SCALABLE DATA SCIENCE WITH SPARKR

Felix Cheung
Principal Engineer & Apache Spark Committer


APACHE
Spark
TM

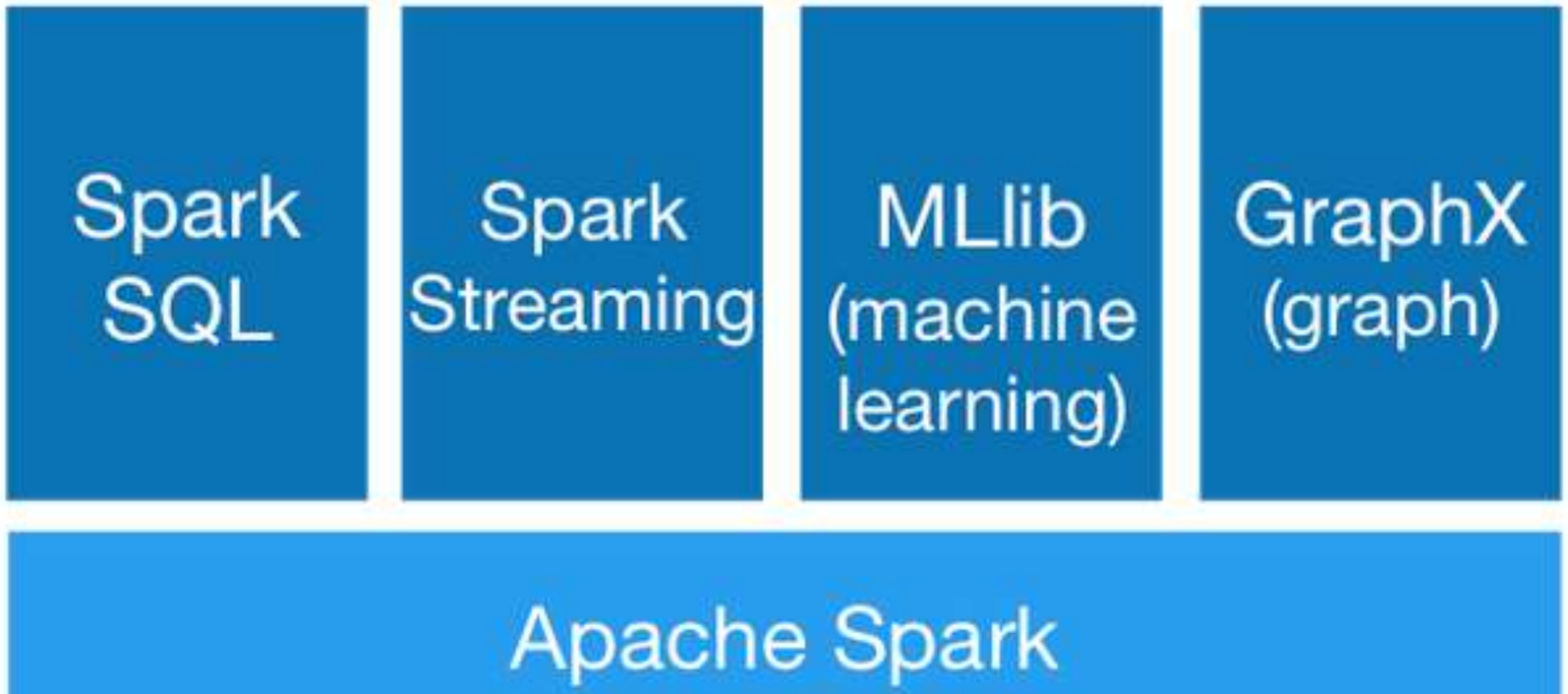
Disclaimer:
Apache Spark community contributions

Agenda

- Spark + R, Architecture
- Features
- SparkR for Data Science
- Ecosystem
- What’s coming

