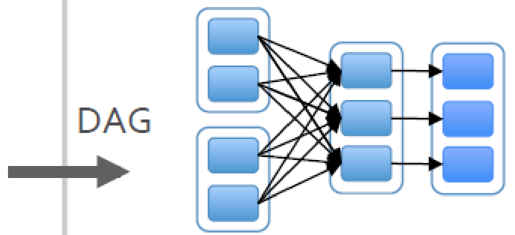
RDD Objects



```
rdd1.join(rdd2)
    .groupBy(…)
    .filter(…)
    .count()
```

build operator DAG

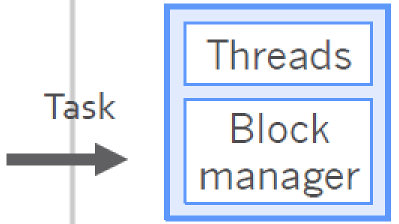
Scheduler (DAGScheduler)



split graph into *stages* of tasks

submit each stage as ready

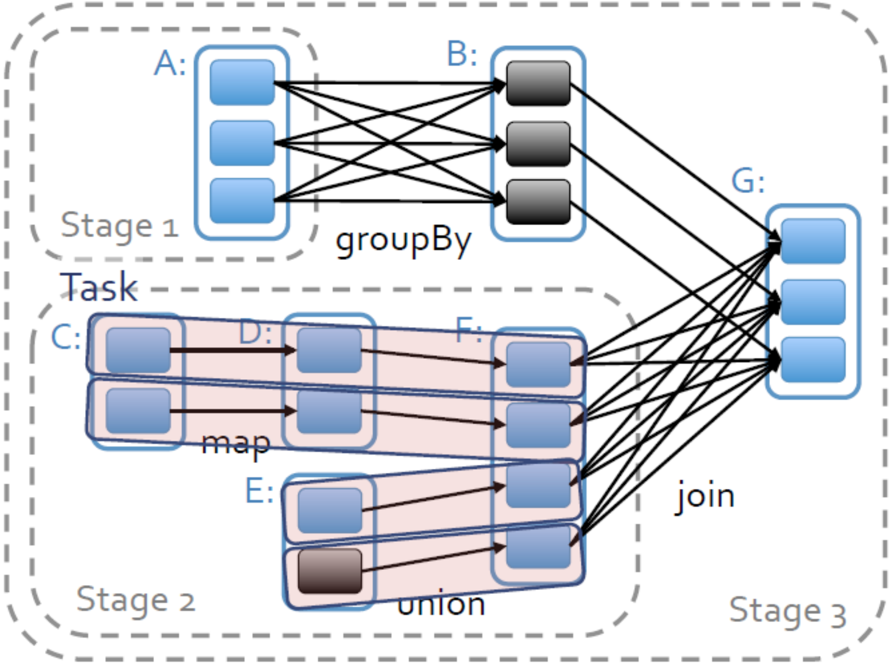
Executors



execute tasks

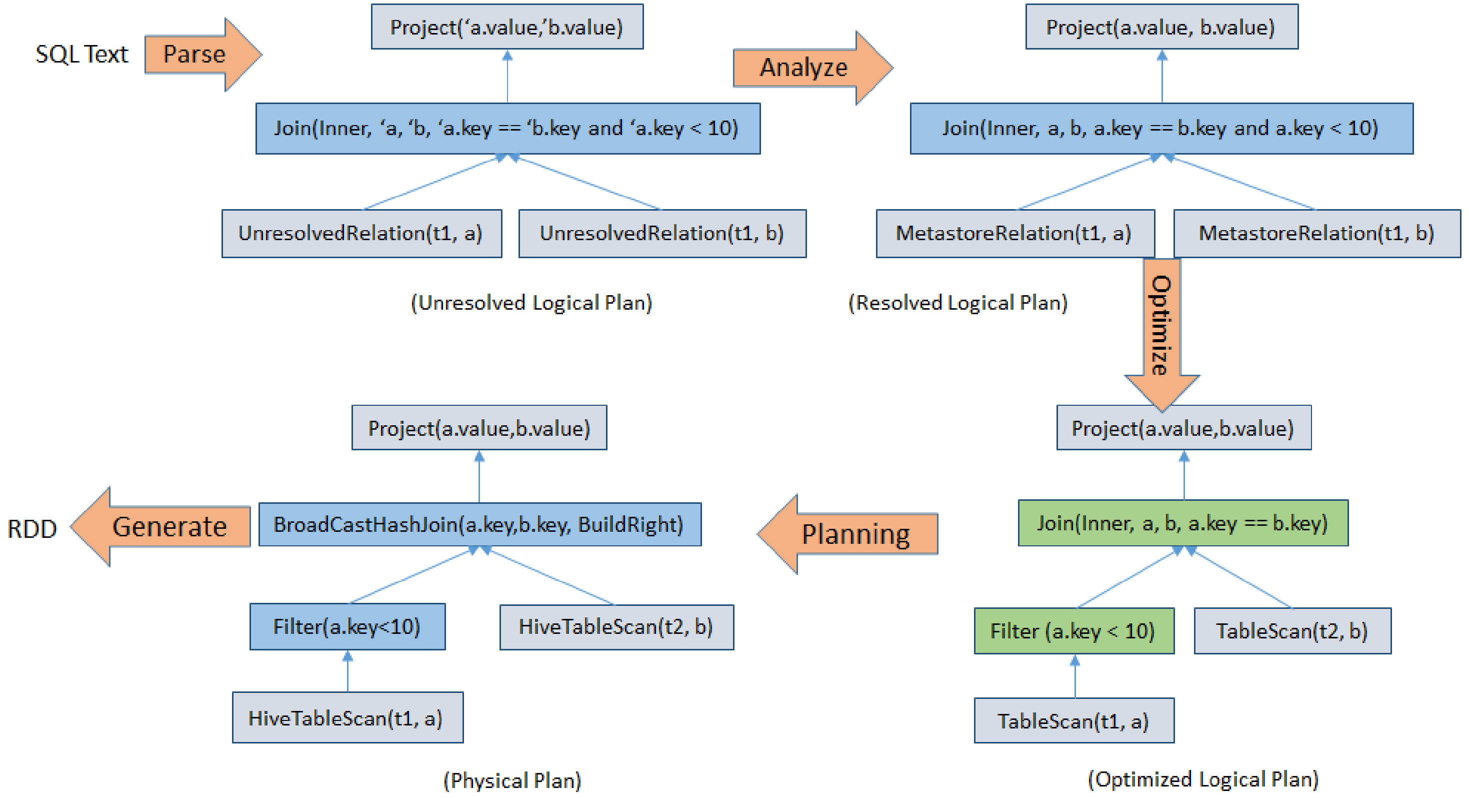
store and serve blocks

Scheduler Optimizations

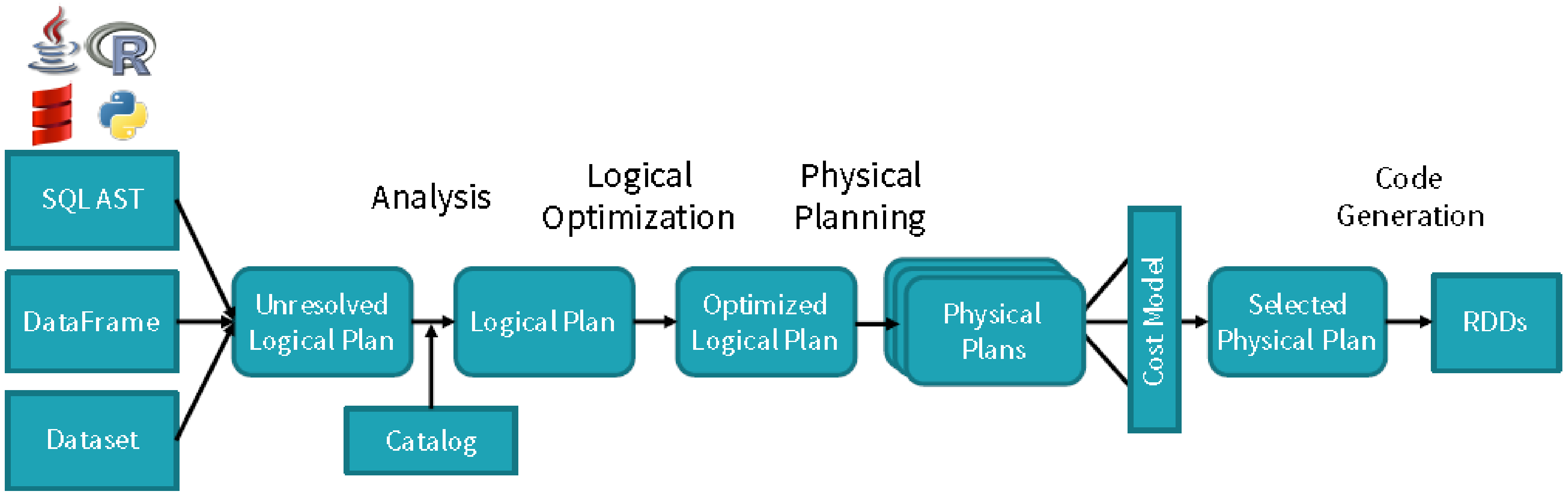


