Applying the Lambda Architecture with Spark, Kafka, and Cassandra

A MODERN BIG DATA ARCHITECTURE

Ahmad Alkilani
DATA ARCHITECT

@akizl

Course Outline

Spark and Spark Streaming

Apache Kafka

Apache Cassandra

What is a Lambda Architecture?

What problems does it address?

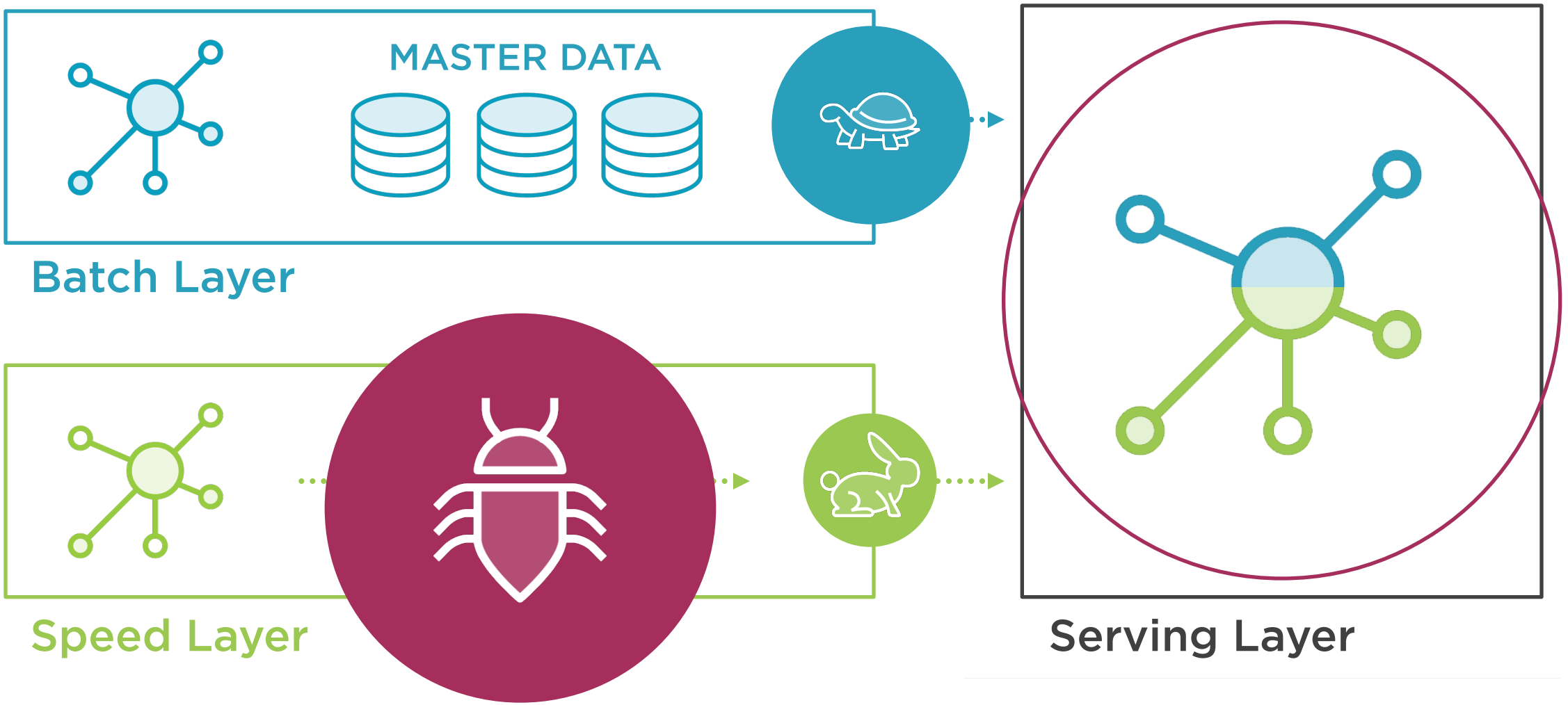
