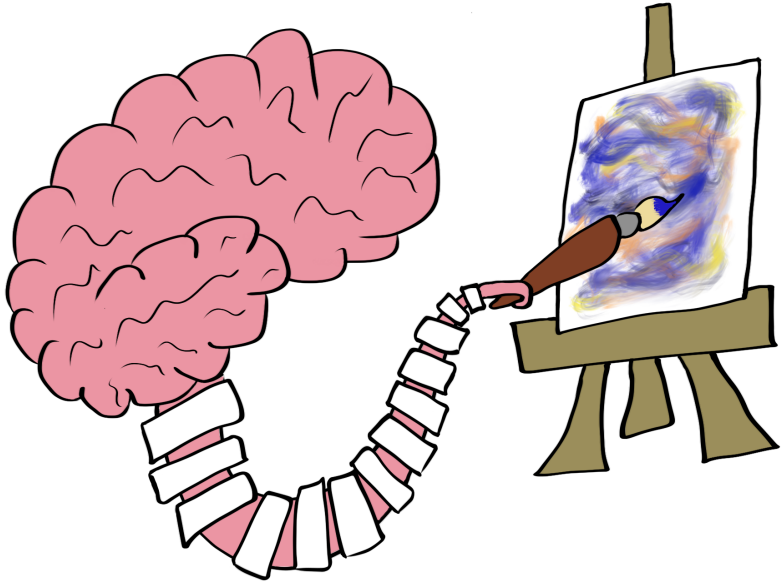
Reactive Feature Generation with Spark and MLlib

