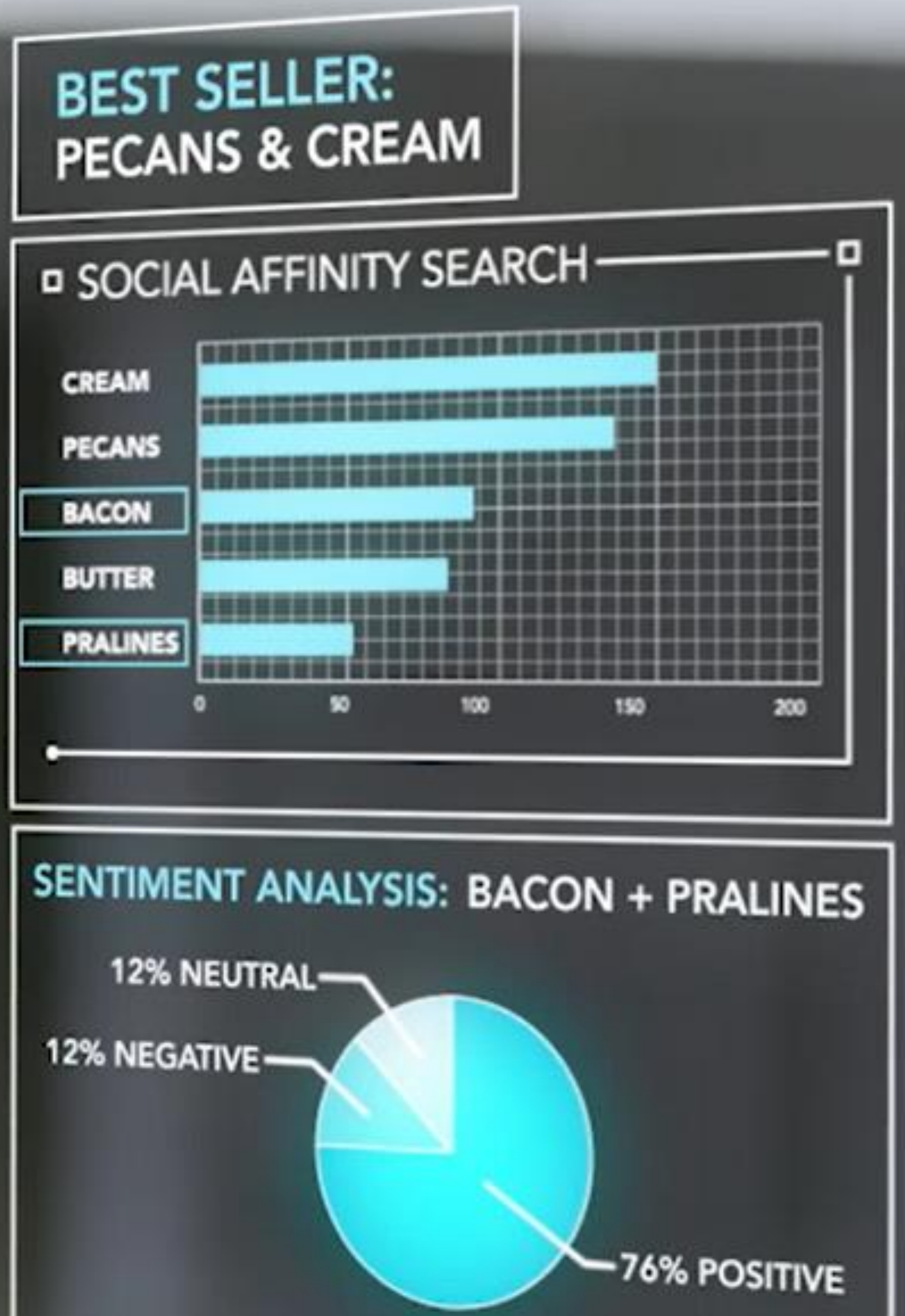


Microsoft R Server

Hadoop

Microsoft

Is your problem big enough for Hadoop ?

When to use Hadoop ?

- Conventional tools won't work on your data
- Your data is really big !
- Won't fit/process in your favourite database or file-system
- Data is really diverse !
- Semi-structured – JSON, XML, Logs, Images, Sounds
- You're a whiz at programming and sys-admin

When not to use Hadoop ?

- !(When to use Hadoop ?)
- You're in a hurry !

Jargon <-

hadoop

hadoop

hadoop Map Reduce

cascading

HDFS

redis

cassandra

HIVE

FLUME

scribe

APACHE HBASE

HCatalog

nutch

Spark

HUE

Sqoop

MESOS

HBASE

Apache Solr

mahout

OOZIE

AVRO

Parquet

Jargon <- sort(hadoop)

Data Input

- Sqoop
- Flume
- nutch
- scribe

File-Systems

- HDFS

File-Formats

- Text delimited
- Text fixed
- RC Files
- ORC Files
- AVRO
- Parquet
- Apache HBase

