Interacting with Distributed Data from R using SparkR

Hossein Falaki
@mhfalaki

Outline

- SparkR Architecture & API
  - Data distribution
  - R and JVM RPC
- Distributing R functions
- Use cases and best practices
  - Code & platform unification
  - Parallelizing existing logic
  - Distributed ML

SparkR Architecture & API

What is SparkR

An R package released with Apache Spark:

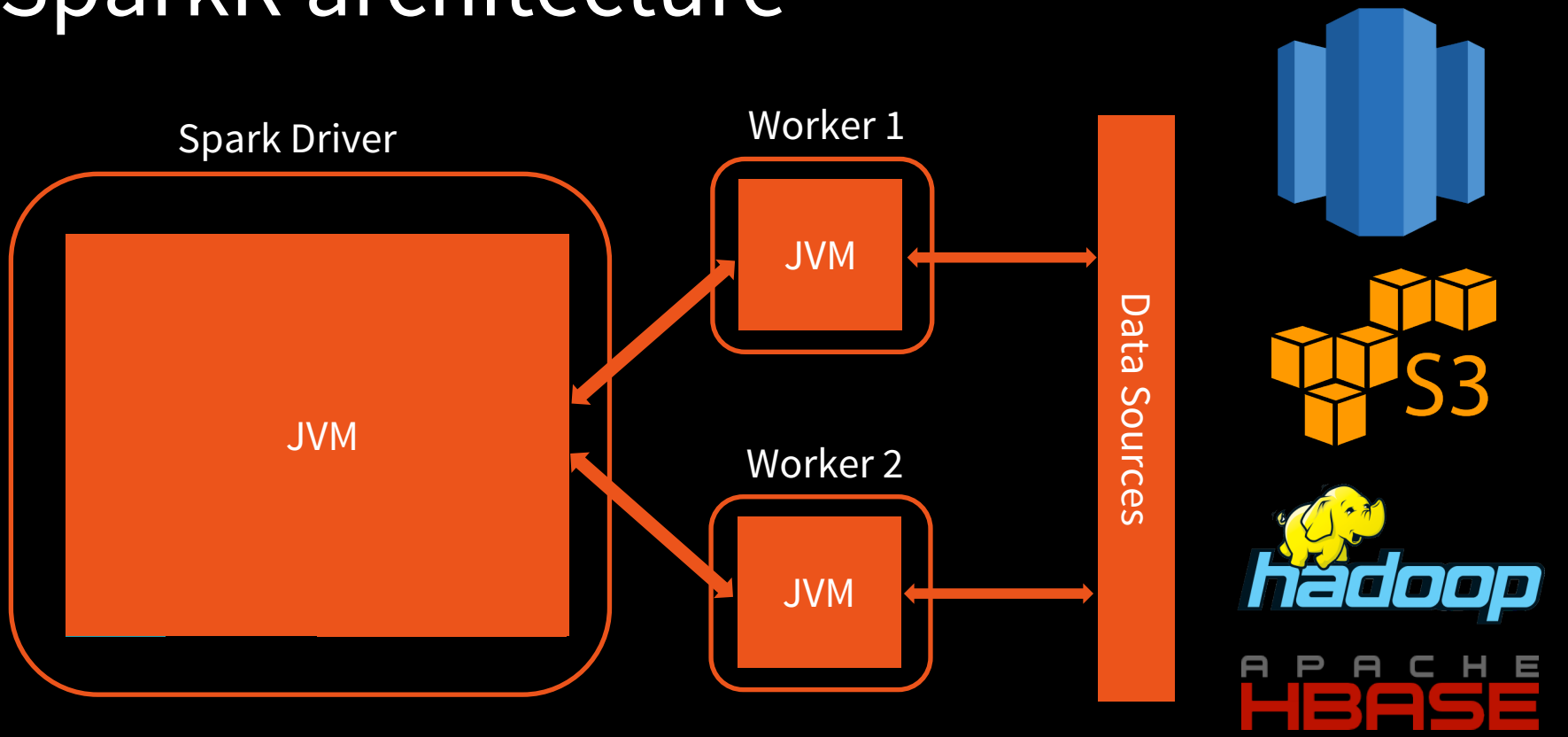
- Provides R front-end to Apache Spark
- Exposes Spark DataFrames (inspired by R and Pandas)
- Enables distributed execution of R functions