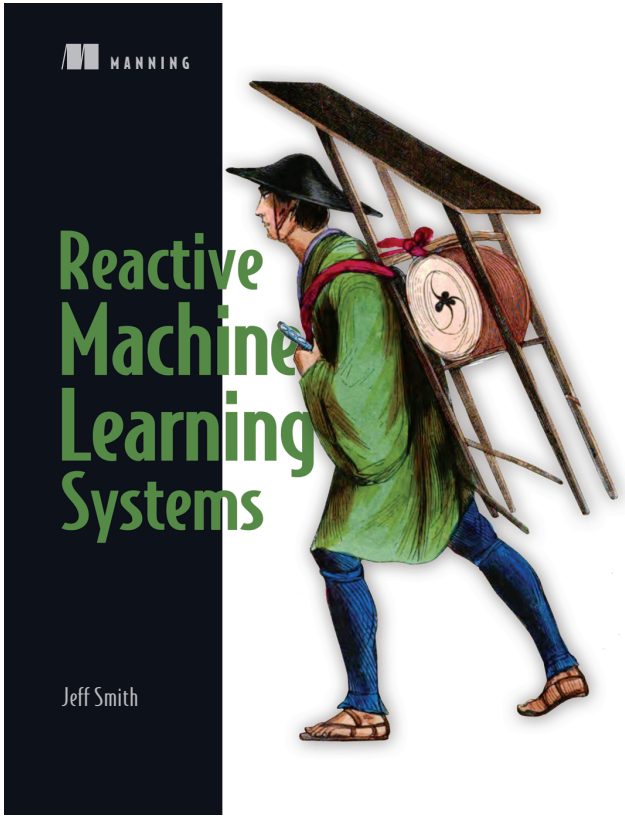
Jeff Smith
x.ai

SPARK SUMMIT EAST
DATA SCIENCE AND ENGINEERING AT SCALE
FEBRUARY 16-18, 2016 NEW YORK CITY

x.ai is a personal assistant

who schedules meetings for you

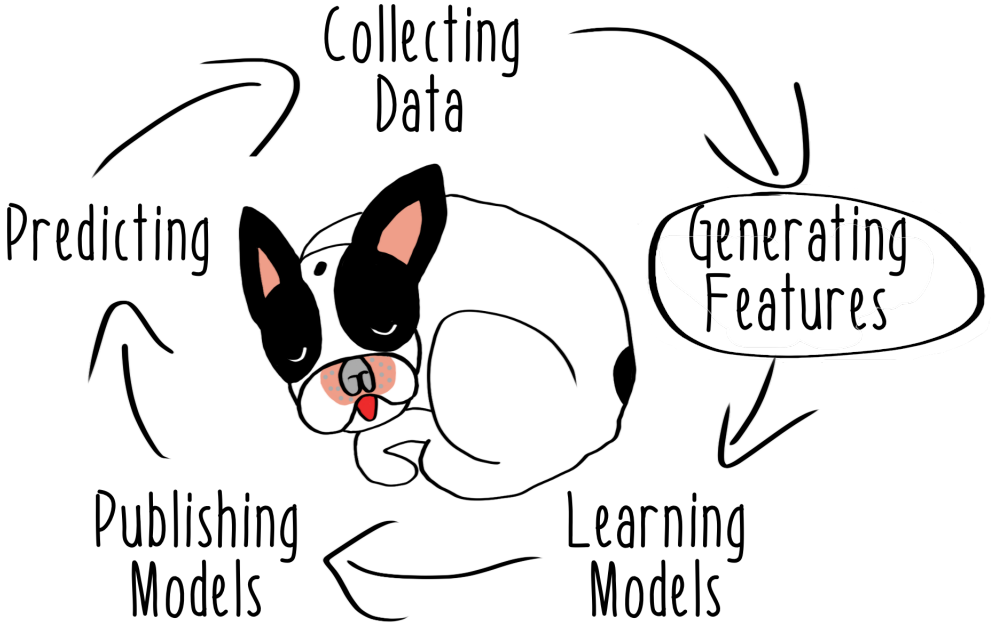



MANNING
Reactive Machine Learning Systems
Jeff Smith

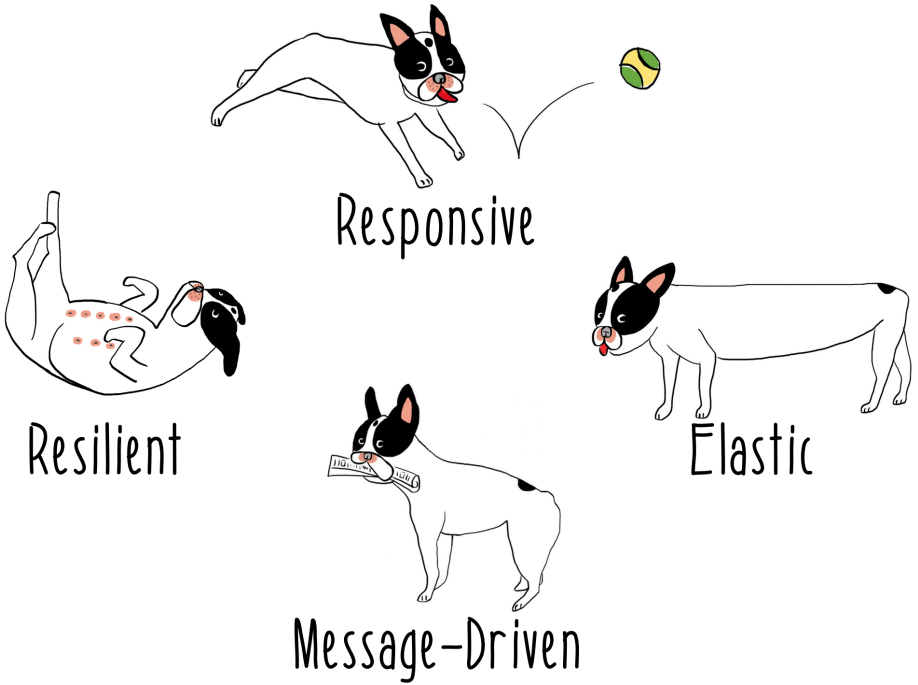
REACTIVE MACHINE LEARNING

Reactive Machine Learning

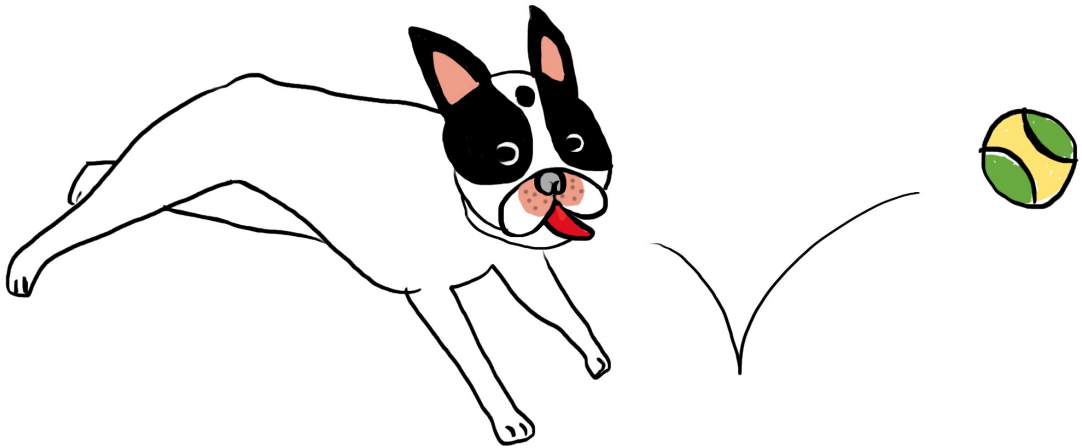
Machine Learning Systems



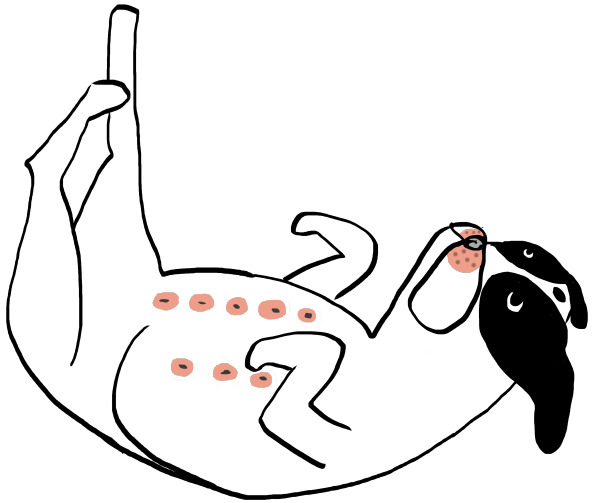
Traits of Reactive Systems



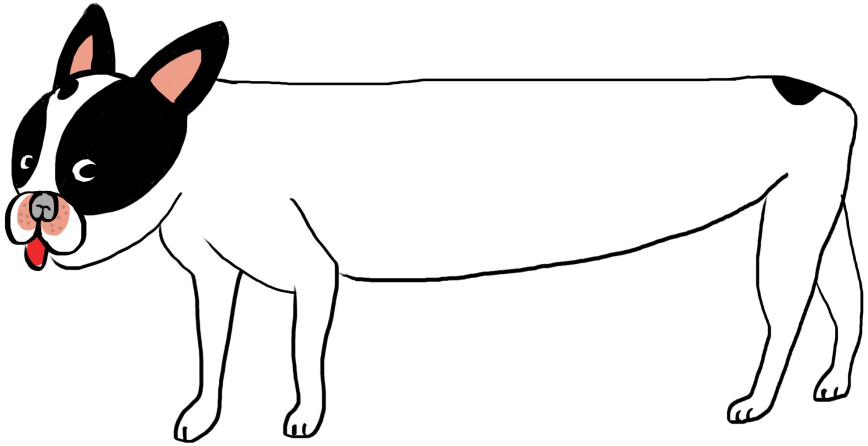
Responsive



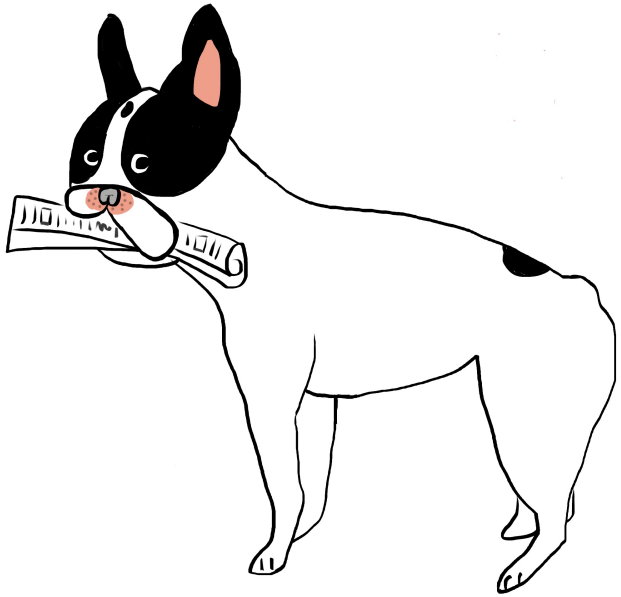
Resilient



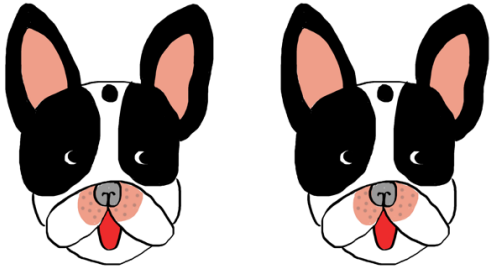
Elastic



Message-Driven

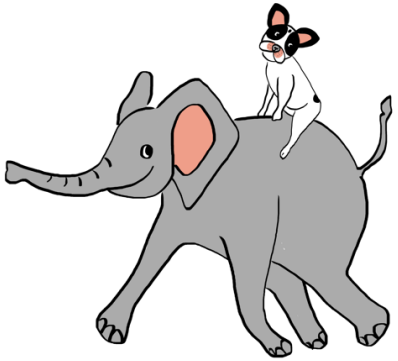


Reactive Strategies



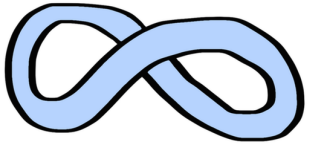
Replication

Containment

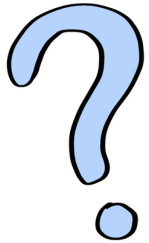


Supervision

Machine Learning Data

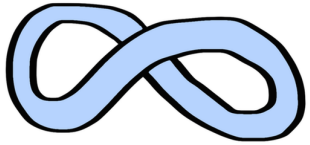


