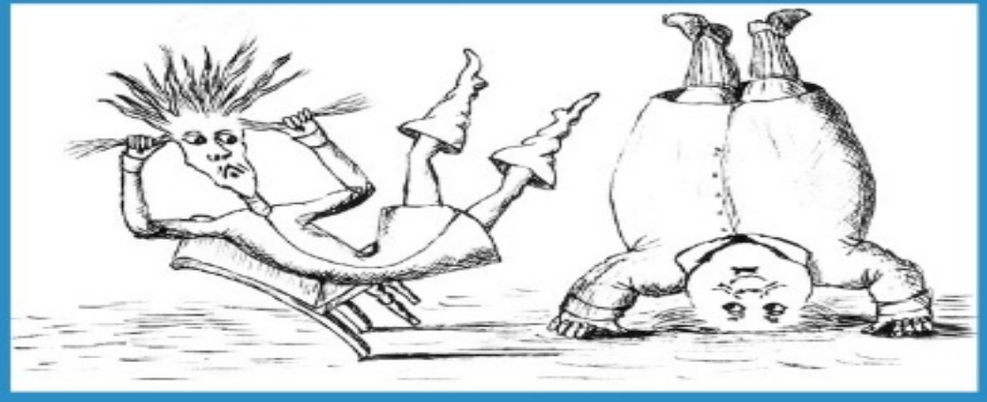
Spark as the Gateway Drug To Typed Functional Programming