General distributed processing, data source, off-memory DataFrames

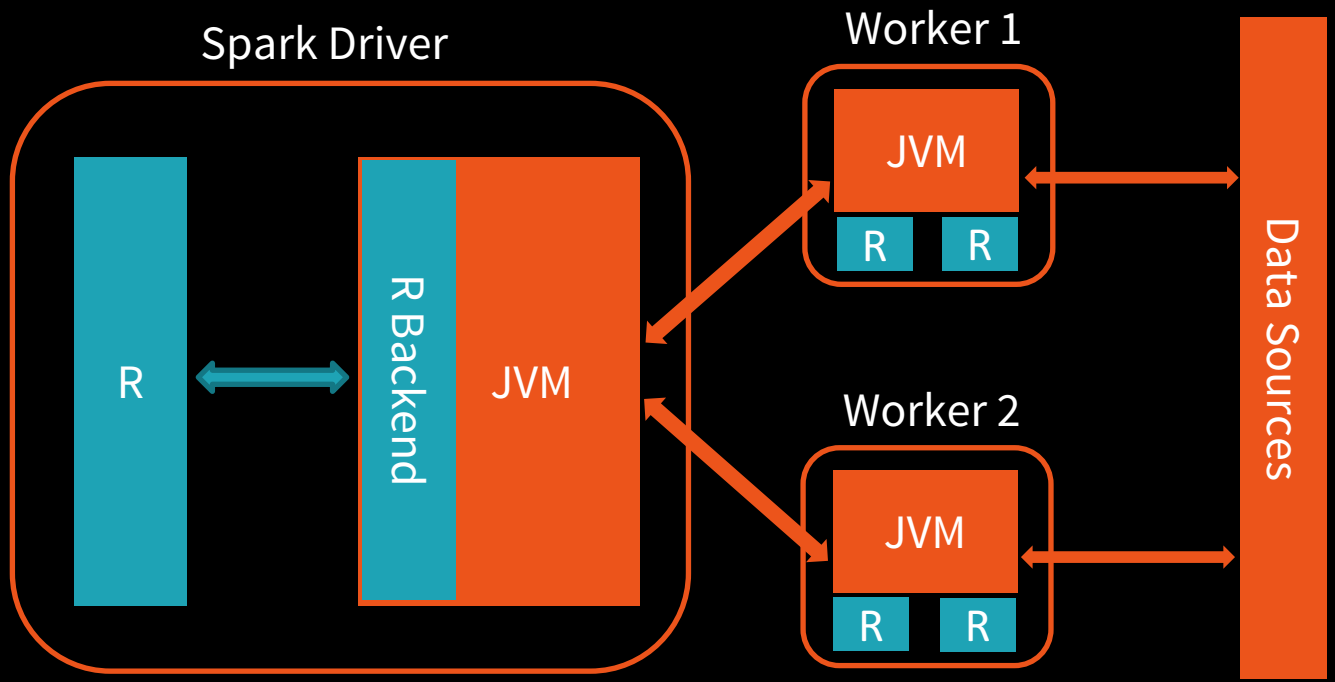
+

dynamic environment, interactivity, +10K packages, visualizations

SparkR architecture



SparkR architecture



Overview of SparkR API

http://spark.apache.org/docs/latest/api/R/

IO

read.df / write.df

createDataFrame / collect

Caching

cache / persist / unpersist

cacheTable / uncacheTable

SQL

sql / tableToDf

registerTempTable / tables

ML Lib

glm / kmeans / Naïve Bayes

survival regression / ...

DataFrame API

select / subset / groupBy

head / avg / nrow / dim

UDF functionality

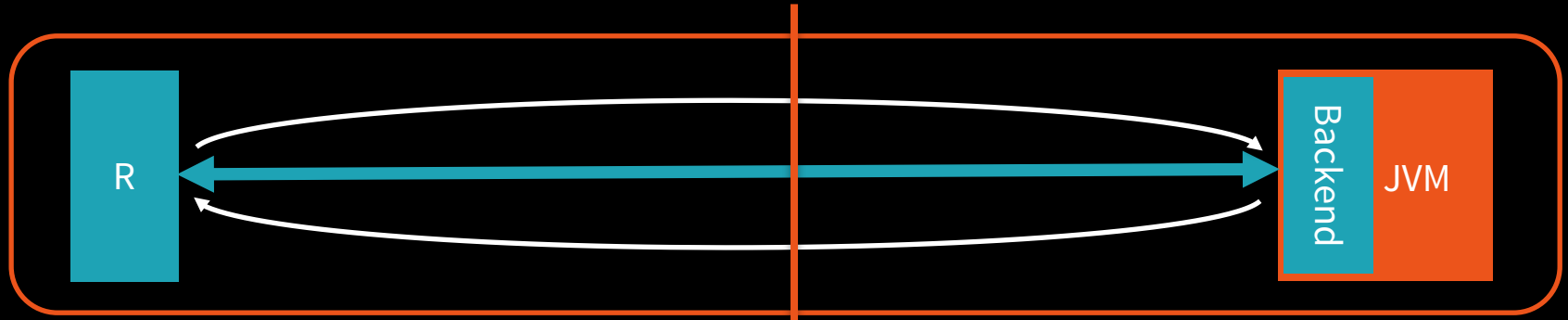
spark.lapply / dapply / gapply / ...

SparkR API :: Spark Session

```
# Importing the library
> library(SparkR)
Attaching package: ‘SparkR’

# Initializing connection to Spark
> sparkR.session()
Spark package found in SPARK_HOME: /databricks/spark
Java ref type org.apache.spark.sql.SparkSession id 1
```

Session Initialization



1. RBackend opens a server port and waits for connections

2. SparkR establishes a TCP connection

3. Each SparkR call sends serialized data over the socket and waits for response

4. RBackendHandler handles and process requests

SparkR Serialization

R and JVM use a proprietary serialization format as wire protocol.

Basic type:

type	binary data

List:

type	size	element 1, element 2, element 3, ...