Databases

- HBase
- redis
- cassandra
- Hive

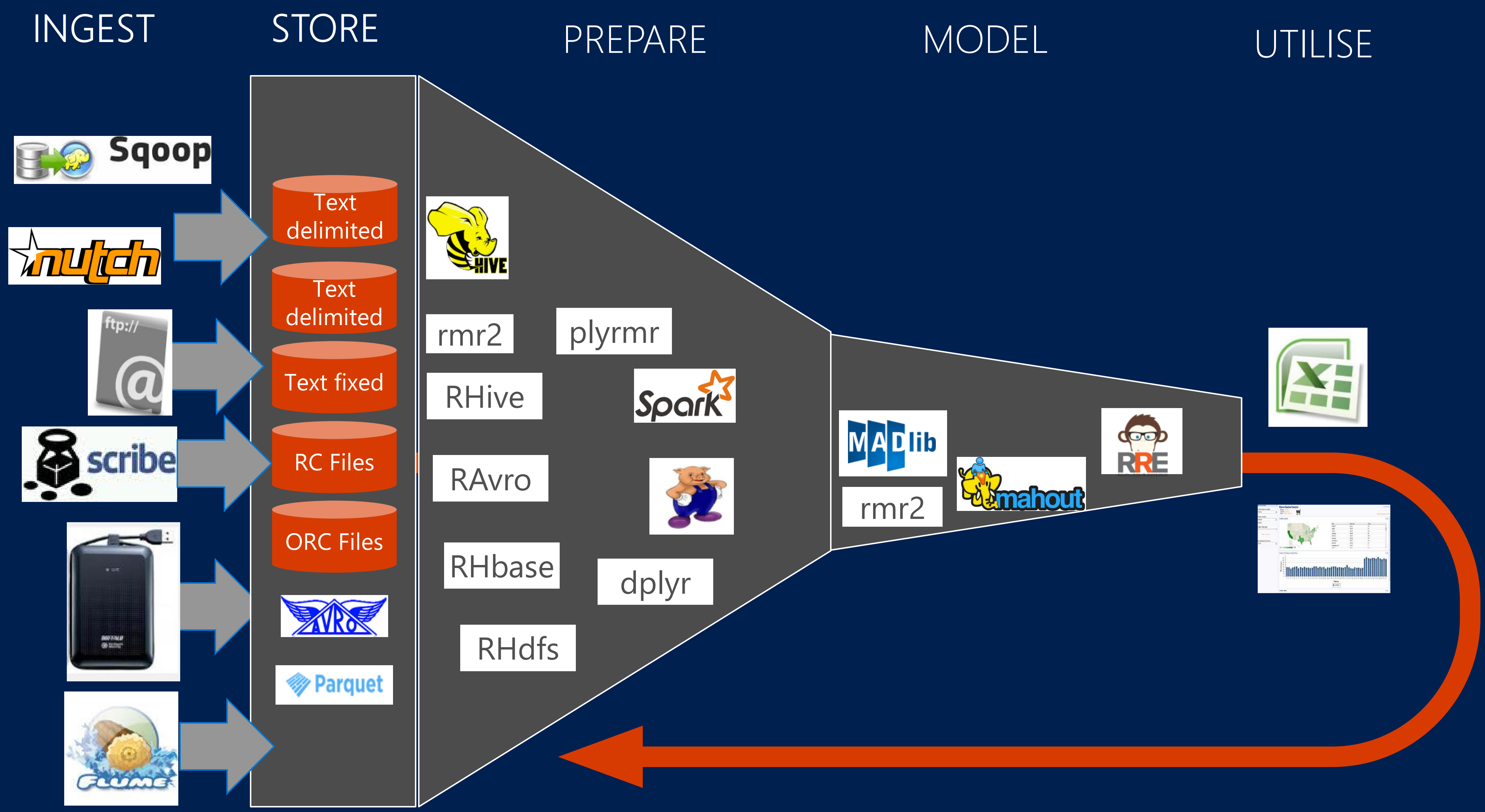
Processing Engines / Languages

- hadoop MapReduce
- Spark
- mahout
- Thrift
- Apache Solr

Management / Control / Workflow/ Metadata

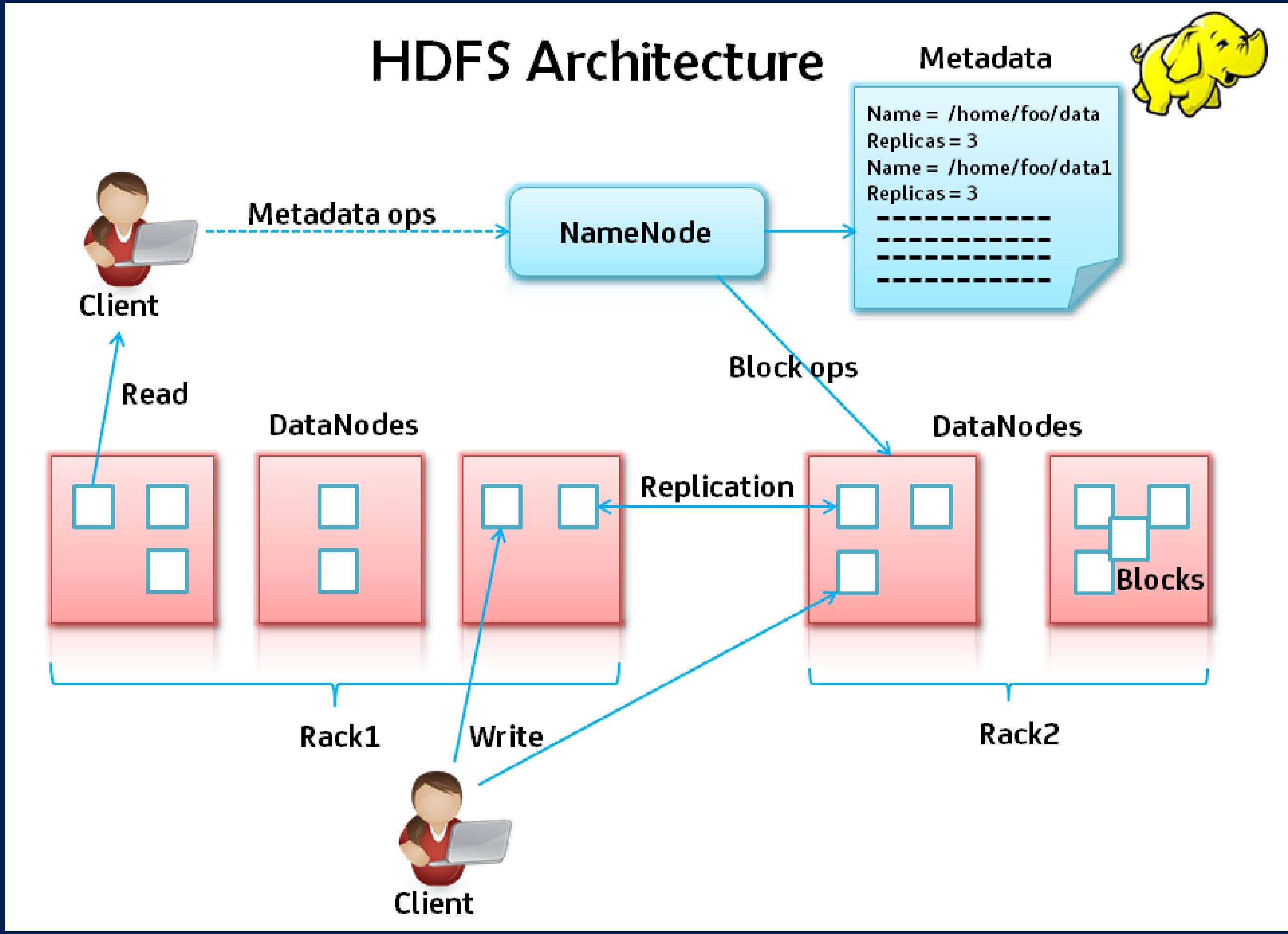
- HCatalog
- Oozie
- MESOS
- cascading
- HUE