Infinite Data

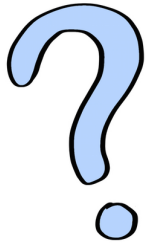


Uncertain Data

Machine Learning Data



Infinite Data



Uncertain Data

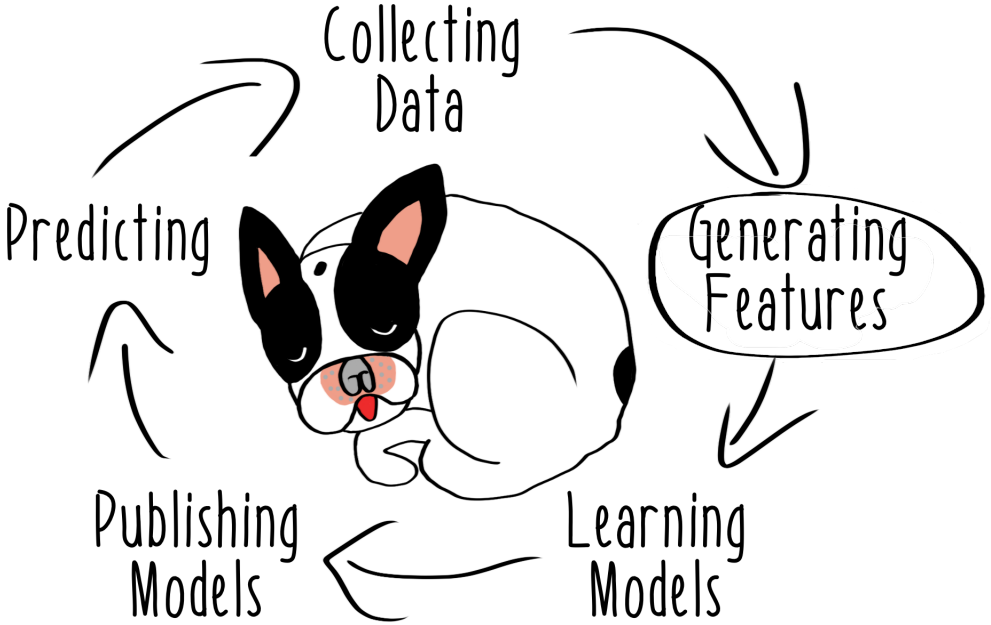
Laziness

Pure Functions

Immutable Facts

Possible Worlds

Machine Learning Systems



INTRODUCING FEATURES

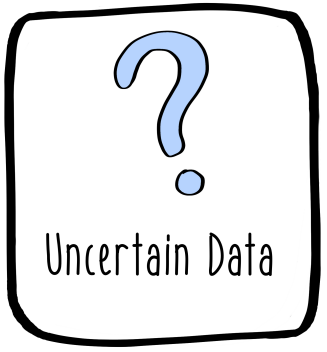
Microblogging Data

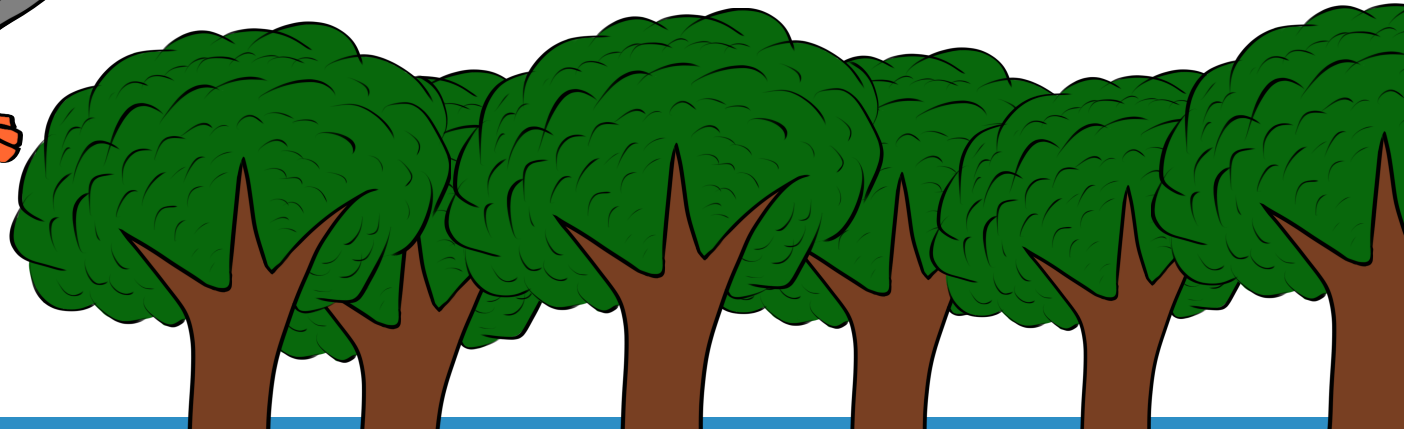
Squawks

Squawkers

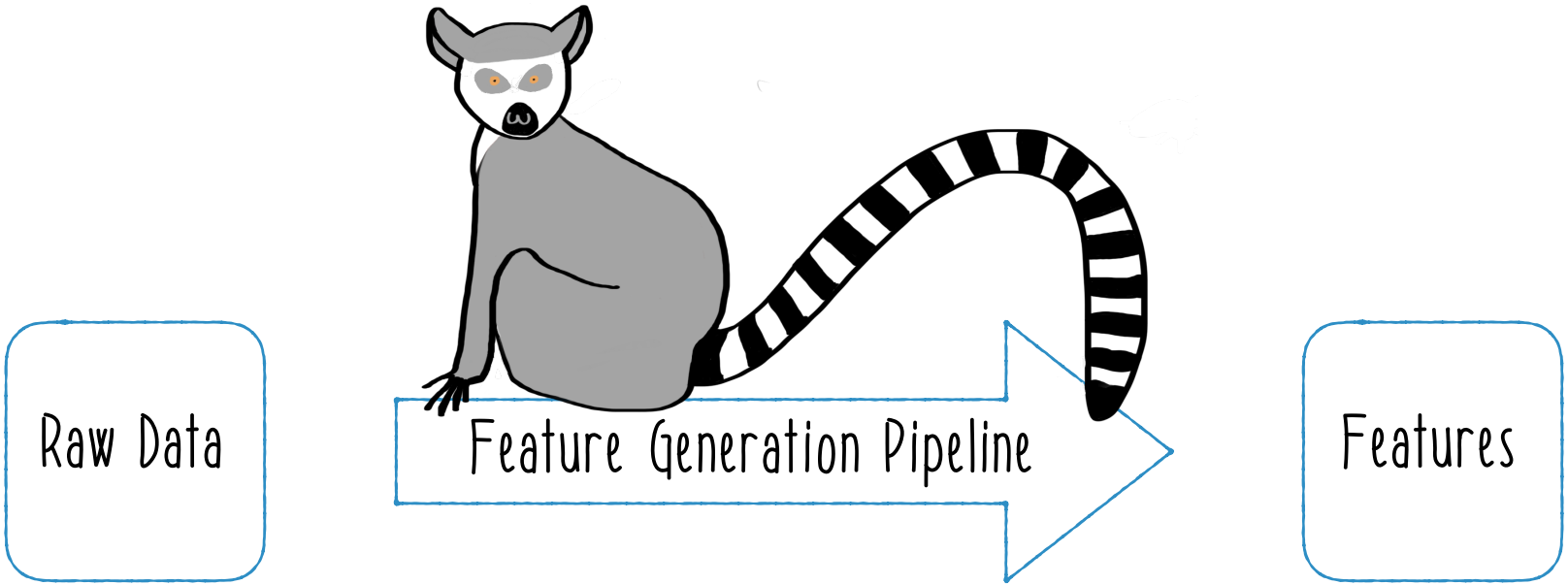
Super

Pidg 'n



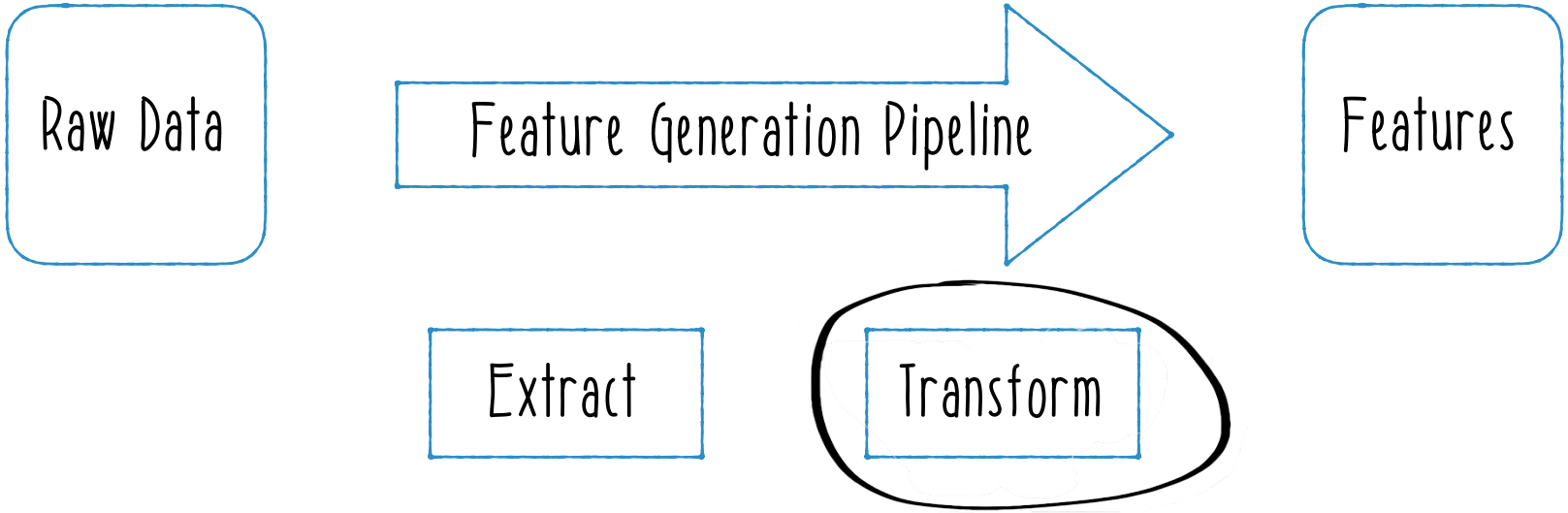


Feature Generation



FEATURE TRANSFORMS

Feature Generation



Features

```
trait FeatureType[V] {
  val name: String
}

trait Feature[V] extends FeatureType[V] {
  val value: V
}
```

Features

```
case class IntFeature(name: String, value: Int)
  extends Feature[Int]

case class BooleanFeature(name: String, value: Boolean)
  extends Feature[Boolean]
```

Named Transforms

```
def binarize(feature: IntFeature, threshold: Double) = {
