DYNAMIC ON-THE-FLY MODIFICATIONS OF SPARK APPLICATIONS

Elena Lazovik

TNO (The Netherlands)

SPARK SUMMIT EUROPE 2016

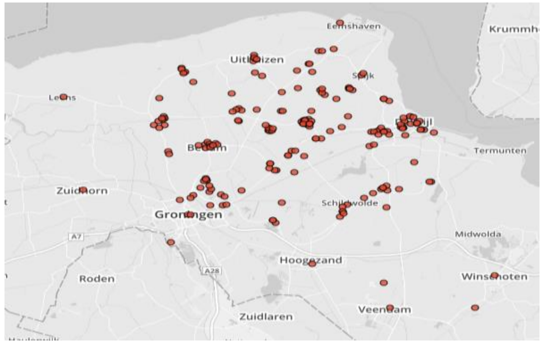
Self.me

- **Scientist innovator**
  Monitoring & Control Services group
  @TNO, from January 2012

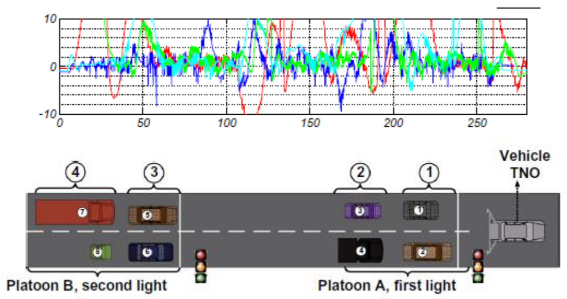
Applied Research Organization of NL

TNO innovation for life

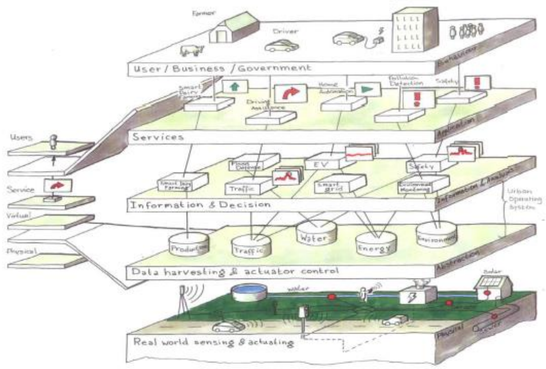
Earthquakes real-time monitoring:
350 buildings in Groningen (NL)



Cooperative automated driving



Live monitoring of water and gas pipes for whole NL


User / Business / Government
Services
Information & Decision
Data harvesting & actuator control
Real world sensing & actuating
Traffic
Water
Energy