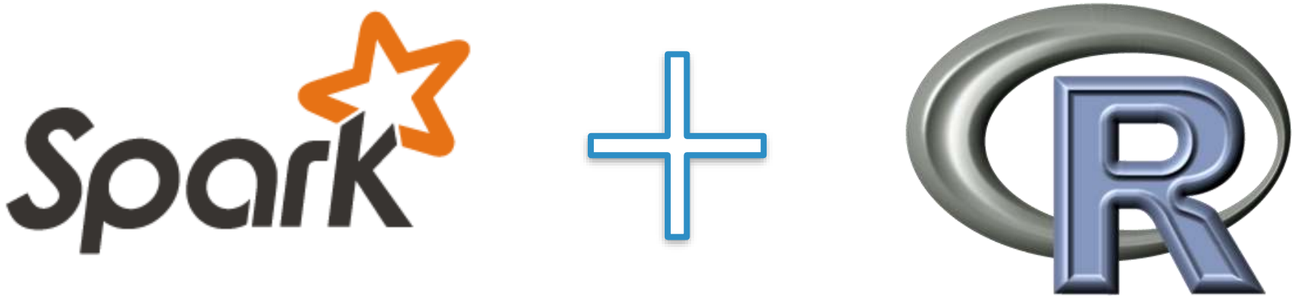
Spark in 5 seconds

- General-purpose cluster computing system
- Spark SQL + DataFrame/Dataset + data sources
- Streaming/Structured Streaming
- ML
- GraphX



R

- A programming language for statistical computing and graphics
- S – 1975
- S4 - advanced object-oriented features
- R – 1993 = S + lexical scoping
- Interpreted
- Matrix arithmetic
- Comprehensive R Archive Network (CRAN) – 10k+ packages


Spark
R

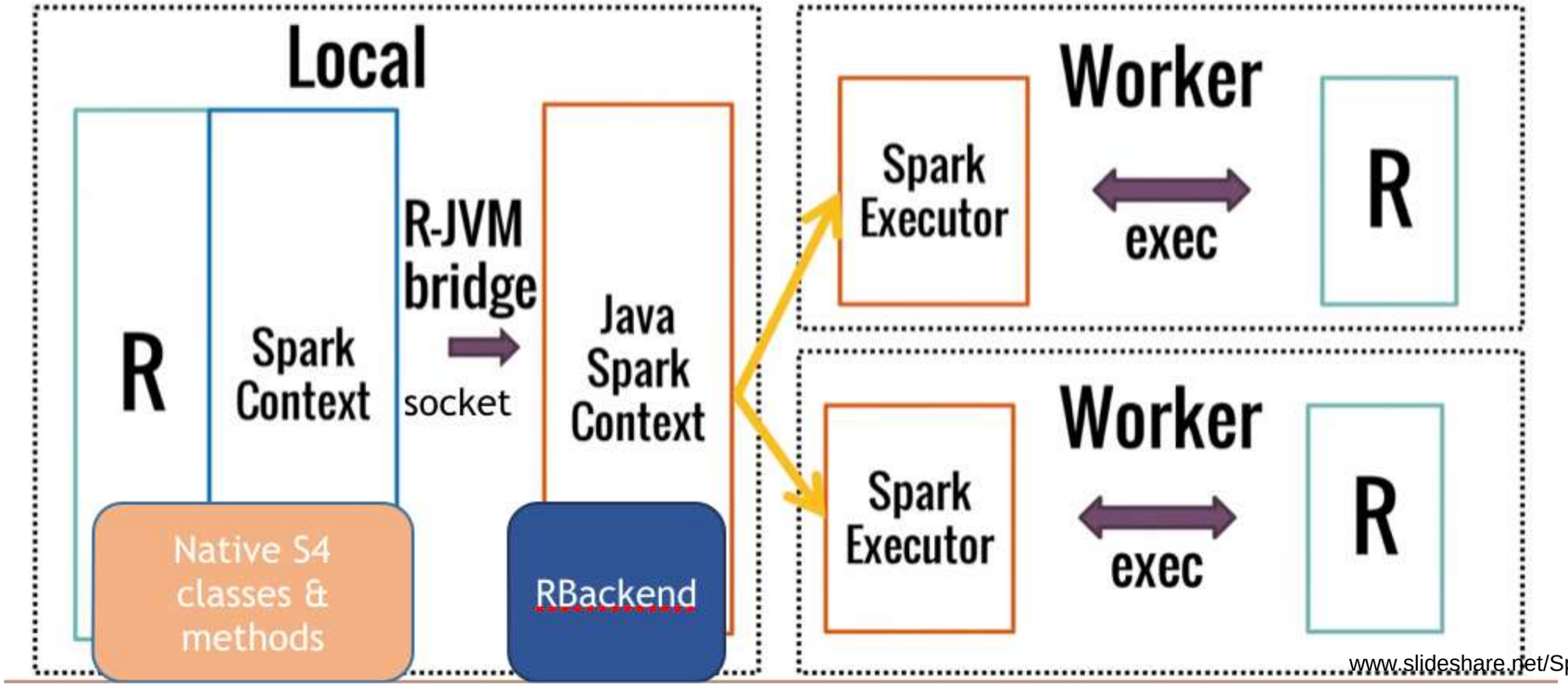
SparkR

- R language APIs for Spark and Spark SQL
- Exposes Spark functionality in an R-friendly DataFrame APIs
- Runs as its own REPL `sparkR`
- or as a R package loaded in IDEs like RStudio

```
library(SparkR)
sparkR.session()
```