  BooleanFeature("binarized-" + feature.name,
                 feature.value > threshold)
}
```

Non-Trivial Transforms

```
def categorize(thresholds: List[Double]) = {
  (rawFeature: DoubleFeature) => {
    IntFeature("categorized-" + rawFeature.name,
      thresholds.sorted
        .zipWithIndex
        .find {
        case (threshold, i) => rawFeature.value < threshold
      }.getOrElse((None, -1))
        ._2)
  }
}
```

Standardizing Naming

```
trait Named {
  def name(inputFeature: Feature[Any]) : String = {
    inputFeature.name + "-" +
      Thread.currentThread.getStackTrace()(3).getMethodName
  }
}

object Binarizer extends Named {
  def binarize(feature: IntFeature, threshold: Double) = {
    BooleanFeature(name(feature), feature.value > threshold)
  }
}
```

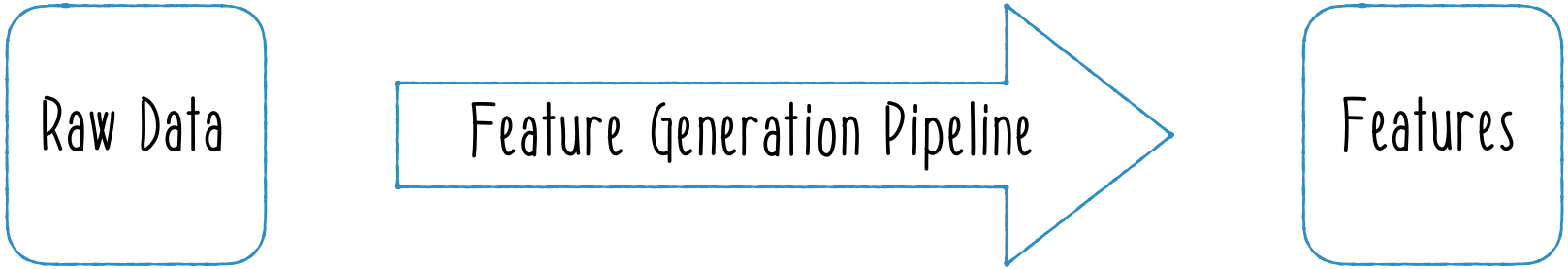
Lineages

```
"cleaned-normalized-categorized-interactions"
```

```
interactions
  categorized
    normalized
      cleaned
```

