IOT, Timeseries and prediction with Android, Cassandra and Spark

Amira Lakhal
@Miralak

Jfokus 2016

About me

Agile Java Developer and 

valtech_

Running addict

Paris 2016 Marathon

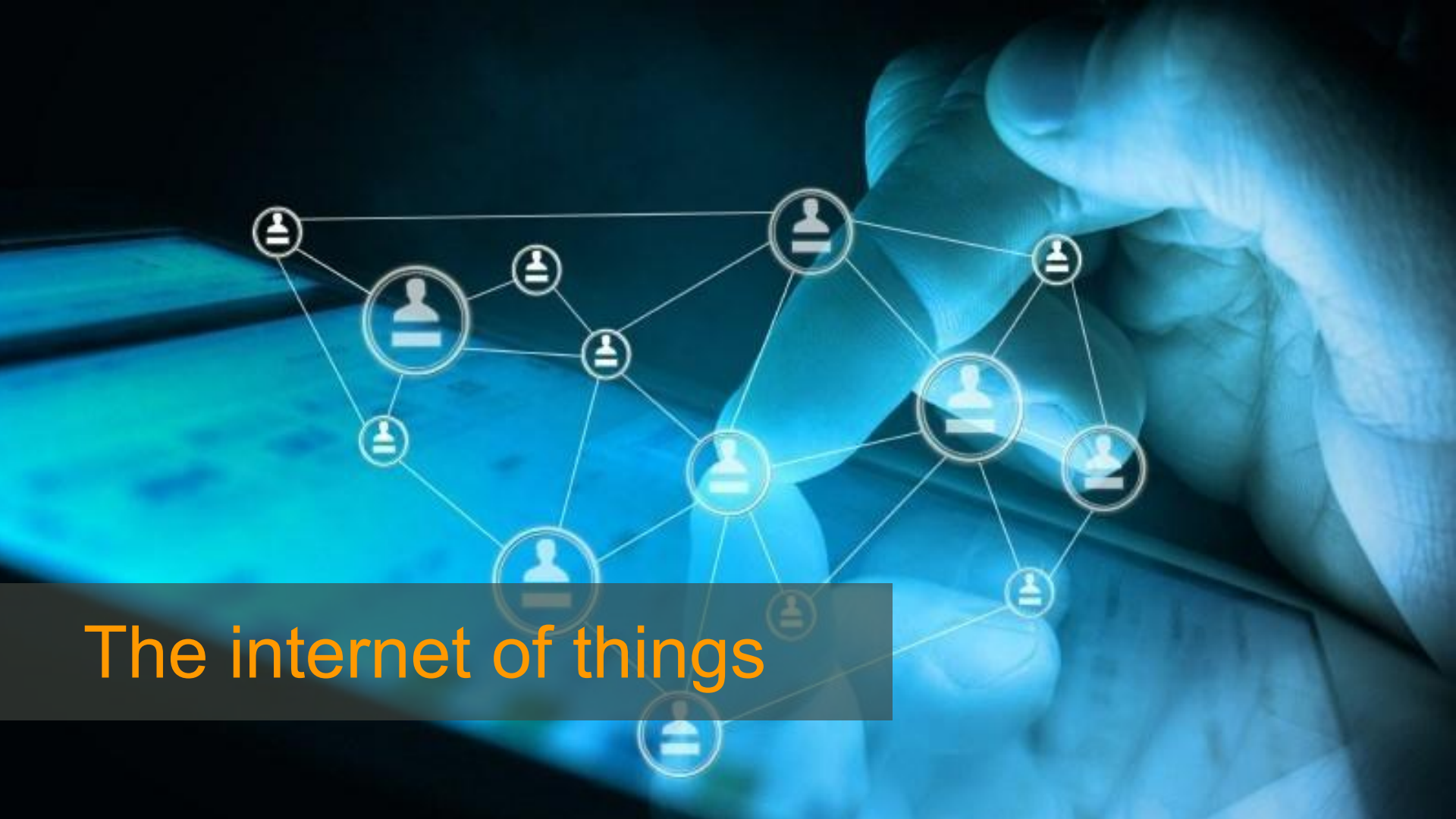
@Miralak

github.com/MiraLak

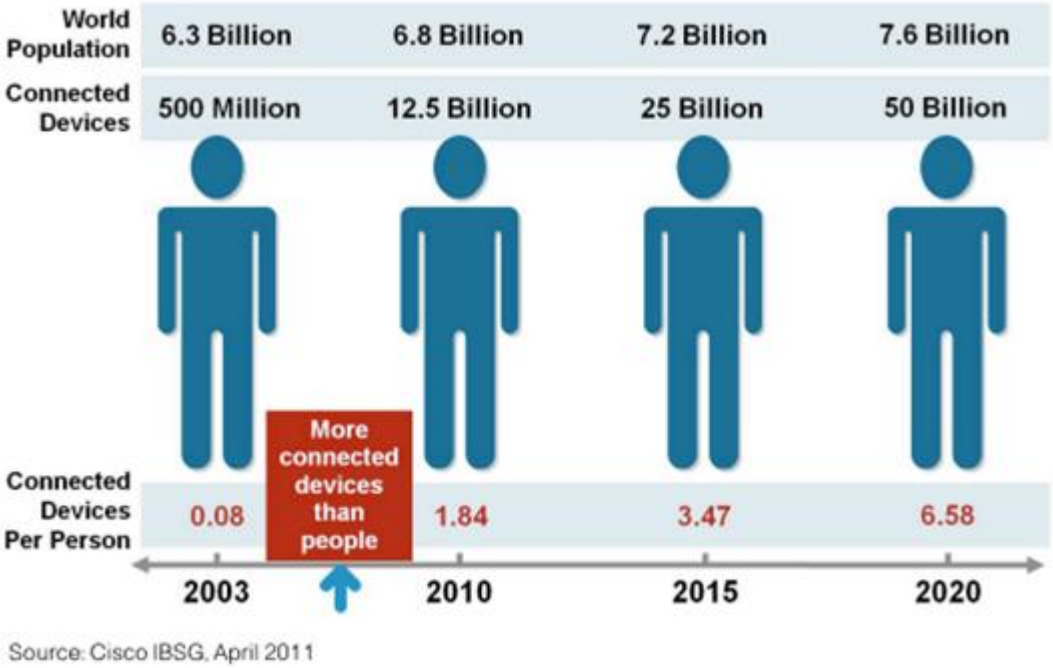
Duchess France

www.duchess-france.org

@duchessfr


The internet of things

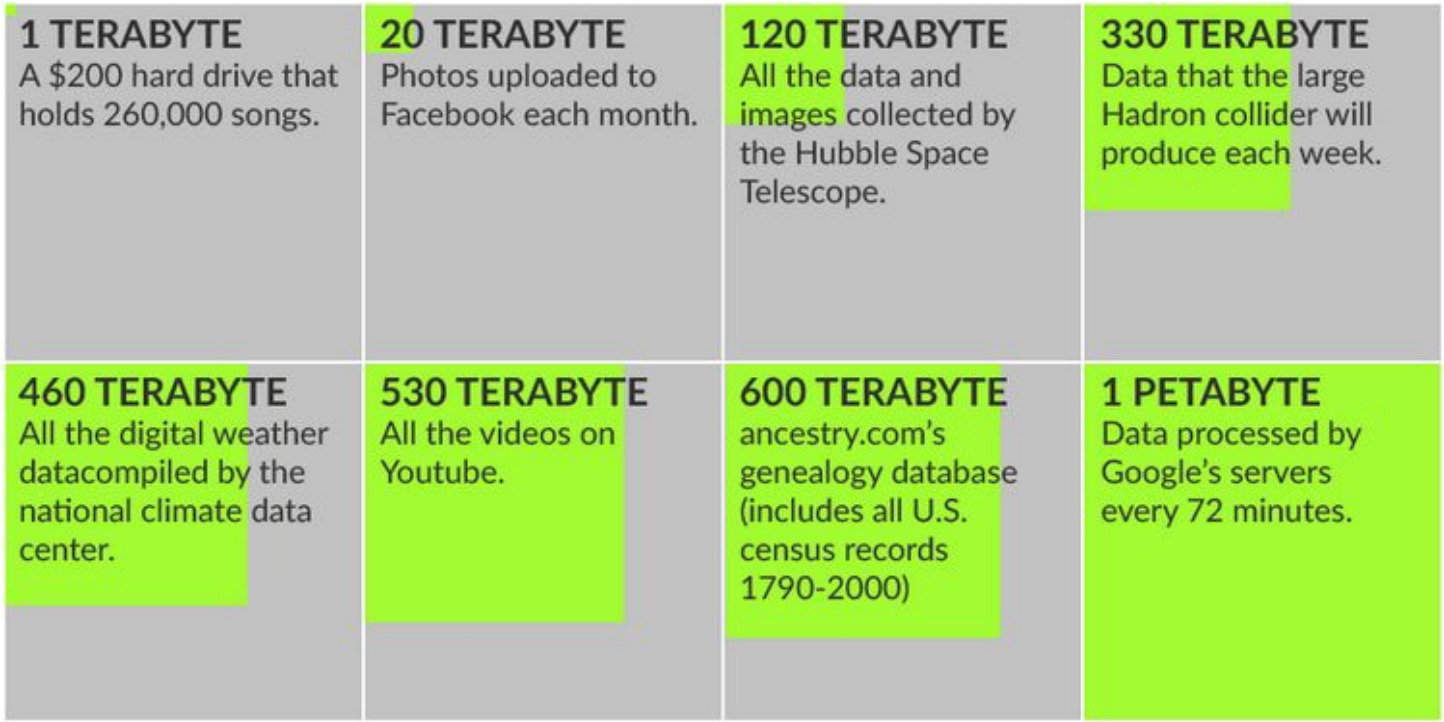
Internet of things (IoT)