Hadoop Analytic Eco-System – Data Pipeline

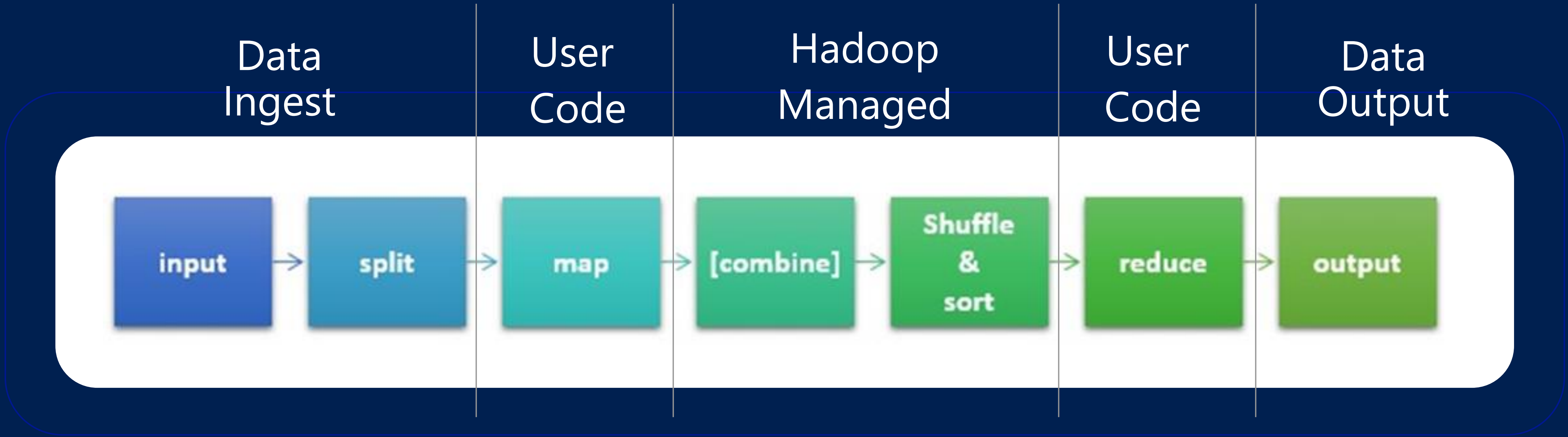


- What is it ?

- Distributed File-System
- Data replication for resiliency – default 3 copies!
- Optimised for
  - Write Once – Read Many
  - Large files ! – big block size – 64MB default!
  - Sequential I/O
- No updates, append only!
- Abstraction over local file-system on servers