■ = previously computed partition

Catalyst Optimizer for DataFrames



Unified Logical Plan



Bytecode

DataFrame Code / SQL

```
df.where(df("year") > 2015)
```

Catalyst Expressions

```
GreaterThan(year#234, Literal(2015))
```

Low-level bytecode

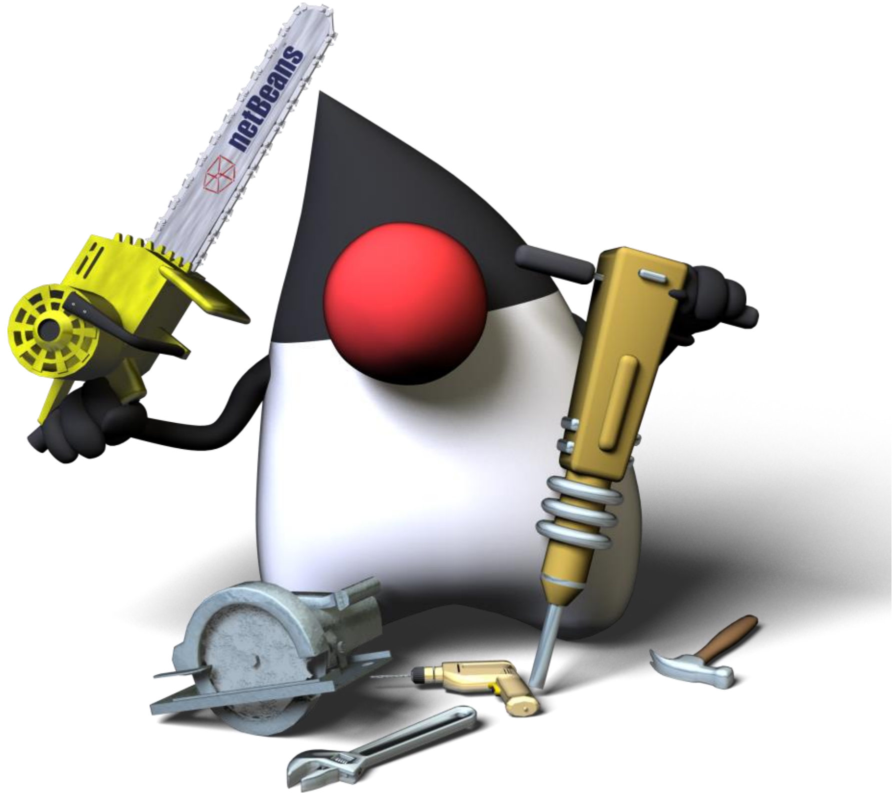
```
bool filter(Object baseObject) {
   int offset = baseOffset + bitSetWidthInBytes + 3*8L;
   int value = Platform.getInt(baseObject, offset);
   return value34 > 2015;
}
```

