

The logo for Spark Summit Europe 2016 is located in the top left corner. It features a stylized, circular graphic composed of many thin, white lines radiating from a central point, creating a sunburst or network-like effect. To the right of this graphic, the text "SPARK SUMMIT" is written in a bold, white, sans-serif font, and "EUROPE 2016" is written below it in a slightly smaller, bold, white, sans-serif font.

SPARK SUMMIT
EUROPE 2016

Lambda Architecture in the IoT

Fast Data Analytics with Spark Streaming and MLlib

Bas Geerdink
ING

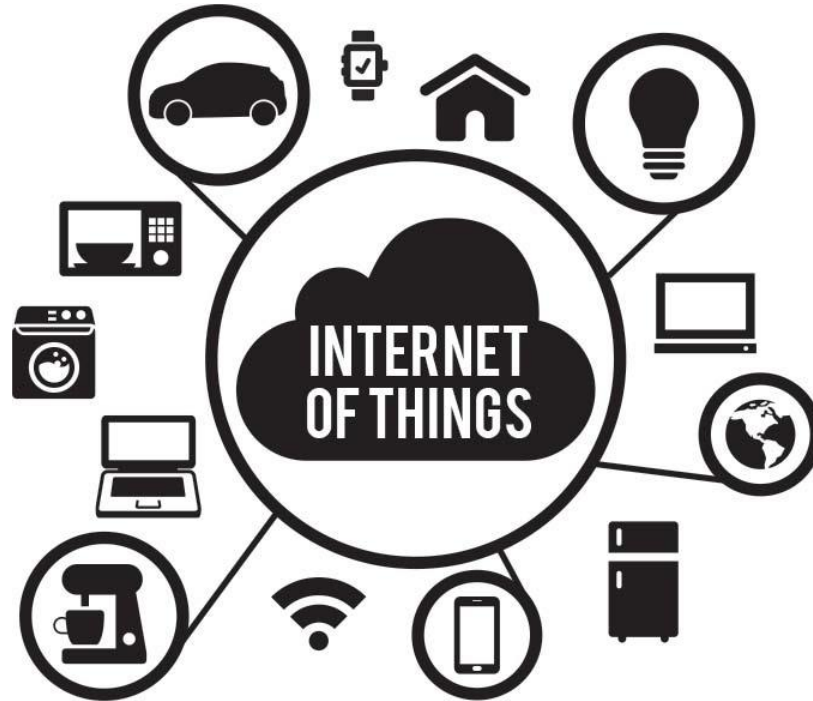
Who am I?

Bas Geerdink

- Chapter Lead in Analytics area at ING
- Master degree in Artificial Intelligence and Informatics
- Spark Certified Developer
- @bgeerdink
- <https://www.linkedin.com/in/geerdink>



The Internet of ...



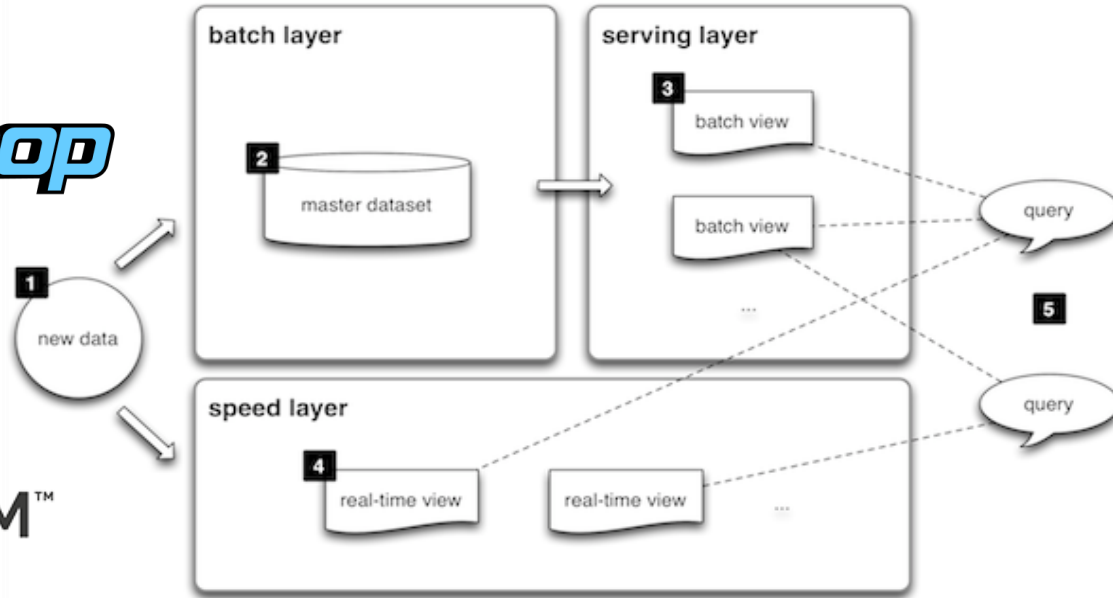
What's new in the IoT?

- Data
 - Streaming data from more sources
- Use cases
 - Combining data streams
- Technology
 - Fast processing and scalability
- Challenges
 - Security: encrypt the sensors/network/server

What do we want in the IoT?