SparkR API :: I/O

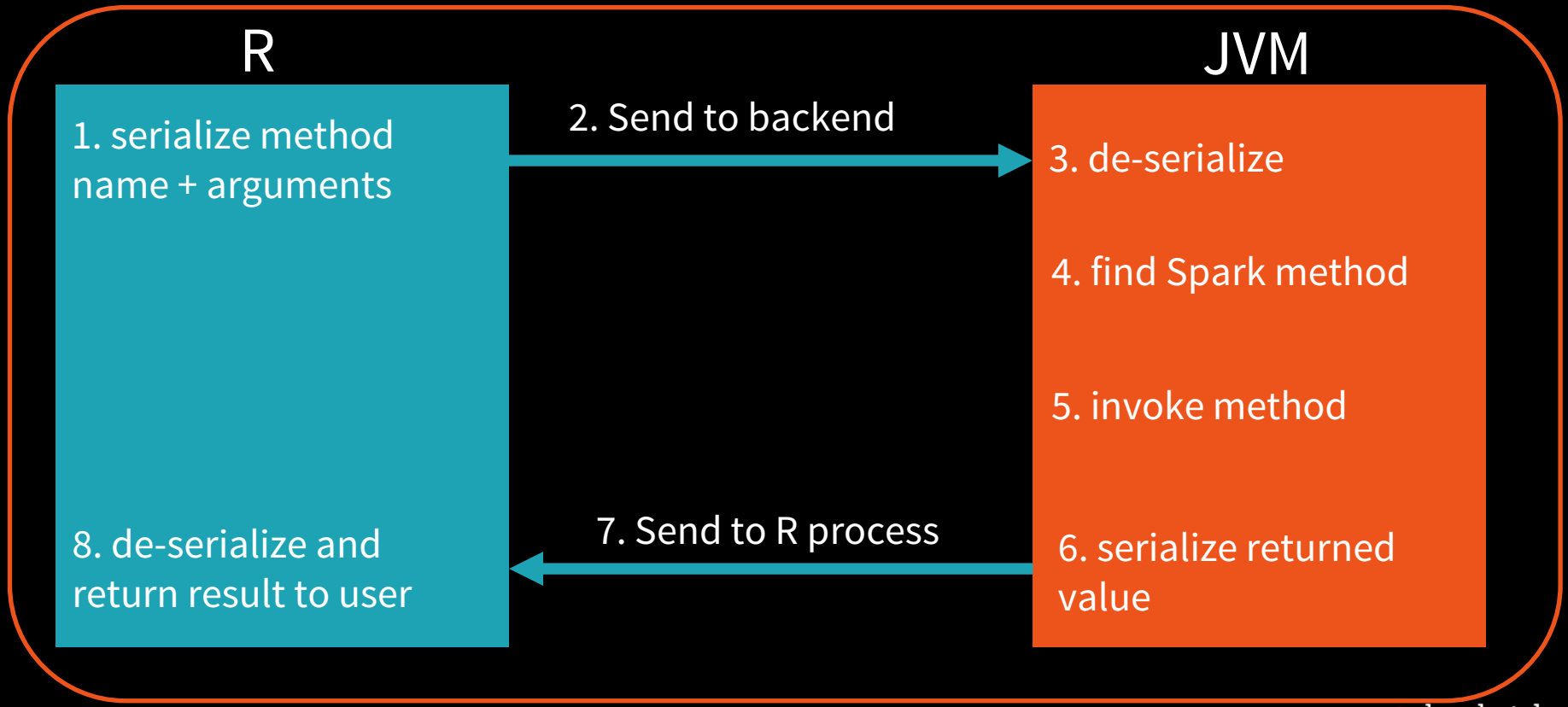
Reading

```
> spark.df <- read.df(path = “...”, source = “csv”)
> cache(spark.df)
> dim(spark.df)
[1] 1235349690   31
```

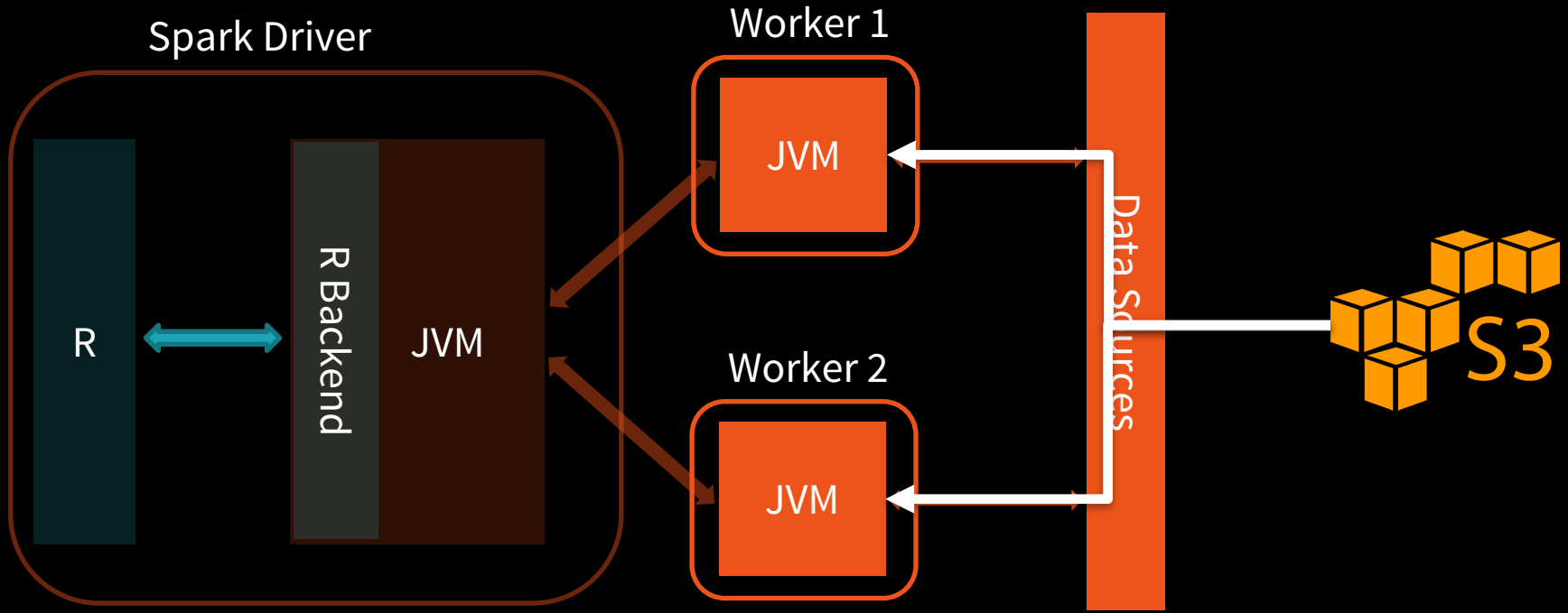
Writing

```
> write.df(spark.df, path = “...”, source = “json”)
```