PIPELINES

Feature Generation

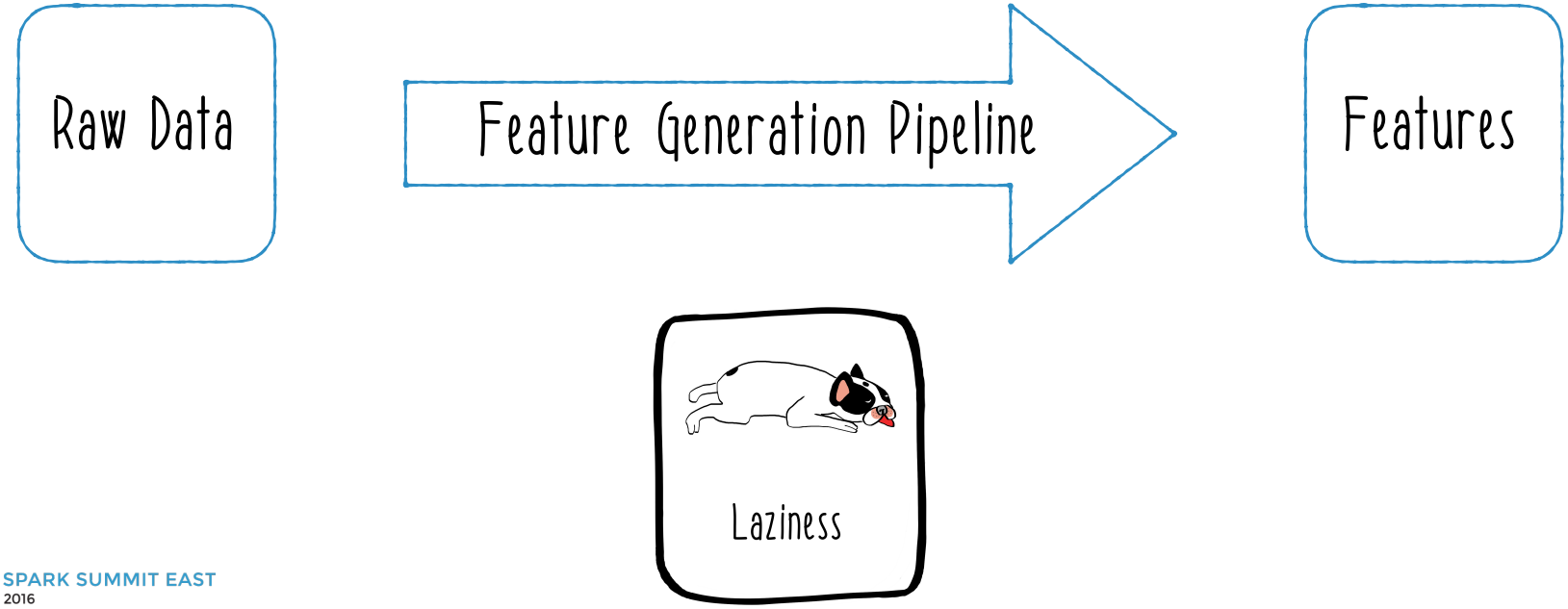


Multi-Stage Generation

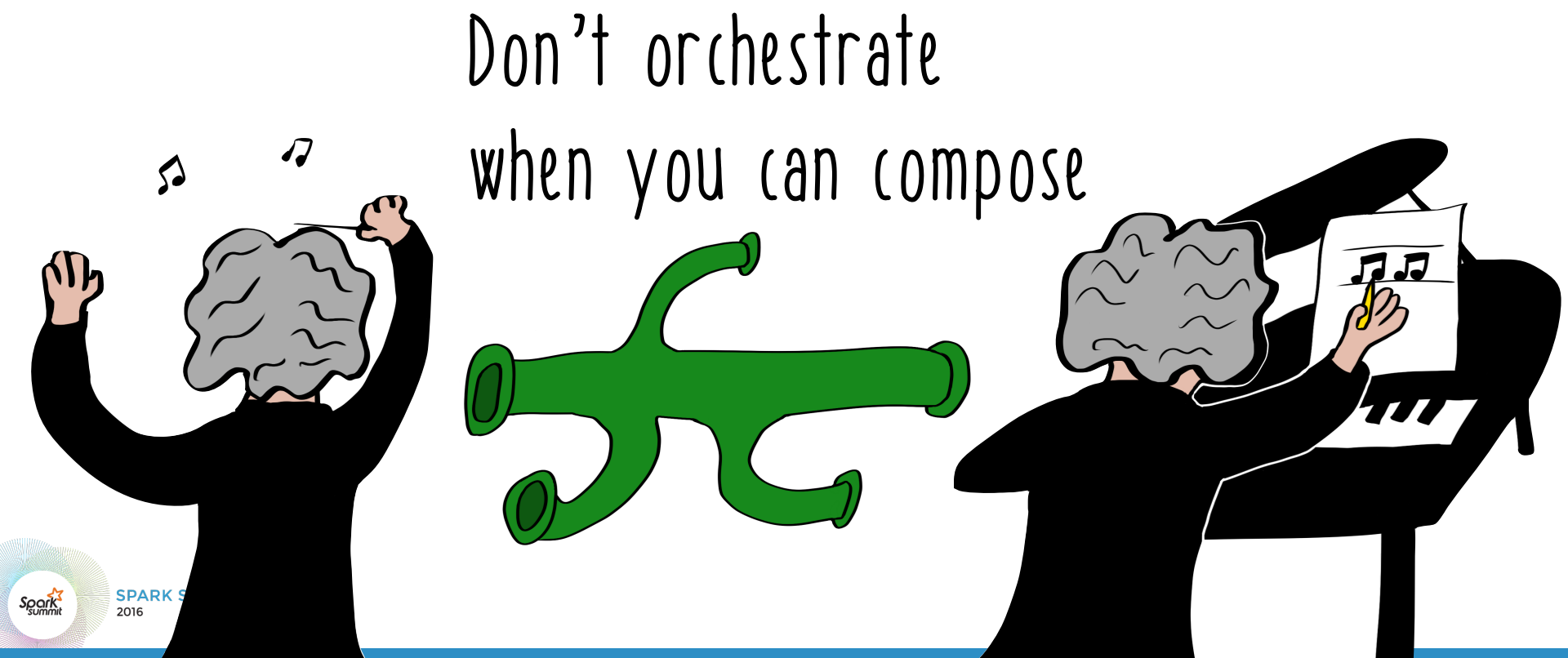
Pipeline Composition

```
def extract(rawSquawks: RDD[JsonDocument]):
      RDD[IntFeature] = {
  ???
}

def transform(inputFeatures: RDD[Feature[Int]]):
      RDD[BooleanFeature] = {
  ???
}

val trainableFeatures = transform(extract(rawSquawks))
```