Architecture

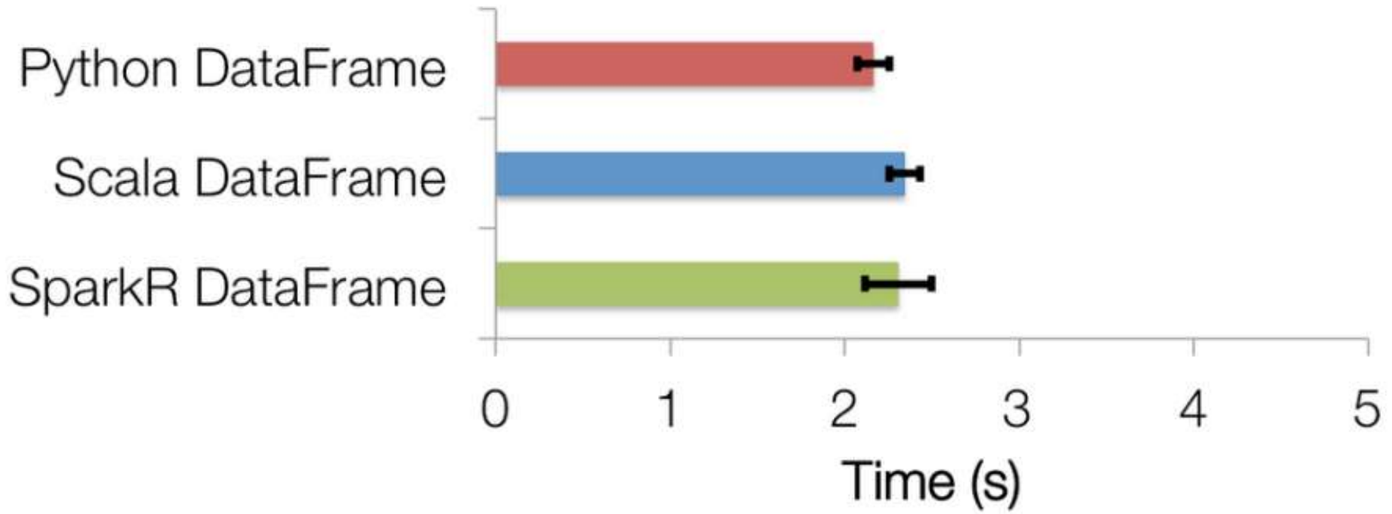
- Native R classes and methods
- RBackend
- Scala “helper” methods (ML pipeline etc.)



www.slideshare.net/SparkSummit/07-venkataraman-sun

Key Advantage

- JVM processing, full access to DAG capabilities and Catalyst optimizer, predicate pushdown, code generation, etc.



