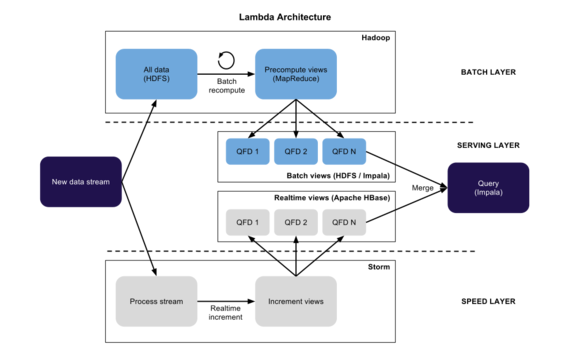


****

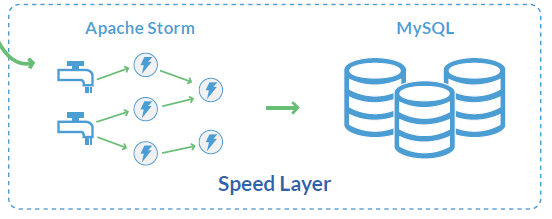
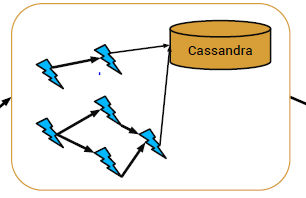
**Serving Layer:**

1. **Raw Data :**
2. **Apache kafka –**

* Low-latency platform for handling real-time, open source stream processing platform.
* Performance : 2 million writes per second
* Zookeeper Required

1. **ActiveMQ**
2. **Real Time & Batch Views :**
   1. **HBase**
   2. **Cassandra**
   3. **mongoDB**

**Speed Layer:**

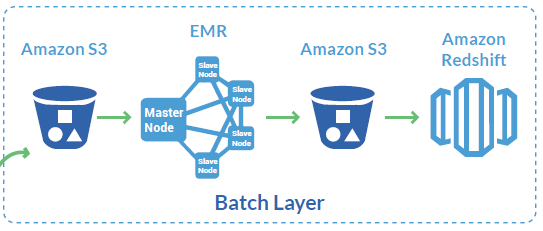
****

1. **Apache Storm (To Process Data and create Real Time views)**

* Apache Storm is focused on stream processing or what some call complex event processing.
* Storm implements a fault tolerant method for performing a computation or pipelining multiple computations on an event as it flows into a system.
* One might use Storm to transform unstructured data as it flows into a system into a desired format.

1. **Cassandra/MySQL/ HBase (To Store Data to use further for queries)**

**Batch Layer:**

****

1. **Apache Spark with HDFS/Amazon S3 (To Process Data and create Batch views)**

* Apache Storm is focused on stream processing or what some call complex event processing.
* If we have to use Amazon S3 then we will have to use Amazon EMR.

1. **Cassandra / HBase (To Store Data to use further for queries)**

**Connectors:**

**-Going to Spark + Kafka + Cassanra – One line code**

**-Main thing is we have to use scala or Java. I will prefer scala.**

1. To take the data in kafka – use kafka source
2. To send data from kafka - use kafka sink
3. Spark Streaming + Kafka Integration Guide (Use messages from kafka queue)
4. Spark Habse connector
5. Spark Cassandra connector – Have support

PySpark Cassandra Connector – Not maintained now

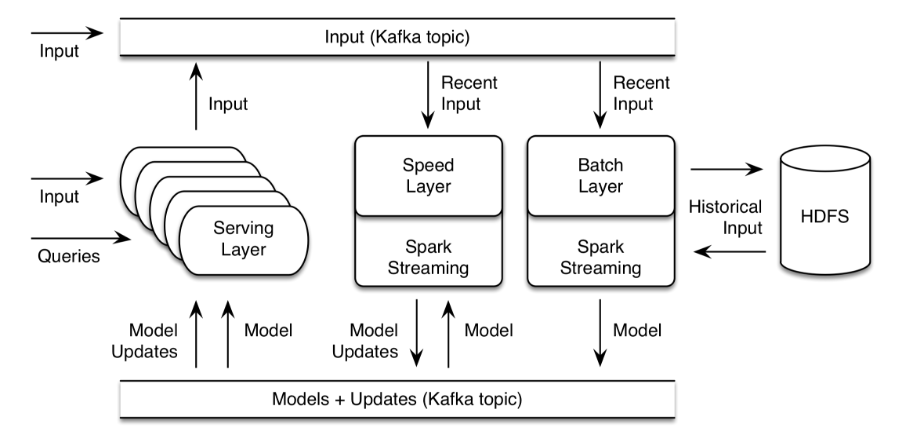
What you are asking about is, in my view, the main problem of implementing a lambda architecture. Here are some suggestions on how to solve it.

The combination of Spark and [Spark Streaming](https://spark.apache.org/docs/1.1.1/streaming-programming-guide.html) largely supersedes the original lambda architecture (which usually involved Hadoop and Storm). [Read here](http://blog.cloudera.com/blog/2014/08/building-lambda-architecture-with-spark-streaming/) an example of how to use a SparkContextand a separate StreamingContext to produce *different* RDDs, one for batch processed results and another for real-time results.

Once you have replicated that in your system, you still have to think about *how to query* both kind of RDDs. The trivial case would be to just union both of them:

scala> rdd1.union(rdd2).collect

Or maybe you can create a new DStream, similar to stateStream in the linked example, where some keys are kept for real-time results, and others for batch results.

****