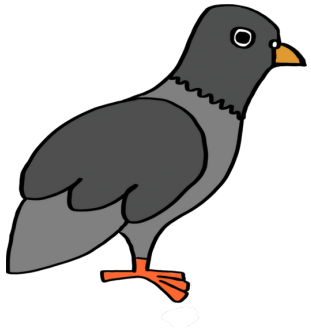
Feature Generation



Pipelines



Pipeline Failure



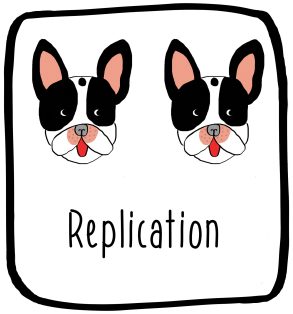
Raw Data

Feature Generation Pipeline

Features

Raw Data

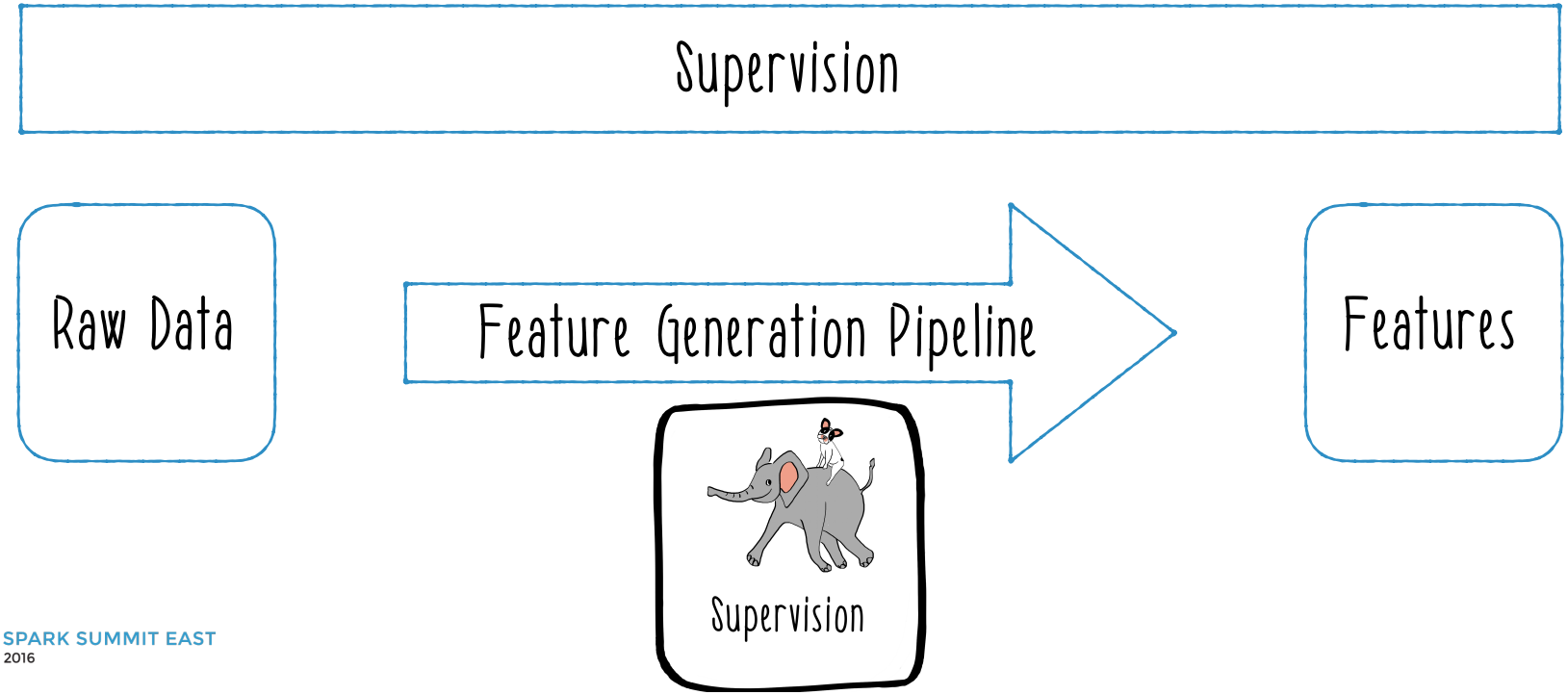
Featı



ipeline

Features

Supervising Feature Generation



Supervising Feature Generation

Reactive DB Drivers

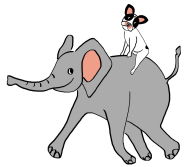
Cluster Managers

Feature Validation

Raw Data

Feature Generation Pipeline

Features



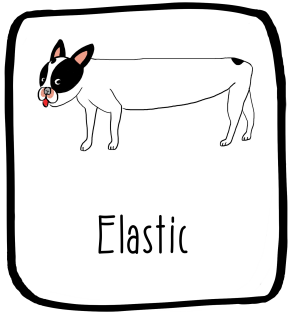
Supervision

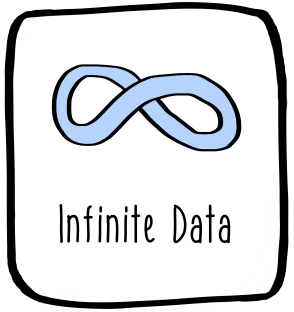
Reactive Database Drivers

Couchbase

MongoDB

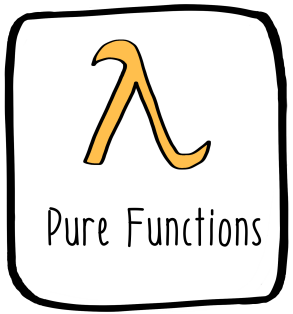
Cassandra





Reactive Database Drivers

```
val rawSquawks: RDD[JsonDocument] = sc.couchbaseView(
  ViewQuery.from("squawks", "by_squawk_id"))
  .map(_.id)
  .couchbaseGet[JsonDocument]()
```

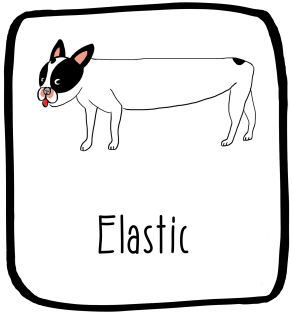


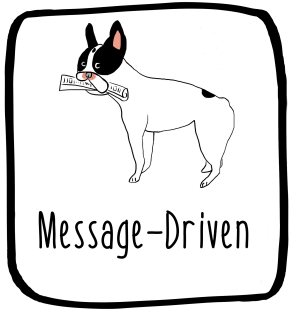
Cluster Managers

Spark Standalone

Mesos

YARN





Supervising Feature Generation

Reactive DB Drivers

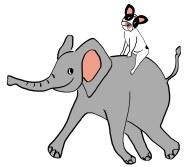
Cluster Managers

Feature Validation

Raw Data

Feature Generation Pipeline

Features



Supervision

FEATURE COLLECTIONS

Original Features

```
object SquawkLength extends FeatureType[Int]

object Super extends LabelType[Boolean]

val originalFeatures: Set[FeatureType] = Set(SquawkLength)
val label = Super
```