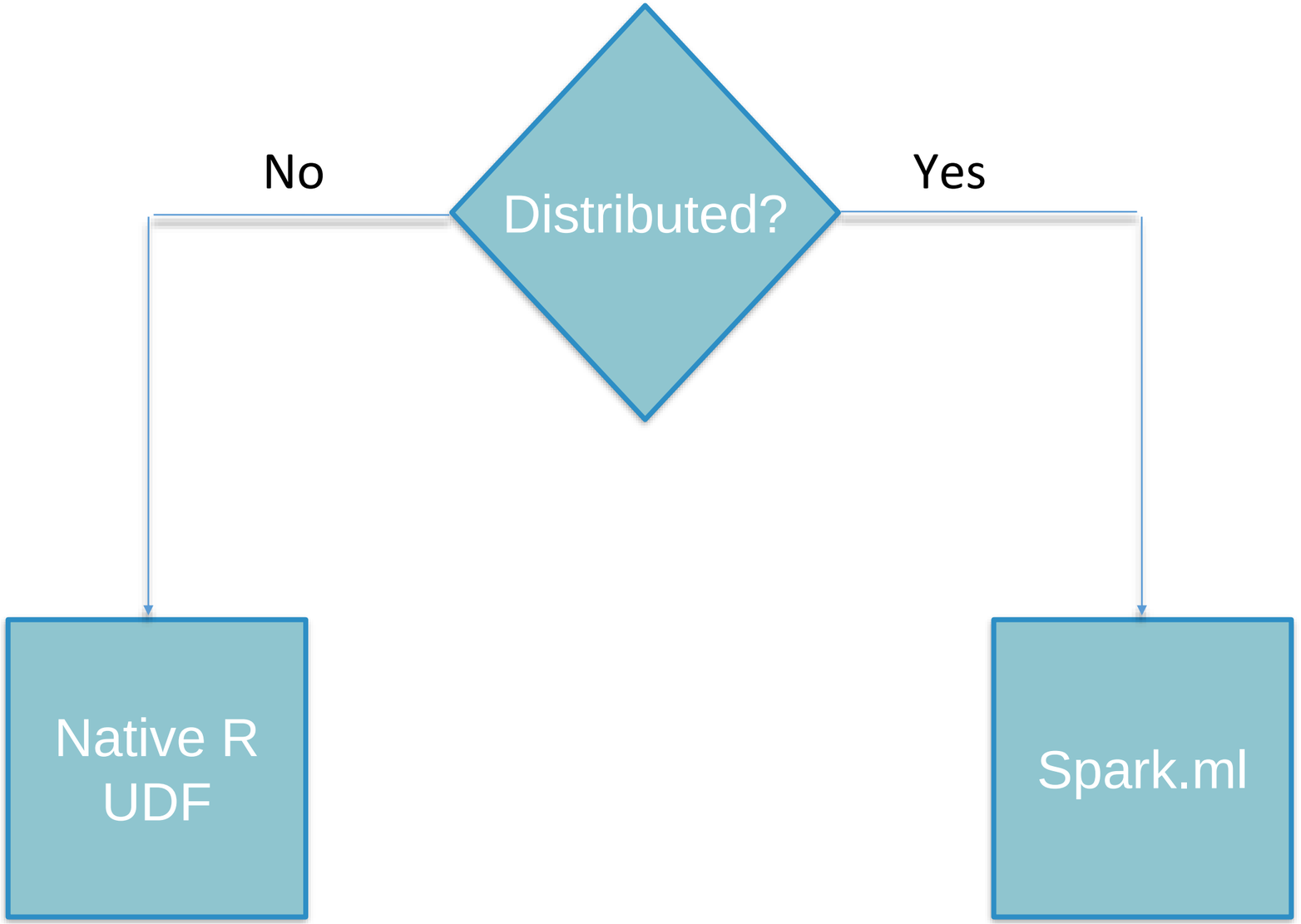
databricks.com/blog/2015/06/09/announcing-sparkr-r-on-spark.html

Features - What's new in SparkR

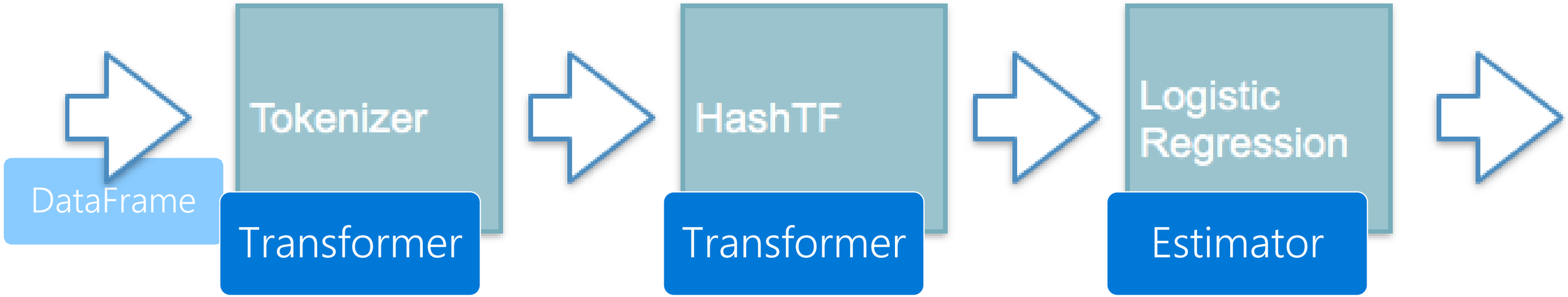
- SQL & Data Source (JSON, csv, JDBC, libsvm)
- SparkSession & default session
- Catalog (database & table management)
- Spark packages, `spark.addFiles()`
- `install.spark()`
- ML
- R-native UDF
- **Structured Streaming**
- Cluster support (YARN, mesos, standalone)

SparkR for Data Science

Decisions, decisions?