Joint work: TNO and RuG

Prof. dr. Alexander Lazovik

Elena Lazovik

TNO innovation for life

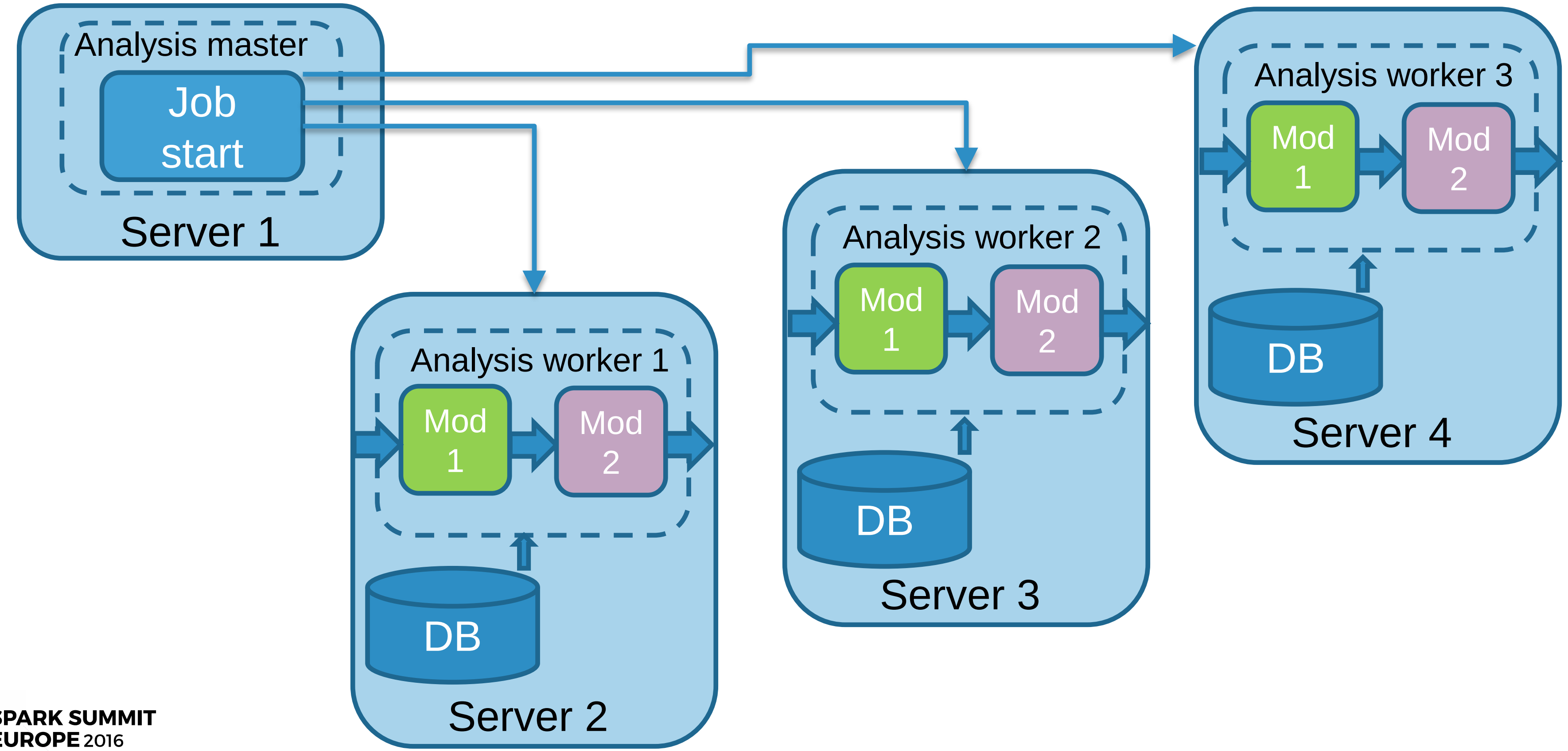
Erik Langius

university of groningen

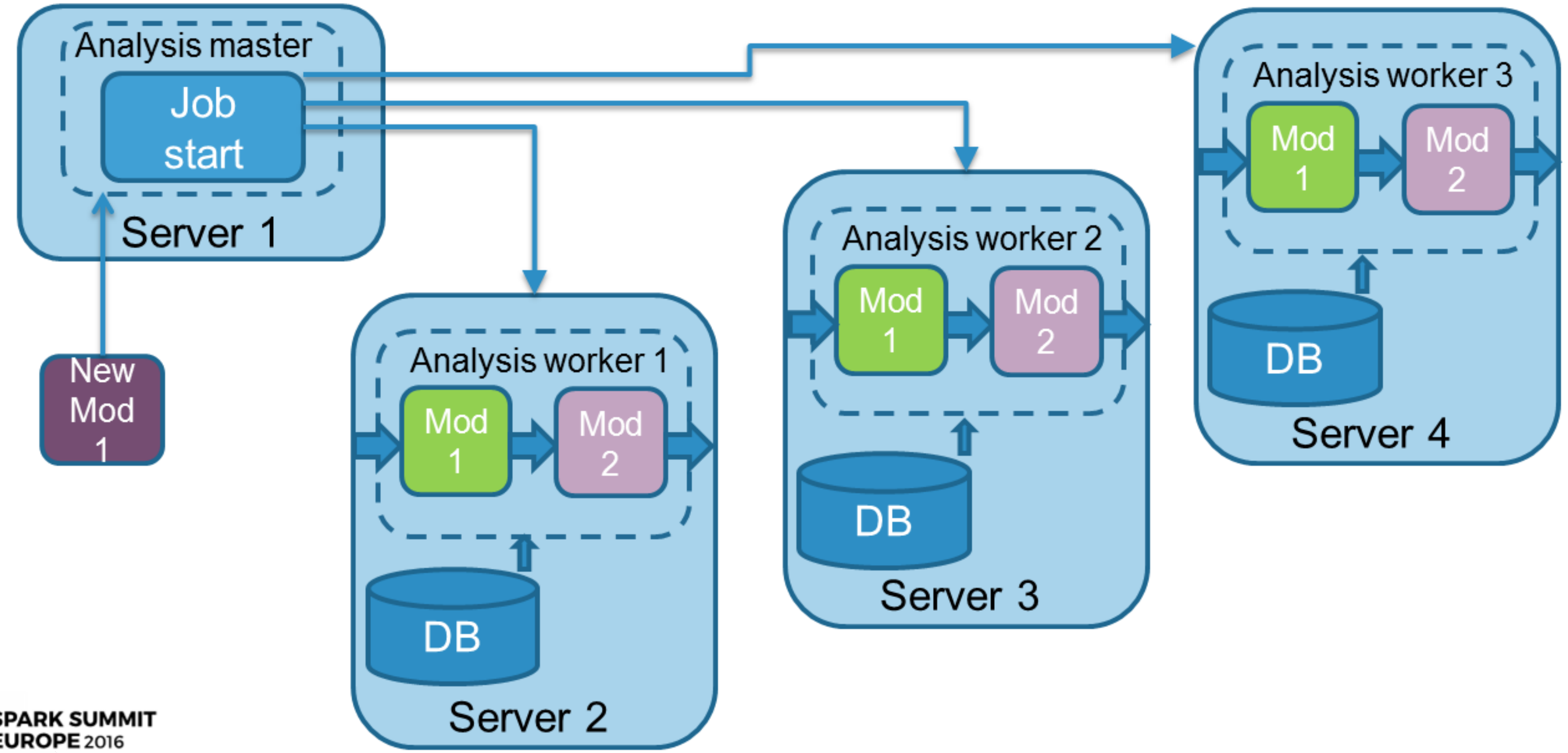
Michel Medema

Toon Albers

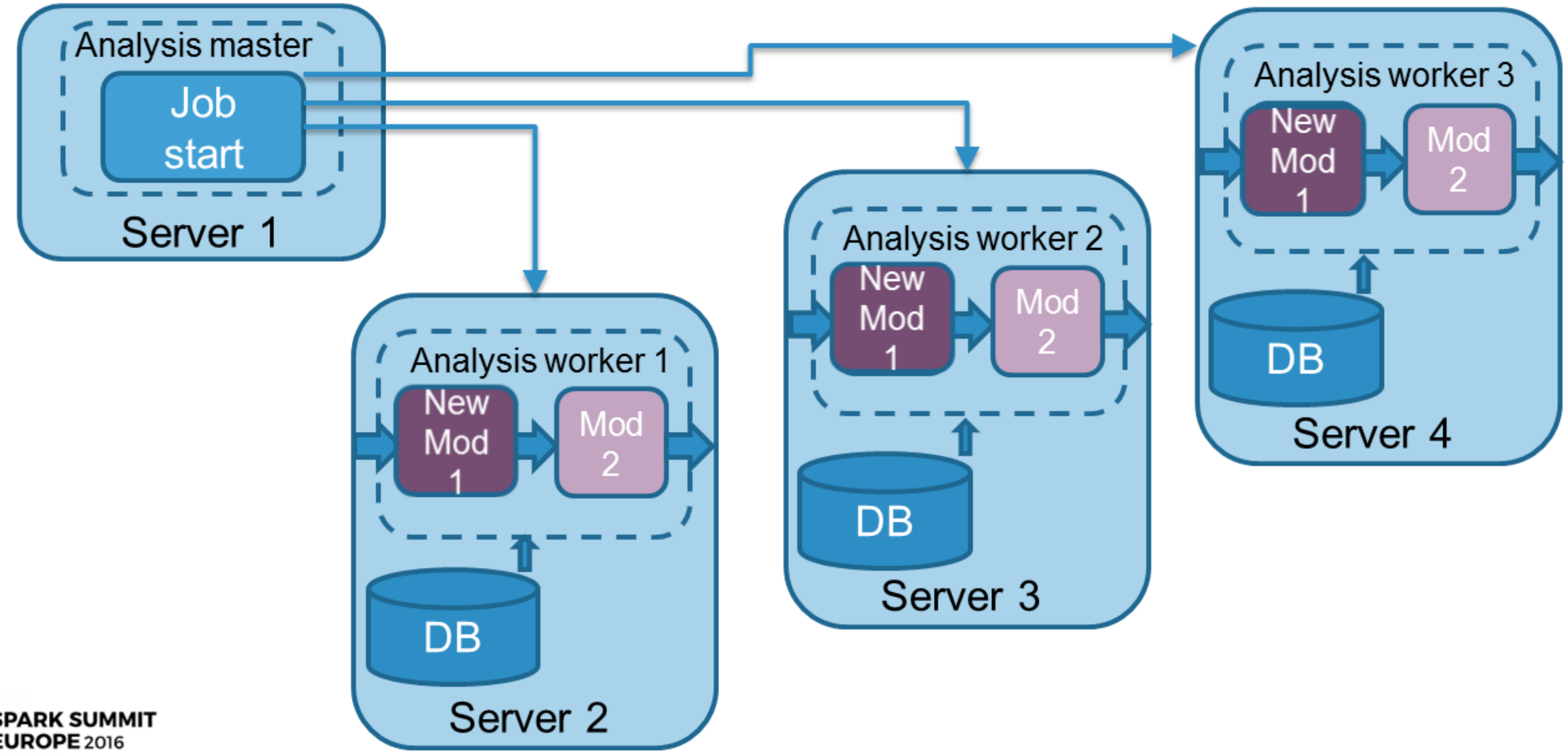
Default Spark application



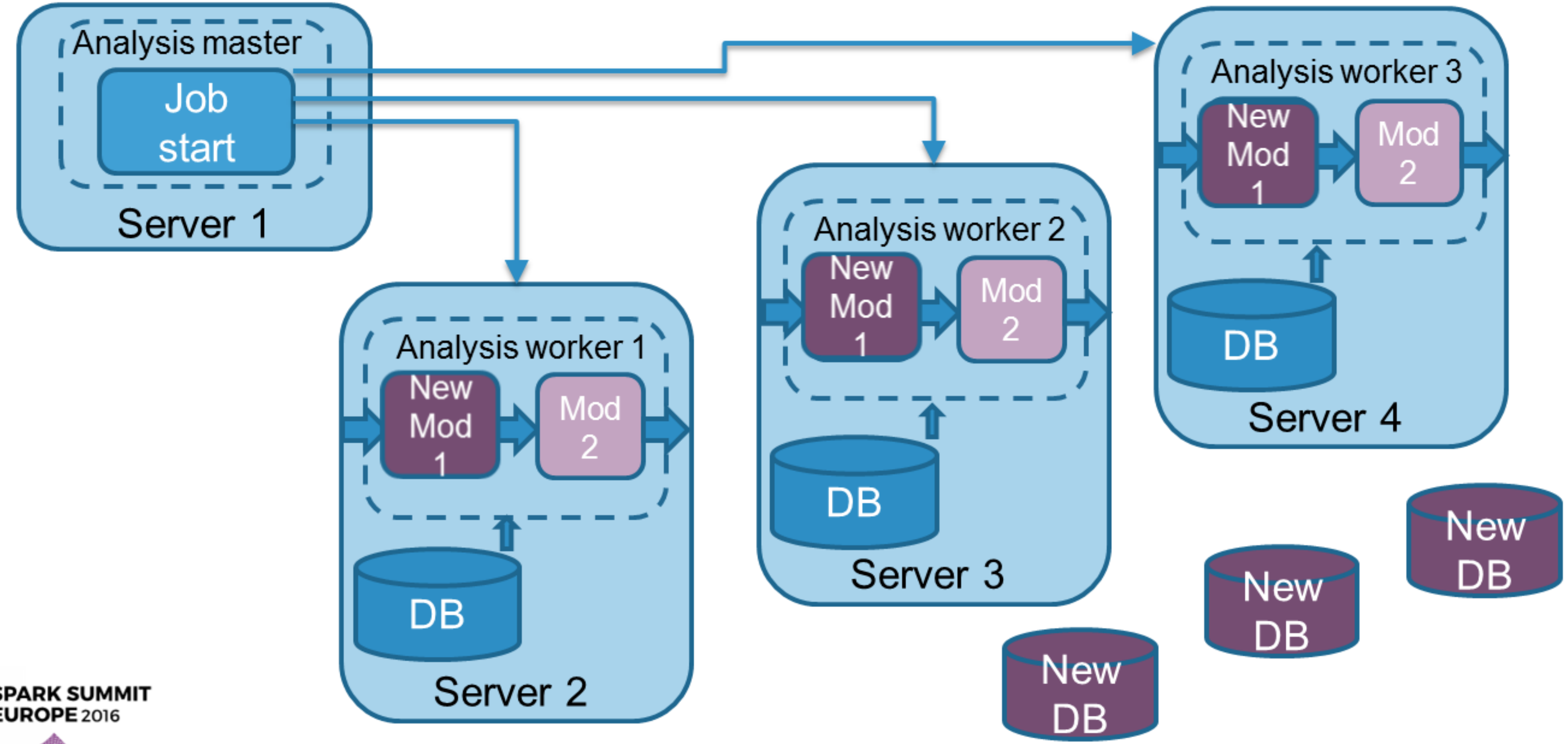
Dynamic On-the-Fly changes 1/4



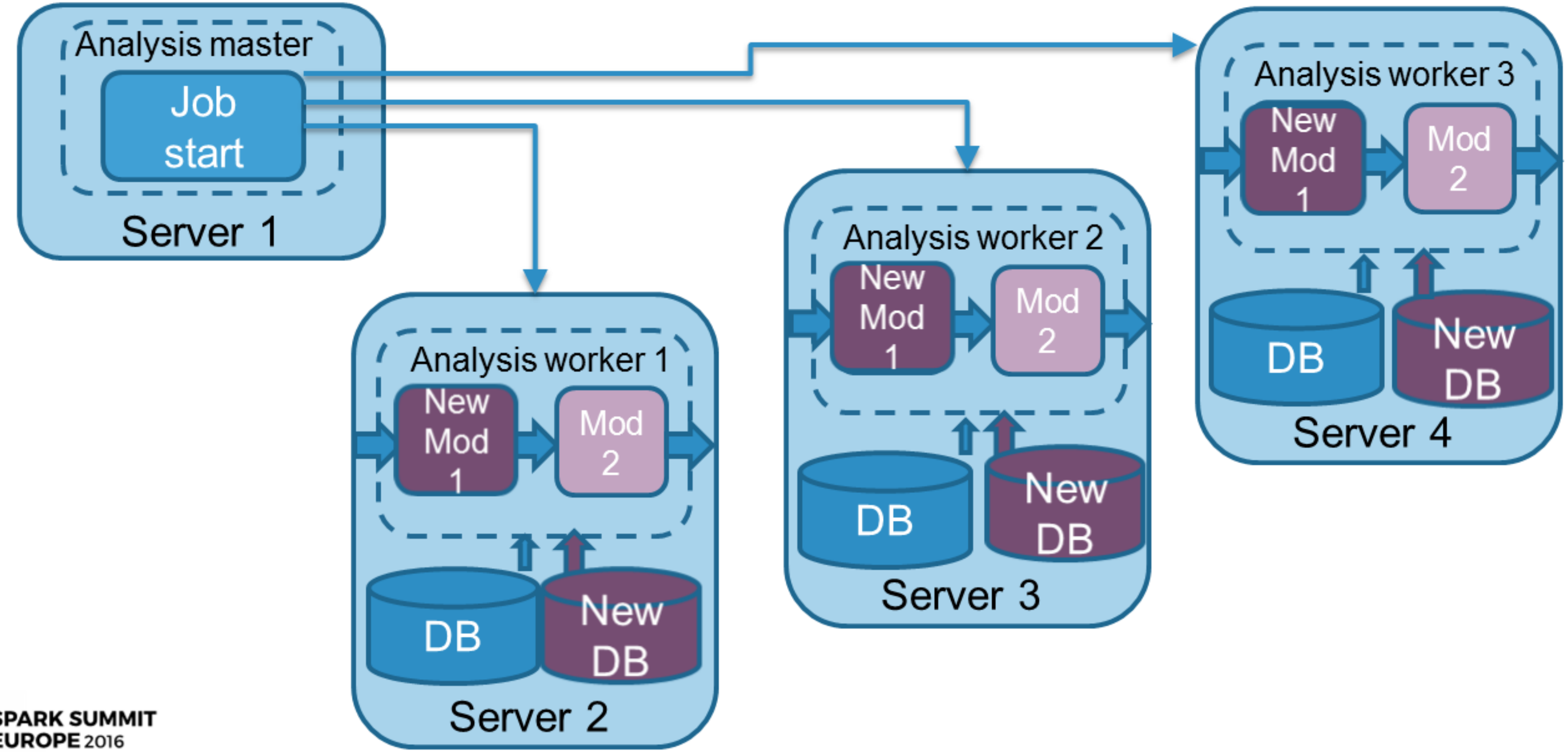
Dynamic On-the-Fly changes 2/4



Dynamic On-the-Fly changes 3/4

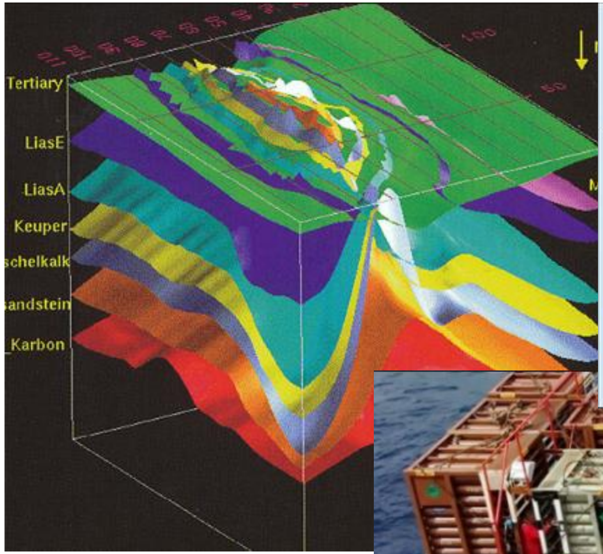


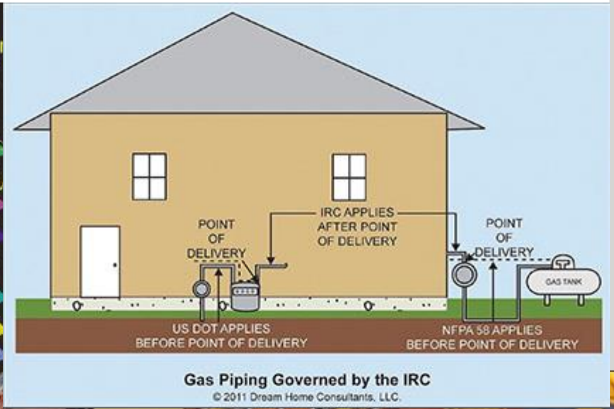