Setup environment and Tools

Introduction to Scala and Spark

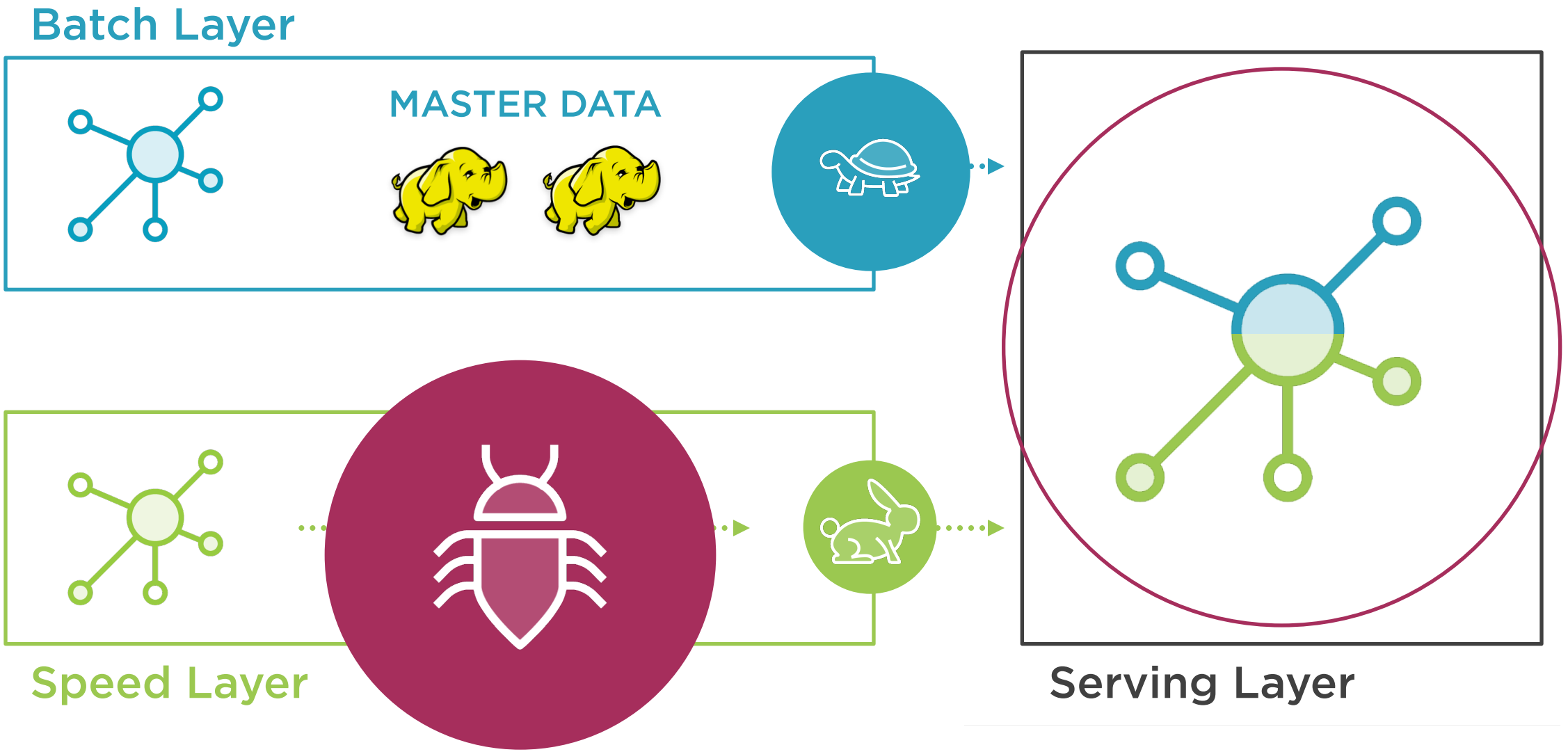
Lambda Architecture λ

Data processing architecture and framework designed to address robustness of the scalability and fault-tolerance (human and machine) of big data systems