JVM *intrinsic* JIT-ed to pointer arithmetic

DataSet.explain()

```
== Physical Plan ==
Project [avg(price)#43,carat#45]
+- SortMergeJoin [color#21], [color#47]
   :- Sort [color#21 ASC], false, 0
   :  +- TungstenExchange hashpartitioning(color#21,200), None
   :     +- Project [avg(price)#43,color#21]
   :        +- TungstenAggregate(key=[cut#20,color#21], functions=[(avg(cast(price#25 as
bigint)),mode=Final,isDistinct=false)], output=[color#21,avg(price)#43])
   :           +- TungstenExchange hashpartitioning(cut#20,color#21,200), None
   :              +- TungstenAggregate(key=[cut#20,color#21],
functions=[(avg(cast(price#25 as bigint)),mode=Partial,isDistinct=false)],
output=[cut#20,color#21,sum#58,count#59L])
   :                 +- Scan CsvRelation(-----)
   +- Sort [color#47 ASC], false, 0
      +- TungstenExchange hashpartitioning(color#47,200), None
         +- ConvertToUnsafe
            +- Scan CsvRelation(----)
```

Case #9 : Why does explain() show so many Tungsten things?

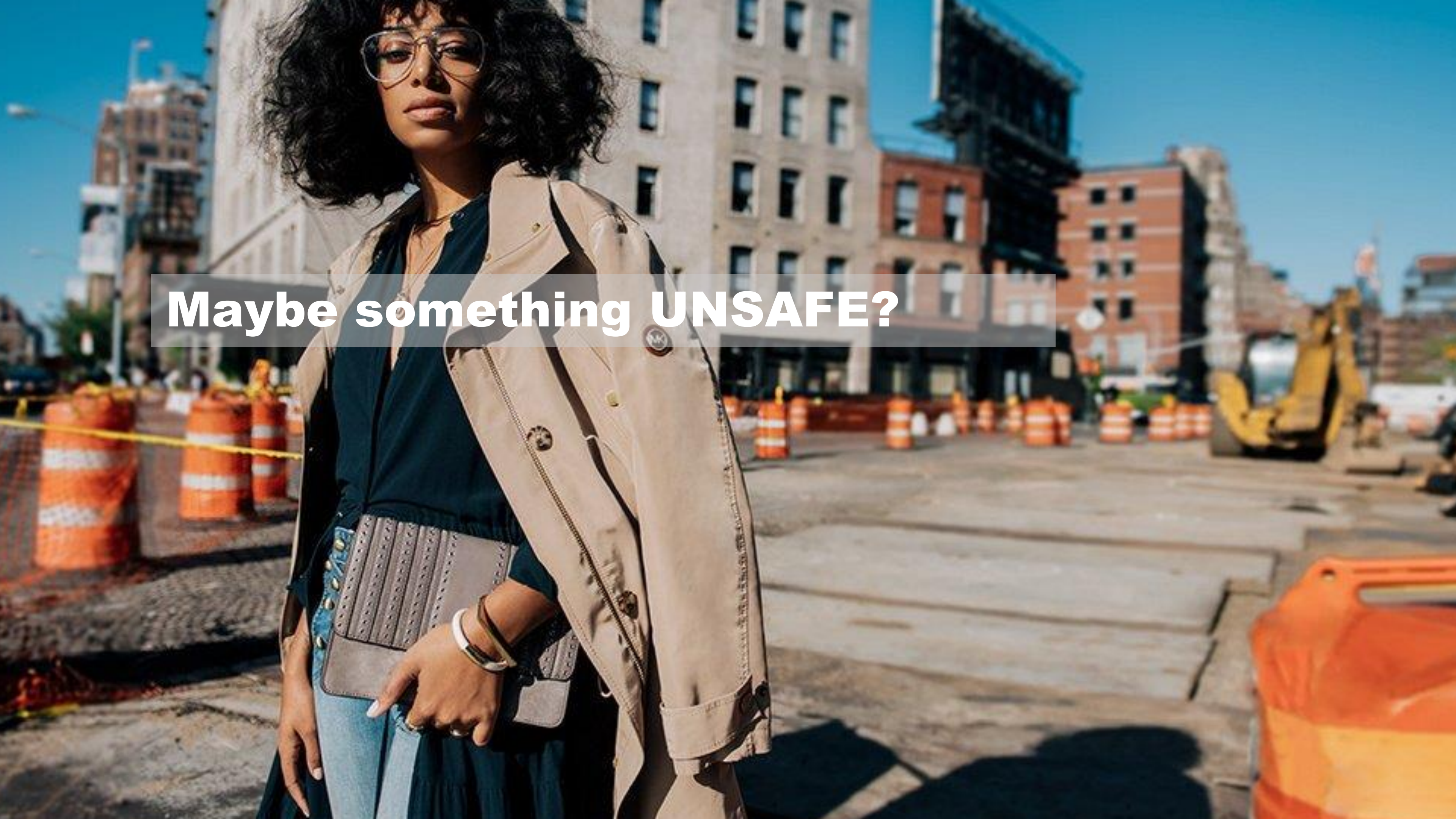


How to be effective with CPU

- Runtime code generation
- Exploiting cache locality
- Off-heap memory management