- FAST data: process stream of events (sensory data)
- BIG data: process files/tables in batches (static data)
- ONE analytics engine
- ONE querying/visualization tool
- Scalable environment
- Reactive software

Lambda Architecture



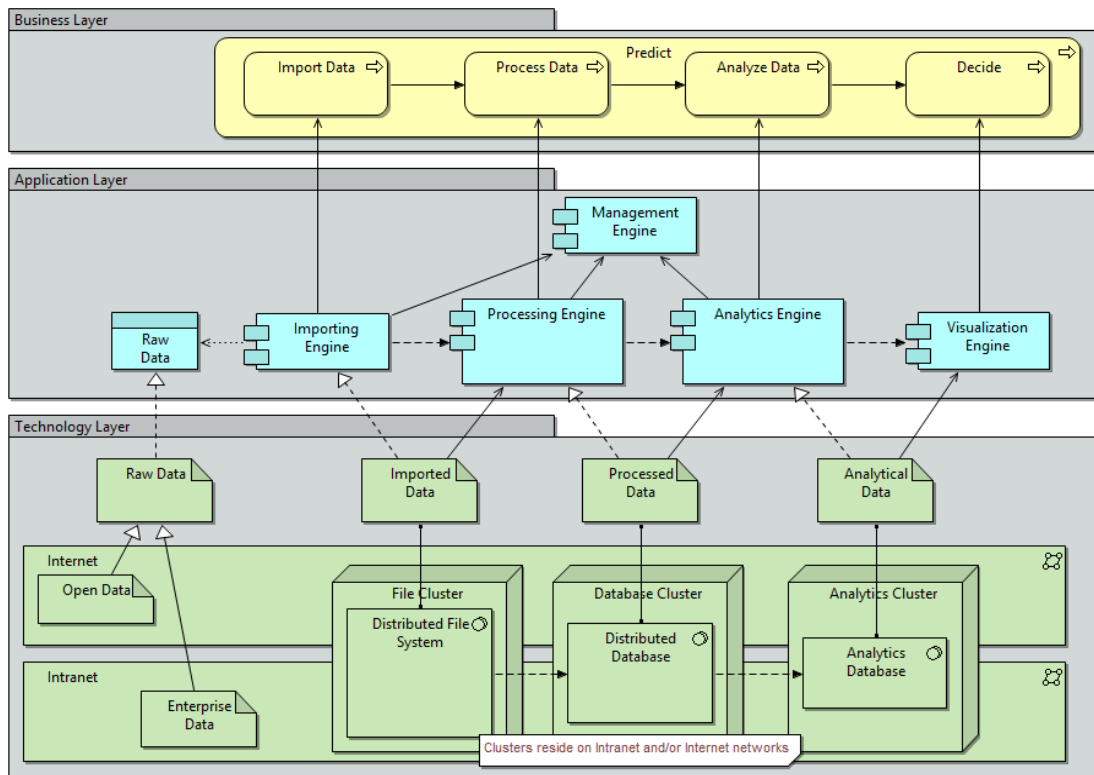
Source: Nathan Marz (2013)

Gamma/Kappa/Omega/...

Architecture

- Lambda Criticism:
 - Too complex
 - Two code bases
 - Two data stores
- Alternatives:
 - Treat a stream as a mini-batch
 - Treat a batch as a stream of records
 - Combine batch and stream within one system

Big Data Reference Architecture



Source: Geerdink (2013)

Use case: Smart Parking

Recommend the best car park when driving to Amsterdam

- Show the car park with the highest score, determined by
- Batch (every x minutes), data from car parks
 - Stream (every x seconds), GPS data of cars

Parkeren + Reizen (P+R)

Parkeer uw auto voor € 8,00 per 24 uur (na 10.00 uur voor € 1,00 per 24 uur) aan de rand van Amsterdam en reis met openbaar vervoer naar het centrum.

P+R locaties

Meer informatie per P+R locatie en uitleg hoe het werkt:

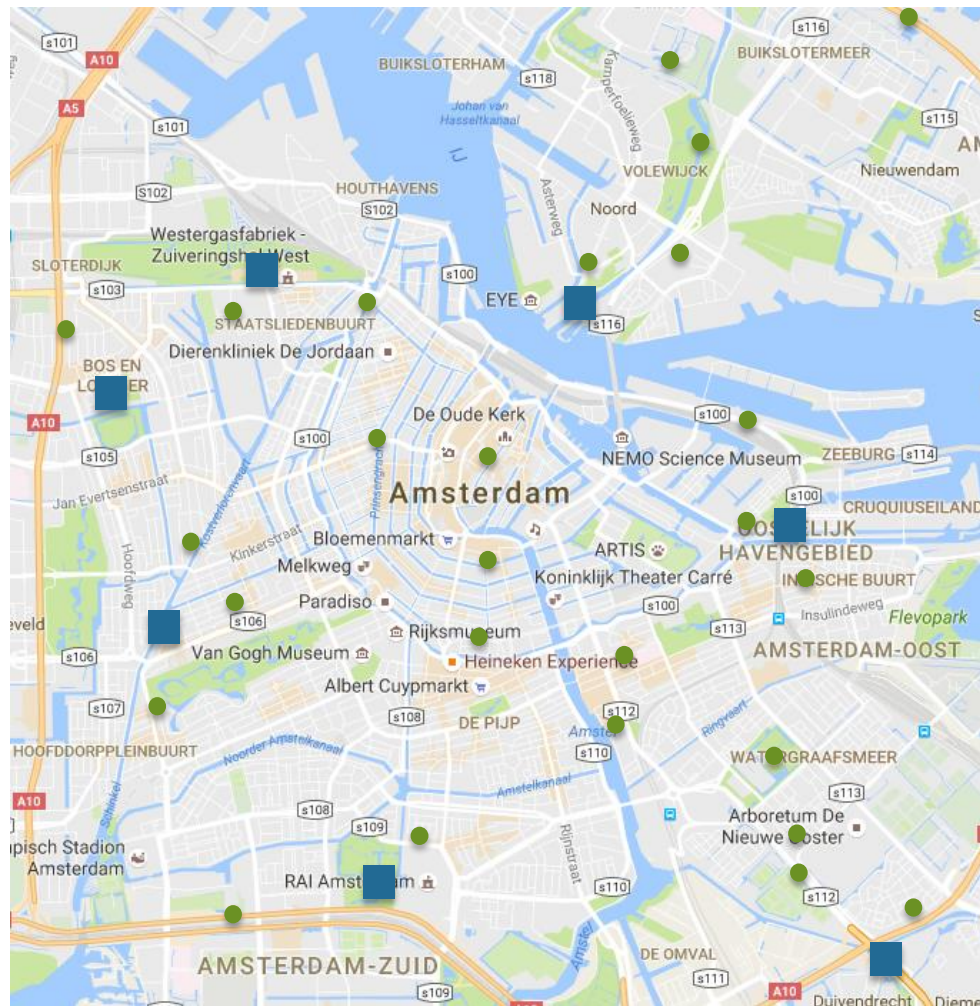
- > P+R ArenA
- > P+R Zeeburg I en II
- > Weekend P+R VUMc
- > P+R Bos en Lommer
- > P+R RAI
- > P+R Sloterdijk
- > P+R Olympisch Stadion
- > Welke P+R kies ik?