Spark ML Pipeline



Spark ML Pipeline

- Pre-processing, feature extraction, model fitting, validation stages
- Transformer
- Estimator
- Cross-validation/hyperparameter tuning

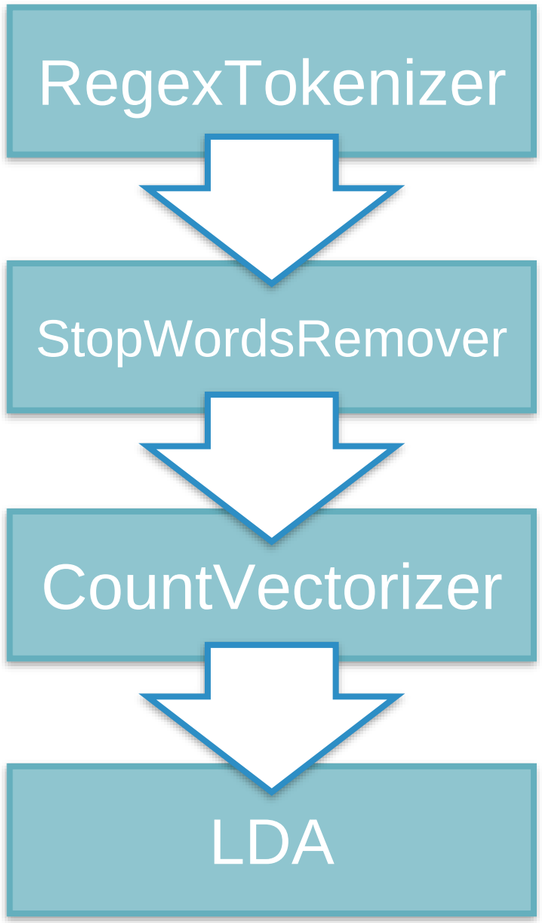
SparkR API for ML Pipeline

R

```
spark.lda(
data = text, k =
20, maxIter = 25,
optimizer = "em")
```

API
builds
ML Pipeline

JVM



Model Operations

- `summary` - print a summary of the fitted model
- `predict` - make predictions on new data
- `write.ml`/`read.ml` - save/load fitted models (slight layout difference: pipeline model plus R metadata)

Spark.ml in SparkR 2.0.0

- Generalized Linear Model (GLM)
- Naive Bayes Model
- k-means Clustering
- Accelerated Failure Time (AFT) Survival Model

Spark.ml in SparkR 2.1.0

- Generalized Linear Model (GLM)
- Naive Bayes Model
- k-means Clustering
- Accelerated Failure Time (AFT) Survival Model
- Isotonic Regression Model
- Gaussian Mixture Model (GMM)
- Latent Dirichlet Allocation (LDA)
- Alternating Least Squares (ALS)
- Multilayer Perceptron Model (MLP)
- Kolmogorov-Smirnov Test (K-S test)
- Multiclass Logistic Regression
- Random Forest
- Gradient Boosted Tree (GBT)

RFormula

- Specify modeling in symbolic form

  ```
  y ~ f0 + f1
  ```

  response *y* is modeled linearly by *f0* and *f1*
- Support a subset of R formula operators

  `~` , `.` , `:` , `+` , `-`
- Implemented as feature transformer in core Spark, available to Scala/Java, Python
- String label column is indexed
- String term columns are one-hot encoded

Generalized Linear Model

```
# R-like
glm(Sepal_Length ~ Sepal_Width + Species,
gaussianDF, family = "gaussian")

spark.glm(binomialDF, Species ~
Sepal_Length + Sepal_Width, family =
"binomial")
```

- “binomial” output string label, prediction

Naive Bayes

```
spark.naiveBayes(nbDF, Survived ~ Class + Sex
+ Age)
```

- index label, predicted label to string

k-means

```
spark.kmeans(kmeansDF, ~ Sepal_Length +
Sepal_Width + Petal_Length + Petal_Width,
k = 3)
```

Accelerated Failure Time (AFT) Survival Model

```
spark.survreg(aftDF, Surv(futime, fustat) ~
ecog_ps + rx)
```