Jeff Smith
Rohan Aletty
x.ai

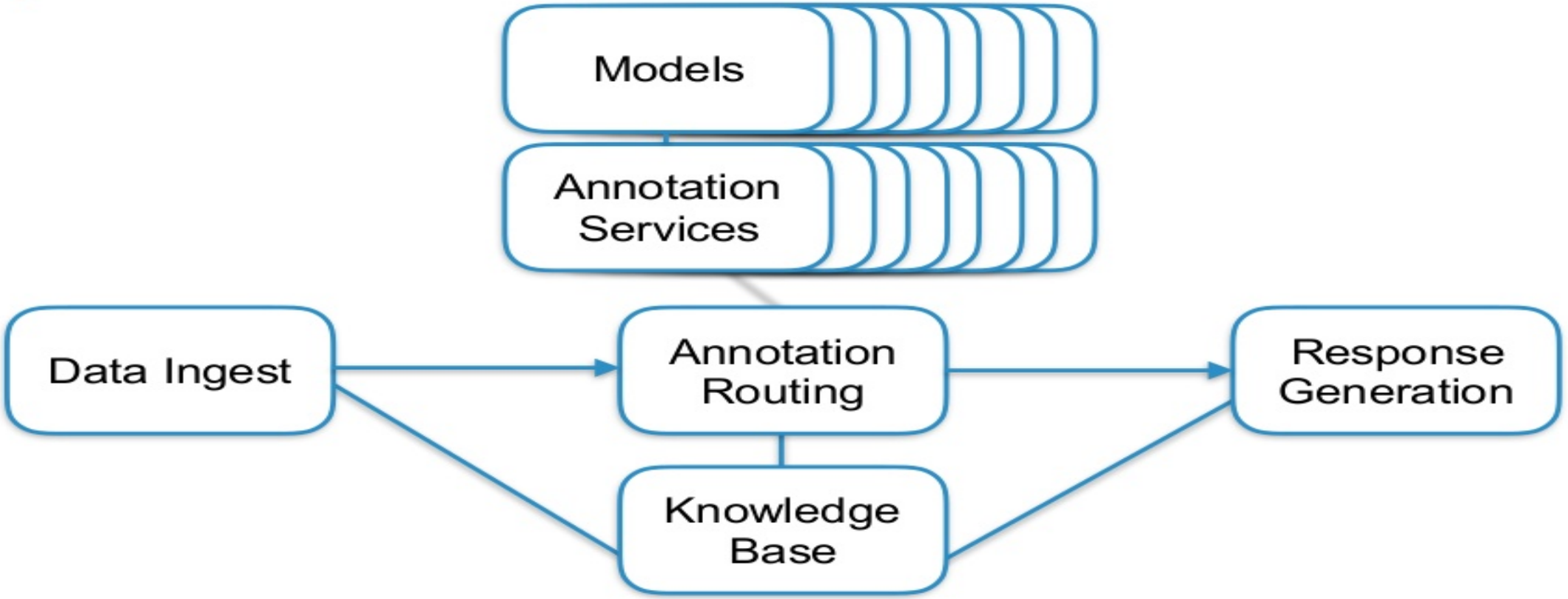


SPARK SUMMIT EAST 2017

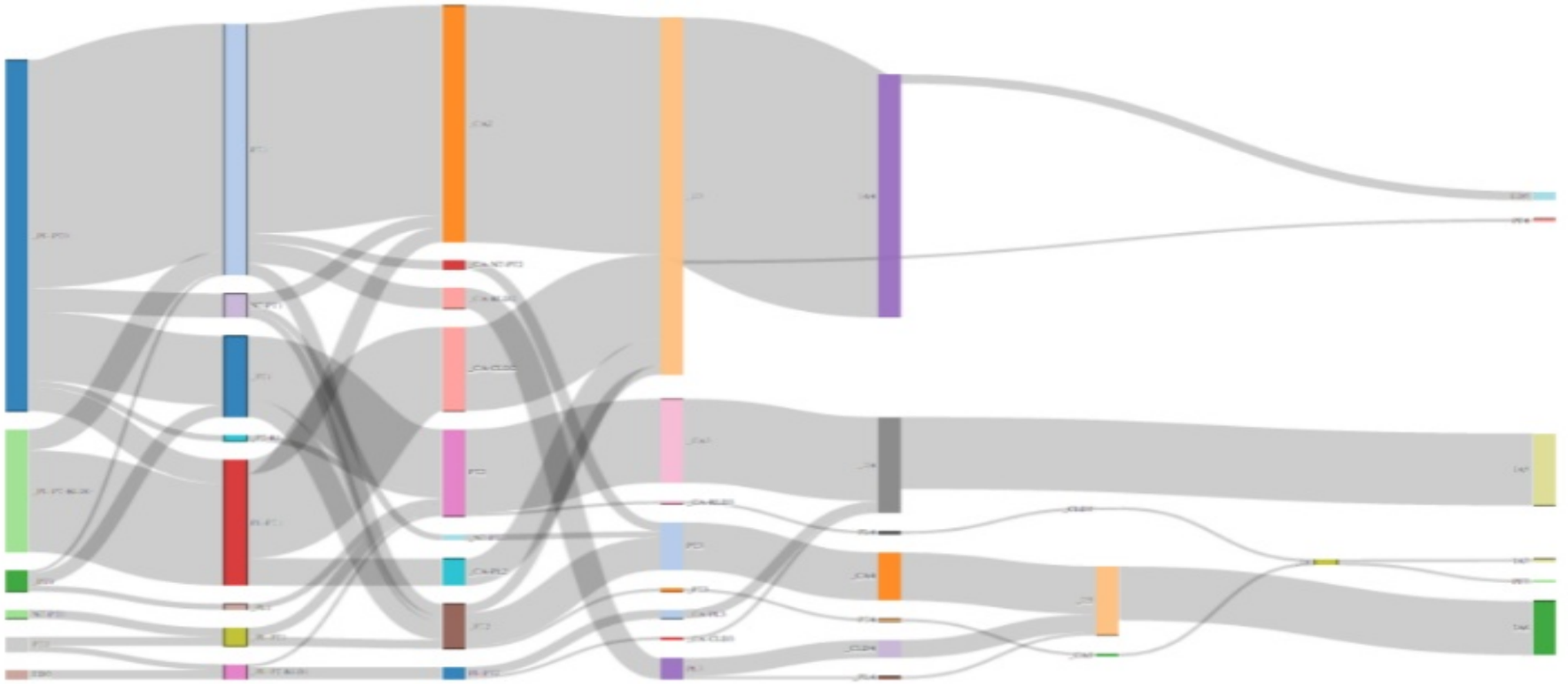
Real World AI

- Scale is increasing
- Complexity is increasing
- Human brain size is constant

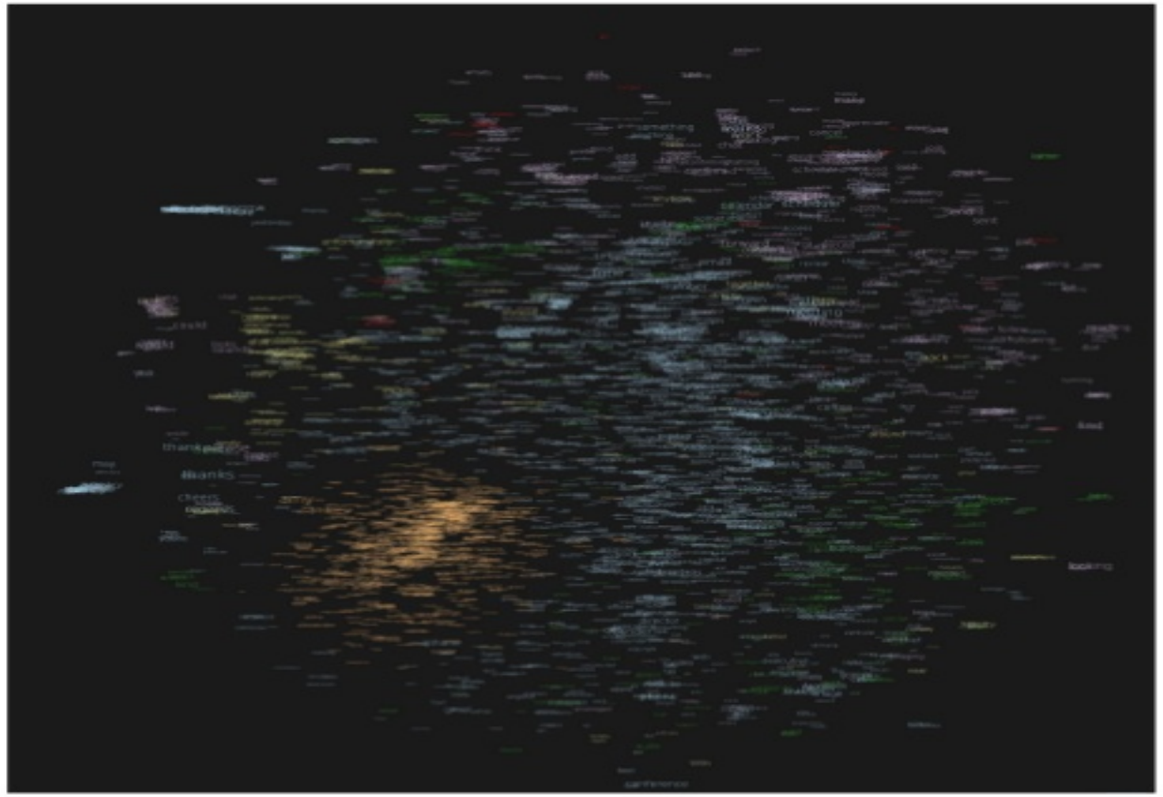
System Complexity



Problem Complexity



Complex Intelligence



Datanauts

Tools

Scala

- Bleeding edge
- Real world

Spark

- Incredibly powerful
- Easy to use

Typed Functional Programming

- Powerful abstractions
- Tough learning curve

Functions

SPARK SUMMIT EAST 2017

Methods

- Collection of statements
- Might have side effects
- On an object

Methods

```
public class Dataset {

    private List<Double> observations;
    private Double average;


    public Dataset(List<Double> inputData) {
        observations = inputData;
    }

}
```

Methods

```
public class Dataset {

    public double getAverage() {
        Double runningSum = 0.0;

        for (Double observation : observations) {
            runningSum += observation;
        }

        average = runningSum / observations.size();

        return average;
    }
}
```