Actuele beschikbaarheid P + R parkeerplekken

Laatste update: 25-okt-2016 00:21 (elke minuut)

P+R locatie	Beschikbaarheid	Parkeerplekken
P+R ArenA	Vrij	1.440
P+R Zeeburg 2	Vrij	322
P+R Olympisch Stadion	Vrij	161
P+R Zeeburg 1	Vrij	60
P+R Sloterdijk	Geen informatie	
P+R Bos en Lommer	Vrij	21
Weekend P+R VUMc	Geplaat	





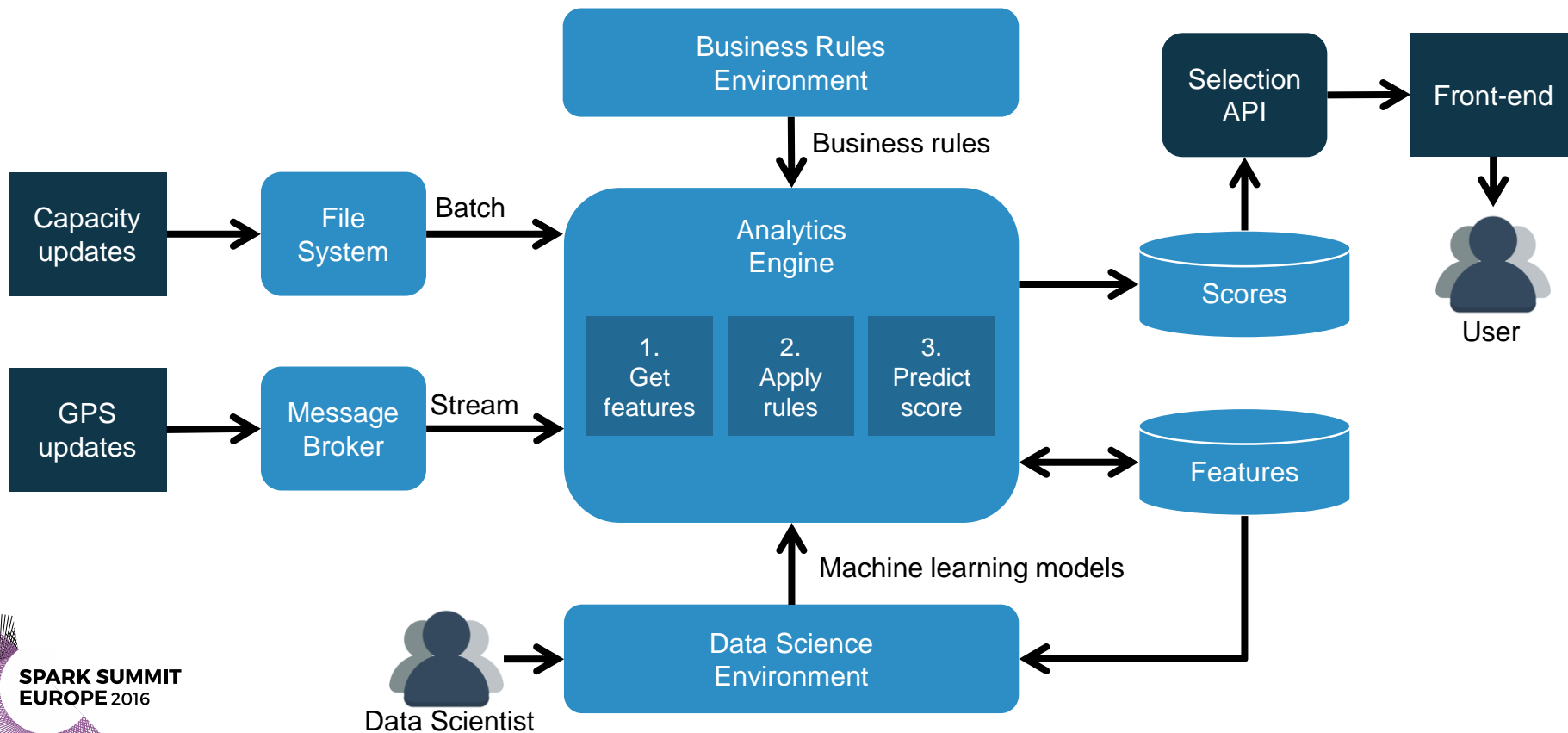
Stream Process

- Get car events (GPS data)
- Filter events (business rules)
- Store events
- For each car park in the neighborhood:
 - Get feature set (location, capacity, usage, ...)
 - Combine event with previous events (running sum)
 - Update feature set
 - Predict score (machine learning) and update database