Control plane



Data plane


Spark Driver
R
R Backend
JVM
Worker 1
JVM
Worker 2
JVM
Data Sources
S3

SparkR API :: I/O

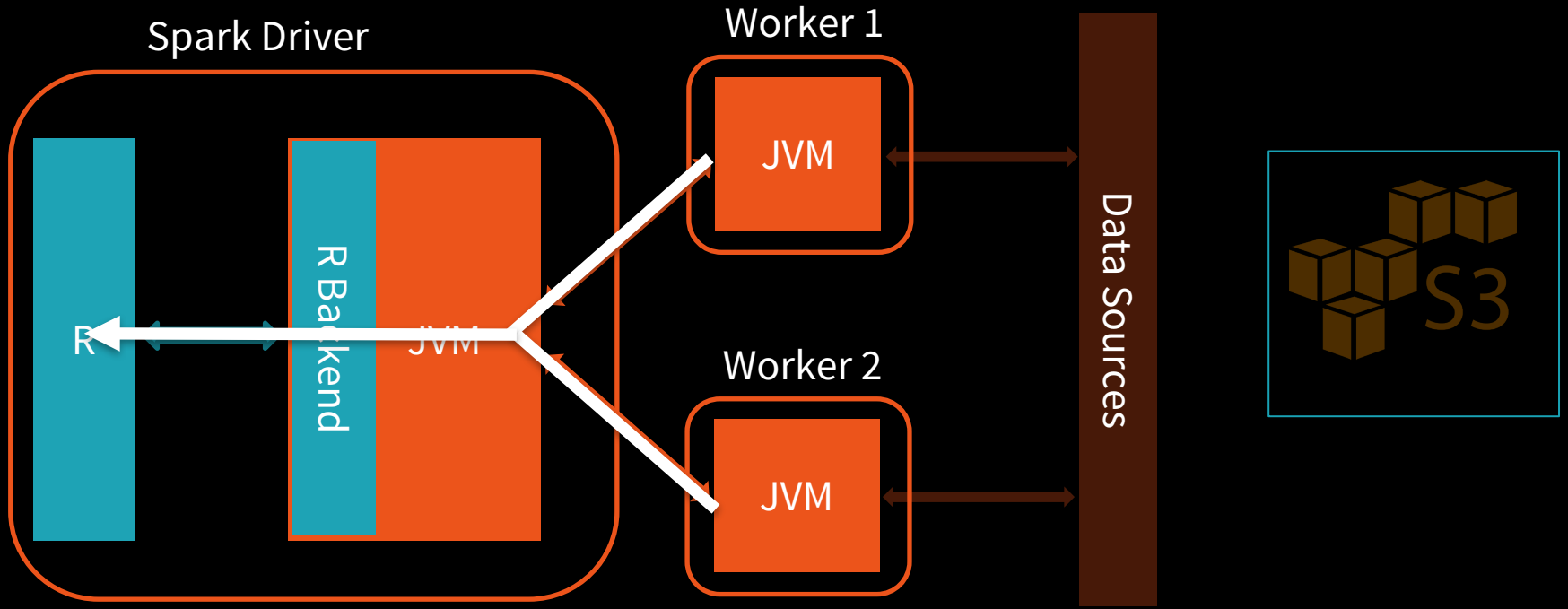
JVM to R

```
> r.df <- collect(spark.df)
```

R to JVM

```
> spark.df <- createDataFrame(iris)
```

Data plane


Spark Driver
R
R Backend
JVM
Worker 1
JVM
Worker 2
JVM
Data Sources
S3

SparkR API :: Transforming data

```
# Filtering
> iad <- subset(df, df$Dest == “IAD”)

# Grouping and aggregation
> arrivals <- count(groupBy(iad, iad$Origin))

# Ordering
> head(orderBy(arrivals, -arrivals$count))
```

SparkR API :: SQL

```
> createOrReplaceTempView (df, “table”)
> arrivals <- sql(“
    SELECT count(FlightNum) as count, Origin
    FROM table
    WHERE Dest == “IAD”
    GROUP BY Origin”)
> plot(collect(arrivals))
```