Methods

```
public class Dataset {

    public void setObservations(List<Double> inputData) {
        observations = inputData;
    }
}
```

Methods

```
public class Dataset {

    private List<Double> observations;
    private Double average;


    public Dataset(List<Double> inputData) {
        observations = inputData;
    }

    public double getAverage() {
        Double runningSum = 0.0;

        for (Double observation : observations) {
            runningSum += observation;
        }

        average = runningSum / observations.size();

        return average;
    }

    public void setObservations(List<Double> inputData) {
        observations = inputData;
    }
}
```

Functions

- Collection of expressions
- Returns a value
- Are objects (in Scala)
- Can be in-lined

Functions in Scala

```
val inputData = List(1.0, 2.0, 3.0)
```

Functions in Scala

```
def average(observations: List[Double]) {
  observations.sum / observations.size
}

average(inputData)
```

Functions in Scala

```
def add(x: Double, y: Double) = {
    x + y
}

val sum = inputData.foldLeft(0.0)(add)

val average = sum / inputData.size
```

Functions in Scala

```
val sum = inputData.foldLeft(0.0)(add)

val average = sum / inputData.size

inputData.foldLeft(0.0)(_ + _) / inputData.size
```

Functions in Spark

```
inputData.foldLeft(0.0)(_ + _) / inputData.size

val observations = sc.parallelize(inputData)

observations.fold(0.0)(_ + _) / observations.count()
```

Immutability

SPARK SUMMIT EAST 2017

Mutation

- Changing an object

Mutation

```
visits = {"Church": 2, "Backus": 1, "McCarthy": 4}

old_value = visits["Backus"]

visits["Backus"] = old_value + 1
```

Immutability

- Never changing objects

Immutability in Scala

```
val visits = Map("Church" -> 2, "Backus" -> 1, "McCarthy" -> 4)

val updatedVisits = visits.updated("Backus", 2)
```

Immutability in Spark

```
val manyVisits = sc.parallelize(visits.toSeq)

val additionalVisit = sc.parallelize(Seq(("Backus", 1)))

val updatedVisits = manyVisits.union(additionalVisit)
                              .aggregateByKey(0)(_ + _, _ + _)
```

Recap

SPARK SUMMIT EAST 2017

Concepts

- Higher-order functions
- Anonymous functions
- Purity of functions

Concepts

- Currying
- Referential transparency
- Closures
- Resilient Distributed Datasets

Lazy Evaluation

SPARK SUMMIT EAST 2017

Functional Programming — Lazy Evaluation

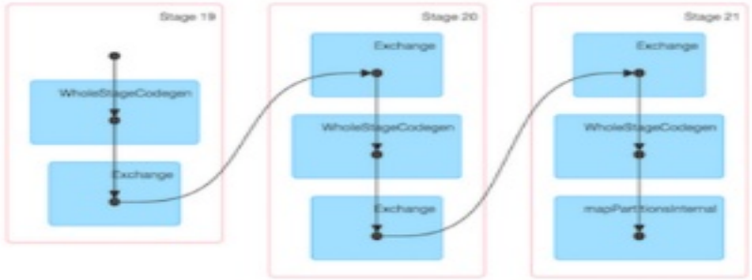
- Delaying evaluation of an expression until a value is needed

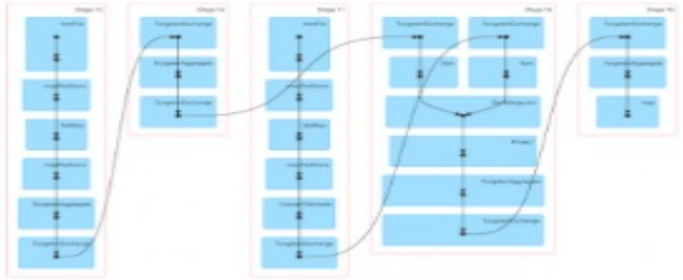
- Two major advantages of lazy evaluation
  - Deferring computation allows program only evaluate what is necessary
  - Changing evaluation scheme into to be more efficient