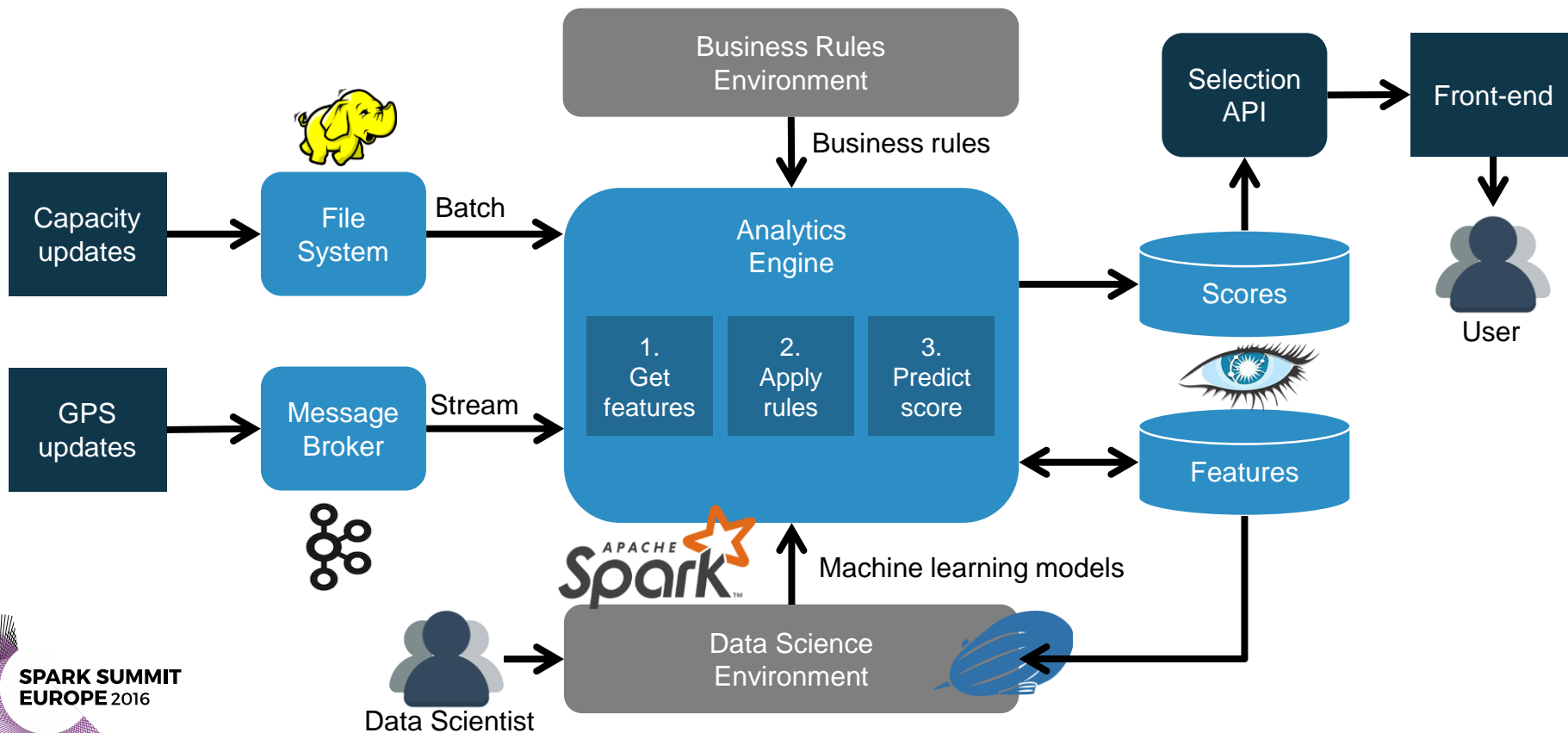
Batch Process

- Get car park data
- Clean up (remove old car data)
- For each car park in the data set:
 - Get recent set of car data (running sum)
 - Update feature set
 - Predict score (machine learning) and update database