Distributing R Computation

Overview of SparkR API

http://spark.apache.org/docs/latest/api/R/

IO

read.df / write.df /
createDataFrame / collect

Caching

cache / persist / unpersist
cacheTable / uncacheTable

SQL

sql / table / saveAsTable
registerTempTable / tables

ML Lib

glm / kmeans / Naïve Bayes
Survival regression

DataFrame API

select / subset / groupBy
head / avg / column / dim

UDF functionality

spark.lapply / dapply
gapply / dapplyCollect

SparkR UDF API

spark.lapply

Runs a function over a list of elements on workers

```
spark.lapply()
```

dapply

Applies a function to each partition of a SparkDataFrame

```
dapply()
dapplyCollect()
```

gapply

Applies a function to each group within a SparkDataFrame

```
gapply()
gapplyCollect()
```

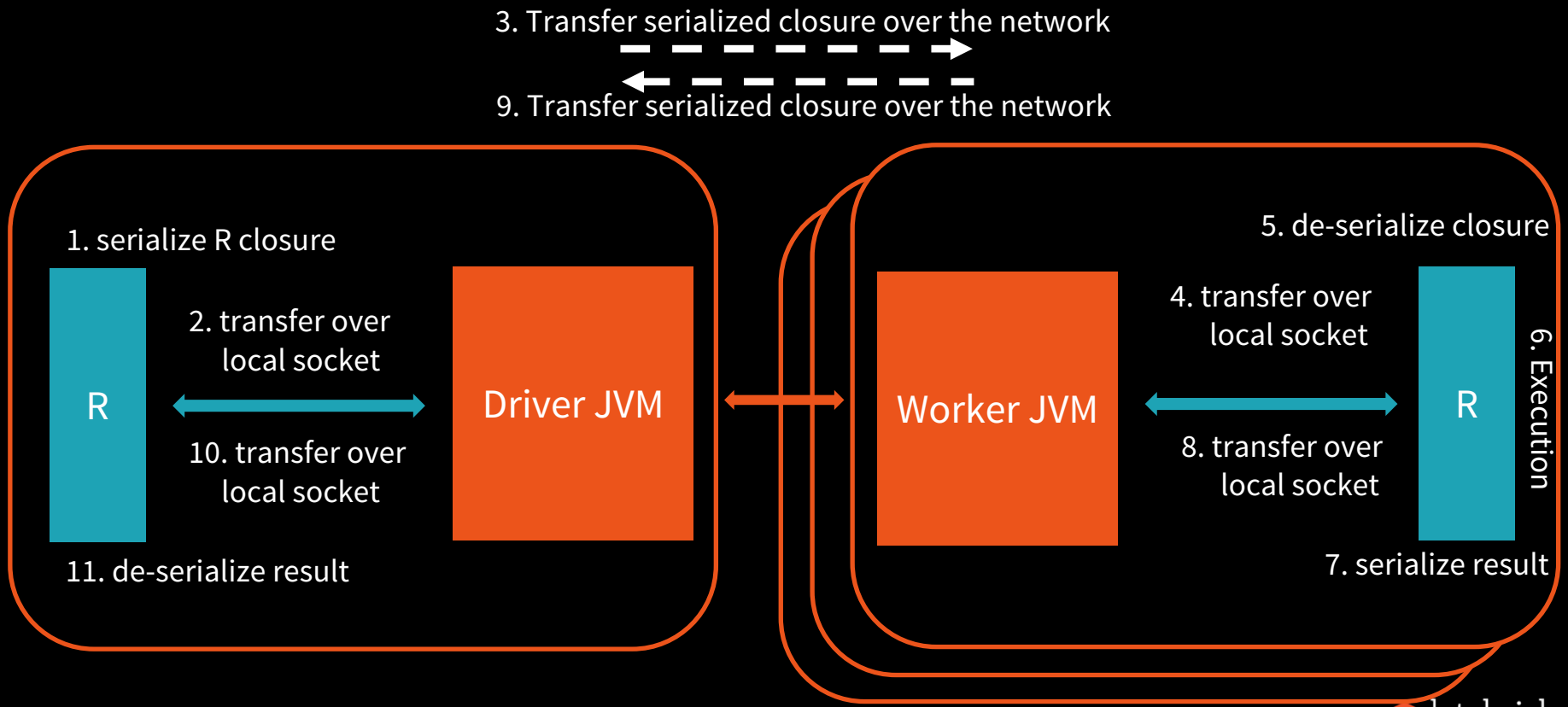
spark.lapply

For each element of a list

1. Sends the function to an R worker
2. Executes the function
3. Returns the result of all workers as a list to R driver

```
spark.lapply(1:100, function(x) {
  runBootstrap(x)
}
```

spark.lapply control flow



dapply

For each partition of the input Spark DataFrame

1. Collects the partition as an R data.frame inside the worker
2. Sends the R function to the R process inside worker
3. Executes the function with R data.frame as input