Dynamic On-the-Fly changes 4/4



Why is it needed?

Long Beach, CA




POINT OF DELIVERY
IRC APPLIES AFTER POINT OF DELIVERY
POINT OF DELIVERY
GAS TANK
US DOT APPLIES BEFORE POINT OF DELIVERY
NFPA 58 APPLIES BEFORE POINT OF DELIVERY
Gas Piping Governed by the IRC
© 2011 Dream Home Consultants, LLC.

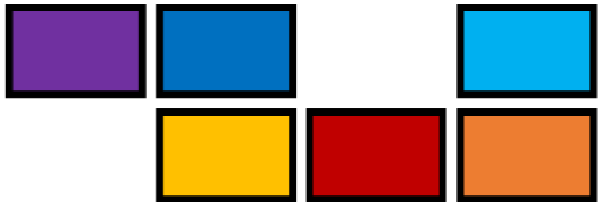
Dynamic changes of functions

- Given a Spark driver program:
  - We want to change a function/parameter in scenario
  - We want to switch one data source to another

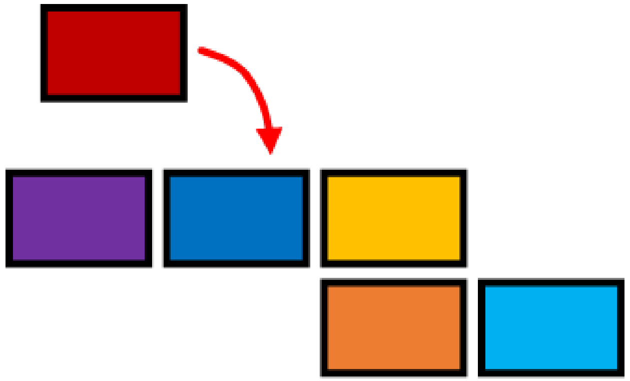
- without stopping the whole application!!!