Lambda Architecture



Lambda Architecture

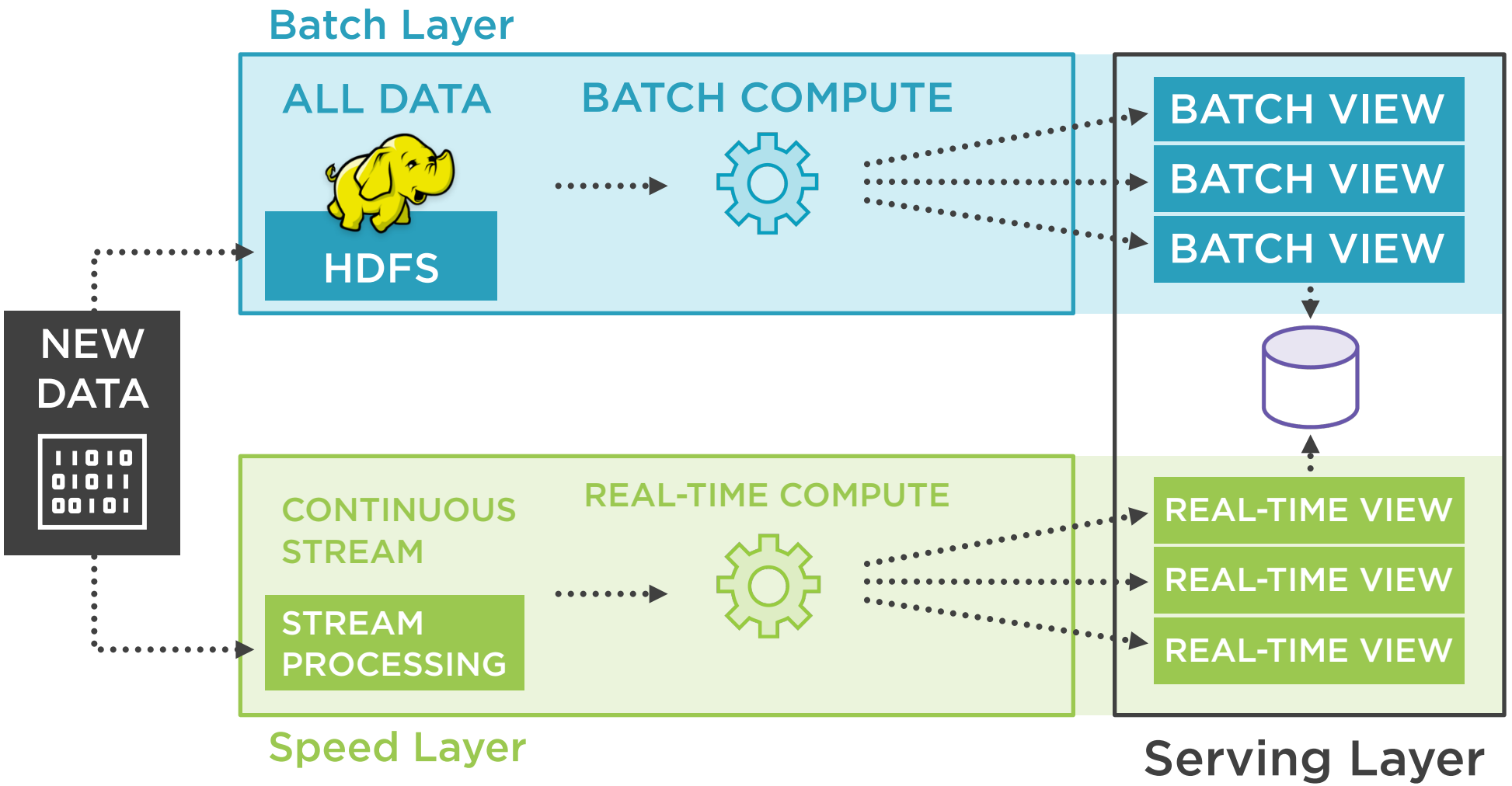


Lambda Architecture

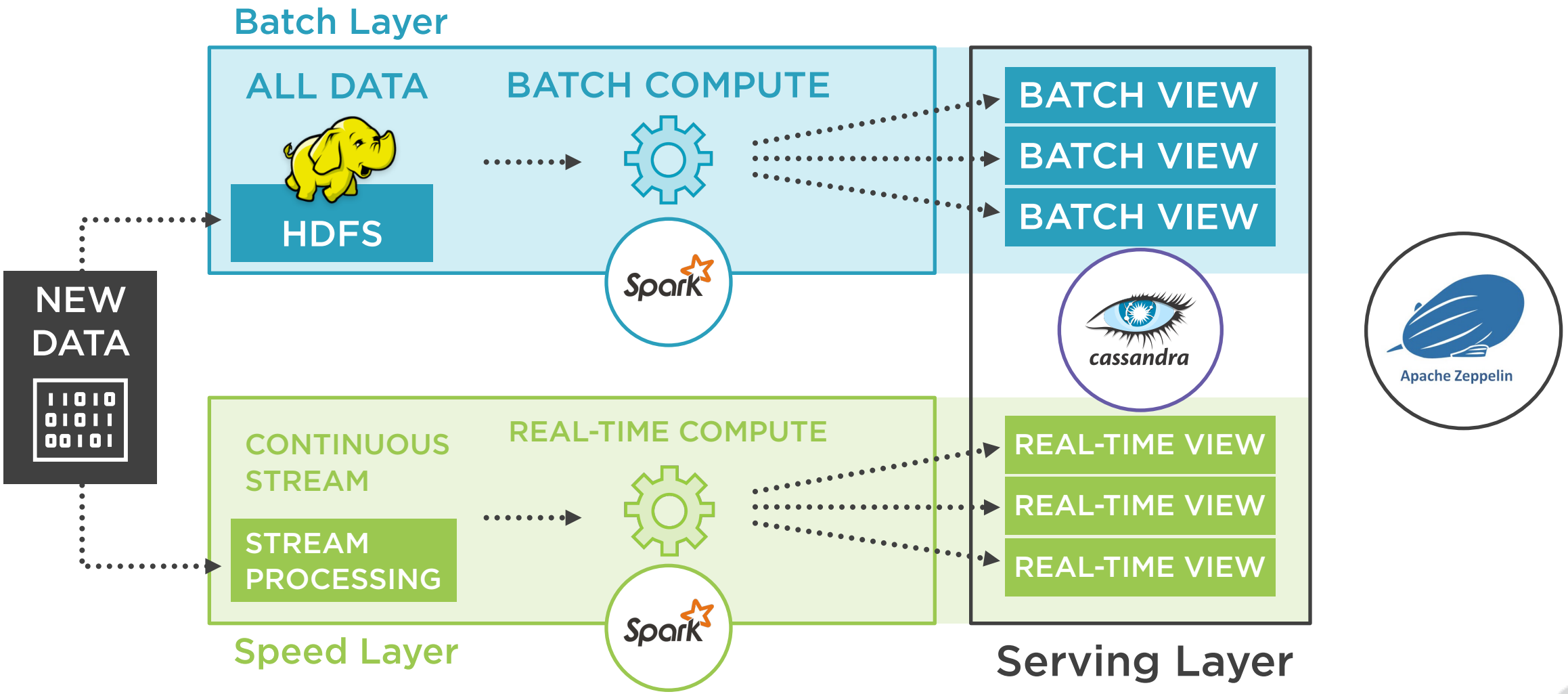
Batch Layer

Speed Layer

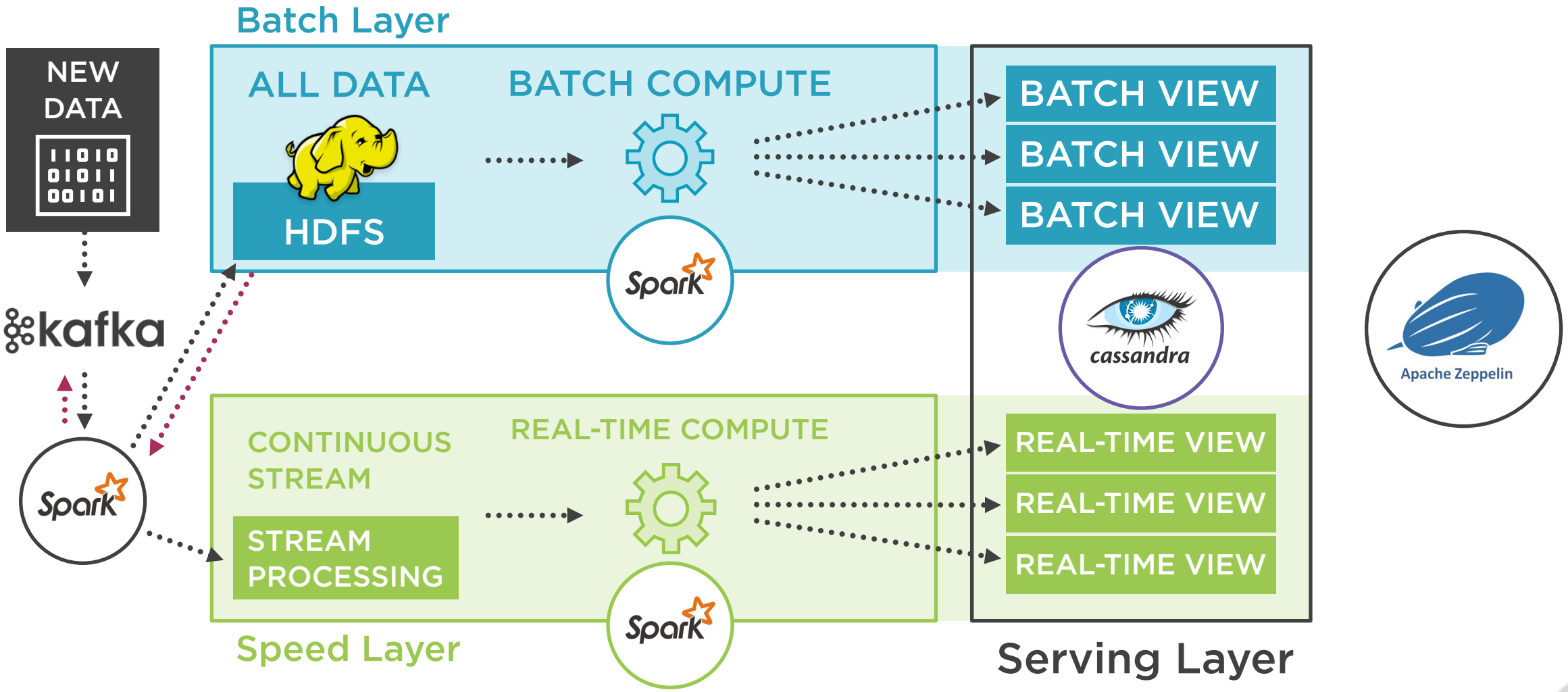
What are we Building



What are we Building



What are we Building



Fast Track to Scala

```
val text : String = "Not another word count example"
text = "try this" (compile error)

var changeMe : String = "This can be changed"
changeMe = "Spark, here I come!"
```

Variable Declaration

- val declares a variable that cannot be changed (preferred). Similar to final in Java
- var declares a variable that can be modified

```scala
val text           = "Not another word count example"

var changeMe           = "This can be changed"
changeMe = "Spark, here I come!"
```

Type Inference

- Use Alt + = in IntelliJ IDE to display snippet of Scala type

```
def sayHello(name: String) : String = {
  s"Hello $name!"
}
```

Function/Method Declaration

- def starts a method declaration
- Return type follows parameter list(s). Drop the “equals” to declare a void
- String interpolation using “**s**” followed by string. Variables placed after **$**
- Every statement in Scala is an expression. No need for “return”

```scala
def sayHello(name: String) = {        = {
  s"Hello $name!"
}
```

With Type Inference

```scala
def sayHello(name: String)(myself: String) = {
  s"Hello $name! My name is $myself"
}

val result = sayHello("Ahmad")("Scala")
```

```
result : java.lang.String = Hello Ahmad! My name is Scala
```

Function Parameter Lists

- Function can have more than one parameter list
- Extremely useful when last parameter is an implicit or another function

```scala
def sayHello(name: String)(whoAreYou : () => String) = {
  s"Hello $name! My name is ${whoAreYou()}"
}

def provideName() = { "Scala" }
val fast = sayHello("test")(provideName)

val faster = sayHello("test") { () =>
  "Anonymous"
}
```