Basic Features

```
object PastSquawks extends FeatureType[Int]

val basicFeatures = originalFeatures + PastSquawks
```

More Features

```
object MobileSquawker extends FeatureType[Boolean]

val moreFeatures = basicFeatures + MobileSquawker
```

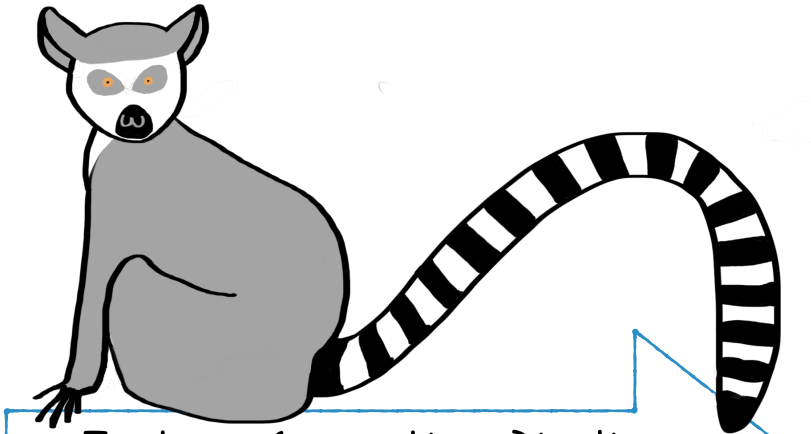
Feature Collections

```
case class FeatureCollection(id: Int,
                             createdAt: DateTime,
                             features: Set[_ <: FeatureType[_]],
                             label: LabelType[_])
```

Feature Collections

```
val earlierCollection = FeatureCollection(101,
                                          earlier,
                                          basicFeatures,
                                          label)

val latestCollection = FeatureCollection(202,
                                         now,
                                         moreFeatures,
                                         label)

val featureCollections = sc.parallelize(
  Seq(earlierCollection, latestCollection))
```

Feature Generation



Raw Data

Feature Generation Pipeline

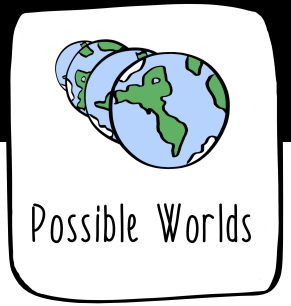
Features

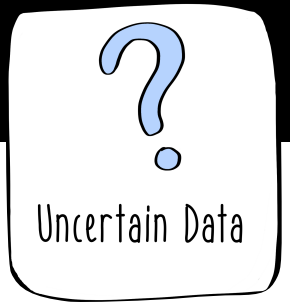
Fallback Collections

```
val FallbackCollection = FeatureCollection(404,
                                           beginningOfTime,
                                           originalFeatures,
                                           label)
```

Fallback Collections

```
def validCollection(collections: RDD[FeatureCollection],
                    invalidFeatures: Set[FeatureType[_]]) = {
  val validCollections = collections.filter(
    fc => !fc.features
      .exists(invalidFeatures.contains))
    .sortBy(collection => collection.id)
  if (validCollections.count() > 0) {
    validCollections.first()
  } else
    FallbackCollection
}
```





VALIDATING FEATURES

Supervising Feature Generation

Reactive DB Drivers

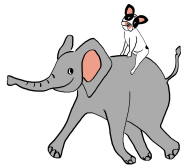
Cluster Managers

Feature Validation

Raw Data

Feature Generation Pipeline

Features



Supervision

Predicting Super Squawkers

Training Instances

```
val instances = Seq(
  (123, Vectors.dense(0.2, 0.3, 16.2, 1.1), 0.0),
  (456, Vectors.dense(0.1, 1.3, 11.3, 1.2), 1.0),
  (789, Vectors.dense(1.2, 0.8, 14.5, 0.5), 0.0)
)
```

Selection Setup

```
val featuresName = "features"
val labelName = "isSuper"

val instancesDF = sqlContext.createDataFrame(instances)
  .toDF("id", featuresName, labelName)

val K = 2
```

Feature Selection

```
val selector = new ChiSqSelector()
  .setNumTopFeatures(K)
  .setFeaturesCol(featuresName)
  .setLabelCol(labelName)
  .setOutputCol("selectedFeatures")
```

Feature Selection

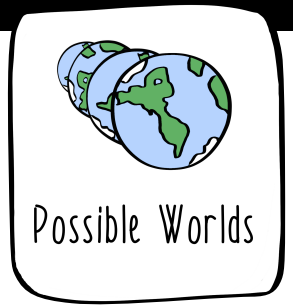
```
val selector = new ChiSqSelector()
  .setNumTopFeatures(K)
  .setFeaturesCol(featuresName)
  .setLabelCol(labelName)
  .setOutputCol("selectedFeatures")

val selectedFeatures = selector.fit(instancesDF)
  .transform(instancesDF)
```

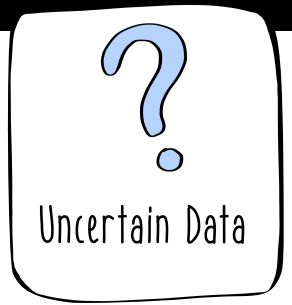
Back to RDDs

```
val labeledPoints = sc.parallelize(instances.map({
  case (id, features, label) =>
    LabeledPoint(label = label, features = features)
}))
```

Validating Features

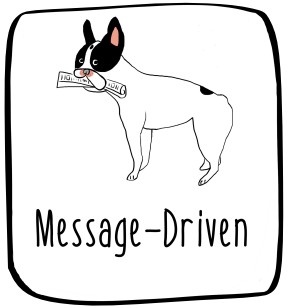
```
def validateSelection(labeledPoints: RDD[LabeledPoint],
                      topK: Int, cutoff: Double) = {
  val pValues = Statistics.chiSqTest(labeledPoints)
    .map(result => result.pValue)
    .sorted
  pValues(topK) < cutoff
}
```