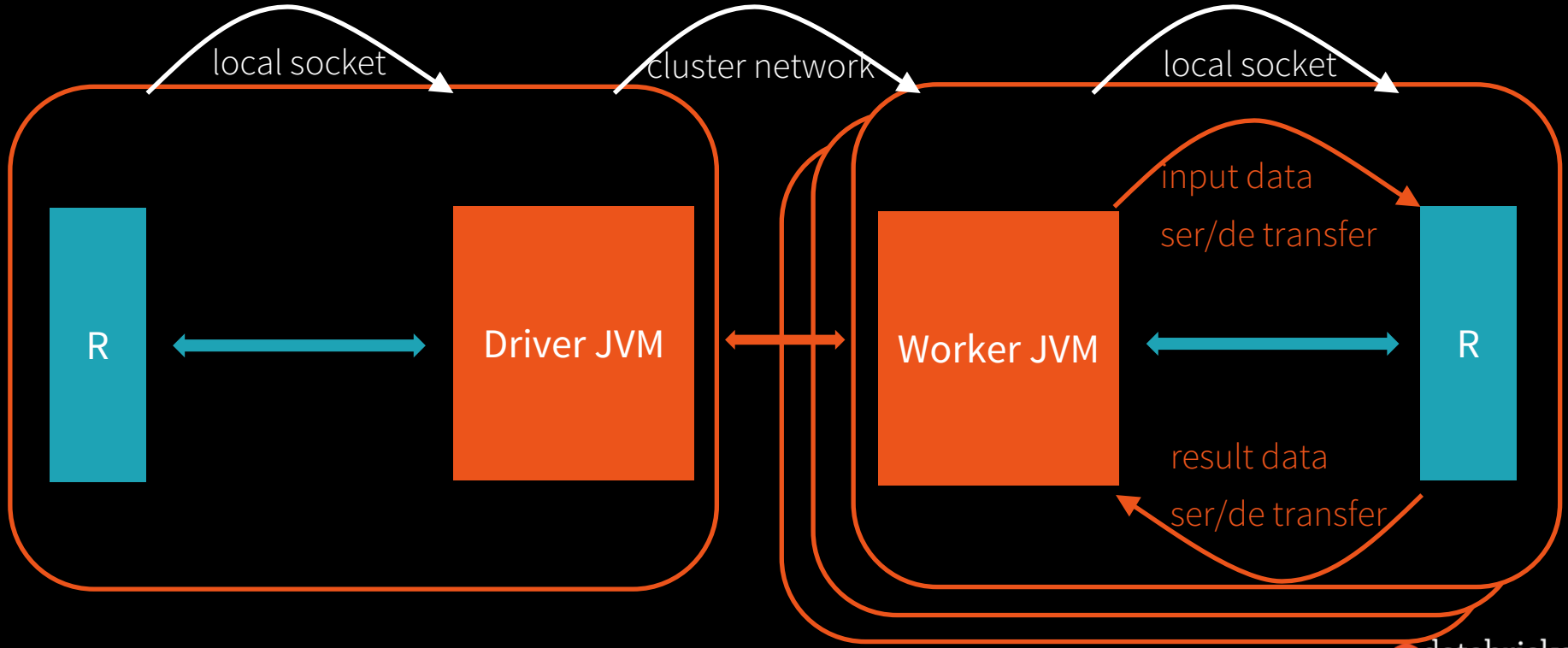
```
dapply(sparkDF, func, schema)
```

Returns results as DataFrame with provided schema

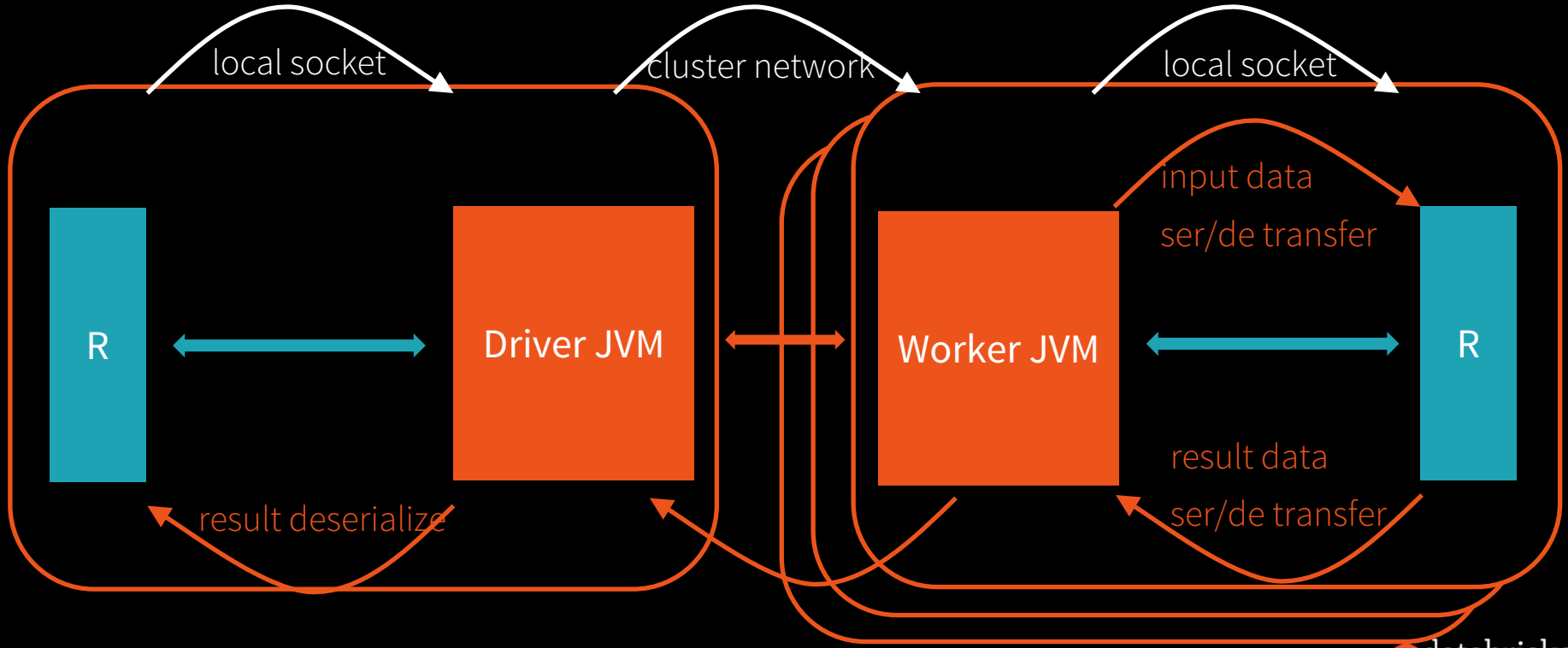
```
dapplyCollect(sparkDF, func)
```

Returns results as an R data.frame

dapply control & data flow



dapplyCollect control & data flow



gapply

Groups the Spark DataFrame on one or more columns