What can we change?

- Spark functional steps
  - `map`, `reduce`, filter, reduceByKey …

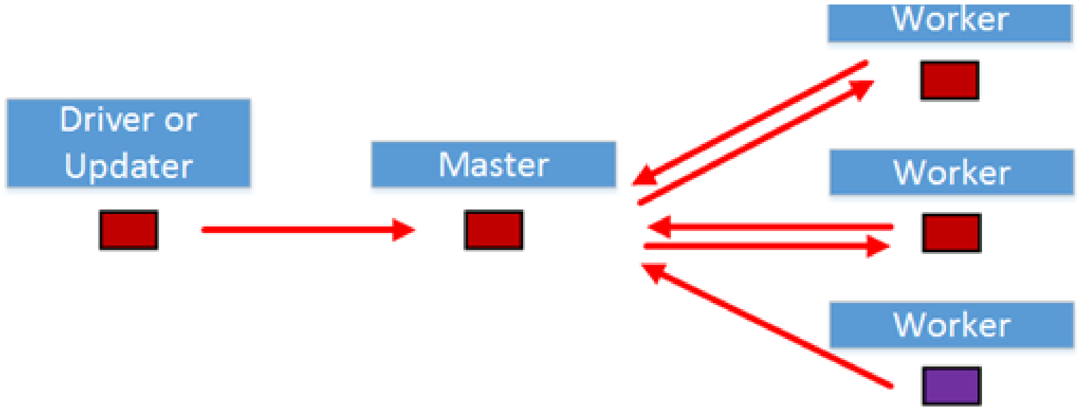


- Functional step content
  - Variables
  - Code (x => x + 1)



How to update?

- Push/pull?
- Variables: serialize
- Code: bytecode?



Typical calculation Spark scenario

```
val conf = new SparkConf()
val sc = new SparkContext(conf)




for (itt <- 1 until 100) {
  val rdd = sc.parallelize(1 until 1000, 10)
  val result = rdd
        .map(i => i * 10)
        .reduce(_ + _)
  println(s"$itt: $result")
}

sc.stop()
```