Tungsten’s goal

Push performance closer to the limits of modern hardware

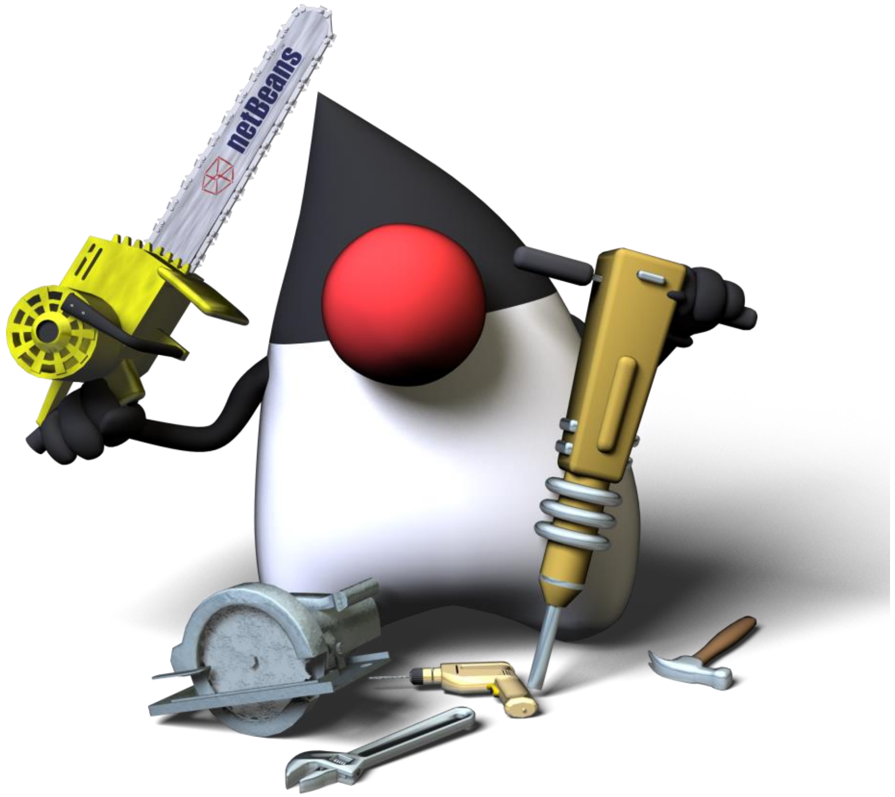


Maybe something UNSAFE?

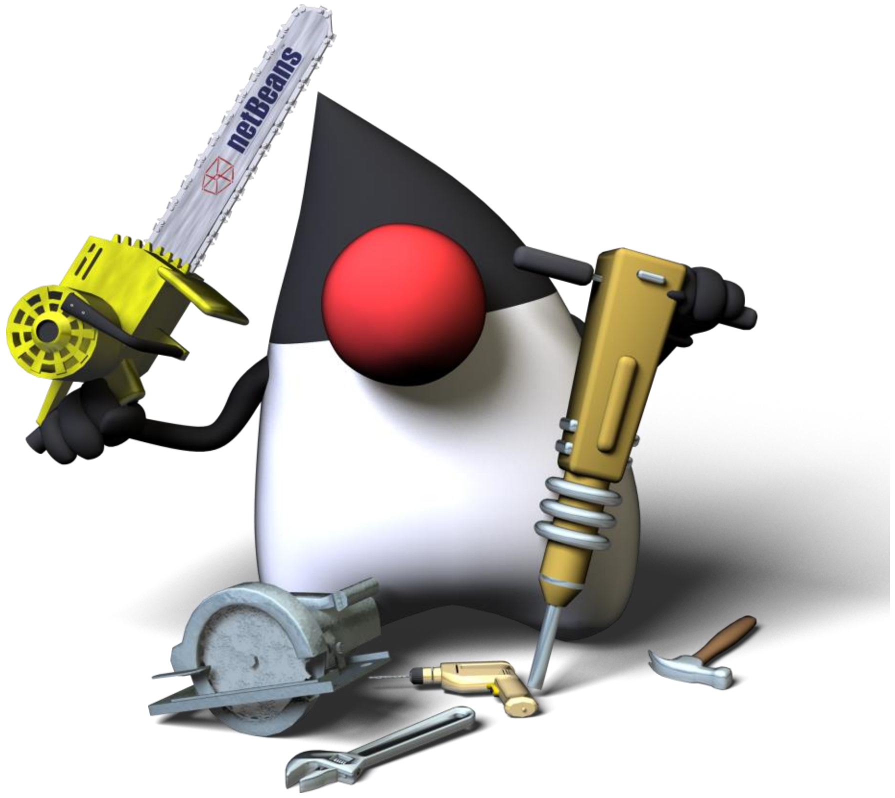
UnsafeRowFormat☺

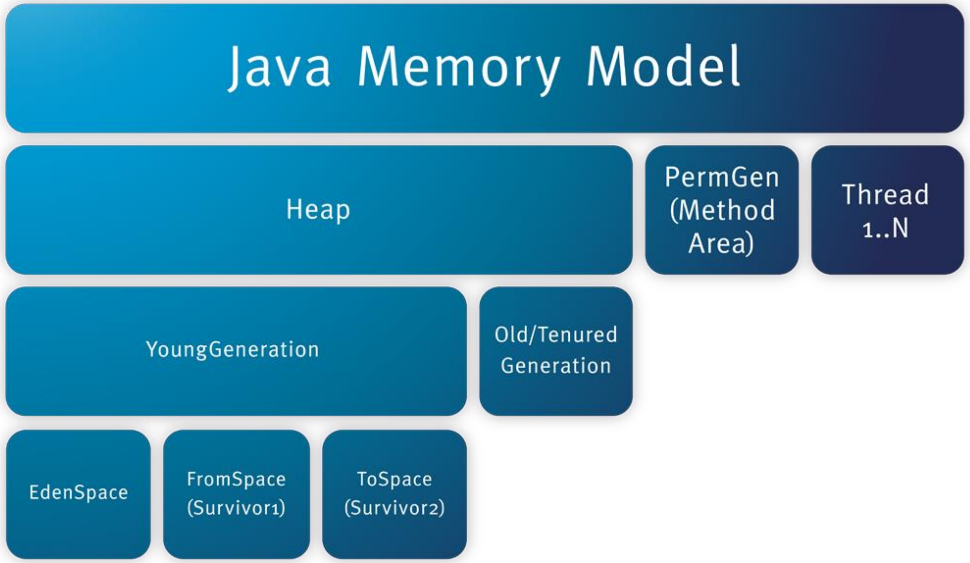
- Bit set for tracking null values
- Small values are inlined
- For variable-length values are stored relative offset into the variablelength data section
- Rows are always 8-byte word aligned
- Equality comparison and hashing can be performed on raw bytes without requiring additional interpretation

Case #10 : Can I influence on Memory Management in Spark?