- formula rewrite for censor

Isotonic Regression

```
spark.isoreg(df, label ~ feature, isotonic =
FALSE)
```

Gaussian Mixture Model

```
spark.gaussianMixture(df, ~ V1 + V2, k = 2)
```

Alternating Least Squares

```
spark.als(df, "rating", "user", "item", rank
= 20, reg = 0.1, maxIter = 10,  nonnegative =
TRUE)
```

Multilayer Perceptron Model

```
spark.mlp(df, label ~ features,
blockSize = 128, layers = c(4, 5, 4,
3), solver = "l-bfgs", maxIter = 100,
tol = 0.5, stepSize = 1)
```

Multiclass Logistic Regression

```
spark.logit(df, label ~ ., regParam =
0.3, elasticNetParam = 0.8, family =
"multinomial", thresholds = c(0, 1, 1))
```

- binary or multiclass

Random Forest

```
spark.randomForest(df, Employed ~ ., type
= "regression", maxDepth = 5, maxBins =
16)
```

```
spark.randomForest(df, Species ~
Petal_Length + Petal_Width,
"classification", numTree = 30)
```

- “classification” index label, predicted label to string

Gradient Boosted Tree

```
spark.gbt(df, Employed ~ ., type =
"regression", maxDepth = 5, maxBins = 16)
```

```
spark.gbt(df, IndexedSpecies ~ ., type =
"classification", stepSize = 0.1)
```

- “classification” index label, predicted label to string
- Binary classification

Modeling Parameters

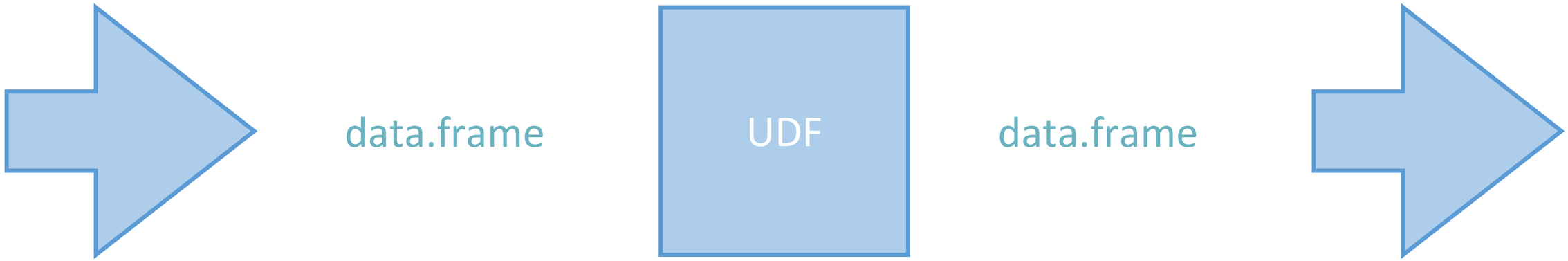
```
spark.randomForest
function(data, formula, type = c("regression", "classification"),
         maxDepth = 5, maxBins = 32, numTrees = 20, impurity = NULL,
         featureSubsetStrategy = "auto", seed = NULL,
         subsamplingRate = 1.0,
         minInstancesPerNode = 1, minInfoGain = 0.0,
         checkpointInterval = 10,
         maxMemoryInMB = 256, cacheNodeIds = FALSE)
```