We want to change hardcoded parameter 10

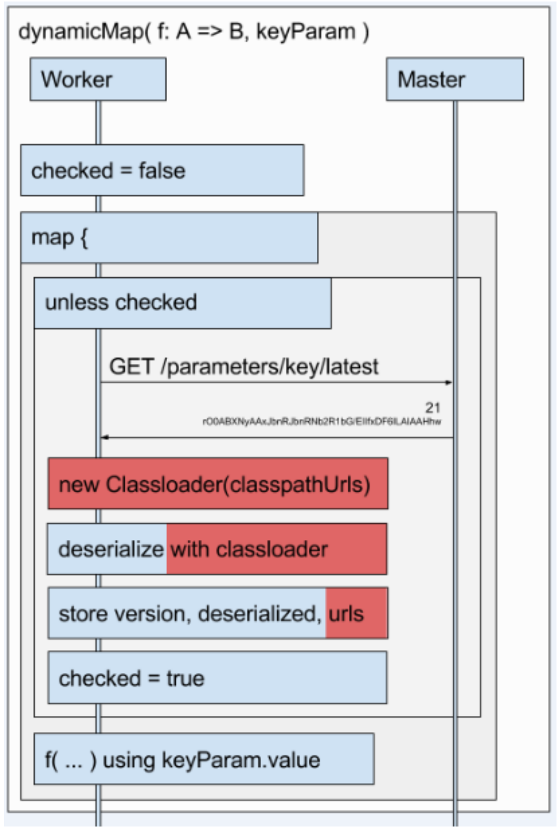
Introducing dynamic-spark

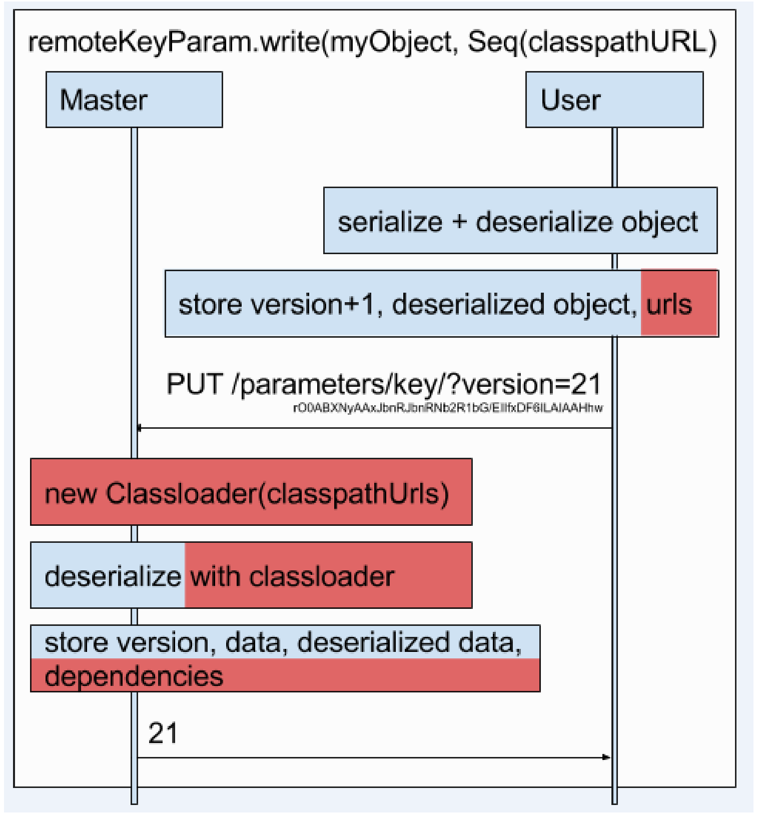
```
val conf = new SparkConf().set("dynamic.server.port", "8090")
val sc = new SparkContext(conf)
val server = new DynamicServer(conf, conf.getInt("dynamic.server.port", 0))
server.start()
val factor = new RESTParameter(server, key = "factor", version = 1, value = 10)
for (itt <- 1 until 100) {
  val rdd = sc.parallelize(1 until 1000, 10)
  val result = rdd
        .dynamicMap({ i => i * factor.value }, factor)
        .reduce(_ + _)
  println(s"$itt: $result")
}
sc.stop()
server.stop()
```

Changing parameter of the function

- val factor = new RemoteRESTParameter(serverUri, "factor", version = 1, value = 10)
- …
- factor.write(15)   // set new value and increment version
- …

DynamicMap flow


dynamicMap(f: A => B, keyParam)
Worker
Master
checked = false
map {
unless checked
GET /parameters/key/latest
21
rO0ABXNyAAxJbnRJbnRNb2R1bG/EllfxDF6ILAIAAHhw
new Classloader(classpathUrls)
deserialize with classloader
store version, deserialized, urls
checked = true
f(...) using keyParam.value


remoteKeyParam.write(myObject, Seq(classpathURL)
Master
User
serialize + deserialize object
store version+1, deserialized object, urls
PUT /parameters/key/?version=21
rO0ABXNyAAxJbnRJbnRNb2R1bG/EllfxDF6ILAIAAHhw
new Classloader(classpathUrls)
deserialize with classloader
store version, data, deserialized data, dependencies
21

Dynamic RDD functions

```
def wrap[A, V](f: A => V, params: DynamicParameter[_]*): A => V = {
    (a: A) => {
      updateParam(params)
      f(a)
    }
  }

def dynamicMap(f: A => B, params: DynamicParameter[_]*): RDD[B] = {
      // Same as RDD.map, but wrapped to check for updates
      val g = wrap(f, params: _*)
      rdd.map(g)
    }
```

And dynamicReduce, dynamicFilter, dynamicReduceByKey, etc.

Passing dynamic function

```
// This object is merely defined as a place to store the functions
object FuncHolder {
  // A value referencing a function
  val compareFunc = (num:Double) => num < 0.1
  // A method that is executed by the client, which returns a
  // function that is to be serialized
  def getFunc:(Double => Boolean) = {
    (num:Double) => num < 0.2
  }
}

val funcParam = new RESTParameter[(Double) => Boolean](server, "filterfunc",
                                    1, (n:Double) => n < 0 )
param.write( FuncHolder.compareFunc,    Seq("file://my/libs/func_holder.jar") )
param.write( FuncHolder.getFunc,        Seq("file://my/libs/func_holder.jar") )
// Here the bytecode is present in the user's client JAR:
param.write( (num:Double) => num < 0.3, Seq("file://this_client.jar") )
```