1. Collects each group as an R data.frame inside the worker
2. Sends the R function to the R process inside worker
3. Executes the function with R data.frame as input

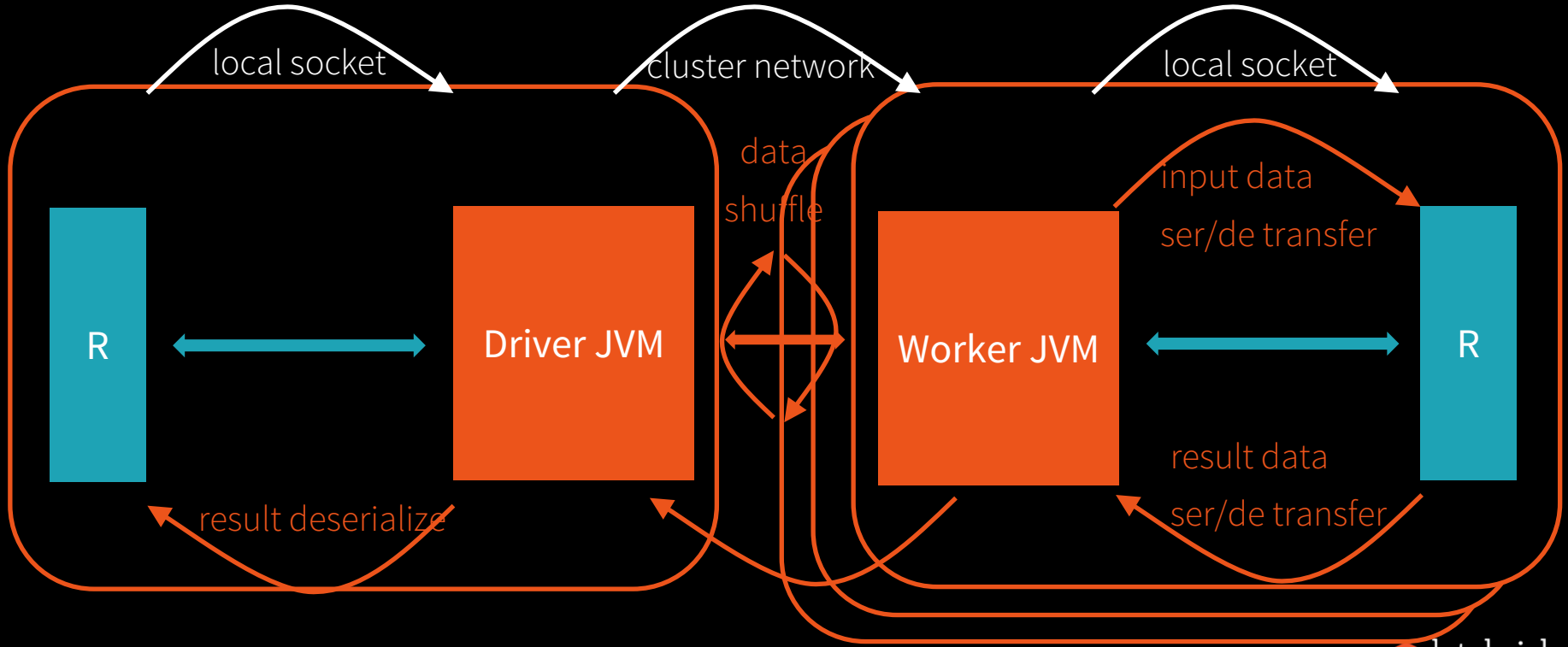
```
gapply(df, cols, func, schema)
```

Returns results as DataFrame with provided schema

```
gapplyCollect(df, cols, func)
```