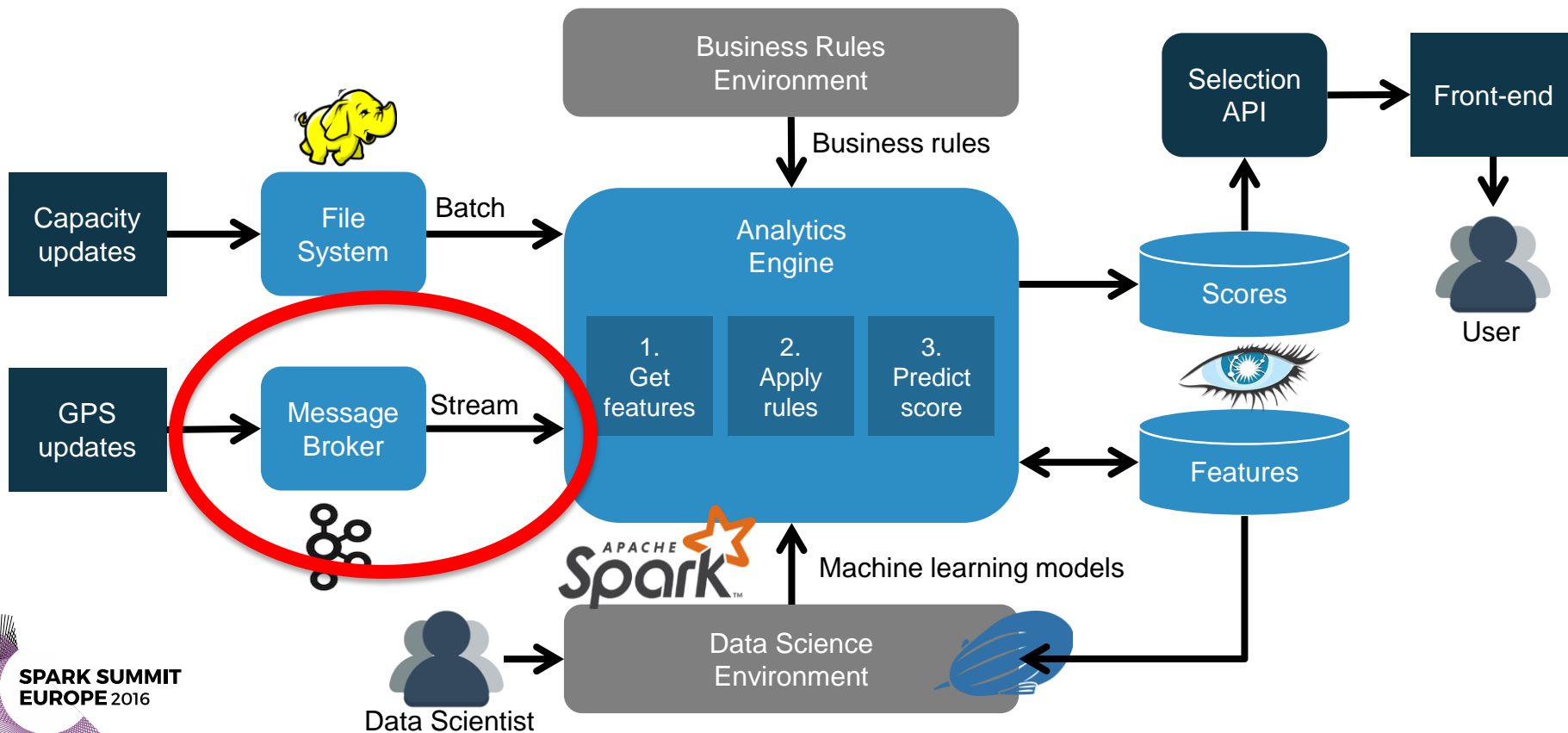
Lambda Architecture



Lambda Architecture



Lambda Architecture



Event Processing with Kafka

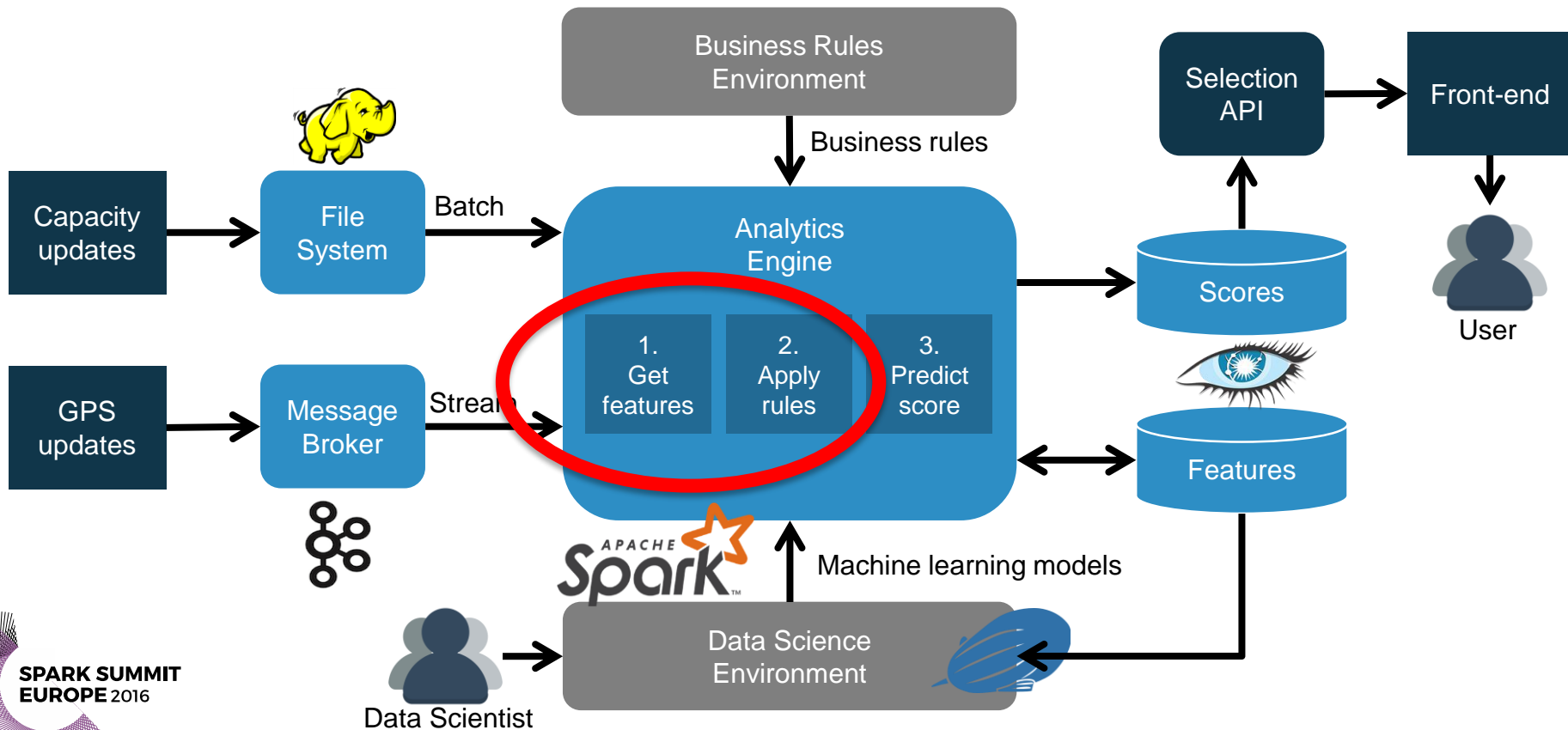
```
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.streaming.kafka010._
import org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent
import org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe

// initialize Spark Streaming
val conf = new SparkConf().setAppName("fast-data").setMaster("local[*]")
val ssc = new StreamingContext(conf, Seconds(1)) // batch interval = 1 sec

// set parameters for Kafka connection
val topics = Array("cars")
val kafkaParams = Map[String, Object]("bootstrap.servers" -> "localhost:9092")

// subscribe to stream -> create Spark DStream
val stream = KafkaUtils.createDirectStream[String, String](
  ssc,
  PreferConsistent,
  Subscribe[String, String](topics, kafkaParams))
```