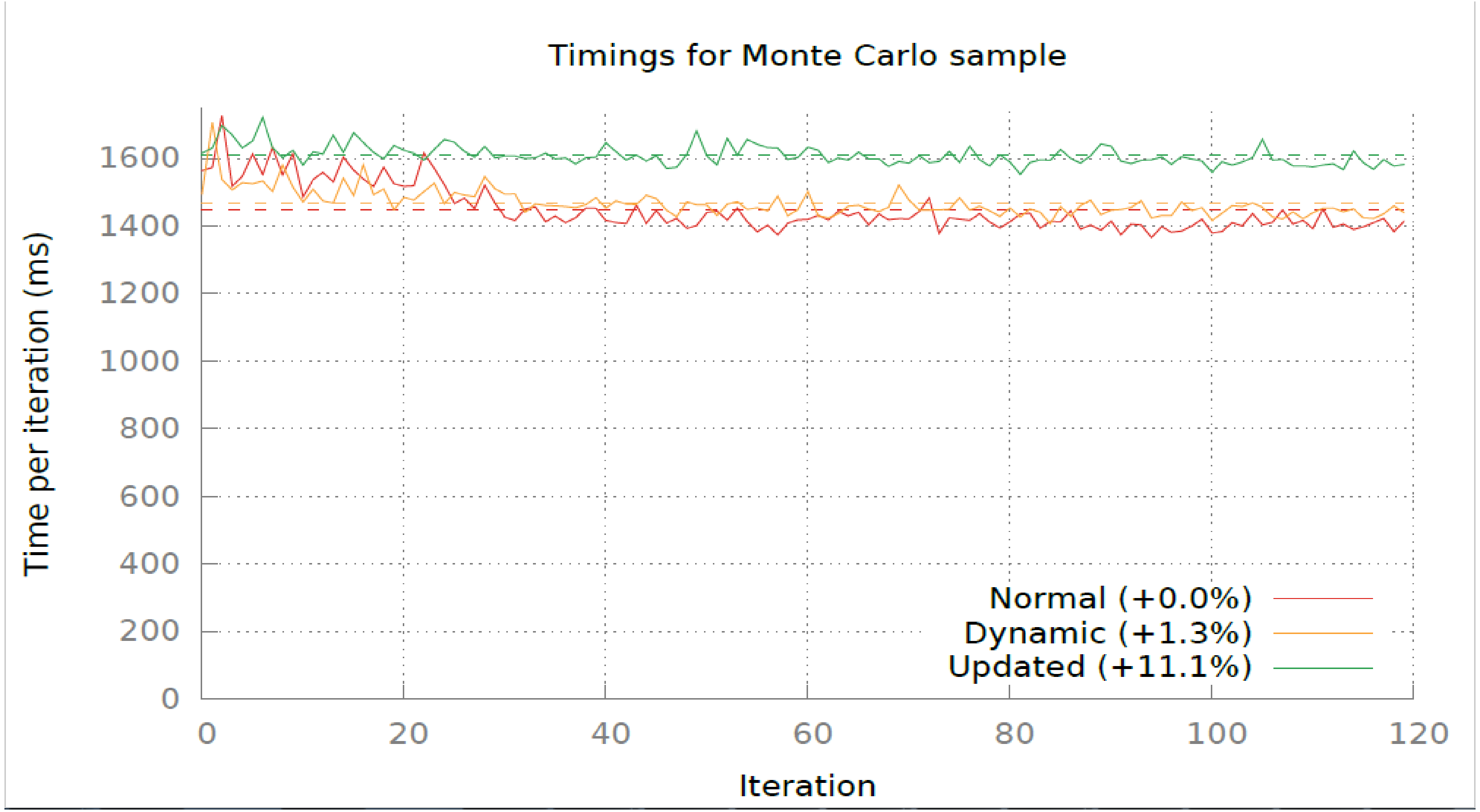
Experiments for dynamic Monte Carlo

$P_f = P(Z<0)$

Limit state function:

$$Z = R - \sum_{i=1}^{25} S_i^2/i$$

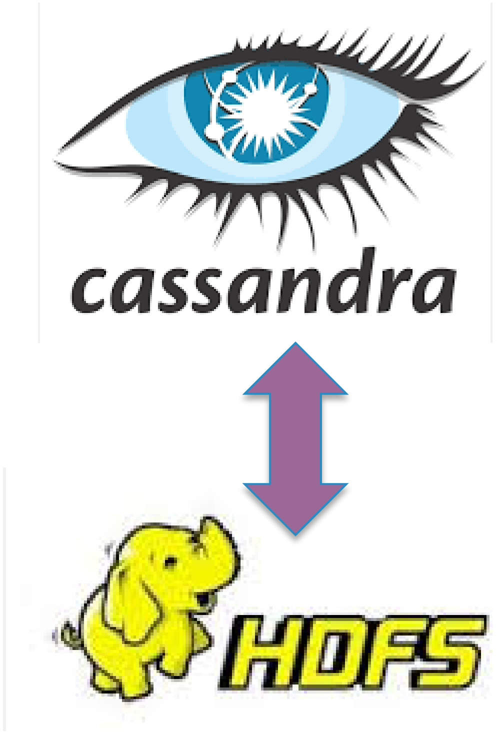
6 physical servers



Dynamic switching of data sources

- Why use dynamic switching?
  - Data source becomes unavailable
  - More accurate data needed

- Research questions
  - Is it feasible?
  - What is the best approach?



Approaches for data sources switch

Two approaches considered

- Extending Apache Spark
- Intermediary system Solution

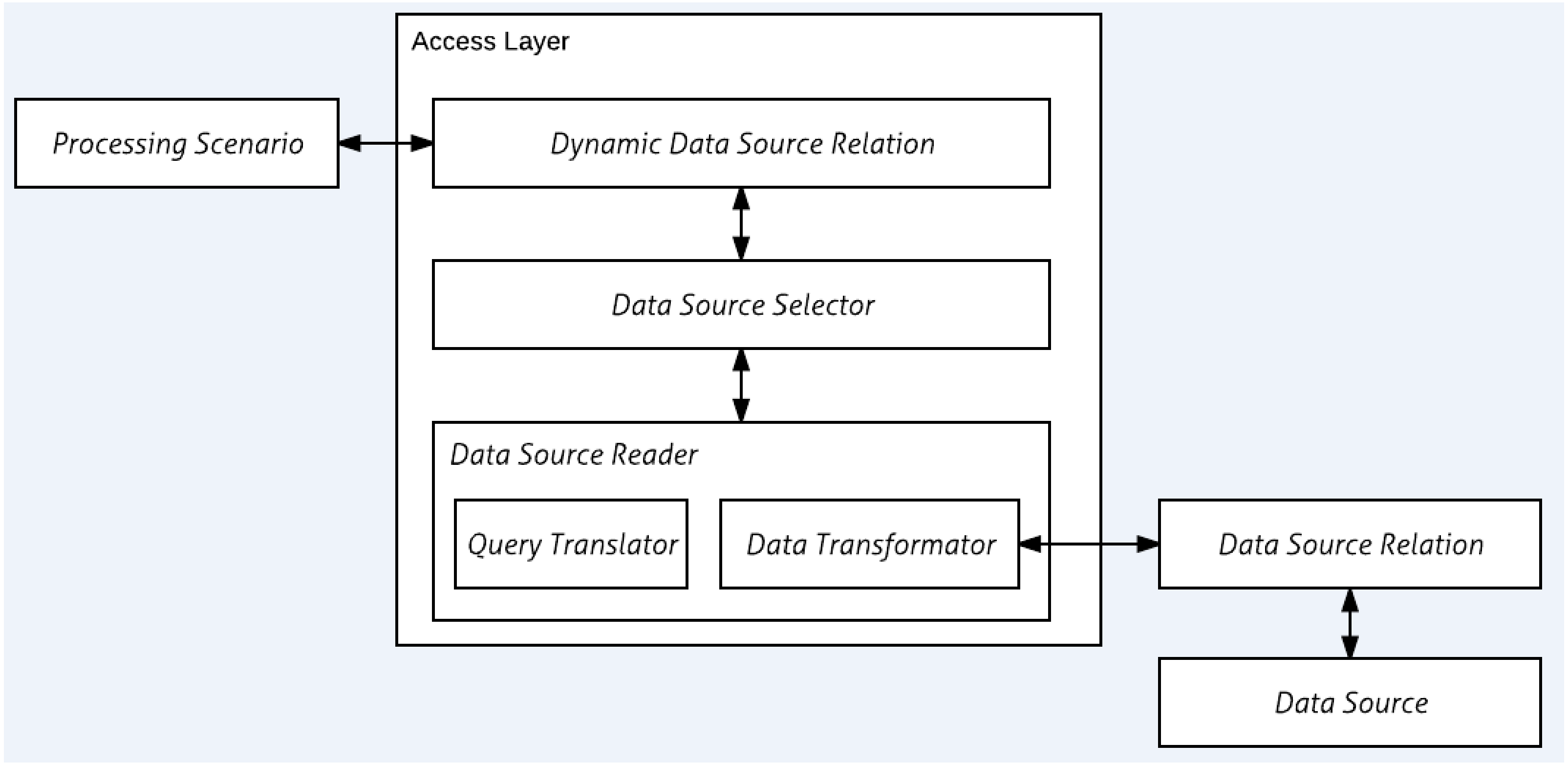
Compare based on key requirements

- Performance
- Usability
- Flexibility
- Efficiency
- Extensibility

Dynamic data sources switching

- Issues with dynamic switching
  - Heterogeneous data sources
    - Accessing Data Source
    - Query Translation
    - Data Transformation
  - Data Locality
    - Process close to data
  - Deployment