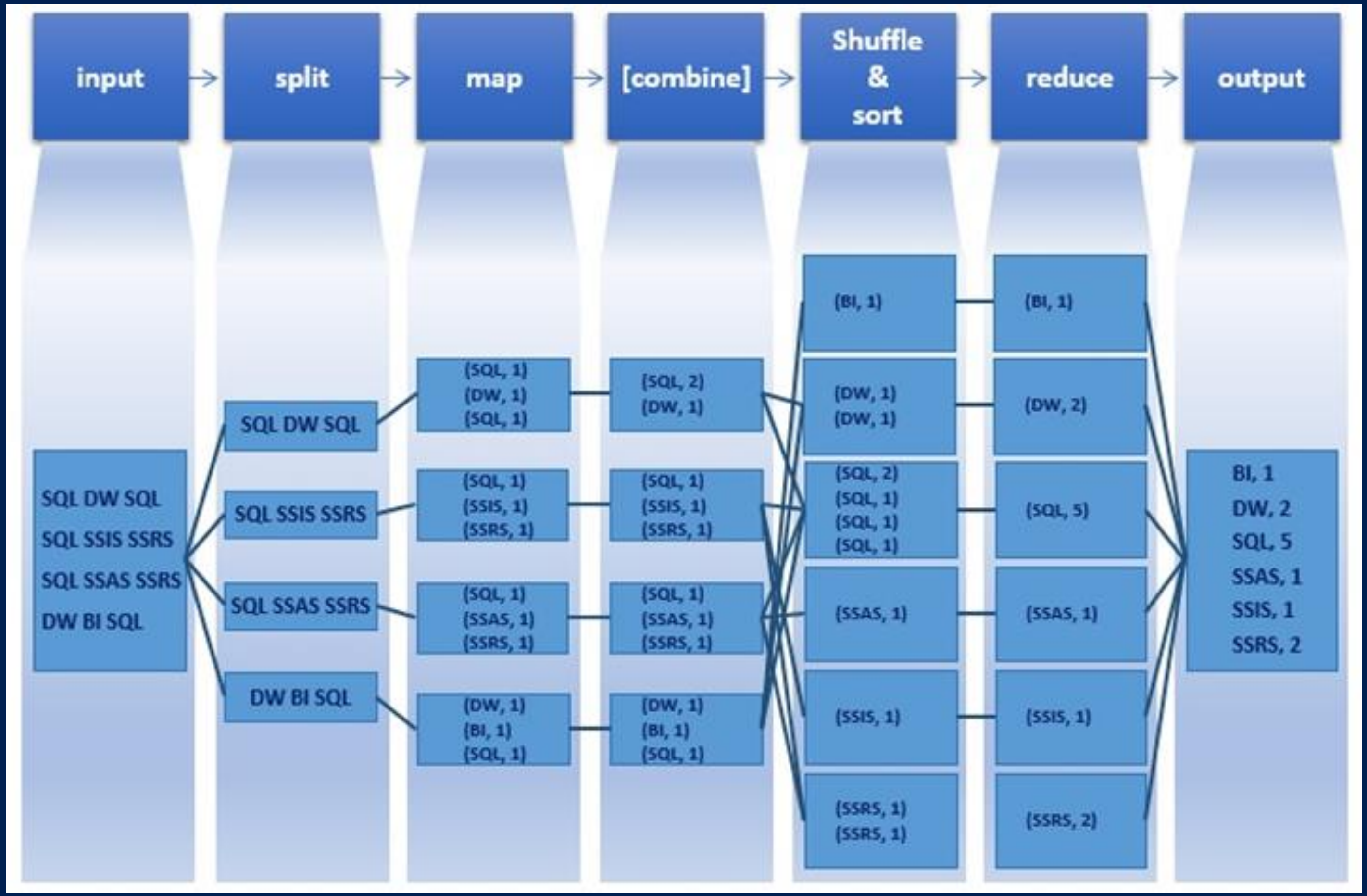
Logical Data Flow



- Functional Programming model – map & reduce
  - Base R {funprog} – Map, Reduce, Filter - http://www.inside-r.org/r-doc/base/Map
  - Rhadoop {rmr2} – supports Map-Reduce programming on Hadoop with R language
  - R Apply family ~= Map

Word Count Example

```
SQL DW SQL
SQL SSIS SSRS
SQL SSAS SSRS
DW BI SQL
```




```
BI, 1
DW, 2
SQL, 5
SSAS, 1
SSIS, 1
SSRS, 2
```

Open Source R on Hadoop

- RHadoop Project
  - rhdfs
  - rhbase
  - ravro
  - rmr2
  - plyrmr
- RHIPE – R Hadoop Integrated Programming Environment
- HadoopStreaming
- SparkR

Open Source R (RHadoop)

K-Means Clustering example…

```
kmeans.mr =
  function( P, num.clusters, num.iter,
    combine,in.memory.combine) {
    dist.fun =
      function(C, P) {
        apply(C,1,function(x)
            colSums((t(P) - x)^2))}
    kmeans.map =
      function(., P) {
        nearest = {
          if(is.null(C))
            sample(1:num.clusters,
              nrow(P),
              replace = T)
          else {
            D = dist.fun(C, P)
            nearest = max.col(-D)}}
        if(!(combine || in.memory.combine))
          keyval(nearest, P)
        else
          keyval(nearest, cbind(1, P))}
    kmeans.reduce = {
      if (!(combine || in.memory.combine) )
        function(., P)
          t(as.matrix(apply(P, 2, mean)))
      else
        function(k, P)
          keyval(
            k, t(as.matrix(apply(P, 2, sum))))}
C = NULL
    for(i in 1:num.iter ) {
      C =
        values(
          from.dfs(
            mapreduce(
              P,
              map = kmeans.map,
              reduce = kmeans.reduce)))
      if(combine || in.memory.combine)
        C = C[, -1]/C[, 1]

      if(nrow(C) < num.clusters) {
        C =
          rbind(
            C,
            matrix(
              rnorm(
                (num.clusters -
                   nrow(C)) * nrow(C)),
              ncol = nrow(C)) %*% C) }}
        C}
out = list()
for(be in c("local", "hadoop")) {
  rmr.options(backend = be)
  set.seed(0)
kmeans.mr( to.dfs(P), num.clusters = 12, num.iter = 5, combine = FALSE, in.memory.combine = FALSE) }
```

MRS (Hadoop)

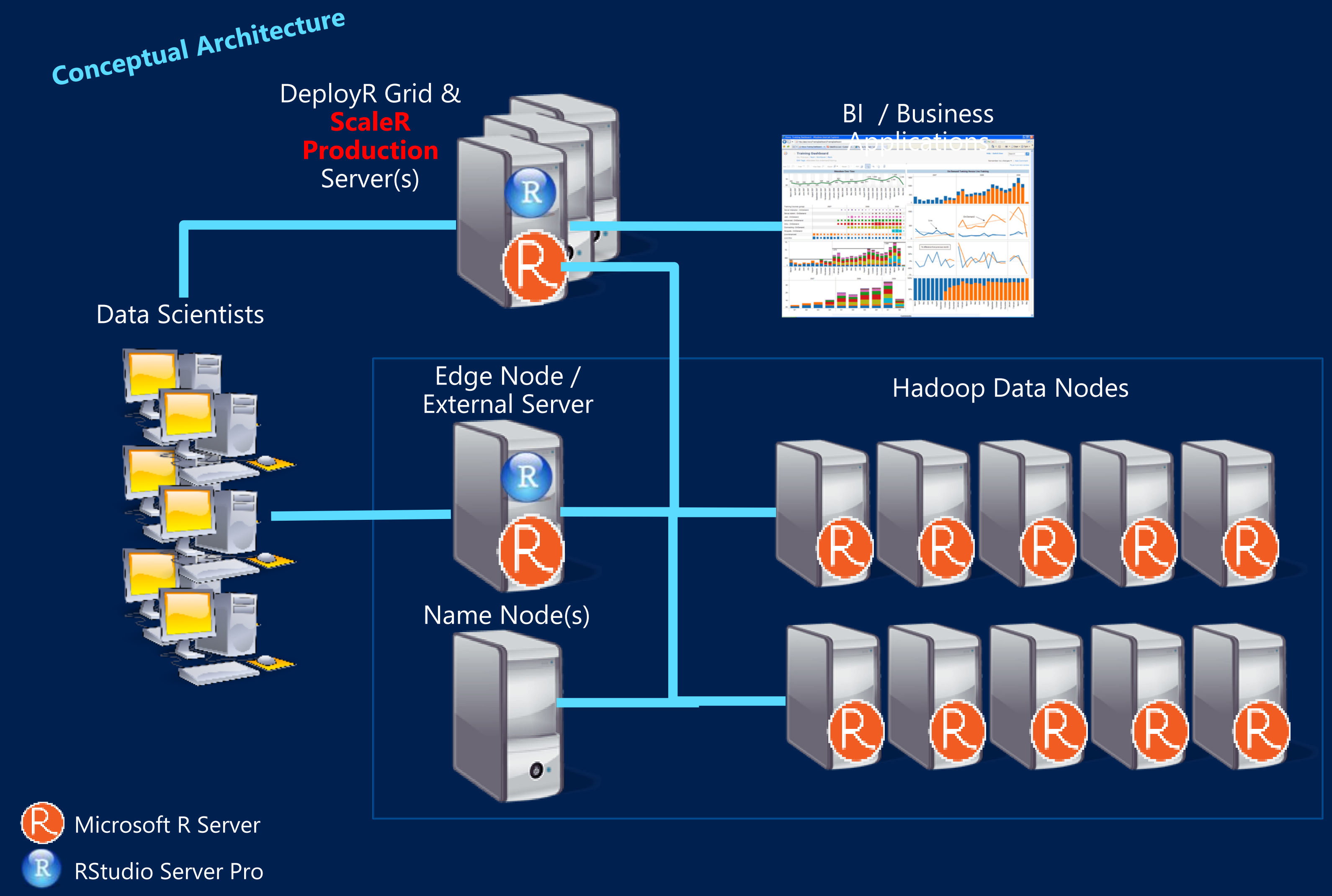
K-Means Clustering example…

```
rxSetComputeContext(myHadoopCluster)
rxKmeans(~ x + y,  data = p.xdf)
```