Source: Cisco IBSG, April 2011

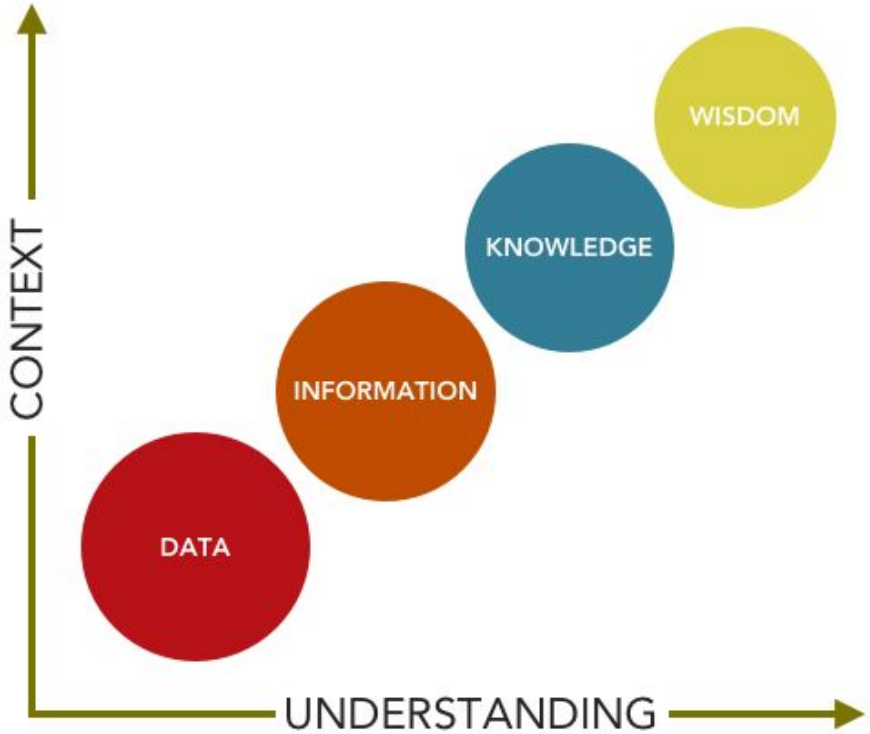
Internet of things

Big Data Era



Source: micronautomata.com

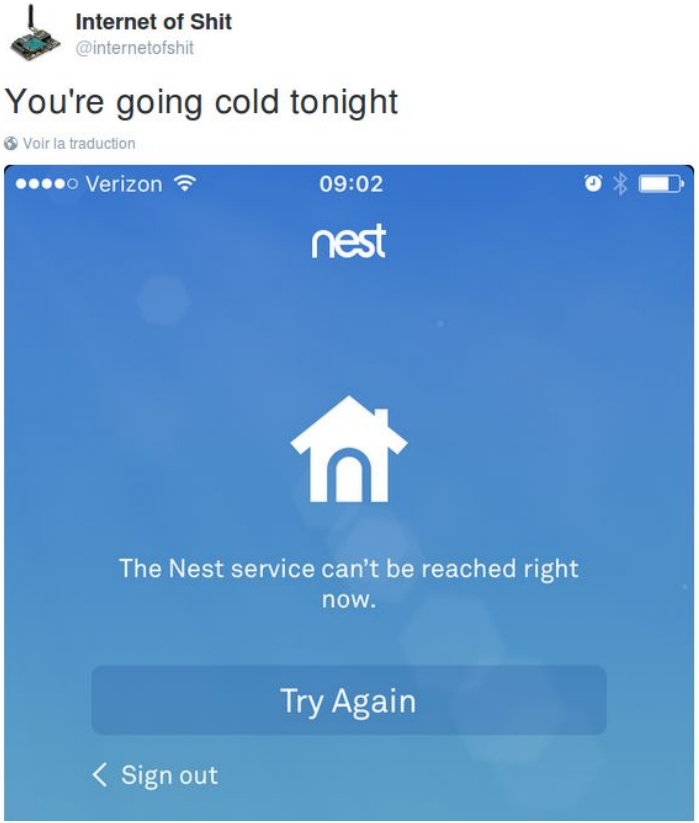
Transform data



Future?

Far far away

Source: aldebaran



My connected objects

Goal

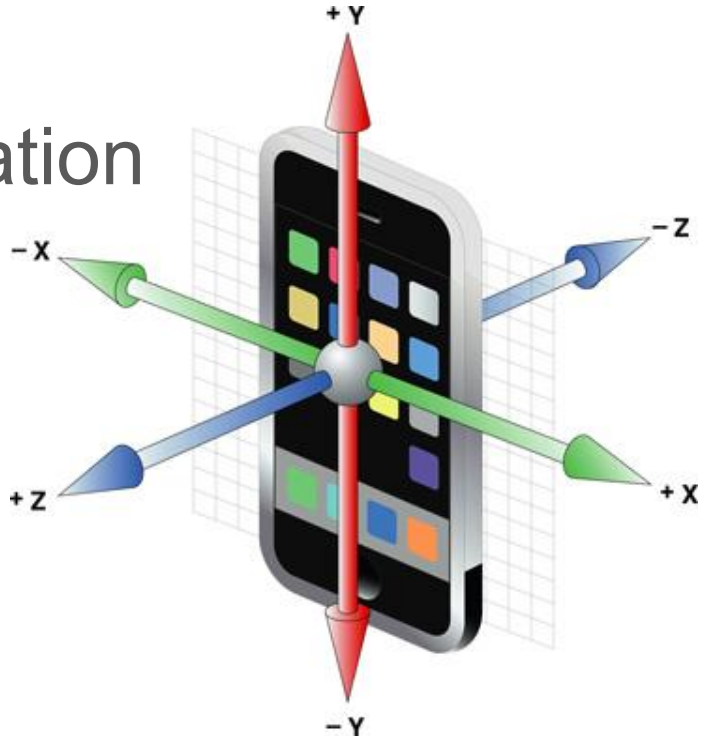
Physical activity recognition with measurement coming from an accelerometer:

walking, jogging, sitting ...

→ Health care

Accelerometer

- A motion sensor
- Measure proper acceleration
- Multiple usages

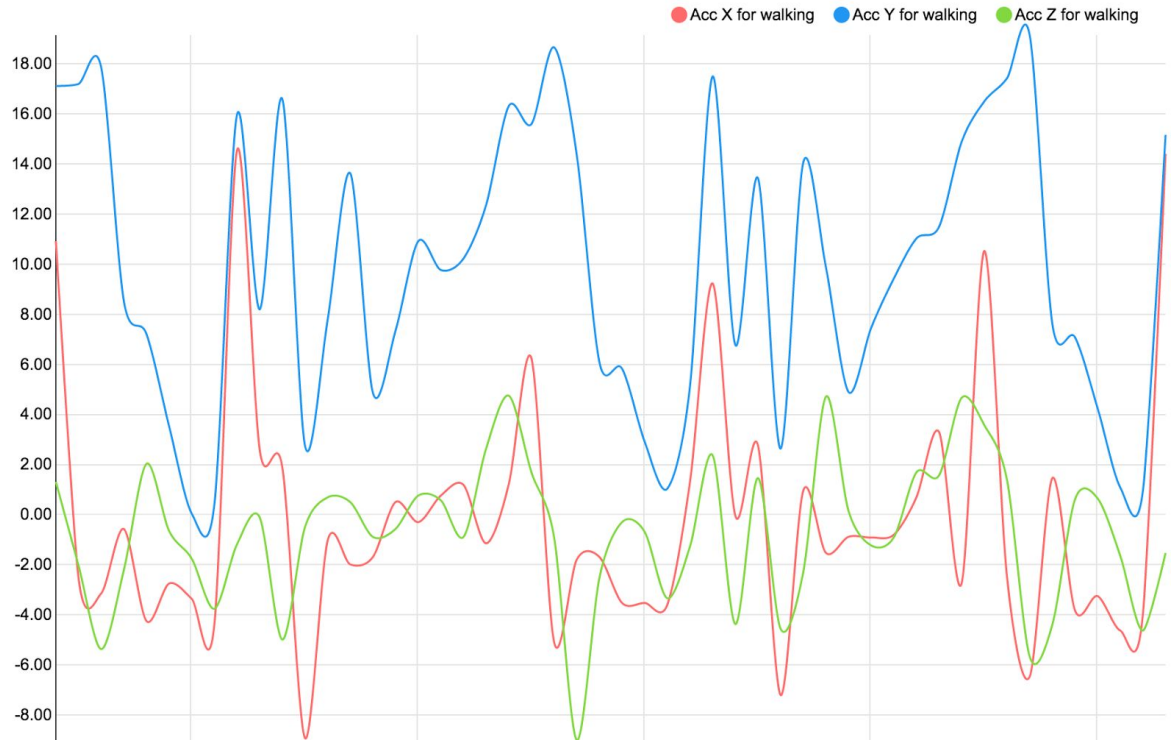


Accelerometer

Each acceleration contains:

- a timestamp (eg, 1428773040488)
- acceleration force along the x axis (unit is m/s^2)
- acceleration force along the y axis (unit is m/s^2)
- acceleration force along the z axis (unit is m/s^2)