Dynamic Data Source API



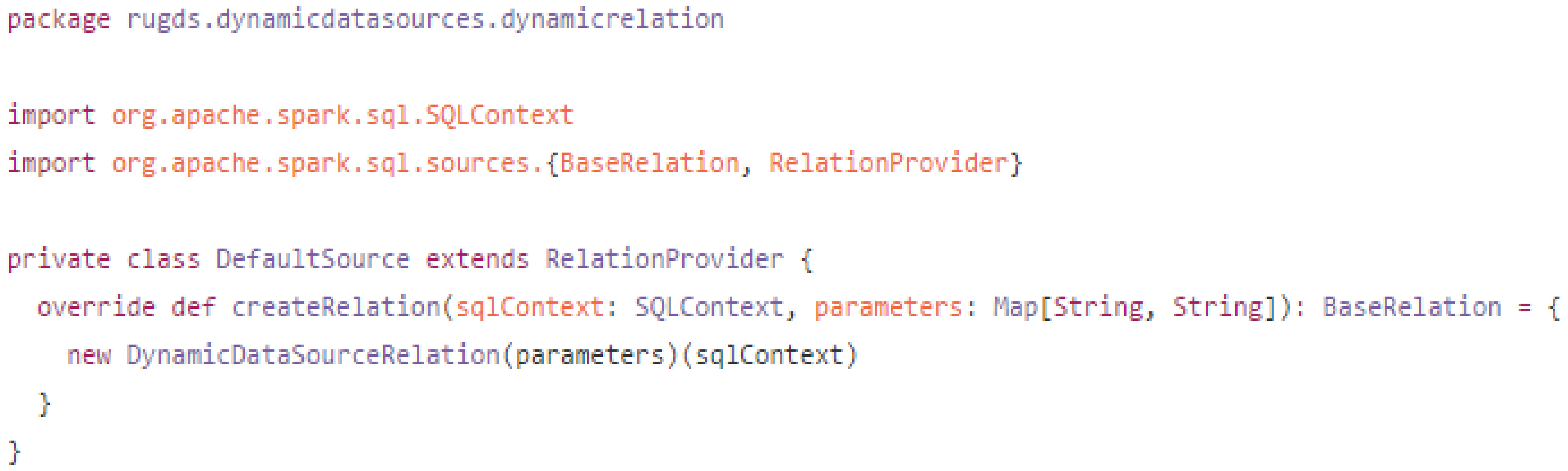
Custom Spark Data Source

- Makes use of the Spark Data Source API: RelationProvider and BaseRelation classes

- Requires just two classes
  - Data Source provider, extending the "RelationProvider"
  - Data Source relation, extending e.g. "BaseRelation"

- Re-uses existing Spark data source relations

Custom Relation Provider



```scala
package rugds.dynamicdatasources.dynamicrelation

import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.sources.{BaseRelation, RelationProvider}

private class DefaultSource extends RelationProvider {
  override def createRelation(sqlContext: SQLContext, parameters: Map[String, String]): BaseRelation = {
    new DynamicDataSourceRelation(parameters)(sqlContext)
  }
}
```

Custom Data Source Relation

```
package rugds.dynamicdatasources.dynamicrelation

import org.apache.spark.rdd.RDD
import org.apache.spark.sql.{Row, SQLContext}
import org.apache.spark.sql.sources.{BaseRelation, Filter, PrunedFilteredScan}
import org.apache.spark.sql.types._
import rugds.Logging
import rugds.dynamicdatasources.dynamicrelation.datasources.readers.DataSourceReader

case class DynamicDataSourceRelation(parameters: Map[String, String])(@transient val sqlContext: SQLContext) extends BaseRelation with PrunedFilteredScan with Logging {
  // Schema that the data should adhere to.
  val schema : StructType = StructType(Seq(
    StructField("value", DoubleType, nullable = false),
    StructField("timestamp", LongType, nullable = false)
  ))
  override def buildScan(requiredColumns: Array[String], filters: Array[Filter]): RDD[Row] = {
    // Select a data source.
    val reader: DataSourceReader = new DataSourceSelector().selectDataSource(sqlContext, schema)

    // Retrieve the RDD
    reader.read
  }
}
```