Returns results as an R data.frame

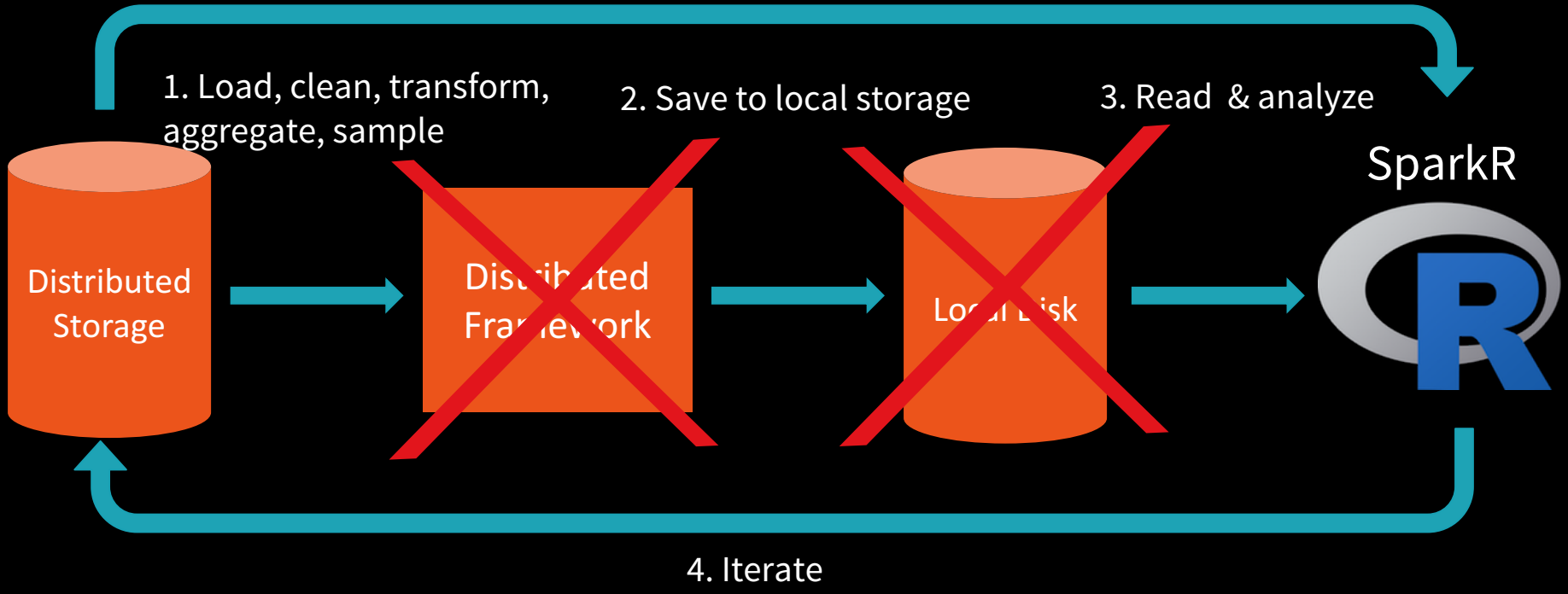
gapplyCollect control & data flow



	gapply	dapply
signature	`gapply(df, cols, func, schema)` `gapply(gdf, func, schema)`	`dapply(df, func, schema)`
user function signature	`function(key, data)`	`function(data)`
data partition	controlled by grouping	not controlled

Use Cases

1. Framework & code unification



Advantages of SparkR

What	Why
No need to save intermediate data on disk	Transfer data directly into R process
No need to manage different codebases	Write entire pipeline in R
Interactive iterations on data	Distributed data can be cached on cluster
Work with any data source & formats	Use Apache Spark data sources ecosystem

Perform interactive analysis on distributed at the speed of thinking from R

What to watch for

1. SparkR functions shadowing other packages
2. How much data can you **`collect()`**?
3. Cache your active Spark DataFrame
4. Push feature engineering to Spark as much as possible
   - Merging datasets
   - Transformations
   - Filtering and projections
   - Sampling

data.frame vs. DataFrame

- Example error messages
  - `... doesn't know how to deal with data of class SparkDataFrame`
  - `no method for coercing this S4 class to a ...`
  - `Expressions other than filtering predicates are not supported in the first parameter of extract operator.`

R function vs. SparkSQL expression

Expressions translate to JVM calls, but functions run in R process of driver or workers

- `filter(logs$type == “ERROR”)`
- `ifelse(df$level > 2, “deep”, “shallow”)`

```
dapply(logs, function(x) {
  subset(x, type == “ERROR”)
}, schema(logs))
```

2. Parallelizing Existing R Programs

Compute-bound

- Complex logic implemented in R
- Multi-threaded to use more than a single core
- Requires large workstation with a lot of RAM and many cores
- Example: simulations, bootstraping

Data-bound

- Usually simple statistics
- Data is growing by volume or complexity
- Number of interactions are growing
- Example: GWAS

SparkR to rescue

- Instead of a single large machine (scaling up) use many commodity computers (scaling out)
  1. Perform data preparation (ETL) using SparkSQL
  2. Distribute input data according to algorithm logic
  3. Parallelize R function to execute on each partition

What to watch for

1. Skew in data/computation
   - Are partitions evenly sized?
   - Is computation uniform across partitions?
2. Packing too much data into the closure
3. Using third-party packages
4. Returned data schema
5. Keeping the pipeline full to minimize impact of fixed costs

Packing too much into the closure

```
Error in invokeJava(isStatic = FALSE, objId$id, methodName,
...):

  org.apache.spark.SparkException: Job aborted due to stage
failure: Serialized task 29877:0 was 520644552 bytes, which
exceeds max allowed: spark.rpc.message.maxSize (268435456
bytes).
```

Using the wrong apply function

- Do not use `spark.lapply()` to distribute data
  - `createDataFrame()`
  - Load from storage
- Know your data partitioning
  - `partitionBy() %>% dapply()`
  - `groupBy() %>% gapply()`

3. Distributed Machine Learning

- SparkR exposes a wide range of distributed learning algorithms implemented in Spark
- Formula-based API that takes SparkDataFrame instead of R data.frame

```
> df <- createDataFrame(as.data.frame(Titanic))
> model <- glm(Freq ~ Sex + Age, df, family = "gaussian")
> summary(model)
```

What to watch for

- Spark MLlib models do not include data
  - You need to compute residuals
- Many cases you need to specify number of iterations
  - Spark uses a distributed iterative optimizer to

Thank You