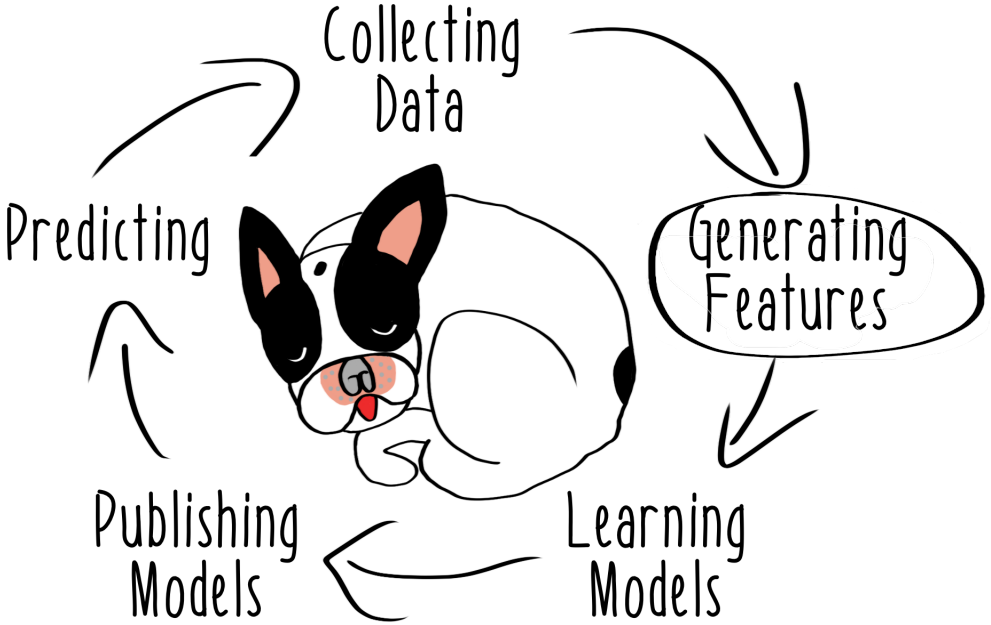
Persisting Validations

```
case class ValidatedFeatureCollection(id: Int,
                                      createdAt: DateTime,
                                      features: Set[_ <: FeatureType[_]],
                                      label: LabelType[_],
                                      passedValidation: Boolean,
                                      cutoff: Double)
```

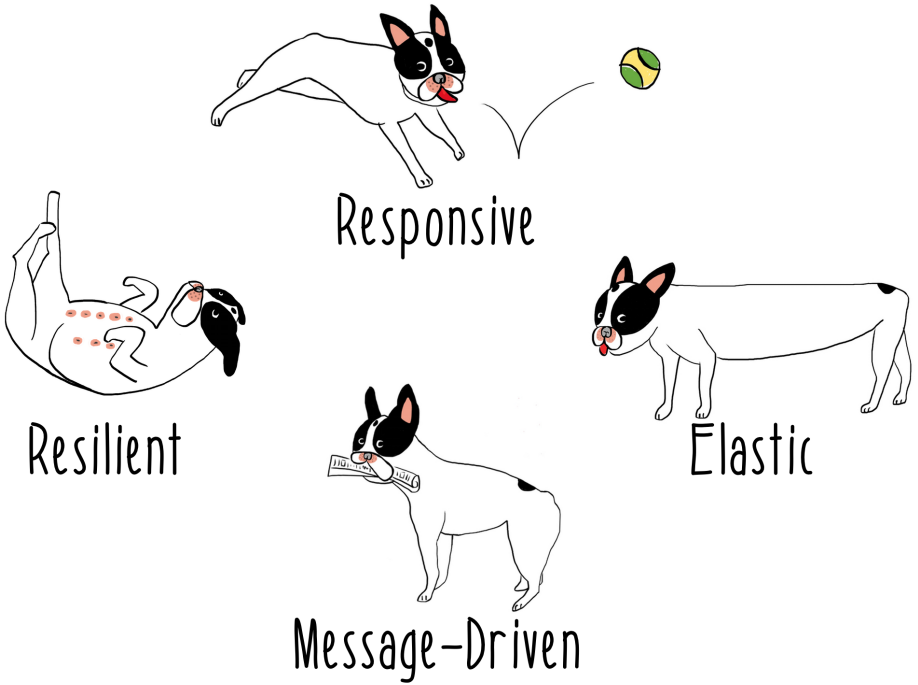


SUMMARY

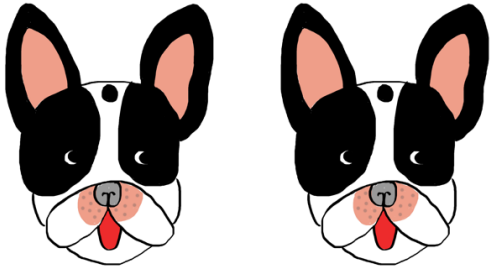
Machine Learning Systems



Traits of Reactive Systems

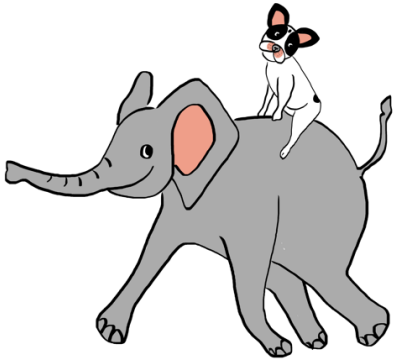


Reactive Strategies



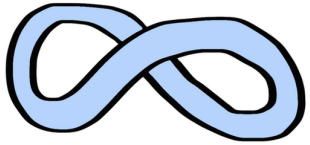
Replication

Containment

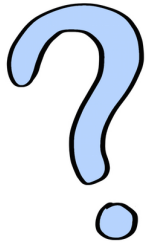


Supervision

Machine Learning Data



Infinite Data



Uncertain Data

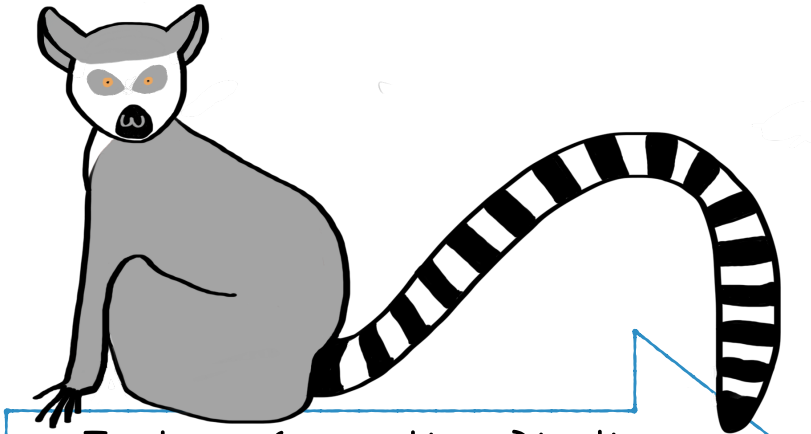
Laziness

Pure Functions

Immutable Facts

Possible Worlds

Feature Generation



Raw Data

Feature Generation Pipeline

Features

Reactive Feature Generation with Spark and MLlib

reactivemachinelearning.com
@jeffksmithjr
@xdotai

SPARK SUMMIT EAST
DATA SCIENCE AND ENGINEERING AT SCALE
FEBRUARY 16-18, 2016 NEW YORK CITY