Function Parameter Lists

- Function can have more than one parameter list
- Extremely useful when last parameter is an implicit or another function

```scala
def sayHello(name: String)(implicit myself: String) = {
  s"Hello $name! My name is $myself"
}

implicit val myString = "implicits"
val fast = sayHello("test")
```

Implicits Example 1

- Compiler only looks for implicits in scope
- Implicits can be imported as we'll see with implicit conversions for Spark

```scala
def sayHello(name: String)(implicit whoAreYou:() => String) = {
  s"Hello $name! My name is ${whoAreYou()}"
}

def provideName() = { "Scala" }
val fast = sayHello("test")
```

Implicits Example 2

- Implicit parameters:
  - Must appear in an argument list of their own
  - Argument list must be last

```
class fastTrack(     name: String,     myself: String) {
  def sayHello(name: String)(myself: String) = {
    s"Hello $name! My name is $myself"
  }
  val greeting = sayHello(name)(myself)
}
val fast = new fastTrack("test", "me")
```

Classes

- Classes take parameter list. Parameters declared private vals by default
- Class body is primary constructor

```scala
class fastTrack(val name: String,var myself: String) {
  def sayHello(name: String)(myself: String) = {
    s"Hello $name! My name is $myself"
  }
  val greeting = sayHello(name)(myself)
}
val fast = new fastTrack("test", "me")
println(fast.name)
fast.myself = "fast"
```

Classes

- Parameters can be made public and or vars as needed
- Prepend private var if you need a private variable instead of a public variable myself as in the example

```scala
case class person(fname: String, lname: String)

val me = person("Ahmad", "Alkilani")

println(me.fname)
```

Case Classes

- Special type of Class (syntactic sugar and adds a lot of functionality to a class)
- Implement equality, toString, hashCode methods among others and are serializable
- Don’t require “new” keyword
- Support for pattern matching
- All arguments are prefixed with val by default

Pattern Matching with Case Classes

```
abstract class Person(fname: String, lname: String) {
  def fullName = {s"$fname-$lname"}
}
case class Student(fname: String, lname: String, id: Int)
      extends Person(fname, lname)

val me = Student("Ahmad", "Alkilani", 23)

def getFullID[T <: Person](something: T) = {
  something match {
    case Student(fname, lname, id) => s"$fname-$lname-$id"
    case p: Person => p.fullName
  }
}

getFullID(me)
```

res1: String = Ahmad-Alkilani-23

```
case class person(fname: String, lname: String)

val me = person(“Ahmad", “Alkilani")

println(me.fname)
```

Pattern Matching with Case Classes

```
implicit class stringUtils(myString : String) {
  def scalaWordCount() = {
    val split = myString.split("\\s+")
    val grouped = split.groupBy(word => word)
    val countPerKey = grouped.mapValues(group => group.length)
    countPerKey
  }
}
```

```
res0: scala.collection.immutable.Map[String,Int] = Map(collections -> 2, Spark -> 1, Scala -> 1, mimic -> 1)
```

```
"Spark collections mimic Scala collections".scalaWordCount()
```

Implicit Conversions

- Allow the compiler to implicitly convert one type to another
- Implicit conversion must be in scope
- IDE will typically provide visual indicator (IntelliJ will underline conversion)

Scala Collections

```scala
val myList = List("Spark", "mimics", "Scala", "collections")

val mapped = myList.map( s => s.toUpperCase )
```

mapped: List[String] = List(SPARK, MIMICS, SCALA, COLLECTIONS)

```scala
val flatMapped = myList.flatMap { s =>
  val filters = List("mimics", "collections")
  if (filters.contains(s))
    None
  else
    Some(s)
}
```

flatMapped: List[String] = List(Spark, Scala)

Summary

Lambda Architecture

Batch/Speed/Serving Layers

Spark

Shared Logic w/Batch & Speed Layers

Scala

Parameter Lists, Case Classes, Implicits, Scala Collections

Environment & VM

IntelliJ, Cygwin, Vagrant, VirtualBox

Spark w/Zeppelin

Spark Context, RDD, DataFrame