Spark.ml Challenges

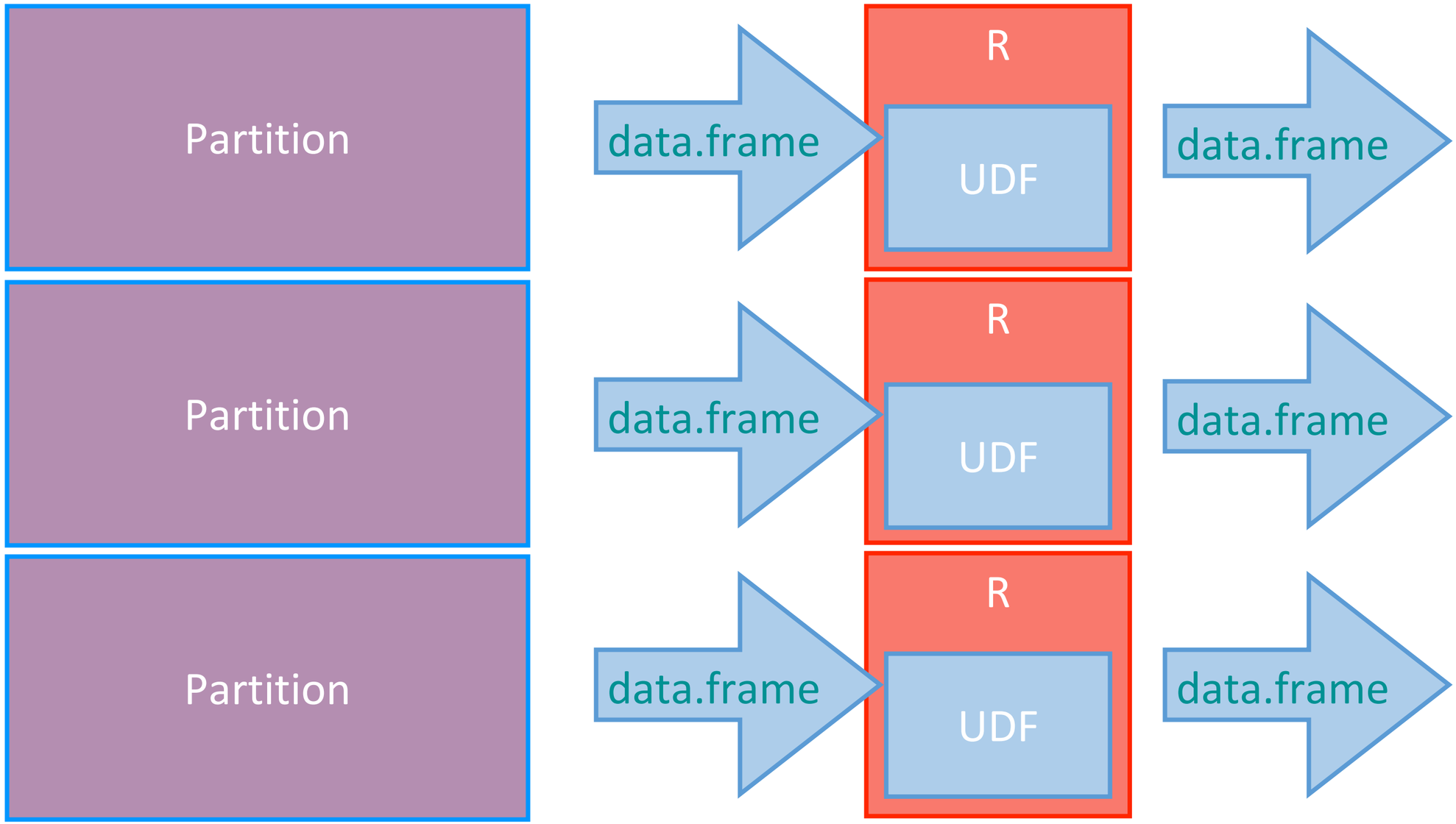
- Limited API sets
  - Keeping up to changes - Almost all
  - Non-trivial to map spark.ml API to R API
  - Simple API, but fixed ML pipeline
- Debugging is hard
  - Not a ML specific problem - Getting better?

Native-R UDF

- User-Defined Functions - custom transformation
- Apply by Partition
- Apply by Group



Parallel Processing By Partition



UDF: Apply by Partition

- Similar to R `apply`
- Function to process each partition of a DataFrame
- Mapping of Spark/R data types

```
dapply(carsSubDF,
       function(x) {
         x <- cbind(x, x$mpg * 1.61)
       },
       schema)
```

UDF: Apply by Partition + Collect

- No schema

```
out <- dapplyCollect(
          carsSubDF,
          function(x) {
            x <- cbind(x, "kmpg" = x$mpg*1.61)
          })
```

Example - UDF

```
results <- dapplyCollect(train,
  function(x) {
    model <-
randomForest::randomForest(as.factor(dep_delayed_1
5min) ~ Distance + night + early, data = x,
importance = TRUE, ntree = 20)
    predictions <- predict(model, t)
    data.frame(UniqueCarrier = t$UniqueCarrier,
delayed = predictions)
  })
```

access package “randomForest::” at each invocation

closure capture - serialize & broadcast “t”

UDF: Apply by Group

- By grouping columns

```
gapply(carsDF, "cyl",
    function(key, x) {
        y <- data.frame(key, max(x$mpg))
    },
    schema)
```

UDF: Apply by Group + Collect

- No Schema

```
out <- gapplyCollect(carsDF, "cyl",
           function(key, x) {
             y <- data.frame(key, max(x$mpg))
             names(y) <- c("cyl", "max_mpg")
             y
      })
```