Spark — Lazy Evaluation

- All transformations are lazy
  - Their existence added to Spark computation DAG

- Example DAGs





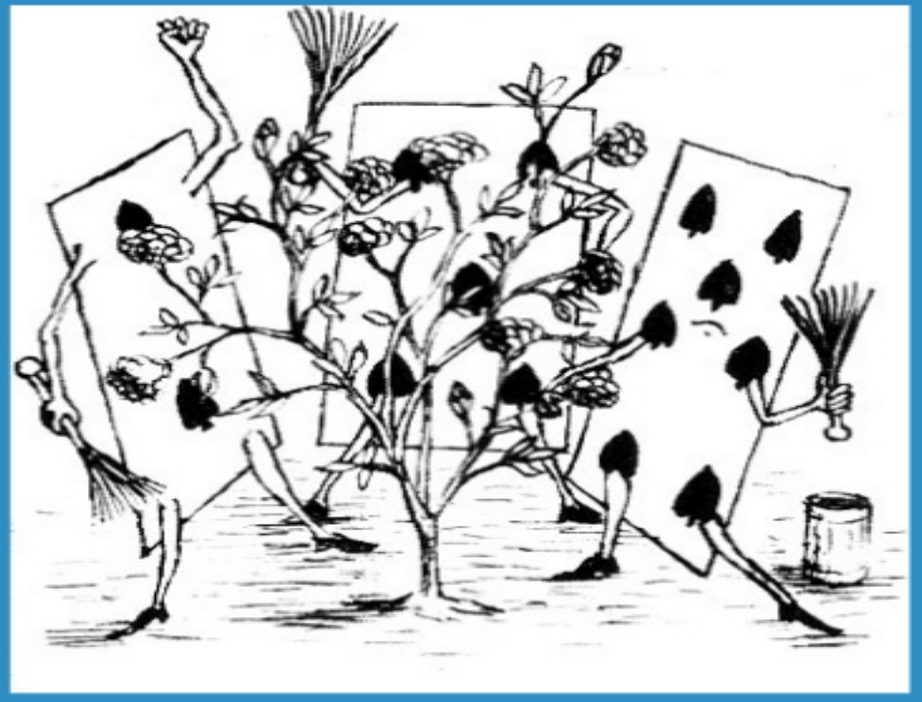
Spark — Lazy Evaluation

```
val rdd1 = sc.parallelize(...)

val rdd2 = rdd1.map(...)

val rdd3 = rdd1.map(...)

val rdd4 = rdd1.map(...)

rdd3.take(5)
```

Spark — Learning Laziness

- Advantage 1: (deferred computation)
  - draws directly from only evaluating parts of DAG that are necessary when executing an action

- Advantage 2: (optimized evaluation scheme)
  - draws directly from pipelining within Spark stages to make execution more efficient



Types

Functional Programming — Type Systems

- Mechanism for defining algebraic data types (ADTs) which are useful for program structure
  - i.e. “let’s group this data together and brand it a new type”
- Compile time guarantees of correctness of program
  - e.g. “no, you cannot add Foo to Bar”

Spark — Types

- RDD's (typed), Datasets (typed), DataFrames (untyped)
- Types provide great schema enforcement on a dataset for preventing unexpected behavior

Spark — Types

```
case class Person(name: String, age: Int)

val peopleDS = spark.read.json(path).as[Person]

val ageGroupedDs = peopleDS.groupBy(_.age)
```

Spark — Learning Types

- Spark through Scala also allows learning of pattern matching
  - ADTs as both product types and union types
- Allows us to reason about code easier
- Gives us compile time safety

Spark — Learning Types

```
trait Person { def name: String }

case class Student(name: String, grade: String) extends Person

case class Professional(name: String, job: String) extends Person

val personRDD: RDD[Person] = sc.parallelize(…)

// working with both union and product types
val mappedRDD: RDD[String] = personRDD.map {
  case Student(name, grade) => grade
  case Professional(name, job) => job
}
```