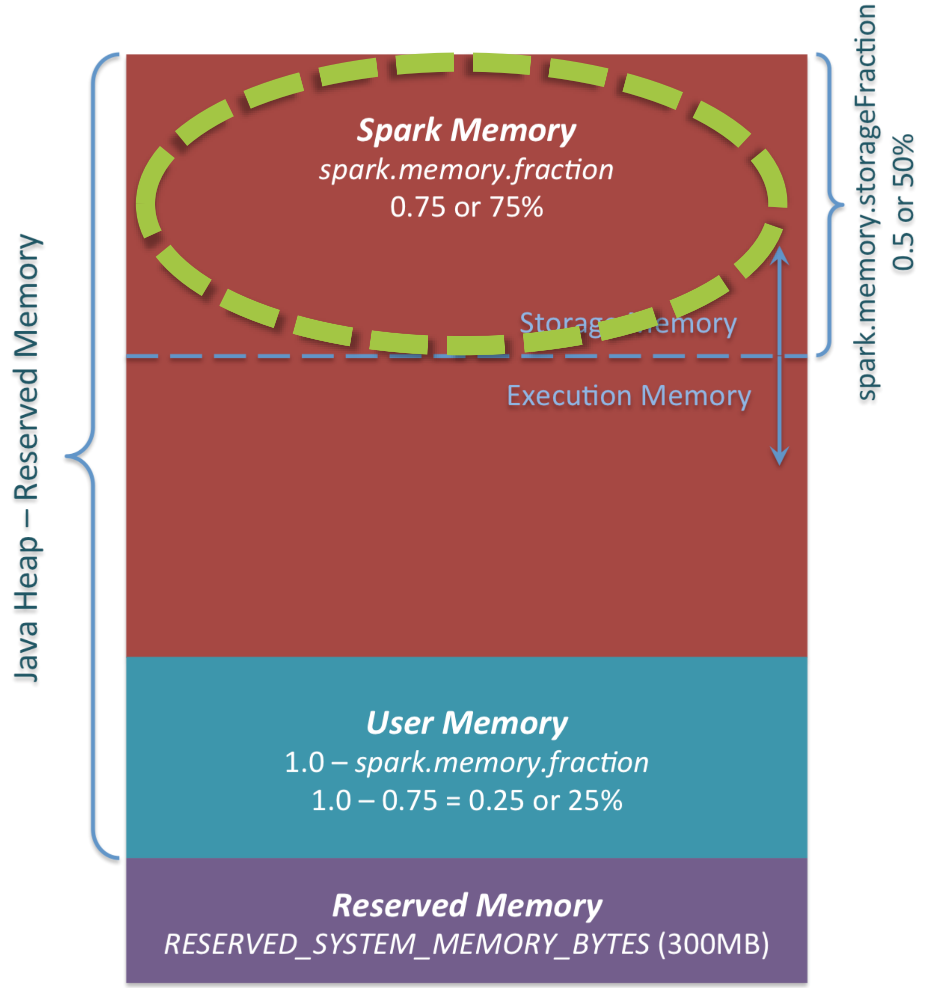


Case #11 : Should I tune generation's stuff?