Timeseries

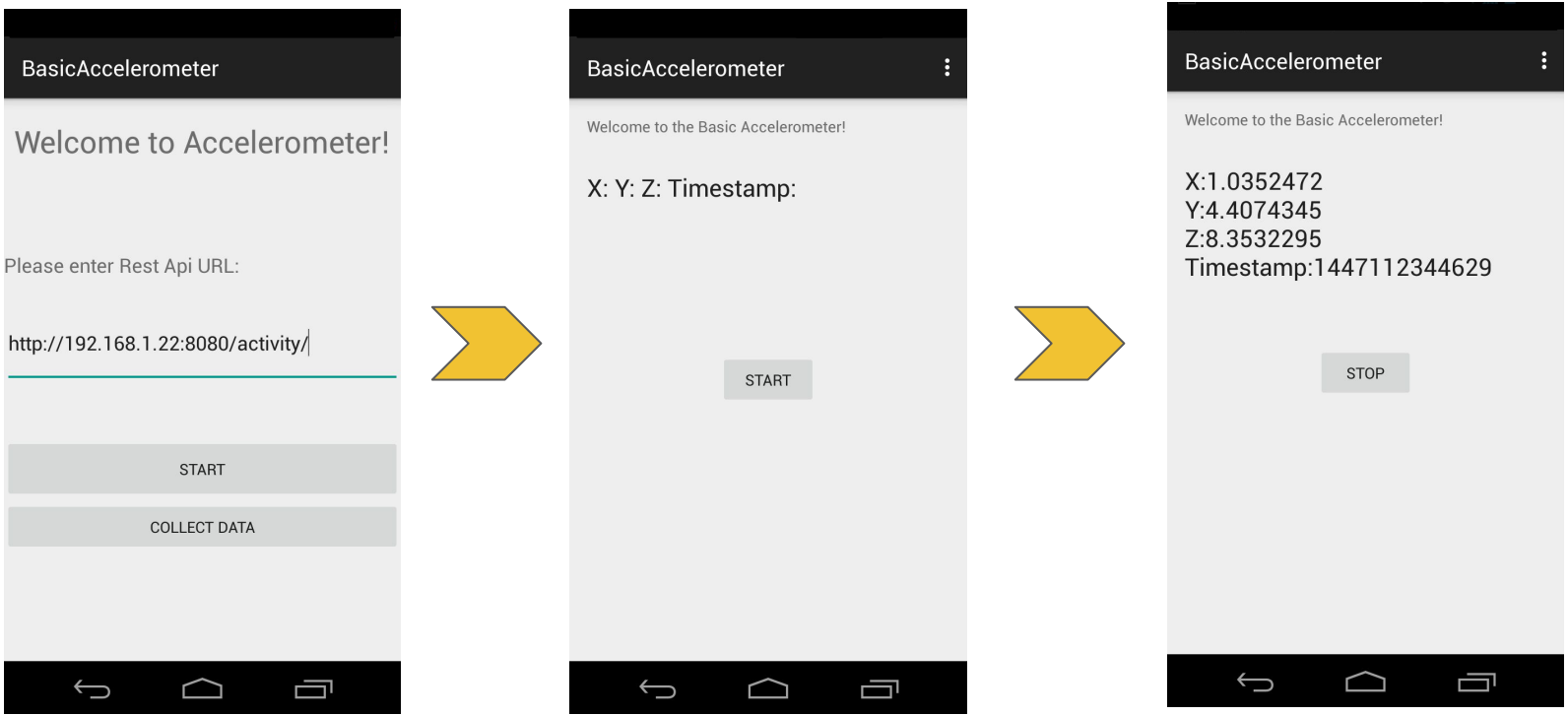


Source: cityzendata.com

Collect data

https://github.com/MiraLak/accelerometer-rest-to-cassandra

Android App



https://github.com/MiraLak/AccelerometerAndroidApp


Store data


cassandra

History

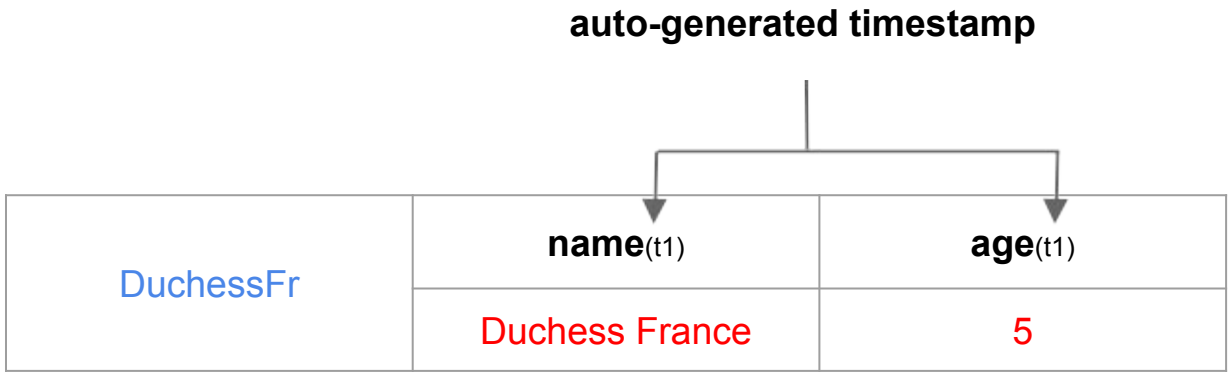
- Created by Facebook
- Open-source in 2008
- current version 3.3
- column-oriented ☞ distributed table

Cassandra Key Facts

- Linear scalability
- Master-less: peer to peer
- Operational simplicity
- Multi-datacenter
- No SPOF ☞ Continuous availability (≈100% up-time)

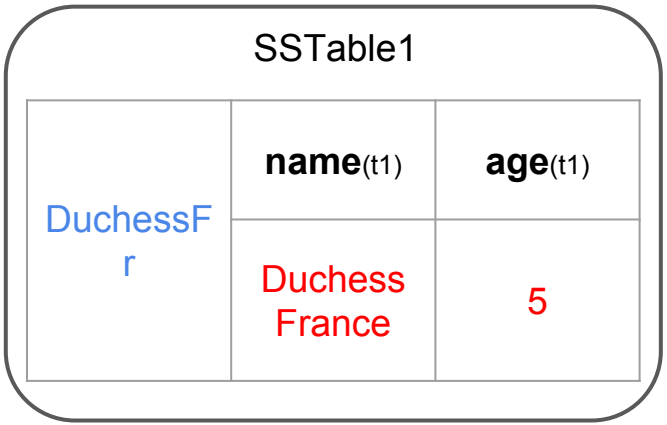
Last Write Win

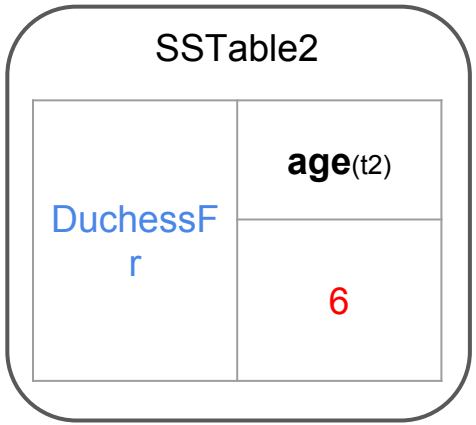
```
INSERT INTO users(login, name, age) VALUES (‘DuchessFr’ , ‘Duchess France’, ‘5’);
```



Last Write Win

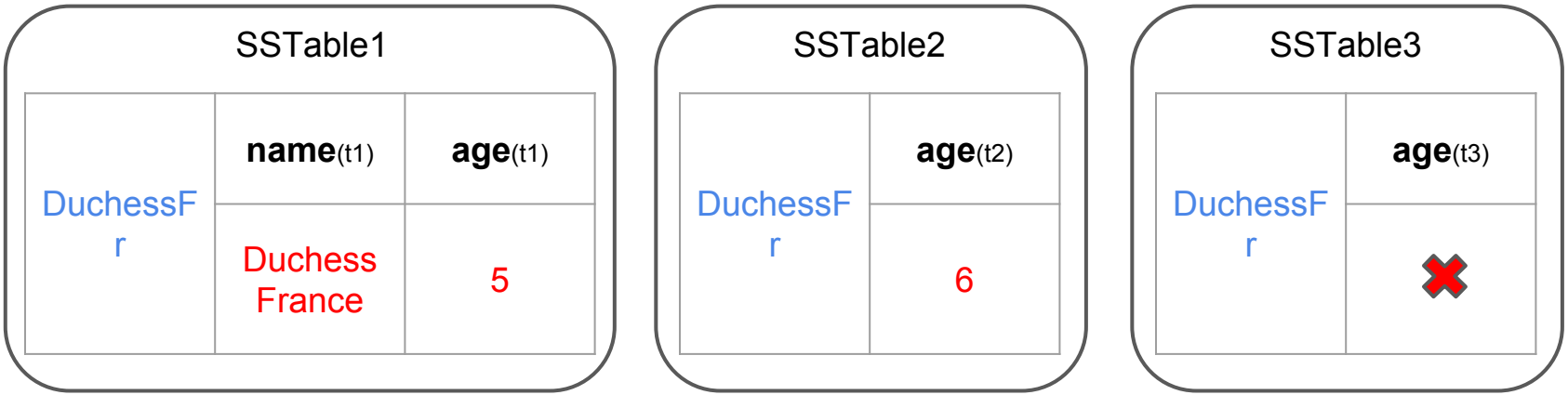
```
UPDATE users SET age = '6' WHERE login='DuchessFr' ;
```





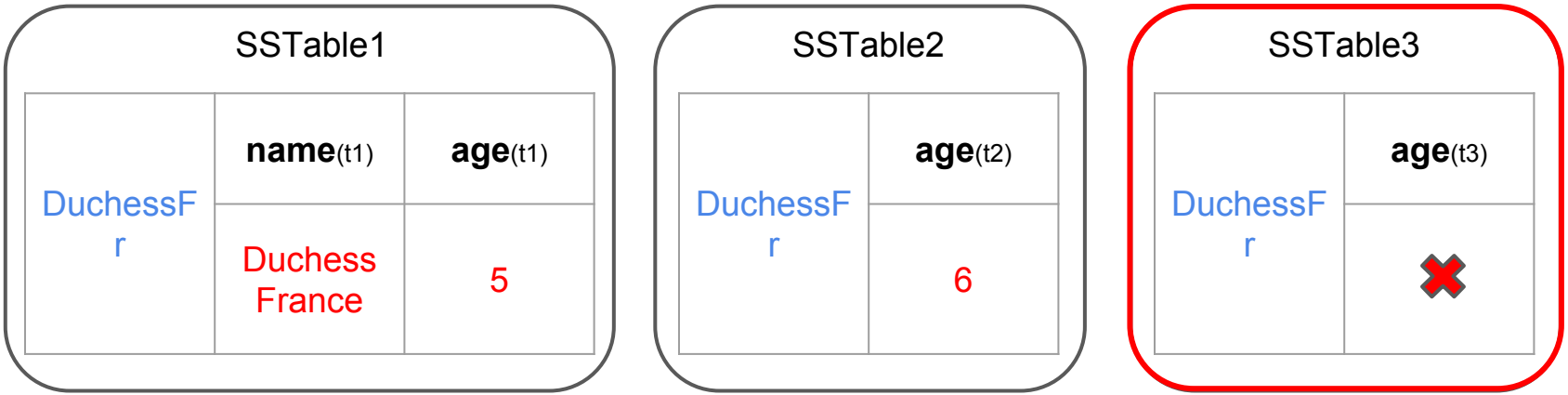
Last Write Win

```
DELETE age from users  WHERE login='DuchessFr' ;
```