Features of MRS on Hadoop

- WODA – No need to recode into MapReduce or Spark
- Supports text delimited data files & Composite (sharded) XDF files
  - Reference directory path containing many files of same format!
- Utilises full parallelism of Hadoop (Mappers & Reducers) for fine-grained parallelism
  - Solve many model execution problems with full cluster parallelism (via rxExec)
- Operational
  - Configure and use YARN resource limits in Compute Context Definition
  - Supports Spark for Job-scheduling
  - Supports HDFS Caching for improved data read/write performance
- Processing platform flexibility – change compute-context and/or data source
  - Local compute context with ODBC to Hive/Impala/SparkSQL etc
  - Local compute context with local file copy
  - Local compute context with direct HDFS read (streaming)
  - Map-Reduce compute context with HDFS files/directories
  - High-Speed data extract support via Teradata Parallel Transporter
  - ODBC for small datasets
- Full Spark Compute Context (Roadmap 1H2016)
  - RDD's as a Data-Source
  - Populate RDDs from Hadoop data sources (e.g. Parquet, ORC, Avro, Composite XDF)
  - Integration and wrappers for existing open-source parallel modelling functions

MRS and Hadoop Architecture options



Microsoft R model run options:

1. **Copy** HDFS data to Edge Node / External Server Linux file system. Use "**Local Parallel**" compute context on that server to run model
2. **Stream** data from HDFS to Edge Node / External Server. Use "**Local Parallel**" compute context on that server to run model and discard data
3. **Send** the Microsoft R model script to run in every data node and return model output object to the Edge Node / External Server

Hadoop Processing Methods – Rx code

- Method 1 - Copy
  - rxSetComputeContext("RxLocalParallel")
  - rxHadoopCopyToLocal("myhadoopfile", "myfile")
  - fs <- RxFileSystem("native")
  - myFile <- RxTextData(..., fileSystem = fs)
  - rxSummary(~., myFile)
- Method 2 – Stream
  - rxSetComputeContext("RxLocalParallel")
  - fs <- RxFileSystem("hdfs")
  - myFile <- RxTextData(..., fileSystem = fs)
  - rxSummary(~., myFile)
- Method 3 – Send to Map Reduce
  - rxSetComputeContext("RxHadoopMR")
  - fs <- RxFileSystem("hdfs")
  - myFile <- RxTextData(..., fileSystem = fs)
  - rxSummary(~., myFile)
- Method 4 – Send to Spark
  - rxSetComputeContext("RxSpark")
  - fs <- RxFileSystem("hdfs")
  - myFile <- RxTextData(..., fileSystem = fs)
  - rxSummary(~., myFile)

New

OR

rxODBCData to Hive, Impala data sources etc

OR

rxSQLServerData to Polybase [external table to Hadoop]

New

Microsoft

Demo: MRS on Hadoop

R Server on Hadoop Spark

New

Higher Performance

- Hadoop MR is slowed by the use of file-based cluster communication
- Spark achieves significantly higher performance through efficient use of distributed memory
  - Resilient Distributed Dataframes (RDD) and Spark Dataframes

Improved Data Access

- R Server on Hadoop MR is limited to text-based data sources
- Spark SQL enables the use of queries to load data from a variety of sources into distributed memory, e.g. HIVE, Parquet, etc.

R Server on Hadoop Spark

- Inherit high-performance big-data processing characteristics from Spark
- Easily integrate with different data sources on Hadoop
- Develop R scripts w/o requiring a detailed understanding of Spark
- Re-use existing R scripts in Spark with only minor changes

Example Code – Using rxSpark()