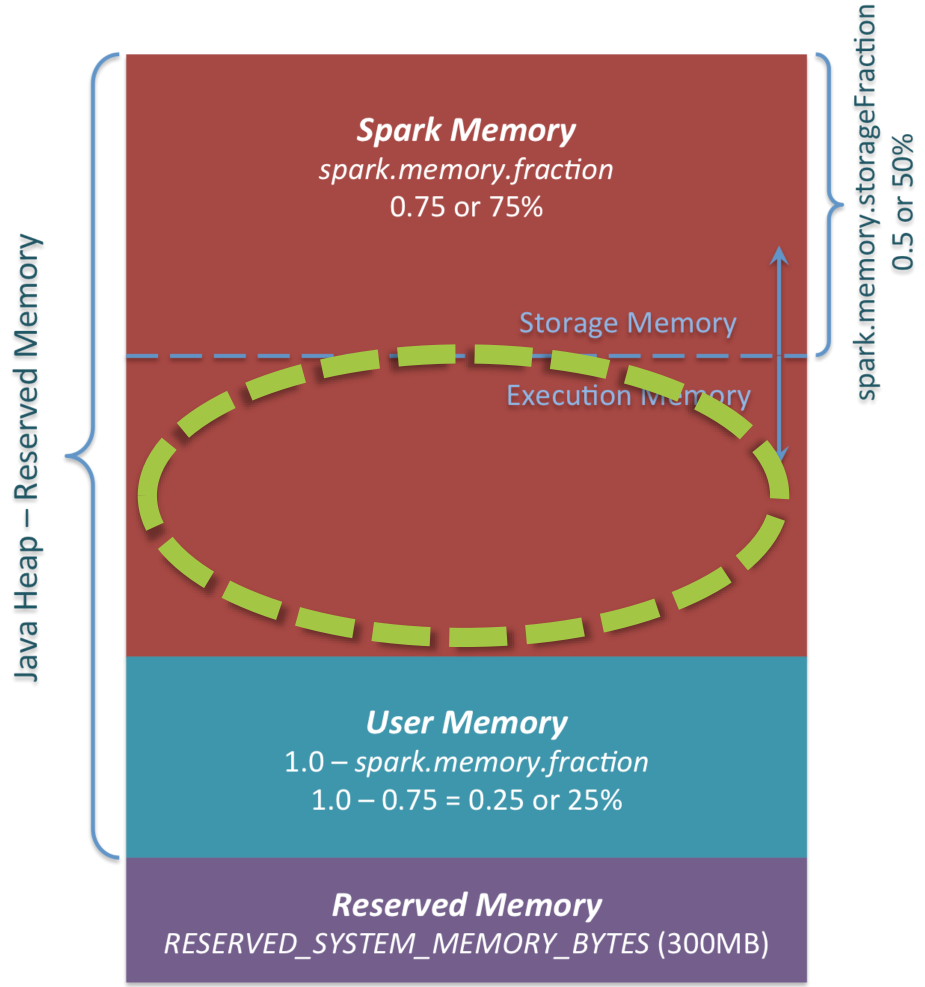


Cached Data

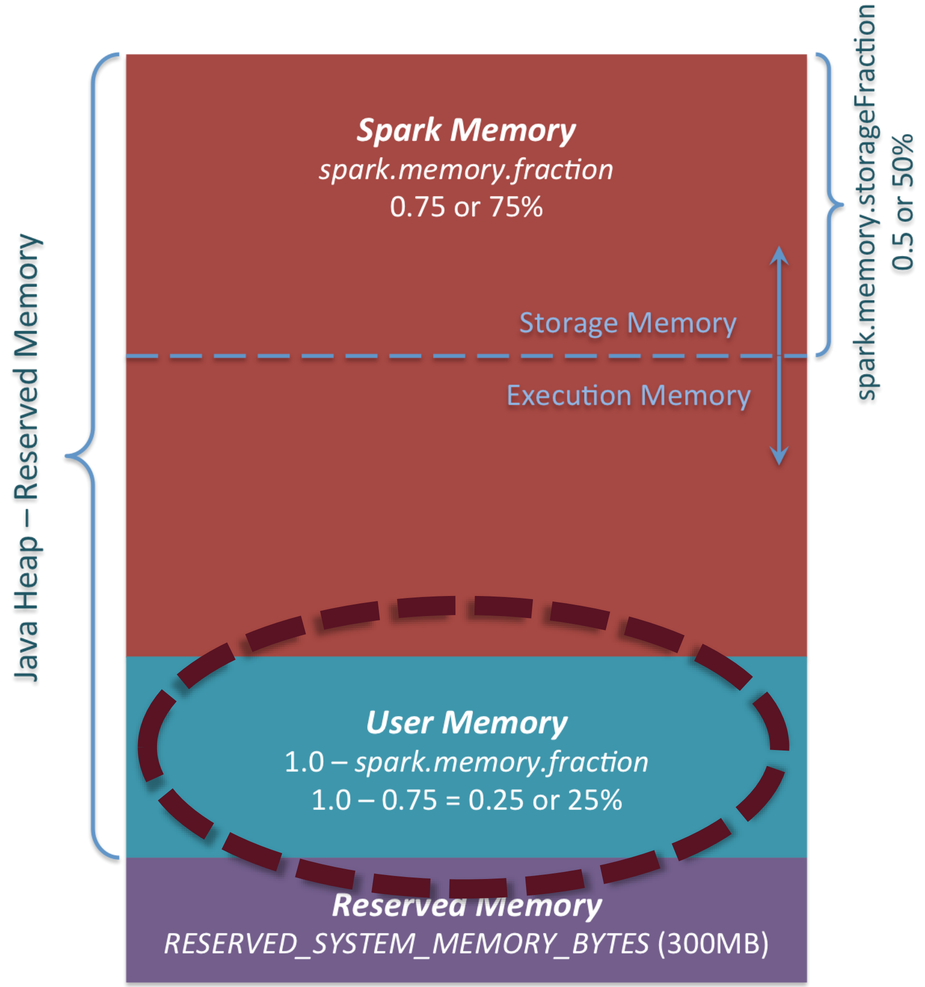


spark.memory.storageFraction
0.5 or 50%
Java Heap – Reserved Memory
Spark Memory
spark.memory.fraction
0.75 or 75%
Storage Memory
Execution Memory
User Memory
1.0 – spark.memory.fraction
1.0 – 0.75 = 0.25 or 25%
Reserved Memory
RESERVED_SYSTEM_MEMORY_BYTES (300MB)