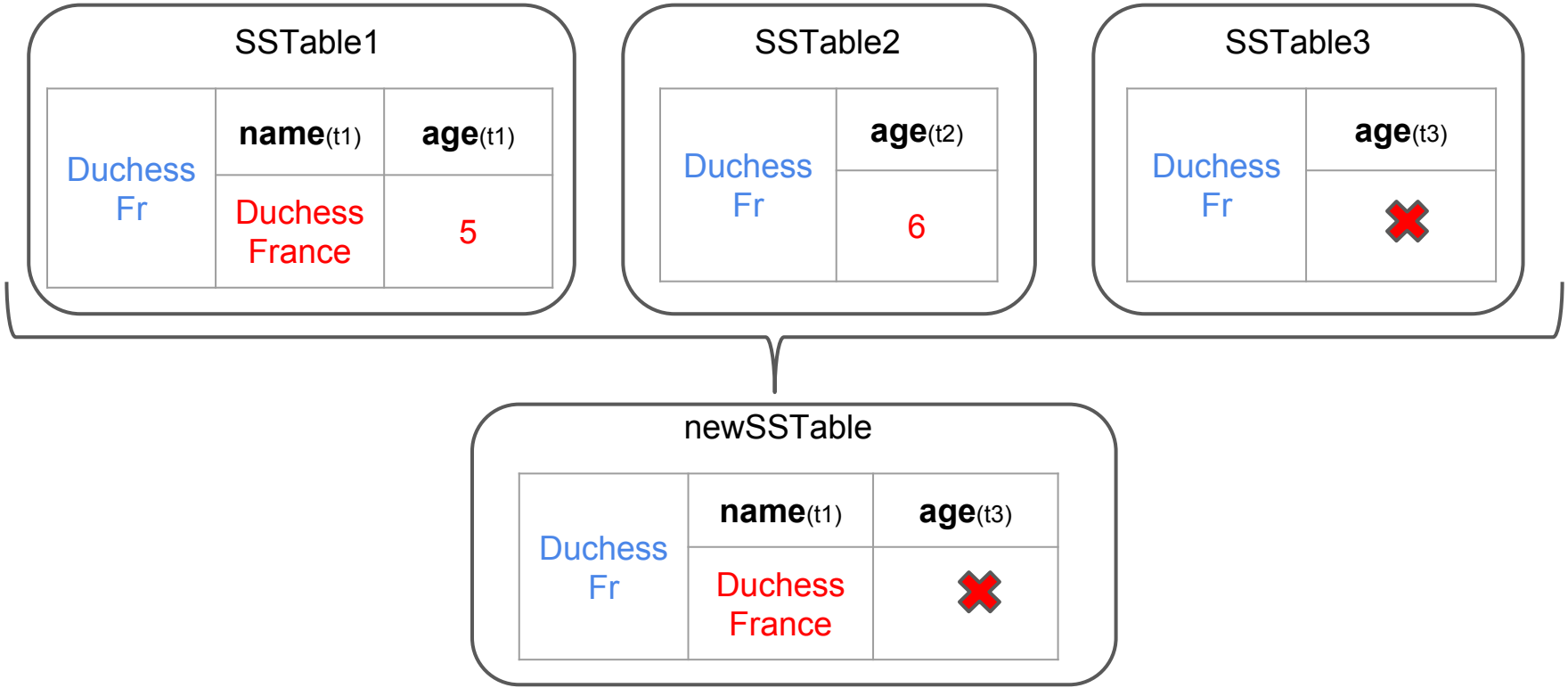


Last Write Win

```
SELECT age from users  WHERE login='DuchessFr' ;
```



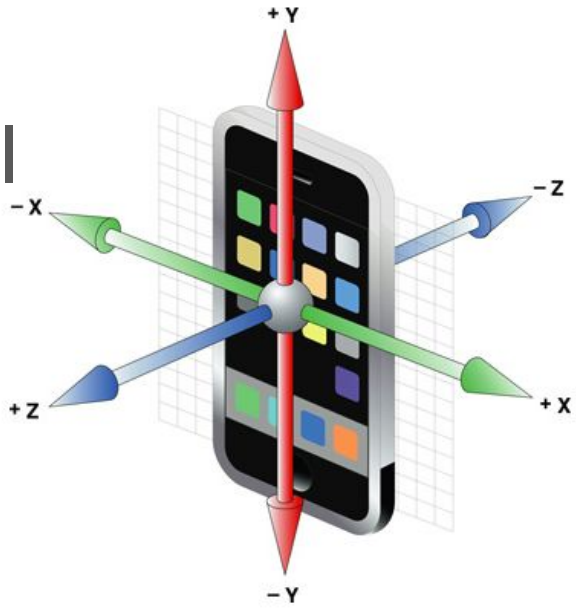
Compaction


SSTable1
Duchess Fr
name(t1)
age(t1)
Duchess France
5
SSTable2
Duchess Fr
age(t2)
6
SSTable3
Duchess Fr
age(t3)
newSSTable
Duchess Fr
name(t1)
age(t3)
Duchess France

Timeseries with Cassandra

We want to use historical data

➡ Use timeseries data model



User ID and timestamp unique

➡ Store many as needed

Timeseries data model

```
CREATE TABLE accelerometer (
    user_id text,
    timestamp bigint,
    x double, y double, z double,
    PRIMARY KEY (( user_id  ),timestamp)
);
```

Storage Model : Logical View

user-id	timestamp	x	y	z
TEST_USER	1447034733210	10.9	34.5	12.6
TEST_USER	1447034733344	15.4	39.9	16.3
TEST_USER	1447034733462	13.5	36.1	20.3
TEST_USER	1447034733556	13.0	41.3	22.8”

Storage model: Disk layout

TEST_USER	1447034733210			1447034733344			1447034733462		
	x	Y	Z	x	y	z	x	y	z
	10.9	34.5	12.6	15.4	39.9	16.3	13.5	36.1	20.3

Timeseries data model

```
CREATE TABLE accelerometer (
      user_id text,
     date text,
     timestamp bigint,
x double, y double, z double,
PRIMARY KEY ((user_id, date),timestamp)
);
```

Storage Model : Logical View

user-id:date	timestamp	x	y	z
TEST_USER: 24-09-2015	1446034733566	10.9	34.5	12.6
TEST_USER: 24-09-2015	1446034733631	15.4	39.9	16.3
TEST_USER: 24-09-2015	1446034733740	13.5	36.1	20.3
TEST_USER: 24-09-2015	1446034733830	13.0	41.3	22.8”

Storage model: Disk layout

TEST_USER :24-09-2015	1446034733566			1446034733631			..	1446034734120		
	x	Y	Z	x	y	z	..	x	y	z
	10.9	34.5	12.6	15.4	39.9	16.3	..	13.5	36.1	20.3

TEST_USER :25-09-2015	1447034733210			1447034733300			..	1447034733810		
	x	Y	Z	x	y	z	..	x	y	z
	13.3	30.5,	12.1	25.2	34.9	16.1	..	15.0	32.1	12.4

Query patterns

Range queries ☞ Slice on disk

Select user_id, date, timestamp, x, y, z

From accelerometer

Where user_id =”TEST_USER”

and date = “24-04-2015”

and timestamp > “1447034733462” and timestamp < “1447034733890”

Timeseries with lastest columns first

```
CREATE TABLE latest_accelerations (
    user_id text,
    timestamp bigint,
    x double, y double, z double,
PRIMARY KEY (user_id, timestamp)
WITH CLUSTERING ORDER BY (timestamp, DESC);
);
```

Timeseries with expiring columns

```
INSERT INTO latest_accelerations( user_id, timestamp, x, y, z )

VALUES ( 'TEST_USER', 1447034733462, 10.9, 34.5, 12.6 )

USING TTL 20;
```

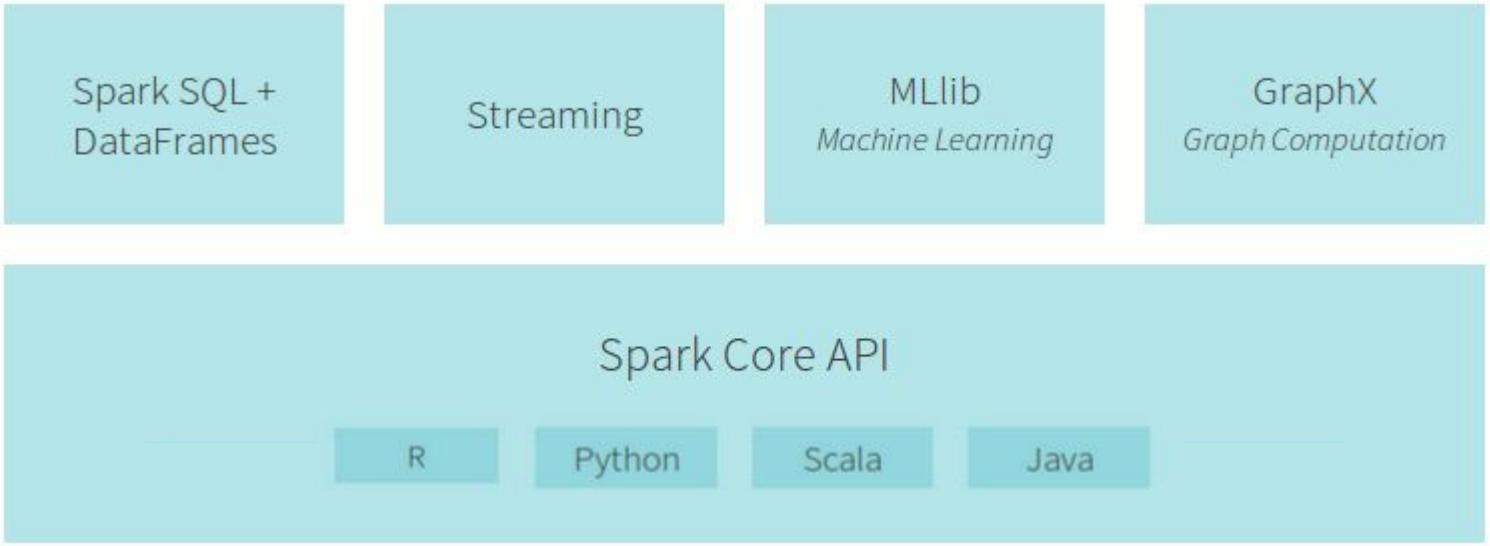
Analyse data


Spark

Spark History

- Created by AMPLab
- Open-source in 2010
- Current version 1.5.2
- Written in Scala
- Fast cluster computing