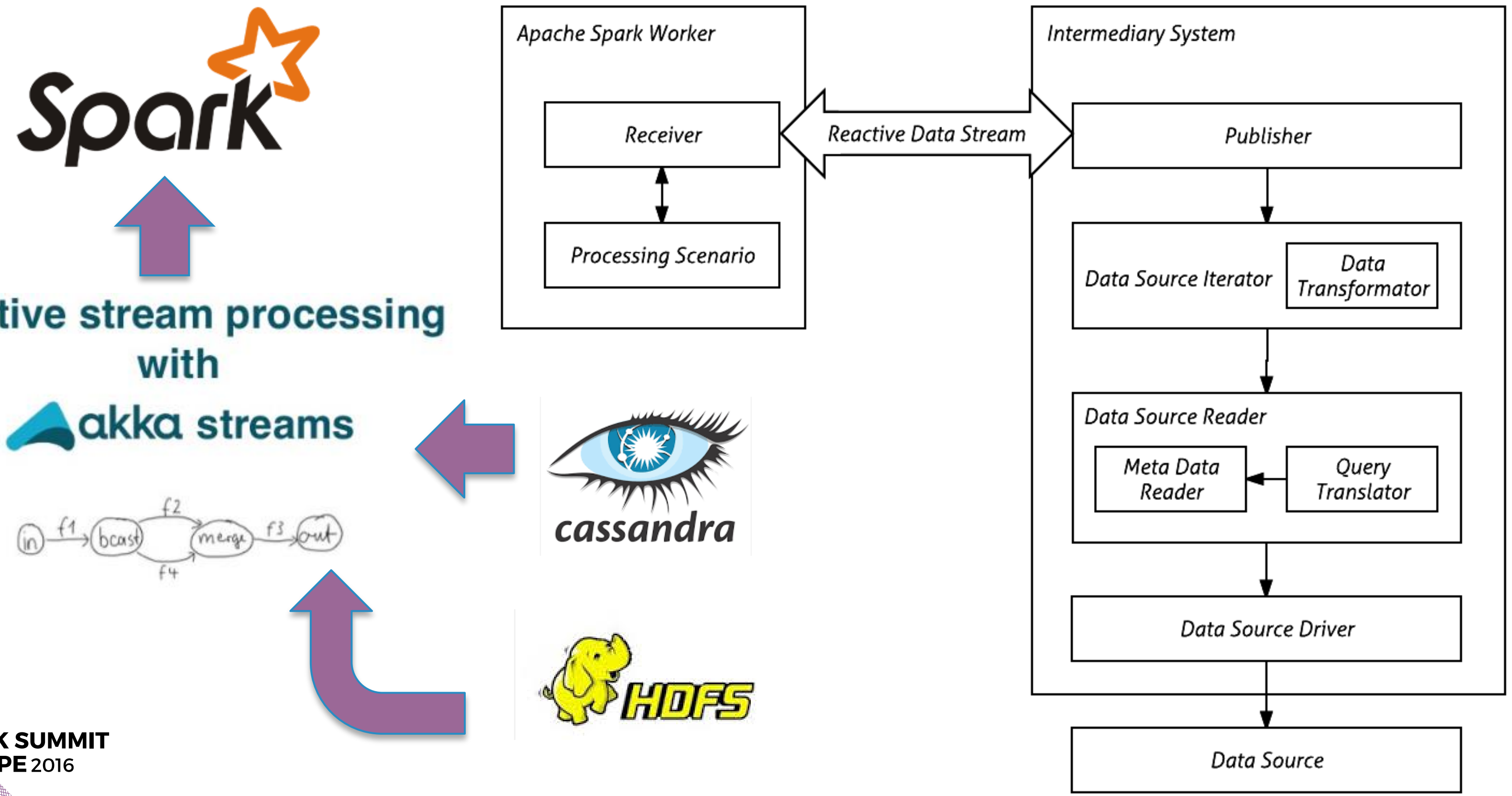
Using The Custom Source

Only **one** line different from normal usage

```
val conf = new SparkConf().setAppName("Dynamic Data Source Client")
val sc = new SparkContext(conf)
val sqlContext = new SQLContext(sc)
val rdd: RDD[String] = sqlContext.sparkContext.textFile(“file.csv")
```

```
val conf = new SparkConf().setAppName("Dynamic Data Source Client")
val sc = new SparkContext(conf)
val sqlContext = new SQLContext(sc)
val rows: DataFrame = sqlContext.read.dynamicSource("")
```

Intermediary system version



Experiments results for data sources

Data sources:

- HDFS
- Apache Cassandra

Intermediary:

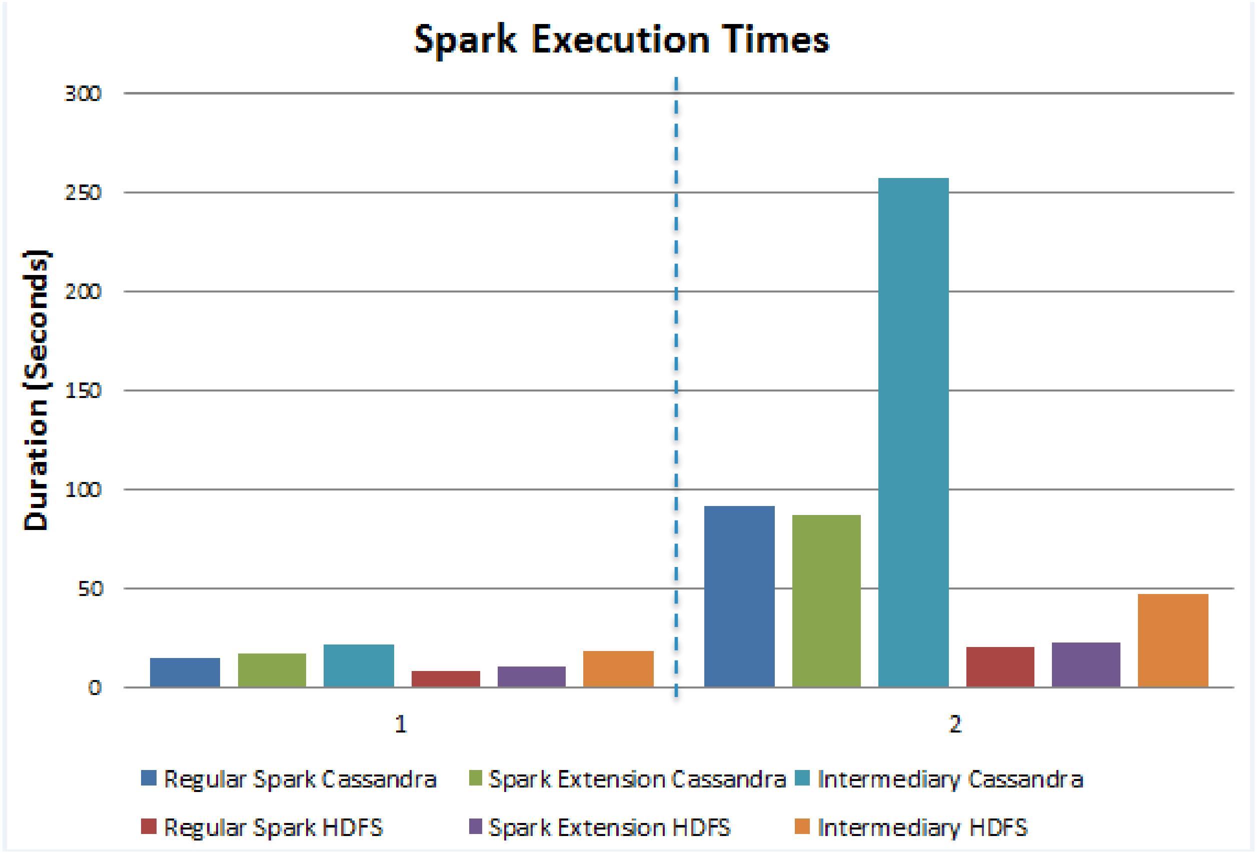
reactive-streams (akka)

Two data sets

Small data set:
(58508 records)

Larger data set:
(33 million records)

6 Spark workers



SPARK SUMMIT EUROPE 2016

Future plans

- dynamic-spark open-source library for Spark 2.0.x
  - cleaner API
  - use of datasets instead of RDD

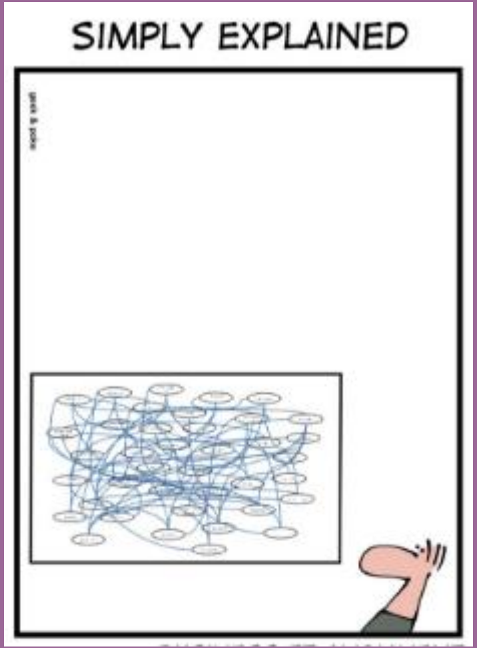
- Data switch for streams
  - more experiments with streaming data

- More efficient dynamic code update
  - more alternatives to REST

THANK YOU.

elena.lazovik@tno.nl

nl.linkedin.com/in/elenalazovik
nl.linkedin.com/in/michel-medema-04238b12b
nl.linkedin.com/in/toon-albers-6b168b20
nl.linkedin.com/in/eriklangius
nl.linkedin.com/in/alexander-lazovik-0a2b934



SPARK SUMMIT EUROPE 2016