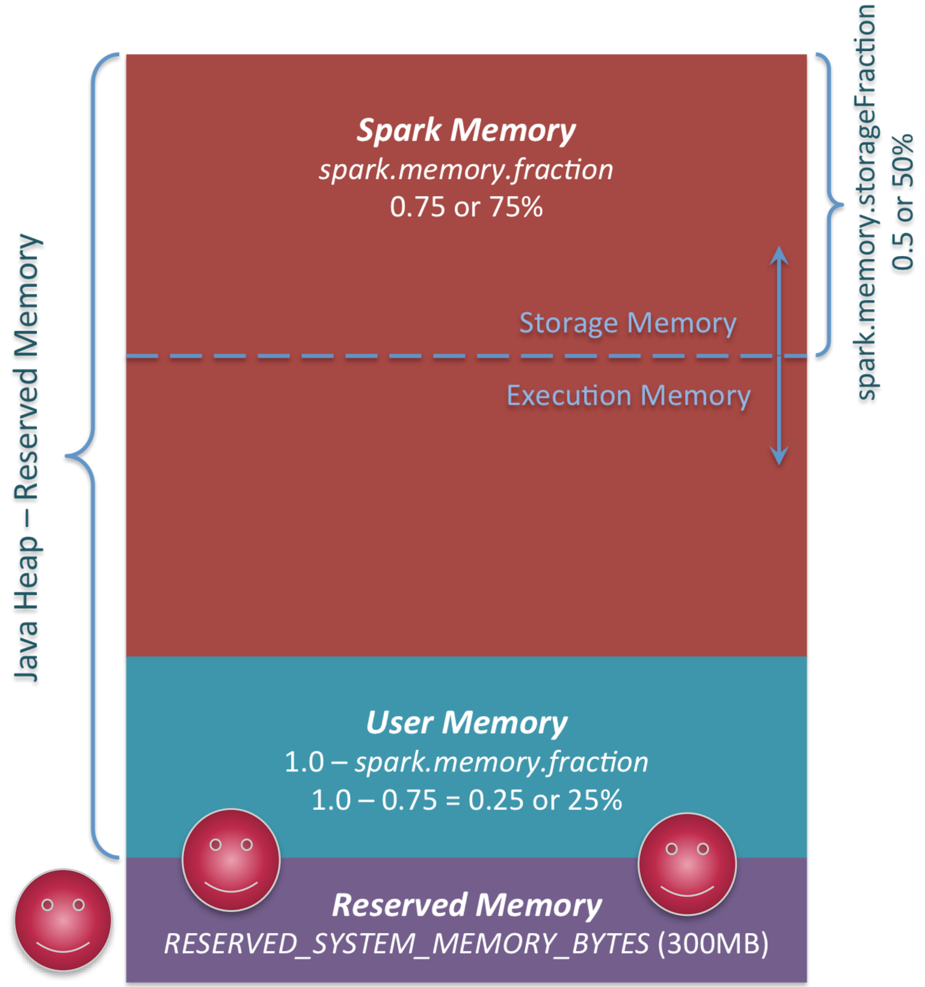
During operations



For your needs



For Dark Lord



IN CONCLUSION

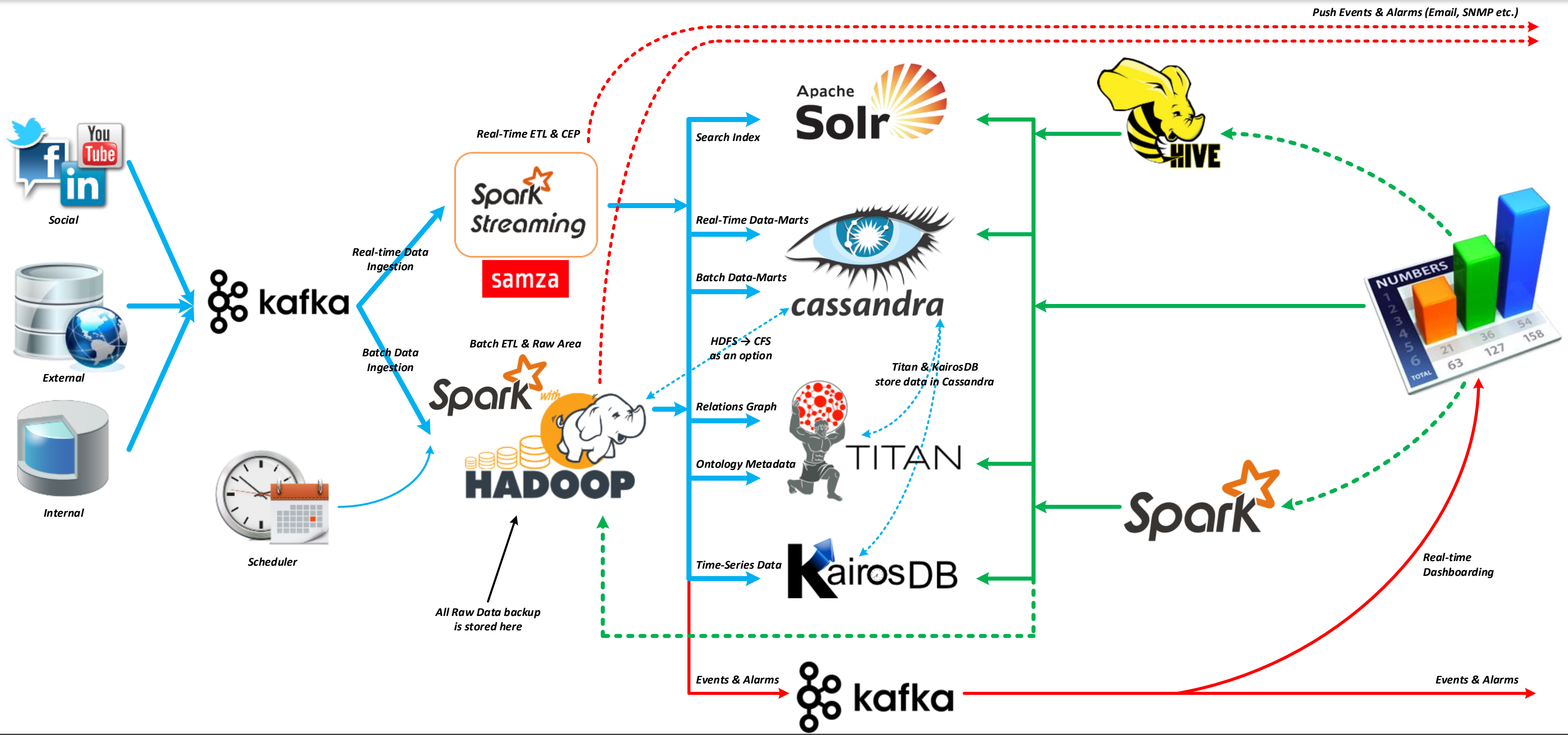
We have no ability...