Spark ecosystem



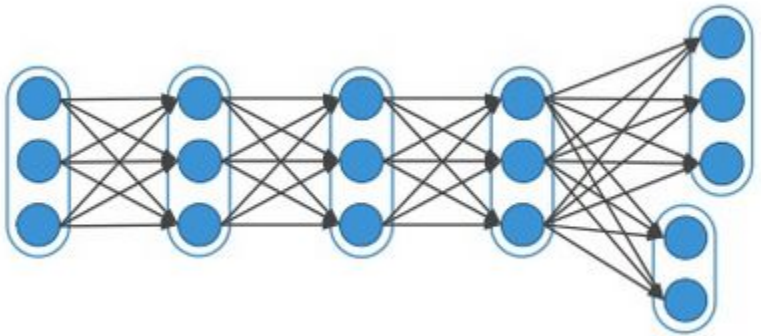
Source: Databricks

Spark data sources

Source: Databricks

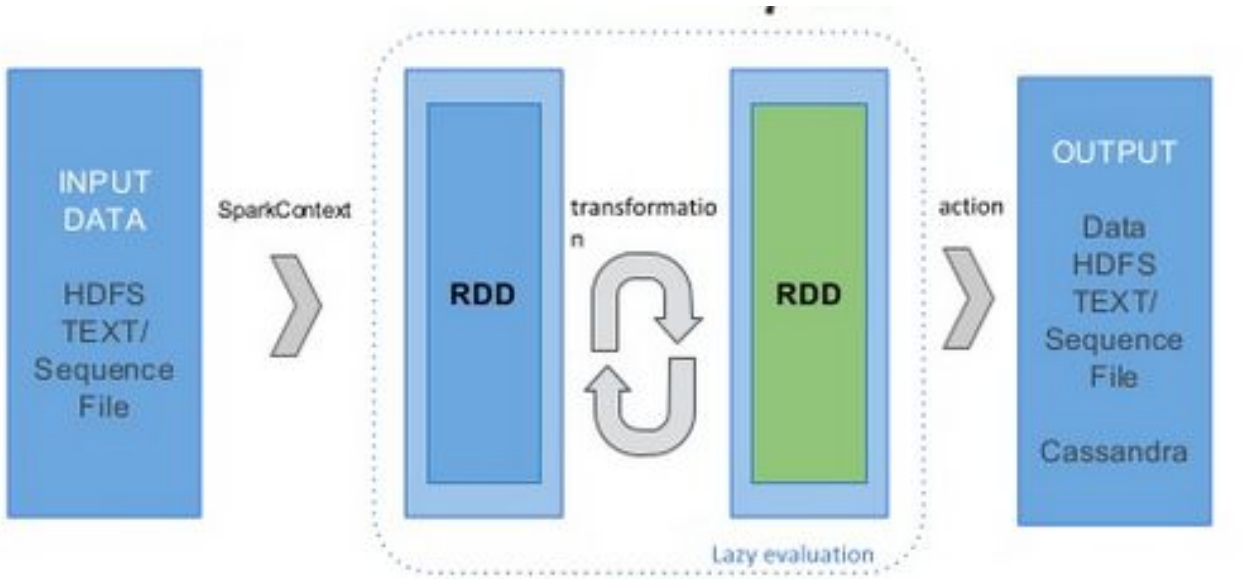
Resilient Distributed Dataset (RDD)

- Abstraction : collection of objects
- Operated in parallel
- Fault tolerant without replication



Source: Databricks

RDD transformations and actions



Source: http://www.bogotobogo.com

Word count sample

```
SparkConf conf = new SparkConf()
  .setAppName("Wordcount")
  .setMaster("local[*]");
JavaSparkContext sc = new JavaSparkContext(conf);
```

```
//Transformations
JavaRDD<String> words = sc.textFile(myFilePath)
                          .flatMap(line -> Arrays.asList(line.split(" ")));
JavaPairRDD<String, Integer> pairs = words.mapToPair(x -> new Tuple2(x,1))
                                          .reduceByKey((a,b) -> a+b)
                                          .filter(tuple -> tuple._2() > 3);


//Action
List<Tuple2<String, Integer>> result = words.collect();
```

Word count sample

```
SparkConf conf = new SparkConf()
  .setAppName("Wordcount")
  .setMaster("local[*]");
JavaSparkContext sc = new JavaSparkContext(conf);
```

```
//Transformations
JavaRDD<String> words = sc.textFile(myFilePath)
                                    .flatMap(line -> Arrays.asList(line.split(" ")));
JavaPairRDD<String, Integer> pairs = words.mapToPair(x -> new Tuple2(x,1))
                                                .reduceByKey((a,b) -> a+b)
                                                .filter(tuple -> tuple._2() > 3);
```

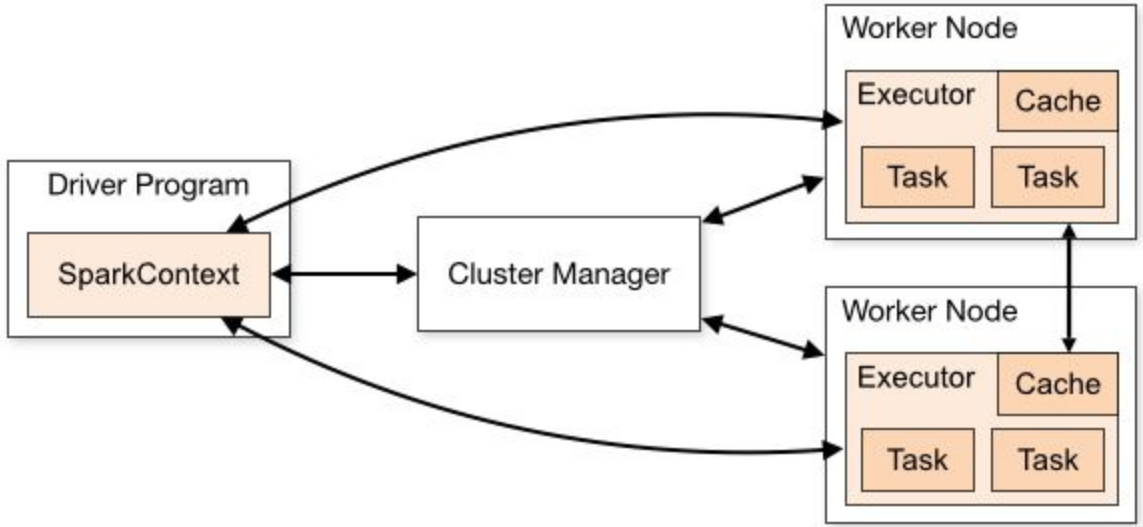
```
//Action
List<Tuple2<String, Integer>> result = words.collect();
```

Word count sample

```
SparkConf conf = new SparkConf()
  .setAppName("Wordcount")
  .setMaster("local[*]");
JavaSparkContext sc = new JavaSparkContext(conf);



//Transformations
JavaRDD<String> words = sc.textFile(myFilePath)
                                    .flatMap(line -> Arrays.asList(line.split(" ")));
JavaPairRDD<String, Integer> pairs = words.mapToPair(x -> new Tuple2(x,1))
                                                .reduceByKey((a,b) -> a+b)
                                                .filter(tuple -> tuple._2() > 3);


//Action
List<Tuple2<String, Integer>> result = words.collect();
```

Spark on cluster

- mesos
- YARN
- standalone



Source: http://spark.apache.org

Spark key facts

- fast
- flexible
- easy to use


Real time analytics


Spark

&


cassandra

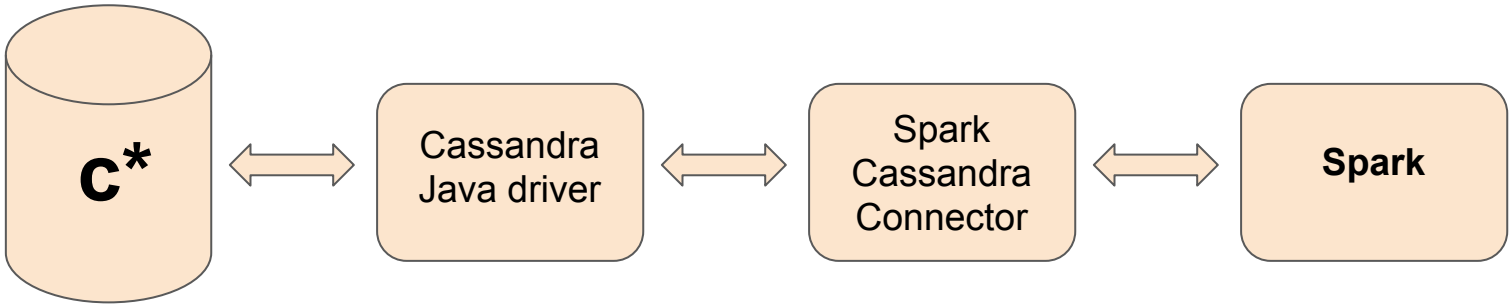
Spark Cassandra Connector

- Open source
- Version 1.5
- Implemented in Scala
- Loads data from Cassandra to Spark
- Writes data from Spark to Cassandra

https://github.com/datastax/spark-cassandra-connector

Spark Cassandra connector

Exposes Cassandra tables as Spark RDD



Spark connection setup

```
SparkConf sparkConf = new SparkConf()
 .setAppName("activityRecognition")
 .set("spark.cassandra.connection.host", "127.0.0.1"))
 .set("spark.cassandra.connection.native.port", "9142")
 .setMaster("local");

JavaSparkContext sc = new JavaSparkContext(sparkConf);
```

From Cassandra to Spark

```
CassandraJavaRDD<CassandraRow> cassandraRowsRDD = javaFunctions
(sc).cassandraTable("activityrecognition", "training");
```

```
JavaRDD<Long> times = cassandraRowsRDD.select("timestamp")
                        .where("user_id=? AND activity=?", user, activity)
                        .map(CassandraRow::toMap)
                        .map(entry -> (long) entry.get("timestamp"))
                    .cache();
```

From Cassandra to Spark

```
CassandraJavaRDD<CassandraRow> cassandraRowsRDD = javaFunctions
(sc).cassandraTable("activityrecognition", "training");
```

```
JavaRDD<Long> times = cassandraRowsRDD.select("timestamp")
                        .where("user_id=? AND activity=?", user, activity)
                        .map(CassandraRow::toMap)
                        .map(entry -> (long) entry.get("timestamp"))
                    .cache();
```