Lambda Architecture



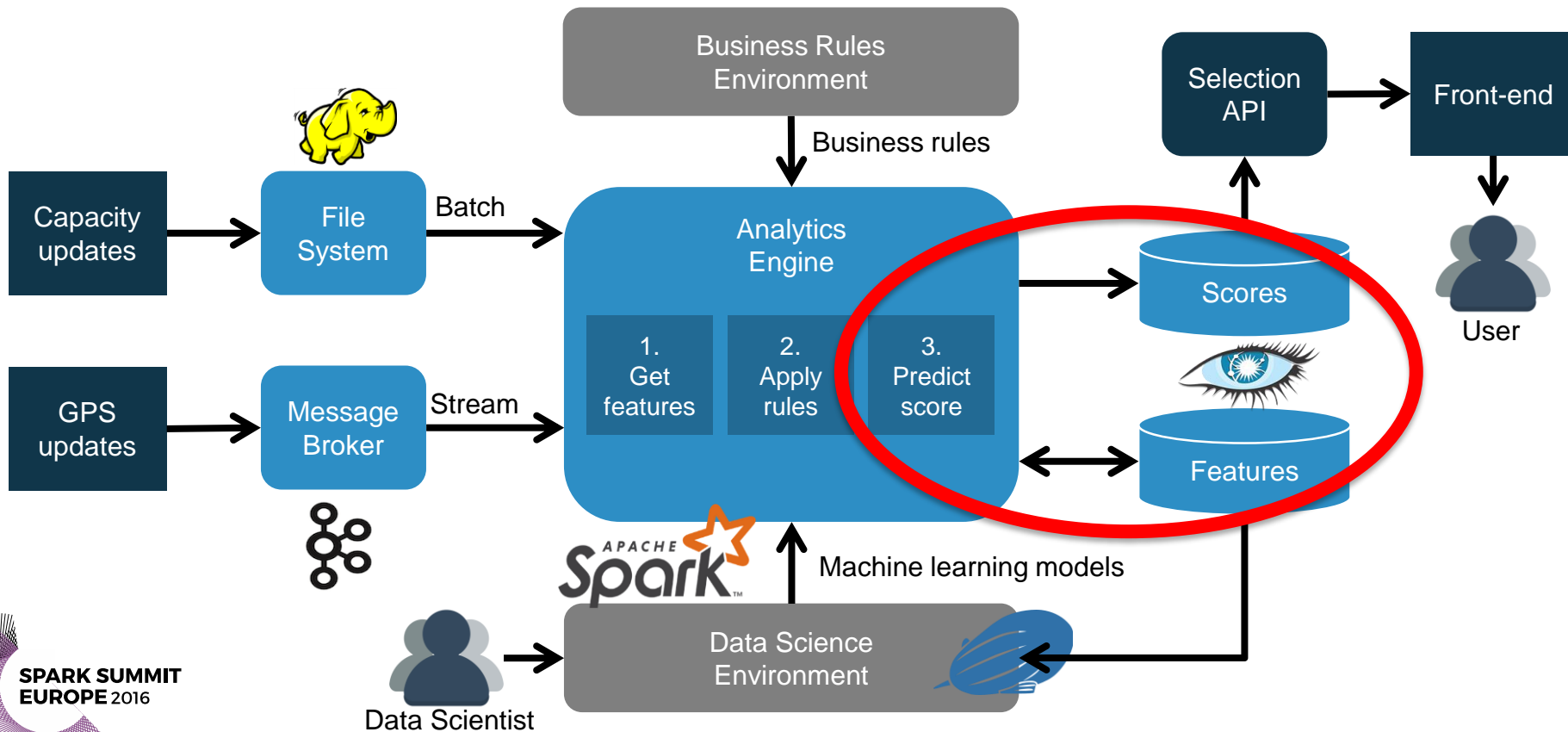
Stream: Data Preparation

```
// change raw data to business events
val events = stream
| .map(event => CarLocationHelper.createCarLocation(event.value))

// apply business rules
val filtered = events
| .filter(Location.filterLocalArea)    // only select cars in local area

// store car locations (update or create)
filtered
| .foreachRDD(rdd => rdd.foreach(cle => { CassandraHelper.insertCarLocation(cle) })))
```