- join structured streaming and other sources to handle it
- one unified ML API
- GraphX rethinking and redesign
- Custom encoders
- Datasets everywhere
- integrate with something important

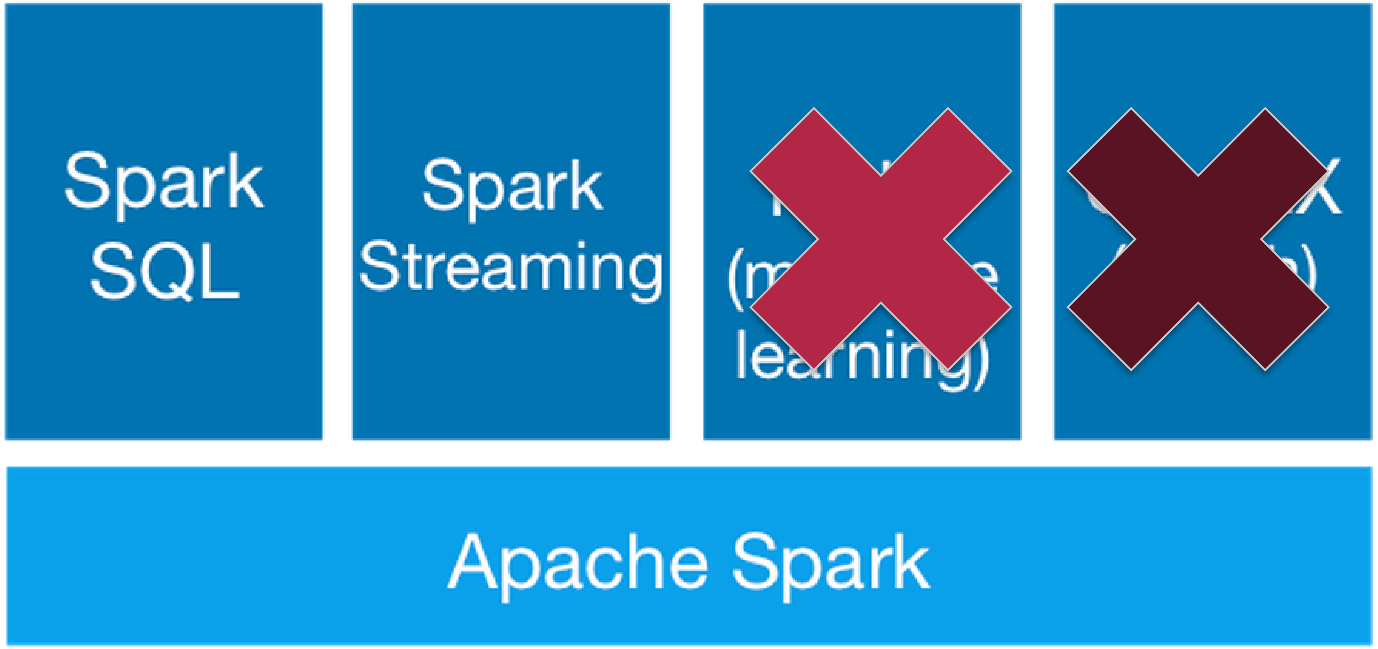
Roadmap

- Support other data sources (not only S3 + HDFS)
- Transactional updates
- Dataset is one DSL for all operations
- GraphFrames + Structured MLLib
- Tungsten: custom encoders
- The RDD-based API is expected to be removed in Spark

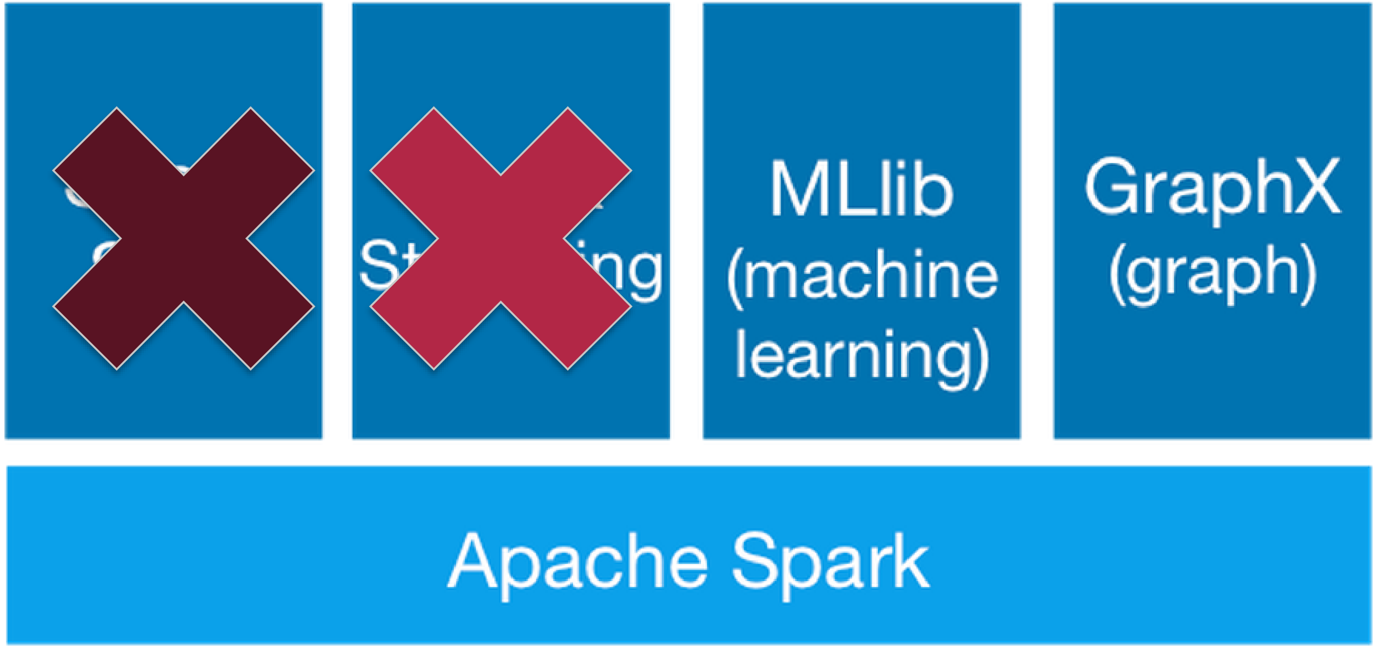
And we can DO IT!



First Part



Second Part



Contacts

E-mail : Alexey_Zinovyev@epam.com

Twitter : @zaleslaw @BigDataRussia

Facebook: https://www.facebook.com/zaleslaw

vk.com/big_data_russia **Big Data Russia**

vk.com/java_jvm **Java & JVM langs**

Any questions?