Spark streaming

Scalable, high-throughput, fault-tolerant stream processing

Source: http://spark.apache.org

Spark Streaming

```
// declare a streaming context
JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, Durations.seconds(2));
```

```
JavaReceiverInputDStream<String> cassandraReceiver = ssc.receiverStream(
                new CassandraReceiver(StorageLevel.MEMORY_ONLY(), ssc.sparkContext()) // custom Cassandra receiver
        );


… // The transformations are done here
cassandraReceiver.print(); // print the receiver value


ssc.start();
ssc.awaitTermination(); // Wait for the computation to terminate
```

Spark Streaming

```
// declare a streaming context
JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, Durations.seconds(2));

JavaReceiverInputDStream<String> cassandraReceiver = ssc.receiverStream(
                new CassandraReceiver(StorageLevel.MEMORY_ONLY(), ssc.sparkContext()) // custom Cassandra receiver
        );

…  // The transformations are done here
cassandraReceiver.print(); // print the receiver value


ssc.start();
ssc.awaitTermination(); // Wait for the computation to terminate
```

Spark Streaming

```
// declare a streaming context
JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, Durations.seconds(2));


JavaReceiverInputDStream<String> cassandraReceiver = ssc.receiverStream(
                new CassandraReceiver(StorageLevel.MEMORY_ONLY(), ssc.sparkContext()) // custom Cassandra receiver
        );

… // The transformations are done here
cassandraReceiver.print(); // print the receiver value

ssc.start();
ssc.awaitTermination(); // Wait for the computation to terminate
```

Spark Streaming

```
// declare a streaming context
JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, Durations.seconds(2));


JavaReceiverInputDStream<String> cassandraReceiver = ssc.receiverStream(
                new CassandraReceiver(StorageLevel.MEMORY_ONLY(), ssc.sparkContext()) // custom Cassandra receiver
        );


…  // The transformations are done here
cassandraReceiver.print(); // print the receiver value
```

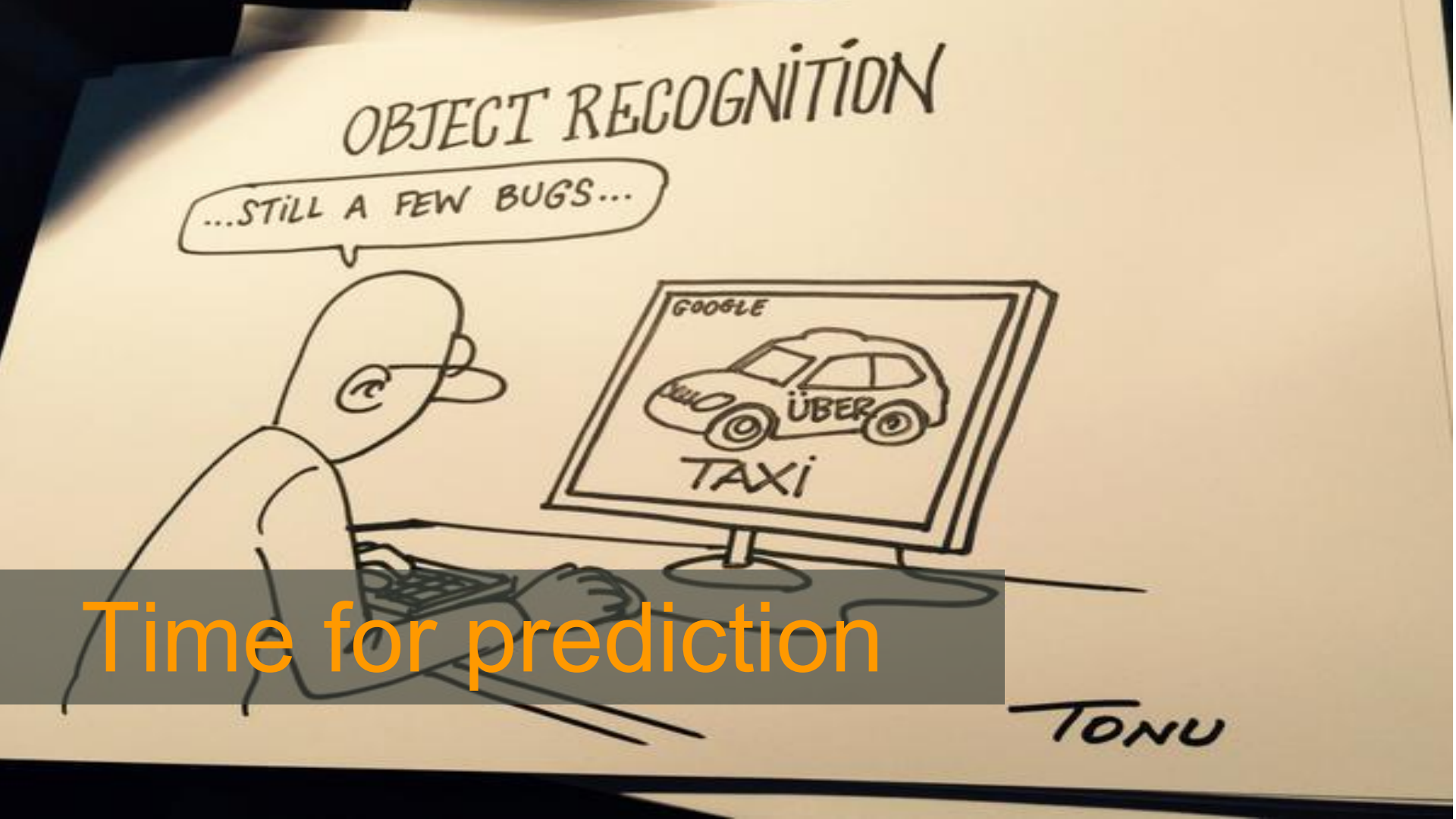
```
ssc.start();
ssc.awaitTermination(); // Wait for the computation to terminate
```

Spark Streaming

```
Public class CassandraReceiver extends Receiver<String>{
 ...

@Override
 public void onStart() {  // Start the thread that receives data over a connection
         new Thread()  {
        @Override public void run() {
           receive(); //Read data from Cassandra, compute features and write prediction into Cassandra
        }
    }.start();
 }
 @Override
 public void onStop() { //Nothing to do   }
...}
```

Spark Streaming

```
Public class CassandraReceiver extends Receiver<String>{
  ...
@Override
  public void onStart() {  // Start the thread that receives data over a connection
          new Thread()  {
         @Override public void run() {
            receive(); //Read data from Cassandra, compute features and write prediction into Cassandra
         }
     }.start();
  }
  @Override
  public void onStop() { //Nothing to do   }
...}
```



Time for prediction

Activity Recognition

Possible activity: walking, jogging, sitting or standing

Measurement from accelerometer : timeseries

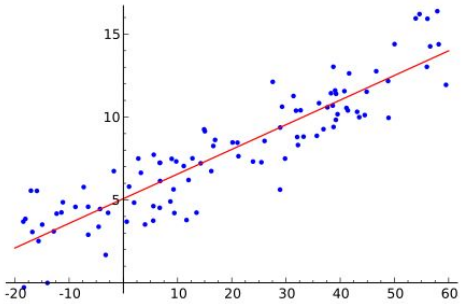
Classifying a timeseries into physical activity classes