Lambda Architecture



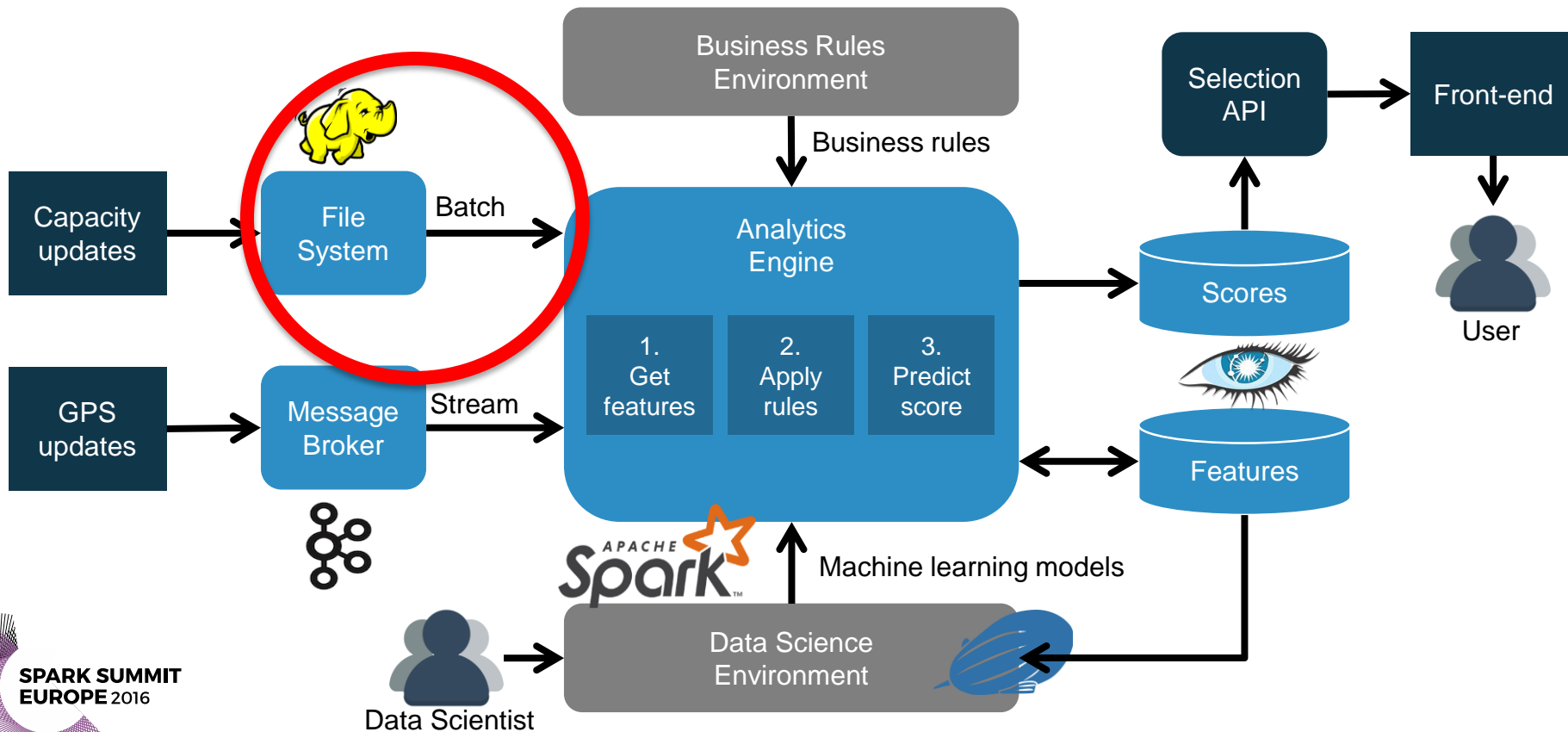
Stream: Create Recommendation

```
// recalculate the distribution of each car park in the neighborhood
filtered
  // get feature set
  .map(cl => Location.getLocalCarParks(cl))
  .foreachRDD(rdd => rdd.foreach(carParks => carParks.foreach { cp =>
    // combine event with previous events (running average)
    val carsInNeighborhood = DensityCalculator.calculateDensity(cp, getCarsInNeighborhood(cp))
    val updatedCarPark = cp.setCarsInNeighborhood(carsInNeighborhood)

    // predict score (machine learning)
    val vector = new DenseVector(updatedCarPark.featureVectorArray)
    val score = model.predict(vector)

    // update feature set and score
    val scoredCarPark = updatedCarPark.setScore(score)
    updateCarParkFeatures(updatedCarPark)
  })
})
```