New

Switching to use of Spark in R Server is straightforward

Using a Spark compute context

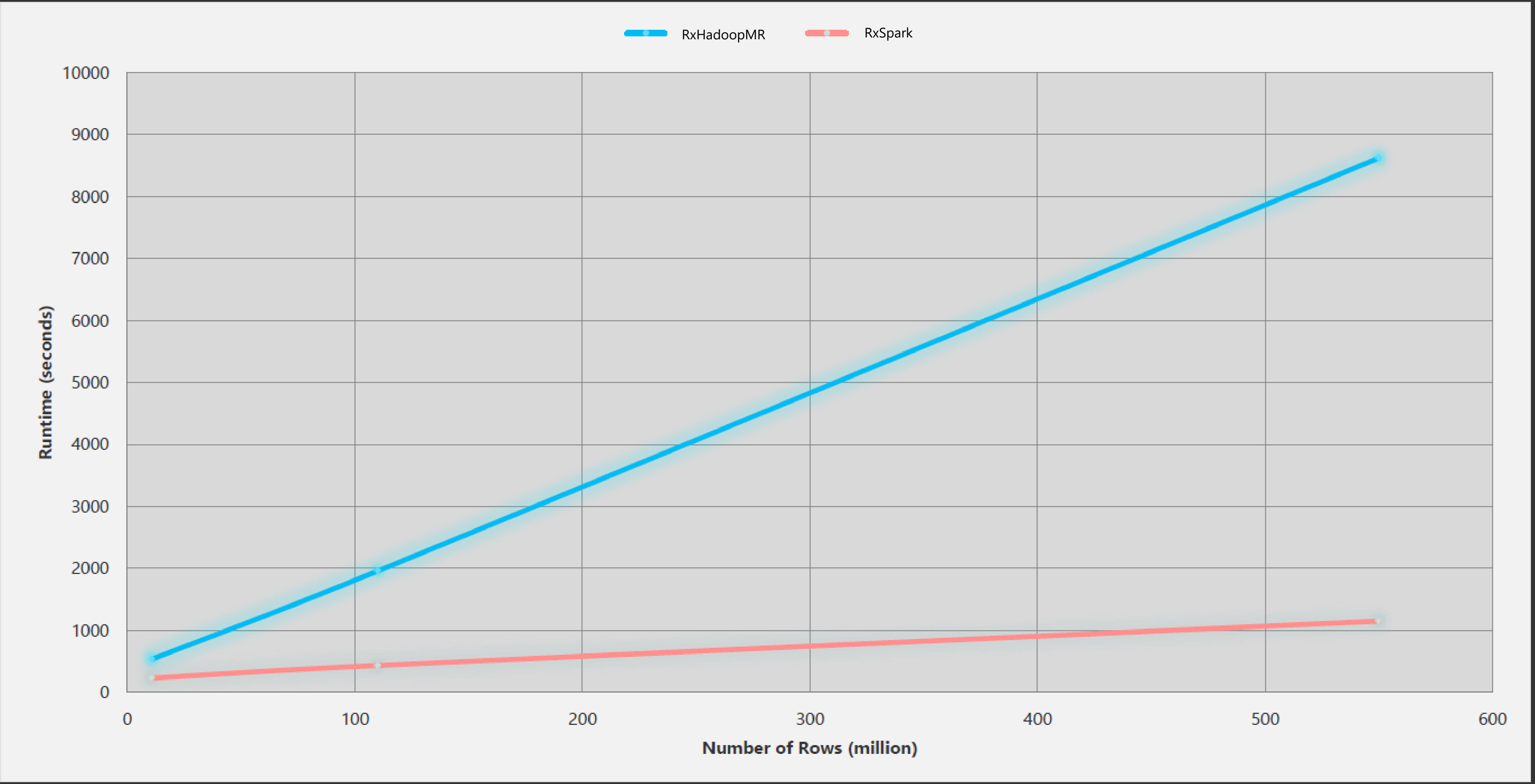
Keep other code unchanged

Sample R Script:

```
rxSetComputeContext(RxSpark(...))
inData <- RxTextData("/ds/AirOnTime.csv", fileSystem = hdfsFS)
model <- rxLogit(DEP_DEL15 ~ DAY_OF_WEEK + UNIQUE_CARRIER, data = inData)
```