Machine Learning

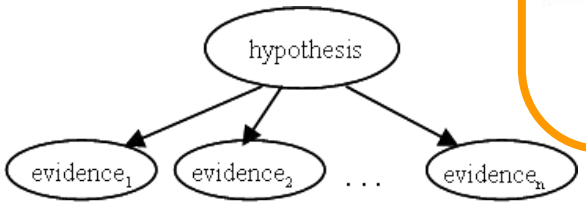
Multiclass classification

labelling unknown pattern based on known patterns

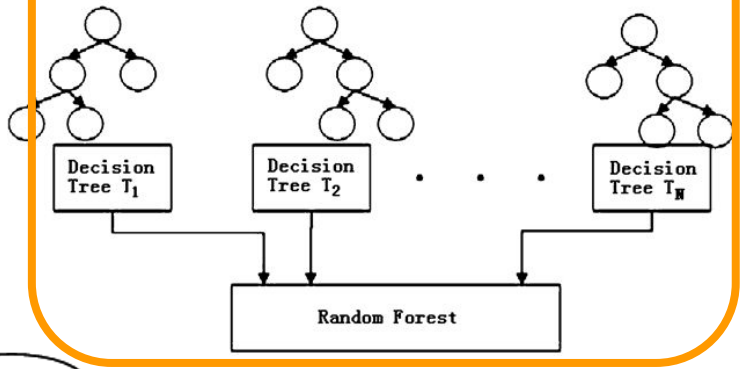
Logistic regression



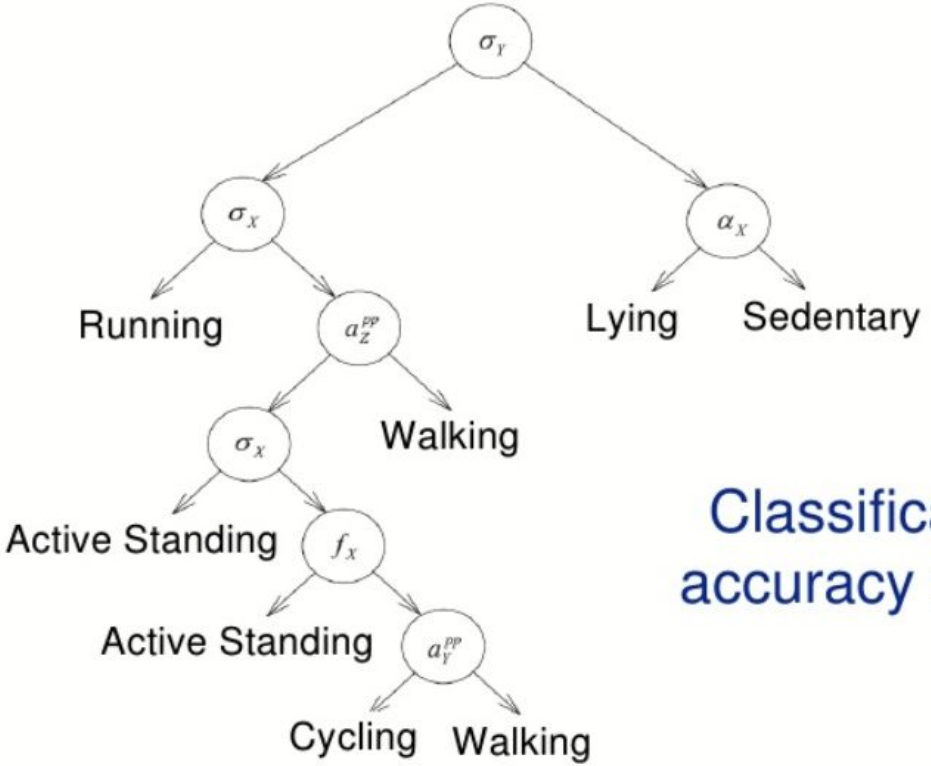
Naive Bayes



Random forest



Decision tree model



Spark MLlib

Classification accuracy > 93%

Supervised learning

Train

Predict

Predictive model

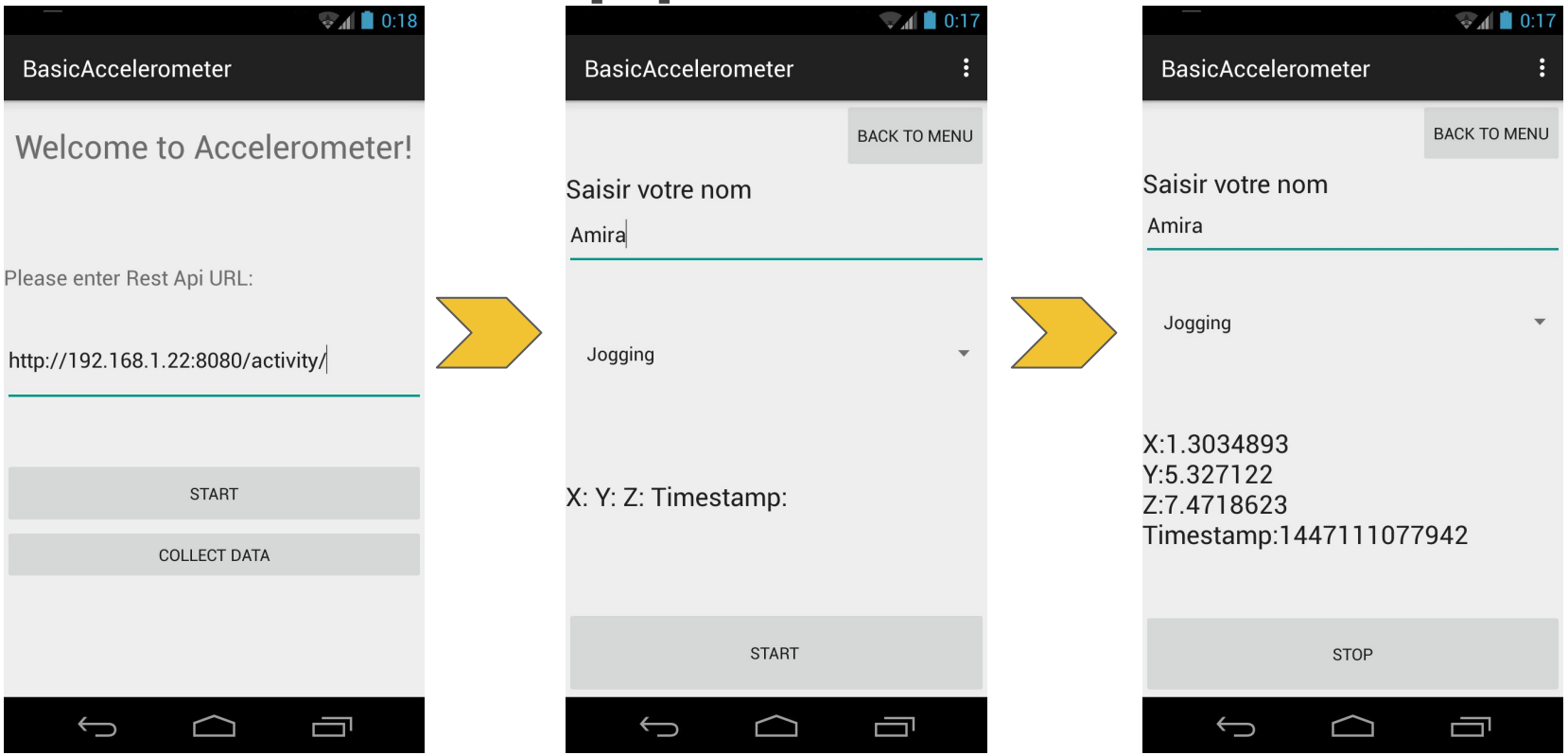
- collect labeled data
- identify the features
- compute the features
- create and train the random forest model
- predict the activity

Special thanks

Collecting data

```
cqlsh:activityrecognition> create table training (
                       ... user_id text,
                       ... activity text,
                       ... timestamp bigint,
                       ... x double,
                       ... y double,
                       ... z double,
                       ... PRIMARY KEY( (user_id, activity), timestamp);
```

Android App


0:18
BasicAccelerometer
Welcome to Accelerometer!
Please enter Rest Api URL:
http://192.168.1.22:8080/activity/
START
COLLECT DATA
0:17
BasicAccelerometer
BACK TO MENU
Saisir votre nom
Amira
Jogging
X: Y: Z: Timestamp:
START
0:17
BasicAccelerometer
BACK TO MENU
Saisir votre nom
Amira
Jogging
X:1.3034893
Y:5.327122
Z:7.4718623
Timestamp:1447111077942
STOP

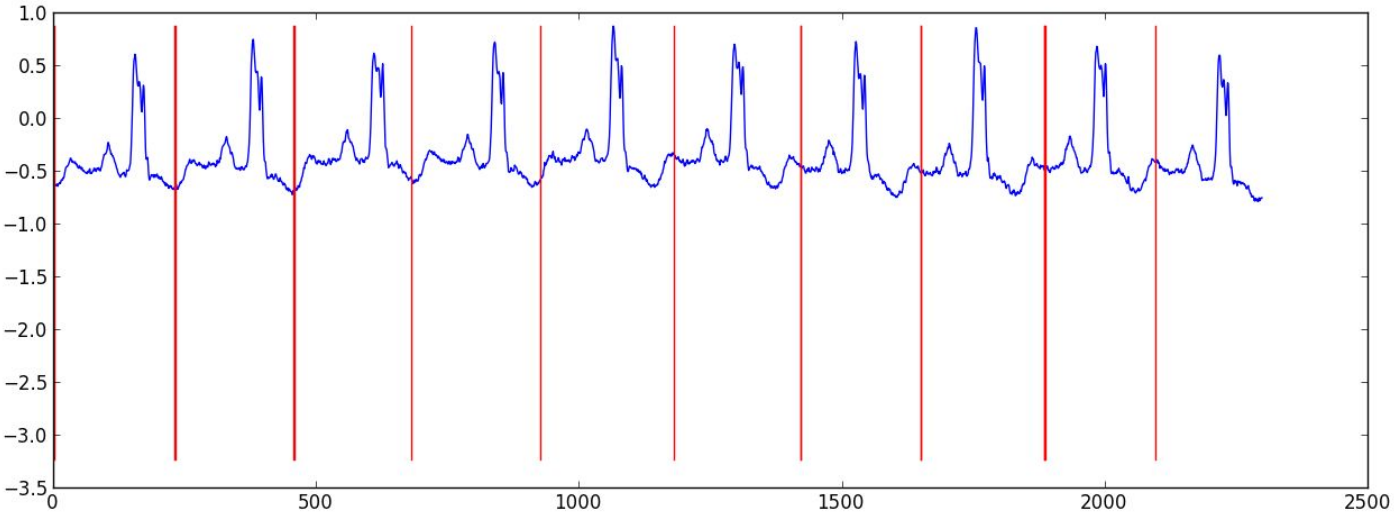
Training data

user_id	activity	timestamp	x	y	z
amira	Standing	1446034733566	-1.49389	-9.31124	0.191003
amira	Standing	1446034733631	-1.45557	-9.34956	0.229323
amira	Standing	1446034733696	-1.45557	-9.54116	0.267643
amira	Standing	1446034733761	-1.45557	-9.4262	-0.000599
amira	Standing	1446034733826	-1.41725	-9.46452	-0.038919
amira	Standing	1446034733890	-1.57053	-9.34956	0.037722
amira	Standing	1446034733955	-1.45557	-9.46452	0.191003
amira	Standing	1446034734019	-1.57053	-9.34956	0.229323
amira	Standing	1446034734085	-1.37893	-9.38788	0.229323
amira	Standing	1446034734149	-1.45557	-9.4262	0.229323

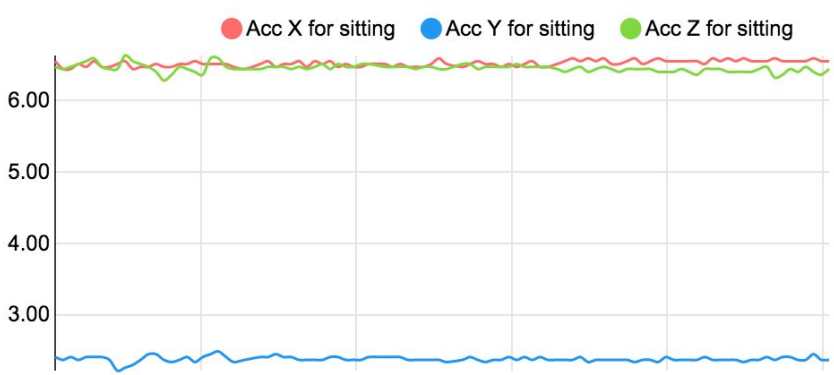
Training data

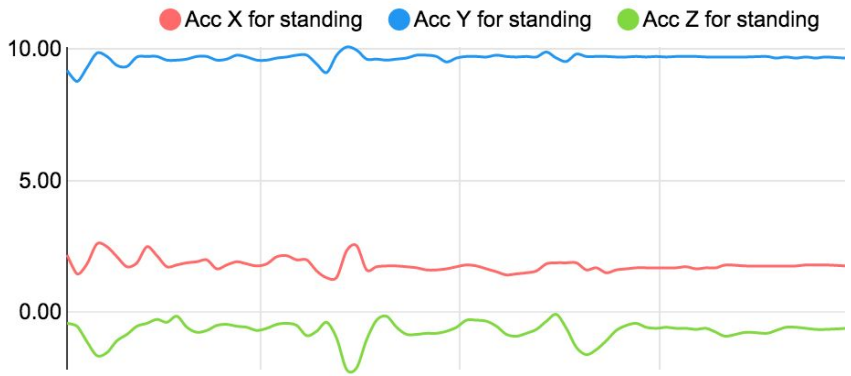
13679 training acceleration => only 1.6 Mo