Lambda Architecture



Batch Processing

```
// initialize Spark for batch processing
val spark = SparkSession.builder.appName("batch-data").master("local[*]")
    .getOrCreate()

// clean up (remove car data older than 60 seconds)
CassandraHelper.removeOldCarLocations(60)

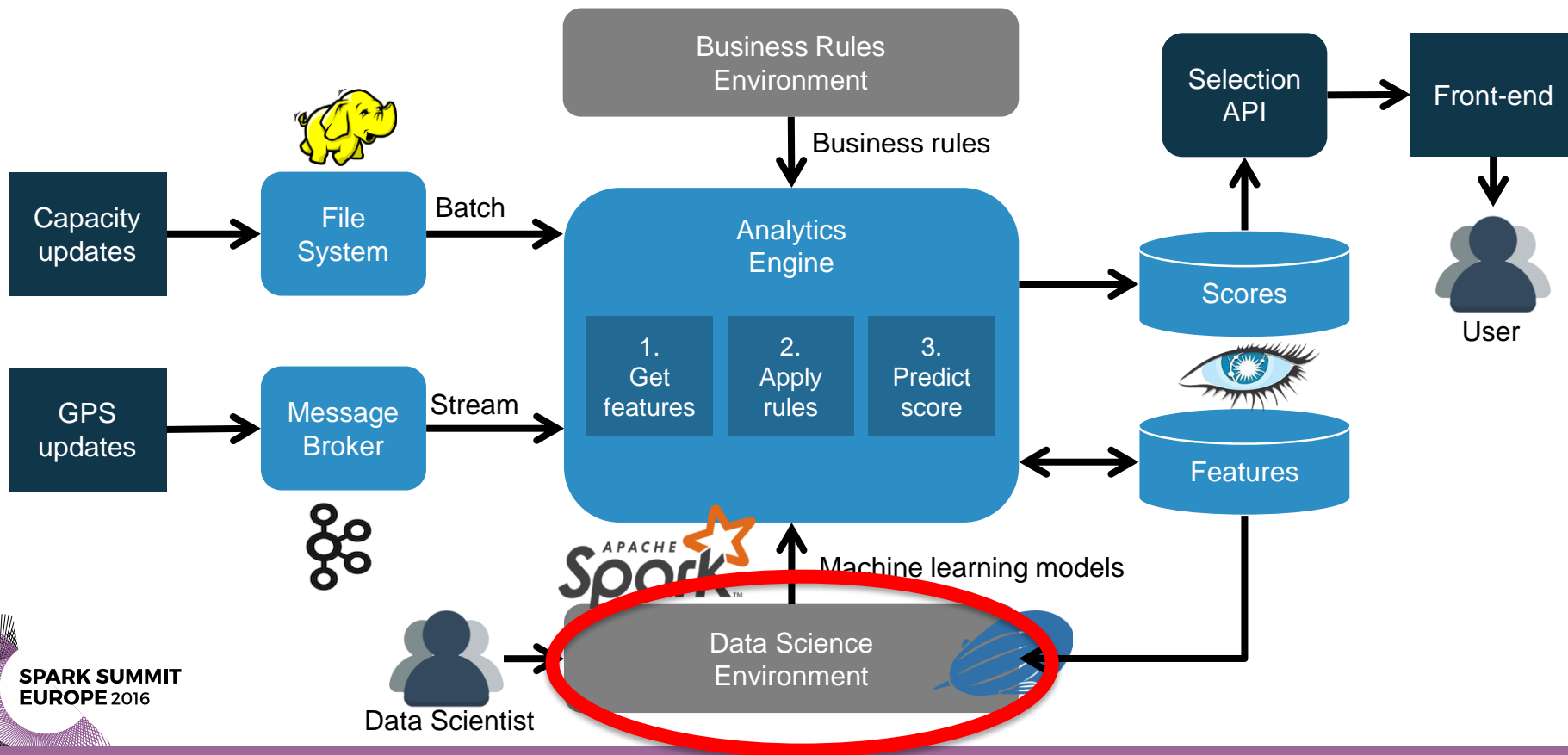
// get car park data
val textFile = spark.sparkContext.textFile("hdfs://data/latest.csv")

// iterate over all car parks in the data set
textFile.map(line => CarParkHelper.createCarPark(line)).map{cp =>
    // get recent set of car data (running average)
    val carsInNeighborhood = DensityCalculator.calculateDensity(cp, getCarsInNeighborhood(cp))
    val updatedCarPark = cp.setCarsInNeighborhood(carsInNeighborhood)

    // predict score (machine learning)
    val vector = new DenseVector(updatedCarPark.featureVectorArray)
    val score = model.predict(vector)

    // update feature set and score
    val scoredCarPark = updatedCarPark.setScore(score)
    updateCarParkFeatures(updatedCarPark)
}
```

Lambda Architecture



Data Science with Zeppelin

The screenshot shows the Zeppelin Notebook web interface in a browser. The address bar shows `http://localhost:9090/` and the notebook path is `localhost:9090/#/notebook/2BZGHQJXH`. The notebook title is "fast-data". The code in the notebook is as follows:

```
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.regression.LinearRegressionModel
import org.apache.spark.mllib.regression.LinearRegressionWithSGD

println("Car park recommendations")

case class CarPark(name: String, latitude: Float, longitude: Float, capacity: Int, usage: Float, accessibility: Float, openFrom: Int, openUntil: Int, rate: Float, cars: Float, score: Float)

// load data
var file = sc.textFile("/home/bas/data.csv")
val records = file
  .map(l => l.split(","))
  .map(parts => CarPark(parts(0), parts(1).toFloat, parts(2).toFloat, parts(3).toInt, parts(4).toFloat, parts(5).toFloat, parts(6).toInt, parts(7).toInt, parts(8).toFloat, 0, 0))

// prepare data, create vector
val data = records.map(record => LabeledPoint(record.score, Vectors.dense(record.cars, record.rate, record.usage))).collect()
val rdd = spark.sparkContext.makeRDD(data)

// separate training and test set
val splits = rdd.randomSplit(Array(0.8, 0.2))
val training = splits(0).cache()
val test = splits(1).cache()

// train the model
val algorithm = new LinearRegressionWithSGD()
algorithm.setIntercept(true)
algorithm.optimizer.setNumIterations(100).setStepSize(0.1)
val model = algorithm.run(training)

println(model.intercept)
println(model.weights)

// do a test run
val prediction = model.predict(test.map(_.features))

// check output
val predictionAndLabel = prediction.zip(test.map(_.label))
predictionAndLabel.count()
predictionAndLabel.take(20).foreach((p) => println("Prediction=" + p._1 + ", actual=" + p._2))

model_health.save(sc, "/home/bas/models/health")
```

The interface includes a search bar for notebooks, a status bar showing "anonymous", and a toolbar with various icons for running and managing the notebook.

Summary

- The IoT requires new architecture principles: in-memory, distributed, reactive and scalable are the new normal
- Lambda/Kappa/Omega reference architectures are designed for combining batch and streaming data flows
- Open source tooling such as Kafka, Cassandra, and Spark adhere to these principles and design patterns

The logo consists of a series of thin, white, curved lines that form a semi-circular shape, resembling a stylized sunburst or a fan.

SPARK SUMMIT
EUROPE 2016

#SparkSummit

A large, purple, semi-transparent rectangular box is centered over the Atomium structure. The text "THANK YOU! ING is hiring 😊" is written in white, bold, sans-serif font inside this box.

THANK YOU!
ING is hiring 😊