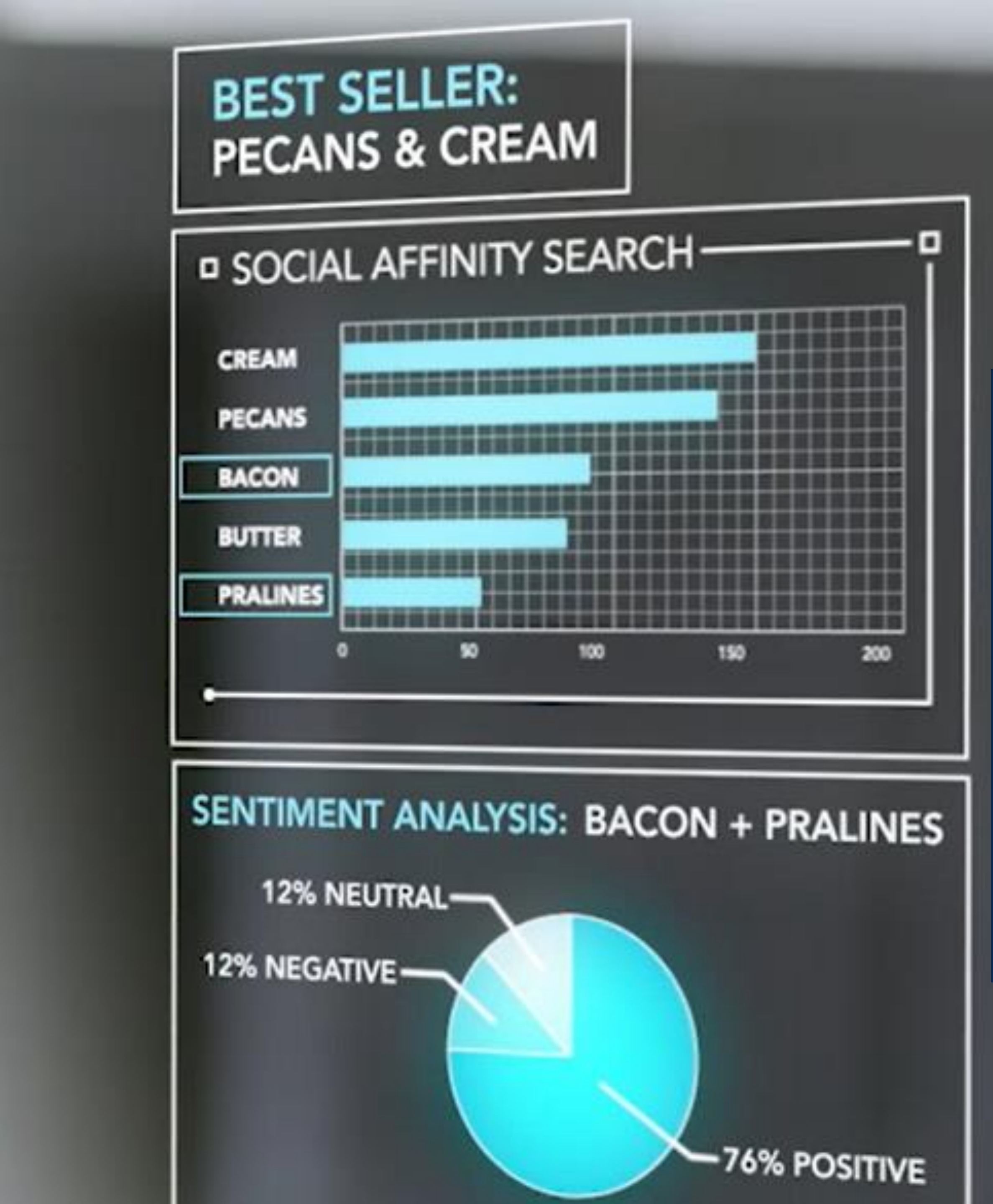
Comparing R Server on MR and Spark

New



MRS On Hadoop

Lab Overview



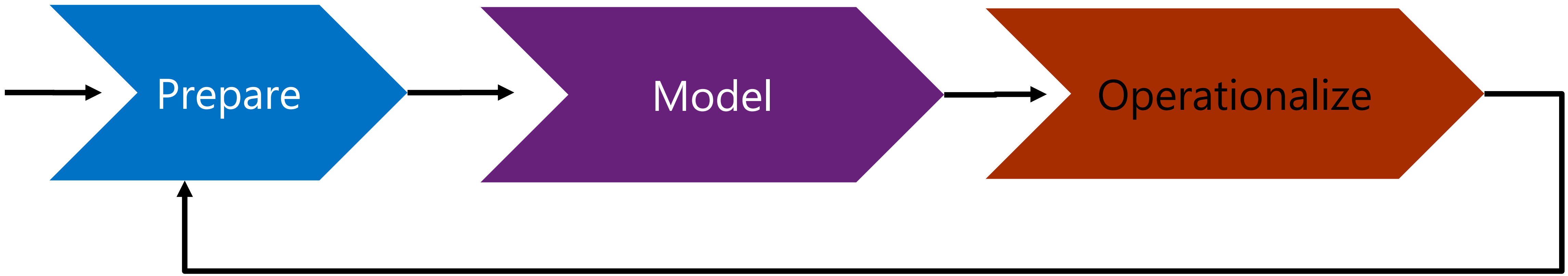
Microsoft

Typical advanced analytics lifecycle

Prepare: Assemble, cleanse, profile and transform diverse data relevant to the subject.

Model: Use statistical and machine learning algorithms to build classifiers and make predictions

Operationalize: Apply predictions and visualizations to support business applications



Airline Arrival Delay Prediction Demo

- Get Rstudio Server set up on the edge node
- Clean/Join – Using SparkR from R Server
- Train/Score/Evaluate – Scalable R Server functions
- Deploy/Consume – Using AzureML from R Server

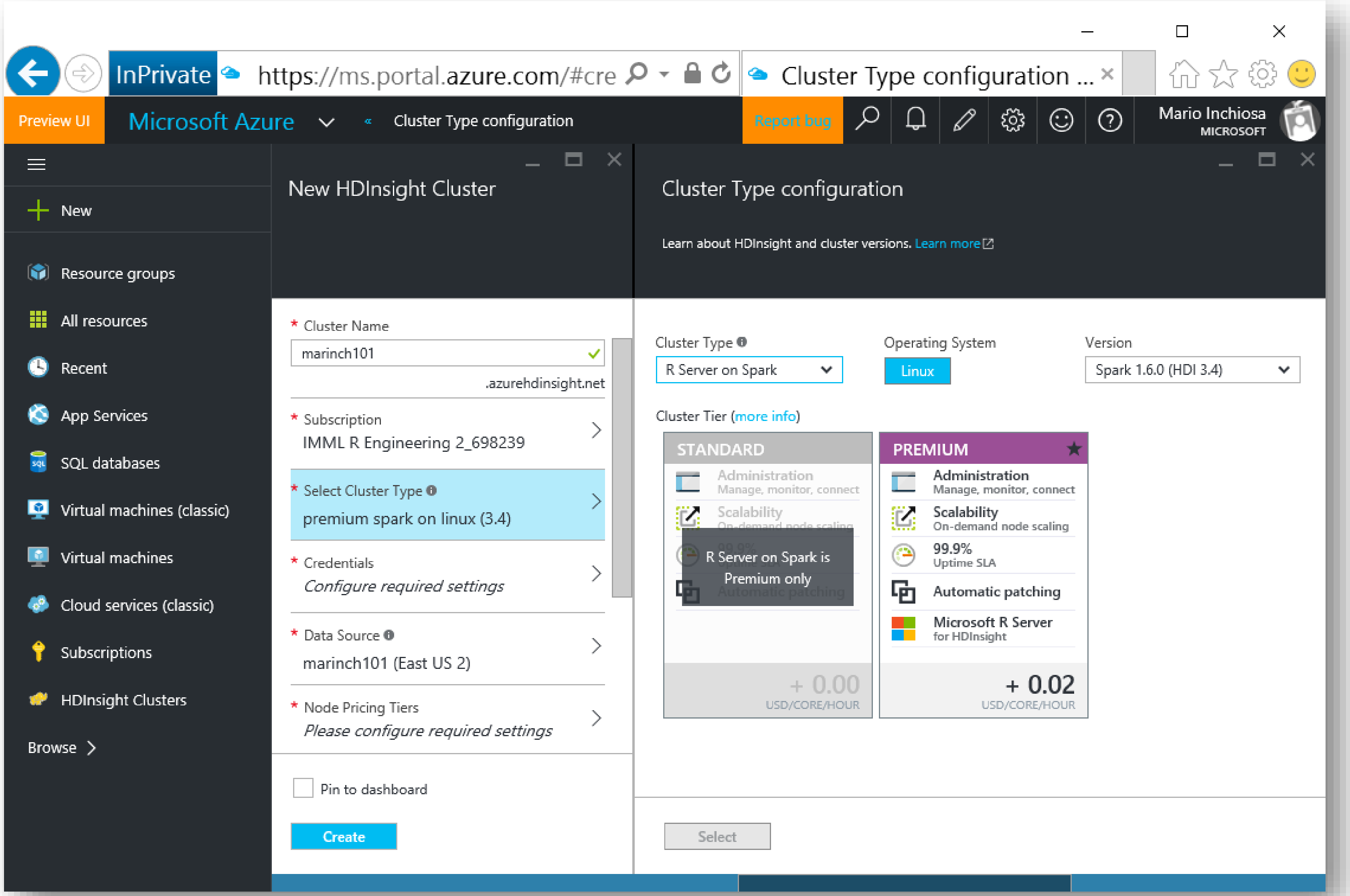
Airline data set

- Passenger flight on-time performance data from the US Department of Transportation’s TranStats data collection
- >20 years of data
- 300+ Airports
- Every carrier, every commercial flight
- http://www.transtats.bts.gov