Prepare data: windows



Timeseries: sitting vs standing

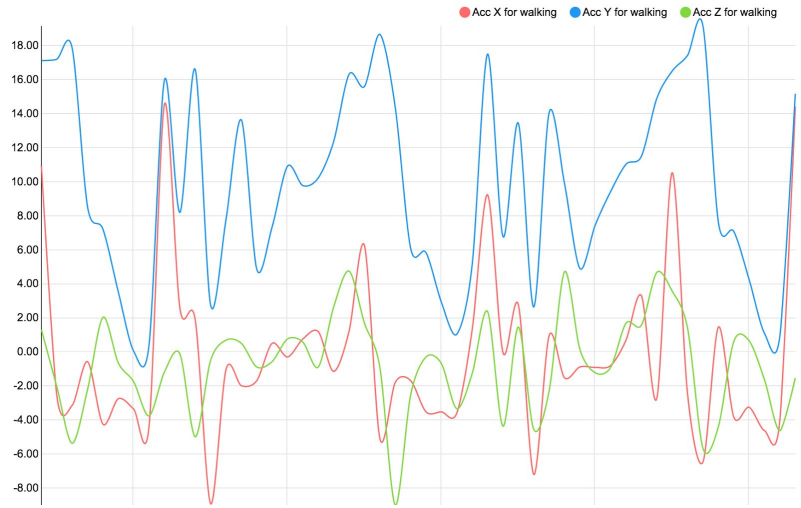


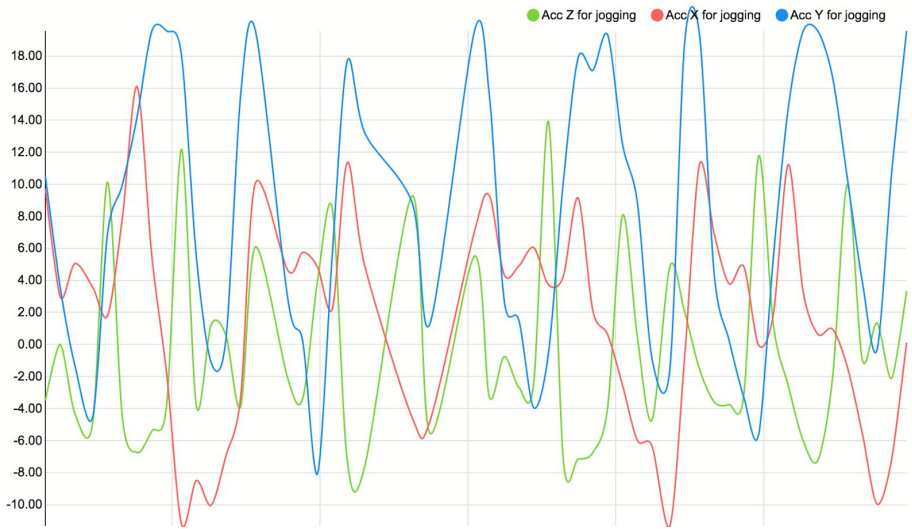


Standing Y> X and Sitting Y <X

Source: cityzendata.com

Timeseries: walking vs Jogging





Y peak to peak amplitude

Source: cityzendata.com

Features extraction

- Average acceleration (for each axis)
- Average difference for X and Y axis
- Variance (for each axis)
- Standard deviation (for each axis) $\sqrt{\ 1/n * \sum (x - mean\_x)}$
- Average absolute difference (for each axis)
- Average resultant acceleration $1/n * \sum \sqrt{(x^2 + y^2 + z^2)}$
- Average peak to peak amplitude for Y axis

Compute features

```
JavaRDD<double[]> accelerationData = dataFromCassandra.map(CassandraRow::toMap).map(
        row -> new double[]{(double) row.get("x"), (double) row.get("y"), (double) row.get("z")});

JavaRDD<Vector> vectorsXYZ = accelerationData.map(Vectors::dense);
```

```
MultivariateStatisticalSummary statisticalSummary  = Statistics.colStats(accelerationData.rdd());
double[] meanArray = statisticalSummary.mean().toArray();
double[] varianceArray = statisticalSummary.variance().toArray();
double difference = extractFeature.computeDifferenceBetweenAxes(meanArray);
```

```
//build LabeledPoint with an activity label
LabeledPoint labeledPoint =new LabeledPoint(label, Vectors.dense(features)
)
```

Compute features

```
JavaRDD<double[]> accelerationData = dataFromCassandra.map(CassandraRow::toMap).map(
        row -> new double[]{(double) row.get("x"), (double) row.get("y"), (double) row.get("z")});

JavaRDD<Vector> vectorsXYZ = accelerationData.map(Vectors::dense);
```

```
MultivariateStatisticalSummary statisticalSummary  = Statistics.colStats(accelerationData.rdd());
double[] meanArray = statisticalSummary.mean().toArray();
double[] varianceArray = statisticalSummary.variance().toArray();
double difference = extractFeature.computeDifferenceBetweenAxes(meanArray);
```

```
//build LabeledPoint with an activity label
LabeledPoint labeledPoint =new LabeledPoint(label, Vectors.dense(features)
)
```

Compute features

```
JavaRDD<double[]> accelerationData = dataFromCassandra.map(CassandraRow::toMap).map(
        row -> new double[]{(double) row.get("x"), (double) row.get("y"), (double) row.get("z")});

JavaRDD<Vector> vectorsXYZ = accelerationData.map(Vectors::dense);


MultivariateStatisticalSummary statisticalSummary  = Statistics.colStats(accelerationData.rdd());
double[] meanArray = statisticalSummary.mean().toArray();
double[] varianceArray = statisticalSummary.variance().toArray();
double difference = extractFeature.computeDifferenceBetweenAxes(meanArray);
```

```
//build LabeledPoint with an activity label
LabeledPoint labeledPoint =new LabeledPoint(label, Vectors.dense(features)
)
```

Decision Tree

```
Map<Integer, Integer> categoricalFeaturesInfo = new HashMap<>();
int numClasses = ActivityType.values().length; //num of classes = num of activity to predict
String impurity = "gini"; //measure of the homogeneity of the labels at the node ∑Ci=1fi(1−fi)
int maxDepth = 20;
int maxBins = 32; //minimum value for bins
```

```
// create model
final DecisionTreeModel model =
DecisionTree.trainClassifier(trainingData, numClasses, categoricalFeaturesInfo, impurity, maxDepth, maxBins);
model.save(sc.sc(), "predictionModel/decisionTree");
```

```
// Compute classification accuracy on test data
final long correctPredictionCount = testData
                                        .mapToPair(p -> new Tuple2<>(model.predict(p.features()), p.label()))
                                        .filter(pl -> pl._1().equals(pl._2()))
                                    .count();

Double classificationAccuracy = 1.0 * correctPredictionCount / testData.count();
```

Decision Tree

```
Map<Integer, Integer> categoricalFeaturesInfo = new HashMap<>();
int numClasses = ActivityType.values().length; //num of classes = num of activity to predict
String impurity = "gini"; //measure of the homogeneity of the labels at the node ∑Ci=1fi(1−fi)
int maxDepth = 20;
int maxBins = 32; //minimum value for bins
```

```
// create model
final DecisionTreeModel model =
DecisionTree.trainClassifier(trainingData, numClasses, categoricalFeaturesInfo, impurity, maxDepth, maxBins);
model.save(sc.sc(), "predictionModel/decisionTree");
```

```
// Compute classification accuracy on test data
final long correctPredictionCount = testData
                                        .mapToPair(p -> new Tuple2<>(model.predict(p.features()), p.label()))
                                        .filter(pl -> pl._1().equals(pl._2()))
                                .count();

Double classificationAccuracy = 1.0 * correctPredictionCount / testData.count();
```

Decision Tree

```
Map<Integer, Integer> categoricalFeaturesInfo = new HashMap<>();
int numClasses = ActivityType.values().length; //num of classes = num of activity to predict
String impurity = "gini"; //measure of the homogeneity of the labels at the node ∑Ci=1fi(1−fi)
int maxDepth = 20;
int maxBins = 32; //minimum value for bins


// create model
final DecisionTreeModel model =
DecisionTree.trainClassifier(trainingData, numClasses, categoricalFeaturesInfo, impurity, maxDepth, maxBins);
model.save(sc.sc(), "predictionModel/decisionTree");
```

```
// Compute classification accuracy on test data
final long correctPredictionCount = testData
                                        .mapToPair(p -> new Tuple2<>(model.predict(p.features()), p.label()))
                                        .filter(pl -> pl._1().equals(pl._2()))
                                    .count();

Double classificationAccuracy = 1.0 * correctPredictionCount / testData.count();
```

Random forest

```
// parameters
Map<Integer, Integer> categoricalFeaturesInfo = new HashMap<Integer, Integer>();
int numTrees = 10;
int numClasses = ActivityType.values().length;
String featureSubsetStrategy = "auto"; //Number of features to consider for splits at each node
String impurity = "gini";
int maxDepth = 20;
int maxBins = 32;
int randomSeeds = 12345; //Random seed for bootstrapping and choosing feature subsets.
```

```
// create model
RandomForestModel model = RandomForest.trainClassifier(trainingData, numClasses,
categoricalFeaturesInfo, numTrees, featureSubsetStrategy, impurity, maxDepth, maxBins, randomSeeds);
model.save(sc.sc(), "predictionModel/randomForest");
```

Accuracy

- 13679 training acceleration
- 15 features

Decision tree accuracy 69.23%

Random forest accuracy 92.3%


Time for the demo!

Conclusion

- Collect data for a device and analyse it.
- Sensors can be combined
- Infinites possibilities with IOT


Thank You