UDF: data type mapping

R	Spark
byte	byte
integer	integer
float	float
double, numeric	double
character, string	string
binary, raw	binary
logical	boolean
POSIXct, POSIXlt	timestamp
Date	date
array, list	array
env	map

* not a complete list

UDF Challenges

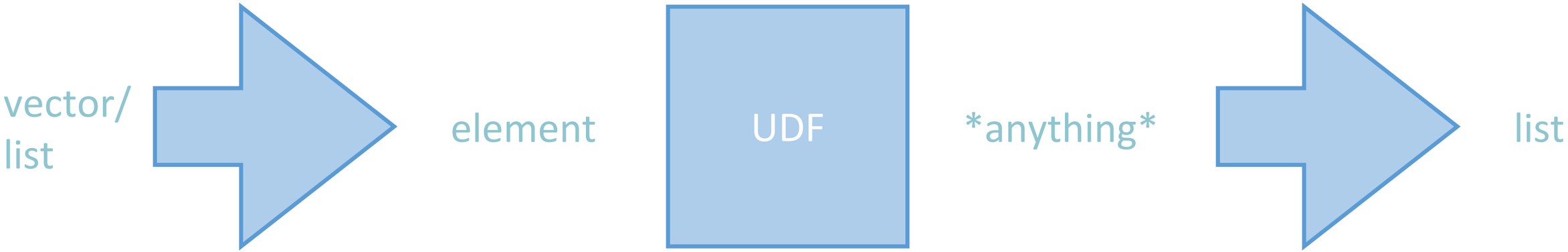
- “struct” - No support for nested structures as columns
- Scaling up / data skew
  - What if partition or *group* too big for single R process?
- Not enough data variety to run model?
- Performance costs
  - Serialization/deserialization, data transfer
  - esp. beware of closure capture
- Package management

UDF: lapply

- Like R `lapply` or `doParallel`
- Good for “embarrassingly parallel” tasks
  - Such as hyperparameter tuning

UDF: lapply

- Take a native R list, distribute it
- Run the UDF in parallel



UDF: parallel distributed processing

- Output is a list - needs to fit in memory at the driver

```
costs <- exp(seq(from = log(1), to = log(1000),
length.out = 5))
train <- function(cost) {
 model <- e1071::svm(Species ~ ., iris, cost = cost)
 summary(model)
}
summaries <- spark.lapply(costs, train)
```

Demo at felixcheung.github.io

SparkR as a Package (target:ASAP)

- Goal: simple one-line installation of SparkR from CRAN

```
install.packages("SparkR")
```

- Spark Jar downloaded from official release and cached automatically, or manually `install.spark()` since Spark 2
- R vignettes
- Community can write packages that depends on SparkR package, eg. `SparkRext`
- Advanced Spark JVM interop APIs

```
sparkR.newJObject, sparkR.callJMethod
sparkR.callJStatic
```

Ecosystem

- RStudio sparklyr
- RevoScaleR/RxSpark, R Server
- H2O R
- Apache SystemML (R-like API)
- STC R4ML
- Renjin (not Spark)
- IBM BigInsights Big R (not Spark!)

Recap: SparkR 2.0.0, 2.1.0, 2.1.1

- `SparkSession`
- ML
  - GLM, LDA
- UDF
- `numPartitions`, `coalesce`
- `sparkR.uiWebUrl`
- `install.spark`

What's coming in SparkR 2.2.0

- More, richer ML
  - Bisecting K-means, Linear SVM, GLM - Tweedie, FP-Growth, Decision Tree (2.3.0)
- Column functions
  - `to_date`/`to_timestamp` format, `approxQuantile` columns, `from_json`/`to_json`
- Structured Streaming
- DataFrame - `checkpoint`, `hint`
- Catalog API - `createTable`, `listDatabases`, `refreshTable`, ...

SS in 1 line

```
library(magrittr)
kbsrvs <- "kafka-0.broker.kafka.svc.cluster.local:9092"
topic <- "test1"

read.stream("kafka", kafka.bootstrap.servers = kbsrvs,
subscribe = topic) %>%
selectExpr("explode(split(value as string, ' ')) as word") %>%
group_by("word") %>%
count() %>%
write.stream("console", outputMode = "complete")
```

SS-ML-UDF

See my other talk….

Thank You.

https://github.com/felixcheung
linkedin: http://linkd.in/1OeZDb7