Spark — Learning Types

```
val rdd1: RDD[Person] = sc.parallelize(...)

val rdd2: RDD[String] = rdd1.map("name: " + _)       // Compilation error!

val rdd3: RDD[String] = rdd2.map("name: " + _.name) // It works!
```

Monads

Functional Programming — Monads

- In category theory:
  - “a monad in X is just a monoid in the category of endofunctors of X”

- In functional programming, refers to a container that can:
  - Inject a value into the container
  - Perform operations on values returning a container with new values
  - Flatten nested containers into a single container

Scala — Monads!

```
trait Monad[M[]] {
  // constructs a Monad instance from the given value, e.g. List(1)
  def apply[T](v: T): M[T]

  // effectively lets you transform values within a Monad
  def bind[T, U](m: M[T])(fn: (T) => M[U]): M[U]
}
```

Scala — Monads!

- Many monads in Scala
  - List, Set, Option, etc.

- Powerful line of thinking
  - Helps code comprehension
  - Reduces error checking logic (pattern matching!)
  - Can build further transformations: map(), filter(), foreach(), etc.

Spark — Learning Monads?

- We have many “computation builders” -- (RDD’s, Datasets, DataFrames)
  - Containers on which transformations can be applied

- Similar to monads though not identical
  - No unit function to wrap constituent values
  - Cannot lift all types into flatMap function unconstrained

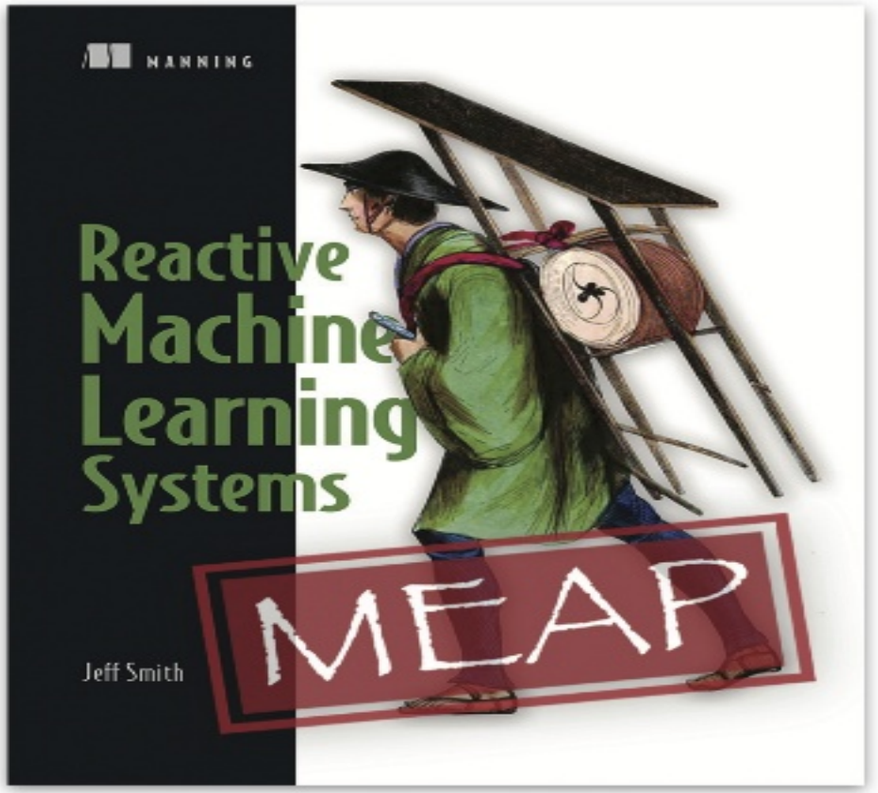
For Later

Conclusions

- Spark introduces all types of devs to Scala
- Scala helps people learn typed functional programming
- Typed functional programming improves Spark development

x.ai
@xdotai
hello@human.x.ai
New York, New York

Use the code
ctwsparks17
for 40% off!



Thank You

SPARK SUMMIT EAST 2017