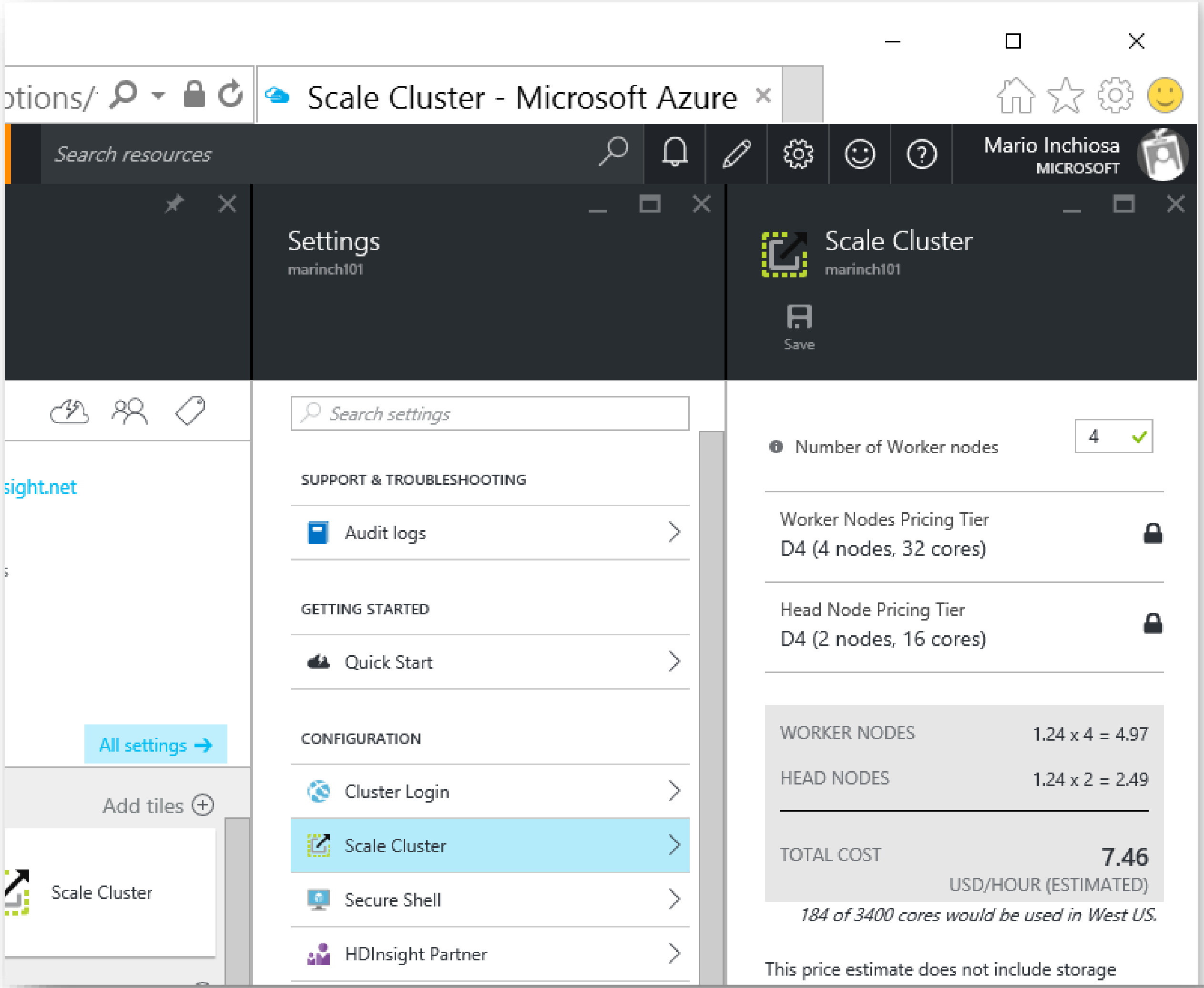
Weather data set

- Hourly land-based weather observations from NOAA
- > 2,000 weather stations
- http://www.ncdc.noaa.gov/orders/qclcd/

Provisioning a cluster with R Server



Scaling a cluster



Clean and Join using SparkR in R Server

```r
#######################################################
# Join airline data with weather at Origin Airport
#######################################################

joinedDF <- SparkR::join(
  airDF,
  weatherDF,
  airDF$OriginAirportID == weatherDF$AirportID &
    airDF$Year == weatherDF$AdjustedYear &
    airDF$Month == weatherDF$AdjustedMonth &
    airDF$DayofMonth == weatherDF$AdjustedDay &
    airDF$CRSDepTime == weatherDF$AdjustedHour,
  joinType = "left_outer"
)
```

Train, Score, and Evaluate using R Server

```
###############################################
# Train and Test a Decision Tree model
###############################################

# Train using the scalable rxDTree function

dTreeModel <- rxDTree(formula, data = trainDS,
                      maxDepth = 6, pruneCp = "auto")

# Test using the scalable rxPredict function

rxPredict(dTreeModel, data = testDS, outData = treePredict,
          extraVarsToWrite = c("ArrDel15"), overwrite = TRUE)
```

Publish Web Service from R

```r
################################################
# Publish the scoring function as a web service
################################################

library(AzureML)

workspace <- workspace(config = "azureml-settings.json")

endpoint <- publishWebService(workspace, scoringFn,
                              name="Delay Prediction Service",
                              inputSchema = exampleDF)

################################################
# Score new data via the web service
################################################

scores <- consume(endpoint, dataToBeScored)
```

Microsoft